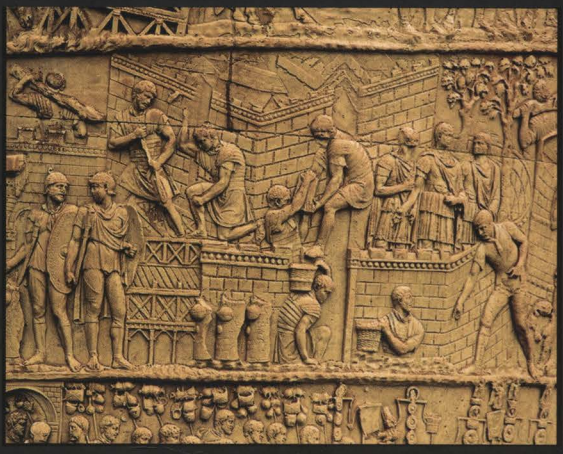
КЛАССИКА COMPUTER SCIENCE

# КОМПЬЮТЕРНЫЕ СЕТИ

## ШЕСТОЕ ИЗДАНИЕ



Э. ТАНЕНБАУМ
Н. ФИМСТЕР
Д. УЭЗЕРОЛЛ

PEARSON Prentice Hall

ПИТЕР

# COMPUTER NETWORKS

SIXTH EDITION

**ANDREW S. TANENBAUM**
*Vrije Universiteit*
*Amsterdam, The Netherlands*

**NICK FEAMSTER**
*University of Chicago*
*Chicago, IL*

**DAVID WETHERALL**
*Google*

# КОМПЬЮТЕРНЫЕ СЕТИ

## ШЕСТОЕ ИЗДАНИЕ

ЭНДРЮ ТАНЕНБАУМ
НИК ФИМСТЕР
ДЭВИД УЭЗЕРОЛЛ

ПИТЕР®

Санкт-Петербург • Москва • Минск

2023

ББК 32.973.202+32.988.02
УДК 004.738.5
Т18

**Таненбаум Эндрю, Фимстер Ник, Уэзеролл Дэвид**

Т18 Компьютерные сети. 6-е изд. — СПб.: Питер, 2023. — 992 с.: ил. — (Серия «Классика computer science»).

ISBN 978-5-4461-1766-6

Перед вами шестое издание самой авторитетной книги по современным сетевым технологиям, написанное признанным экспертом Эндрю Таненбаумом в соавторстве со специалистом компании Google Дэвидом Уэзероллом и профессором Чикагского университета Ником Фимстером. Первая версия этого классического труда появилась на свет в далеком 1980 году, и с тех пор каждое издание книги неизменно становилось бестселлером. В книге последовательно изложены основные концепции, определяющие современное состояние компьютерных сетей и тенденции их развития. Авторы подробно объясняют устройство и принципы работы аппаратного и программного обеспечения, рассматривают все аспекты и уровни организации сетей — от физического до прикладного. Изложение теоретических принципов дополняется яркими, показательными примерами функционирования интернета и компьютерных сетей различного типа. Большое внимание уделяется сетевой безопасности.

Шестое издание полностью переработано с учетом изменений, произошедших в сфере сетевых технологий за последние годы, и, в частности, освещает такие технологии, как DOCSIS, 4G и 5G, беспроводные сети стандарта 802.11ax, 100-гигабитные сети Ethernet, интернет вещей, современные транспортные протоколы CUBIC TCP, QUIC и BBR, программно-конфигурируемые сети и многое другое.

**16+** (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.202+32.988.02
УДК 004.738.5

Права на издание получены по соглашению с Pearson Education Inc. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, такие как Meta Platforms Inc., Facebook, Instagram и др.

ISBN 978-0136764052 англ.

ISBN 978-5-4461-1766-6

© 2021, 2011 by Pearson Education, Inc. or its affiliates, 221 River Street, Hoboken, NJ 07030

© Перевод на русский язык ООО «Прогресс книга», 2023

© Издание на русском языке, оформление ООО «Прогресс книга», 2023

© Серия «Классика computer science», 2023

# Краткое содержание

Предисловие ..... 20

Об авторах ..... 24

Глава 1. Введение ..... 26

Глава 2. Физический уровень ..... 121

Глава 3. Канальный уровень ..... 242

Глава 4. Подуровень управления доступом к среде ..... 312

Глава 5. Сетевой уровень ..... 412

Глава 6. Транспортный уровень ..... 563

Глава 7. Прикладной уровень ..... 684

Глава 8. Сетевая безопасность ..... 813

Глава 9. Рекомендации для чтения и библиография ..... 960

# Оглавление

**Предисловие** ..... **20**

Новое в шестом издании ..... 20
Список аббревиатур ..... 22
Материалы для студентов ..... 22
Благодарности ..... 23

**Об авторах** ..... **24**

От издательства ..... 25

**Глава 1. Введение** ..... **26**

1.1. Применение компьютерных сетей ..... 26
1.1.1. Доступ к информации ..... 27
1.1.2. Общение ..... 30
1.1.3. Электронная коммерция ..... 31
1.1.4. Развлечения ..... 32
1.1.5. Интернет вещей ..... 32
1.2. Типы компьютерных сетей ..... 33
1.2.1. Сети широкополосного доступа ..... 33
1.2.2. Мобильные и беспроводные сети ..... 34
1.2.3. Сети доставки контента ..... 37
1.2.4. Транзитные сети ..... 38
1.2.5. Корпоративные сети ..... 39
1.3. Сетевые технологии, от локальных до глобальных ..... 41
1.3.1. Персональные сети ..... 41
1.3.2. Локальные сети ..... 42

1.3.3. Домашние сети ... 45
1.3.4. Городские сети ... 47
1.3.5. Глобальные сети ... 48
1.3.6. Объединенные сети ... 52
1.4. Примеры сетей ... 53
1.4.1. Интернет ... 53
1.4.2. Мобильные сети ... 65
1.4.3. Беспроводные сети (Wi-Fi) ... 71
1.5. Сетевые протоколы ... 76
1.5.1. Цели проектирования ... 76
1.5.2. Разделение протокола на уровни ... 78
1.5.3. Соединения и надежность ... 82
1.5.4. Примитивы служб ... 85
1.5.5. Службы и протоколы ... 88
1.6. Эталонные модели ... 89
1.6.1. Эталонная модель OSI ... 89
1.6.2. Эталонная модель TCP/IP ... 90
1.6.3. Критика модели и протоколов OSI ... 94
1.6.4. Критика модели и протоколов TCP/IP ... 96
1.6.5. Модель, используемая в этой книге ... 97
1.7. Стандартизация ... 98
1.7.1. Стандартизация и открытый исходный код ... 98
1.7.2. Кто есть кто в мире телекоммуникаций ... 99
1.7.3. Кто есть кто в мире международных стандартов ... 101
1.7.4. Кто есть кто в мире интернет-стандартов ... 104
1.8. Политические, правовые и социальные проблемы ... 106
1.8.1. Свобода слова в интернете ... 106
1.8.2. Сетевой нейтралитет ... 107
1.8.3. Безопасность ... 108
1.8.4. Защита персональной информации ... 109
1.8.5. Дезинформация ... 111
1.9. Единицы измерения ... 111

1.10. Краткий обзор следующих глав ... 112
1.11. Резюме ... 113
Вопросы и задачи ... 115

**Глава 2. Физический уровень ... 121**

2.1. Проводные среды передачи данных ... 121
2.1.1. Запоминающее устройство ... 122
2.1.2. Витая пара ... 123
2.1.3. Коаксиальный кабель ... 125
2.1.4. Линии электропередачи ... 126
2.1.5. Оптоволокно ... 127
2.2. Беспроводная передача данных ... 133
2.2.1. Спектр электромагнитных волн ... 133
2.2.2. Псевдослучайная перестройка рабочей частоты ... 135
2.2.3. Метод прямой последовательности для расширения спектра ... 136
2.2.4. Сверхширокополосная связь ... 137
2.3. Применение спектра электромагнитных волн для передачи данных ... 137
2.3.1. Радиосвязь ... 137
2.3.2. Микроволновая связь ... 139
2.3.3. Передача данных в инфракрасном диапазоне ... 140
2.3.4. Передача данных в видимом диапазоне ... 141
2.4. От форм волн к битам ... 143
2.4.1. Теоретические основы обмена данными ... 143
2.4.2. Максимальная скорость передачи данных по каналу ... 147
2.4.3. Цифровая модуляция ... 149
2.4.4. Мультиплексирование ... 158
2.5. Коммутируемая телефонная сеть общего пользования ... 166
2.5.1. Структура телефонной системы ... 167
2.5.2. Абонентские шлейфы: телефонные модемы, ADSL и оптоволокно ... 170
2.5.3. Соединительные линии и мультиплексирование ... 179
2.5.4. Коммутация ... 186

2.6. Сотовые сети ... 191
2.6.1. Основные понятия: соты, передача обслуживания, пейджинг ... 191
2.6.2. Технология 1G: аналоговая передача голоса ... 193
2.6.3. Технология 2G: цифровая передача голоса ... 195
2.6.4. GSM: Глобальная система мобильной связи ... 196
2.6.5. Технология 3G: цифровая передача голоса и данных ... 200
2.6.6. Технология 4G: коммутация пакетов ... 204
2.6.7. Технология 5G ... 206
2.7. Кабельные сети ... 207
2.7.1. История кабельных сетей: ТВ-системы коллективного приема ... 207
2.7.2. Широкополосный доступ в интернет по кабелю: сети HFC ... 208
2.7.3. DOCSIS ... 211
2.7.4. Совместное использование ресурсов в сетях DOCSIS: узлы и мини-слоты ... 212
2.8. Спутники связи ... 214
2.8.1. Геостационарные спутники ... 216
2.8.2. Среднеорбитальные спутники ... 220
2.8.3. Низкоорбитальные спутники ... 220
2.9. Сравнение различных сетей доступа ... 223
2.9.1. Наземные сети доступа: кабельные, оптоволоконные и ADSL ... 223
2.9.2. Спутники и наземные сети ... 225
2.10. Нормативное регулирование физического уровня ... 227
2.10.1. Распределение частот ... 227
2.10.2. Сотовые сети ... 230
2.10.3. Телефонная сеть ... 231
2.11. Резюме ... 234
Вопросы и задачи ... 235

**Глава 3. Канальный уровень ... 242**

3.1. Ключевые вопросы проектирования канального уровня ... 242
3.1.1. Службы, предоставляемые сетевому уровню ... 243

3.1.2. Формирование фрейма ..... 246
3.1.3. Обработка ошибок ..... 250
3.1.4. Управление потоком ..... 251
3.2. Обнаружение и коррекция ошибок ..... 252
3.2.1. Корректирующие коды ..... 253
3.2.2. Коды для обнаружения ошибок ..... 259
3.3. Элементарные протоколы передачи данных на канальном уровне ..... 266
3.3.1. Исходные упрощающие допущения ..... 266
3.3.2. Базовая схема передачи и приема данных ..... 267
3.3.3. Симплексные протоколы канального уровня ..... 271
3.4. Повышение эффективности ..... 277
3.4.1. Цель: двунаправленная передача, отправка сразу нескольких фреймов ..... 278
3.4.2 Примеры дуплексных протоколов раздвижного окна ..... 280
3.5. Практическое использование протоколов канального уровня ..... 296
3.5.1. Передача пакетов по каналам SONET ..... 296
3.5.2. ADSL ..... 300
3.5.3. DOCSIS ..... 303
3.6. Резюме ..... 305
Вопросы и задачи ..... 306

**Глава 4. Подуровень управления доступом к среде ..... 312**

4.1. Проблема распределения канала ..... 313
4.1.1. Статическое распределение канала ..... 313
4.1.2. Допущения, связанные с динамическим распределением каналов ..... 315
4.2. Протоколы коллективного доступа ..... 317
4.2.1. Система ALOHA ..... 317
4.2.2. Протоколы множественного доступа с контролем несущей ..... 321
4.2.3. Протоколы без коллизий ..... 325
4.2.4. Протоколы с ограниченной конкуренцией ..... 329
4.2.5. Протоколы беспроводных локальных сетей ..... 333
4.3. Сеть Ethernet ..... 336
4.3.1. Физический уровень классического Ethernet ..... 337

4.3.2. Протокол MAC в классическом Ethernet ..... 338
4.3.3. Производительность Ethernet ..... 342
4.3.4. Коммутируемый Ethernet ..... 344
4.3.5. Fast Ethernet ..... 347
4.3.6. Gigabit Ethernet ..... 350
4.3.7. 10-гигабитный Ethernet ..... 354
4.3.8. 40- и 100-гигабитный Ethernet ..... 356
4.3.9. Ретроспективный взгляд на Ethernet ..... 357
4.4. Беспроводные локальные сети ..... 358
4.4.1. Стандарт 802.11: архитектура и стек протоколов ..... 359
4.4.2. Стандарт 802.11: физический уровень ..... 361
4.4.3. Стандарт 802.11: протокол подуровня управления доступом к среде ..... 364
4.4.4. Стандарт 802.11: структура фрейма ..... 371
4.4.5. Службы ..... 373
4.5. Bluetooth ..... 376
4.5.1. Архитектура Bluetooth ..... 376
4.5.2. Применение Bluetooth ..... 377
4.5.3. Стек протоколов Bluetooth ..... 378
4.5.4. Bluetooth: уровень радиосвязи ..... 380
4.5.5. Bluetooth: канальный уровень ..... 380
4.5.6 Bluetooth: структура фрейма ..... 382
4.5.7. Bluetooth 5 ..... 383
4.6. DOCSIS ..... 384
4.6.1. Общие сведения ..... 384
4.6.2. Пристрелка ..... 385
4.6.3. Распределение пропускной способности каналов ..... 385
4.7. Коммутация на канальном уровне ..... 386
4.7.1. Применение мостов ..... 387
4.7.2. Обучающиеся мосты ..... 388
4.7.3. Мосты связующего дерева ..... 391
4.7.4. Повторители, концентраторы, мосты, коммутаторы, маршрутизаторы и шлюзы ..... 394
4.7.5. Виртуальные локальные сети ..... 397

4.8. Резюме ... 404
Вопросы и задачи ... 405

**Глава 5. Сетевой уровень ... 412**

5.1. Вопросы разработки сетевого уровня ... 412
5.1.1. Метод коммутации пакетов с ожиданием ... 412
5.1.2. Службы, предоставляемые транспортному уровню ... 413
5.1.3. Реализация службы без установления соединения ... 415
5.1.4. Реализация службы с установлением соединения ... 416
5.1.5. Сравнение сетей виртуальных каналов и дейтаграммных сетей ... 418
5.2. Алгоритмы маршрутизации в рамках одной сети ... 419
5.2.1. Принцип оптимальности ... 422
5.2.2. Алгоритм поиска кратчайшего пути ... 423
5.2.3. Лавинная адресация ... 426
5.2.4. Маршрутизация по вектору расстояний ... 427
5.2.5. Маршрутизация с учетом состояния линий ... 431
5.2.6. Иерархическая маршрутизация внутри сети ... 437
5.2.7. Широковещательная маршрутизация ... 438
5.2.8. Многоадресная рассылка ... 441
5.2.9. Произвольная маршрутизация ... 444
5.3. Управление трафиком на сетевом уровне ... 445
5.3.1. Необходимость в управлении трафиком: перегрузка ... 445
5.3.2. Методы управления трафиком ... 448
5.4. QoS и QoE приложений ... 463
5.4.1. Требования приложений к QoS ... 463
5.4.2. Избыточное обеспечение ... 465
5.4.3. Планирование пакетов ... 466
5.4.4. Комплексное обслуживание ... 474
5.4.5. Дифференцированное обслуживание ... 477
5.5. Межсетевое взаимодействие ... 480
5.5.1. Интерсети: общие сведения ... 480
5.5.2. Различия сетей ... 481
5.5.3. Объединение гетерогенных сетей ... 483

5.5.4. Соединение конечных точек в гетерогенных сетях ... 485
5.5.5. Межсетевая маршрутизация ... 487
5.5.6. Поддержка различных размеров пакета: фрагментация пакета ... 488
5.6. Программно-конфигурируемые сети ... 492
5.6.1. Общие сведения ... 492
5.6.2. Плоскость управления в SDN: логически централизованное программное управление ... 493
5.6.3. Плоскость данных в SDN: программируемое оборудование ... 495
5.6.4. Программируемая сетевая телеметрия ... 498
5.7. Сетевой уровень интернета ... 499
5.7.1. Протокол IP версии 4 ... 501
5.7.2. IP-адреса ... 505
5.7.3. Протокол IP версии 6 ... 519
5.7.4. Управляющие протоколы интернета ... 530
5.7.5. Коммутация меток и MPLS ... 535
5.7.6. Протокол внутреннего шлюза OSPF ... 538
5.7.7. Протокол внешнего шлюза BGP ... 544
5.7.8. Многоадресная интернет-рассылка ... 551
5.8. Политика сетевого уровня ... 552
5.8.1. Пиринговые споры ... 552
5.8.2. Приоритизация трафика ... 553
5.9. Резюме ... 554
Вопросы и задачи ... 556

**Глава 6. Транспортный уровень ... 563**

6.1. Транспортные службы ... 563
6.1.1. Службы, предоставляемые верхним уровням ... 563
6.1.2. Примитивы транспортных служб ... 565
6.1.3. Сокеты Беркли ... 569
6.1.4. Пример программирования сокета: файл-сервер для интернета ... 571
6.2. Элементы транспортных протоколов ... 575
6.2.1. Адресация ... 577

6.2.2. Установление соединения ..... 580
6.2.3. Разрыв соединения ..... 586
6.2.4. Контроль ошибок и управление потоком данных ..... 590
6.2.5. Мультиплексирование ..... 596
6.2.6. Восстановление после сбоев ..... 597
6.3. Контроль перегрузки ..... 599
6.3.1. Выделение требуемой пропускной способности ..... 599
6.3.2. Регулирование скорости отправки ..... 604
6.3.3. Проблемы беспроводного соединения ..... 608
6.4. Транспортные протоколы интернета: UDP ..... 611
6.4.1. Основы UDP ..... 611
6.4.2. Вызов удаленной процедуры ..... 613
6.4.3. Транспортные протоколы реального времени ..... 616
6.5. Транспортные протоколы интернета: TCP ..... 623
6.5.1. Основы TCP ..... 623
6.5.2. Модель службы TCP ..... 624
6.5.3. Протокол TCP ..... 626
6.5.4. Заголовок TCP-сегмента ..... 628
6.5.5. Установка TCP-соединения ..... 631
6.5.6. Разрыв TCP-соединения ..... 633
6.5.7. Модель управления TCP-соединением ..... 633
6.5.8. Раздвижное окно TCP ..... 636
6.5.9. Управление таймерами в TCP ..... 640
6.5.10. Контроль перегрузки в TCP ..... 643
6.5.11. CUBIC TCP ..... 654
6.6. Транспортные протоколы и контроль перегрузки ..... 654
6.6.1. QUIC ..... 655
6.6.2. BBR: контроль перегрузки на основе пропускной способности узких мест ..... 655
6.6.3. Будущее протокола TCP ..... 658
6.7. Вопросы производительности ..... 658
6.7.1. Проблемы производительности компьютерных сетей ..... 659
6.7.2. Оценка производительности сети ..... 660
6.7.3. Оценка пропускной способности сетей доступа ..... 660

6.7.4. Оценка QoE ... 662
6.7.5. Проектирование хостов для быстрых сетей ... 663
6.7.6. Быстрая обработка сегментов ... 666
6.7.7. Сжатие заголовков ... 670
6.7.8. Протоколы для протяженных сетей с высокой пропускной способностью ... 672
6.8. Резюме ... 676
Вопросы и задачи ... 677

**Глава 7. Прикладной уровень ... 684**

7.1. Служба имен доменов DNS ... 684
7.1.1. История и общие сведения ... 685
7.1.2. Процесс поиска DNS ... 685
7.1.3. Пространство имен и иерархия DNS ... 688
7.1.4. DNS-запросы и DNS-ответы ... 692
7.1.5. Разрешение имен ... 699
7.1.6. Практическое ознакомление с DNS ... 701
7.1.7. DNS и конфиденциальность ... 701
7.1.8. Разногласия по поводу способов именования ... 704
7.2. Электронная почта ... 704
7.2.1. Архитектура и службы ... 706
7.2.2. Пользовательский агент ... 708
7.2.3. Форматы сообщений ... 710
7.2.4. Пересылка сообщений ... 716
7.2.5. Окончательная доставка ... 721
7.3. Всемирная паутина ... 725
7.3.1. Представление об архитектуре ... 726
7.3.2. Статичные веб-объекты ... 735
7.3.3. Динамические веб-страницы и веб-приложения ... 736
7.3.4. HTTP и HTTPS ... 740
7.3.5. Конфиденциальность в интернете ... 753
7.4. Потоковая передача аудио и видео ... 758
7.4.1. Цифровой звук ... 759
7.4.2. Цифровое видео ... 762

7.4.3. Потоковая передача сохраненных медиафайлов ..... 766
7.4.4. Потоковая передача в реальном времени ..... 774
7.5. Доставка контента ..... 785
7.5.1. Контент и интернет-трафик ..... 786
7.5.2. Серверные фермы и веб-прокси ..... 788
7.5.3. Сети доставки контента ..... 792
7.5.4. Одноранговые сети ..... 797
7.5.5. Эволюция интернета ..... 803
7.6. Резюме ..... 807
Вопросы и задачи ..... 808

**Глава 8. Сетевая безопасность ..... 813**

8.1. Основы сетевой безопасности ..... 815
8.1.1. Базовые принципы безопасности ..... 817
8.1.2. Базовые принципы проведения атак ..... 819
8.1.3. Методы борьбы с угрозами ..... 821
8.2. Основные компоненты атаки ..... 822
8.2.1. Разведка ..... 822
8.2.2. Прослушивание и перехват (и немного подмены данных) ..... 825
8.2.3. Подмена данных (помимо ARP) ..... 827
8.2.4. Нарушение работы ..... 840
8.3. Брандмауэры и системы обнаружения вторжений ..... 844
8.3.1. Брандмауэры ..... 844
8.3.2. Обнаружение и предотвращение вторжений ..... 847
8.4. Криптография ..... 852
8.4.1. Основы криптографии ..... 852
8.4.2. Два базовых принципа криптографии ..... 855
8.4.3. Подстановочные шифры ..... 857
8.4.4. Перестановочные шифры ..... 859
8.4.5. Одноразовые блокноты ..... 860
8.5. Алгоритмы с симметричным ключом ..... 866
8.5.1. Стандарт шифрования данных DES ..... 867

8.5.2. Улучшенный стандарт шифрования AES ... 868
8.5.3. Режимы шифрования ... 870
8.6. Алгоритмы с открытым ключом ... 875
8.6.1. Алгоритм RSA ... 876
8.6.2. Другие алгоритмы с открытым ключом ... 878
8.7. Цифровые подписи ... 879
8.7.1. Подписи с симметричным ключом ... 880
8.7.2. Подписи с открытым ключом ... 881
8.7.3. Профили сообщений ... 883
8.7.4. Атака «дней рождения» ... 886
8.8. Управление открытыми ключами ... 888
8.8.1. Сертификаты ... 889
8.8.2. X.509 ... 890
8.8.3. Инфраструктуры систем с открытыми ключами ... 892
8.9. Протоколы аутентификации ... 895
8.9.1. Аутентификация на основе общего секретного ключа ... 896
8.9.2. Установка общего ключа: протокол обмена ключами Диффи — Хеллмана ... 901
8.9.3. Аутентификация с помощью центра распространения ключей ... 904
8.9.4. Аутентификация при помощи протокола Kerberos ... 907
8.9.5. Аутентификация путем шифрования с открытым ключом ... 910
8.10. Защита соединений ... 911
8.10.1. IPsec ... 911
8.10.2. Виртуальные частные сети ... 915
8.10.3. Безопасность в беспроводных сетях ... 917
8.11. Безопасность электронной почты ... 921
8.11.1. PGP ... 921
8.11.2. S/MIME ... 926
8.12. Веб-безопасность ... 926
8.12.1. Угрозы ... 927
8.12.2. Безопасное именование ресурсов и DNSSEC ... 928
8.12.3. Протокол TLS ... 931
8.12.4. Выполнение недоверенного кода ... 935

8.13. Социальные вопросы ..... 938
8.13.1. Приватная и анонимная коммуникация ..... 938
8.13.2. Свобода слова ..... 942
8.13.3. Авторское право ..... 946
8.14. Резюме ..... 949
Вопросы и задачи ..... 951

**Глава 9. Рекомендации для чтения и библиография ..... 960**

9.1. Литература для дальнейшего чтения ..... 960
9.1.1. Введение ..... 961
9.1.2. Физический уровень ..... 962
9.1.3. Канальный уровень ..... 963
9.1.4. Подуровень управления доступом к среде ..... 964
9.1.5. Сетевой уровень ..... 965
9.1.6. Транспортный уровень ..... 966
9.1.7. Прикладной уровень ..... 967
9.1.8. Сетевая безопасность ..... 968
9.2. Алфавитный список литературы ..... 970

Сьюзан, Барбаре, Дэниелу, Арону, Нейтану,
Марвину, Матильде, Оливии и Мирте
*Э. Таненбаум*

Маршини, Миле и Кире
*Н. Фимстер*

Кэтрин, Люси и Пеппер
*Д. Уэзеролл*

# Предисловие

Вот и наступил черед шестого издания нашей книги. Каждая ее предыдущая версия соответствовала определенному этапу эволюции компьютерных сетей. В 1980 году, когда вышло первое издание, сети представляли собой скорее диковинку, интересную больше с теоретической точки зрения. При выходе второго издания, в 1988-м, сети использовались в университетах и крупных компаниях. В год публикации третьего издания, 1996-й, компьютерные сети, особенно интернет, уже стали повседневной реальностью для миллионов людей. К моменту выхода четвертого издания, в 2003-м, стал вполне обыденным доступ в интернет через беспроводные сети и мобильные компьютеры. К пятому изданию на первый план в этой сфере вышло распределение контента (особенно видеоконтента — при помощи CDN и p2p-сетей) и мобильные телефоны. Теперь, на момент выхода шестого издания, основной акцент в отрасли делается на очень высокую производительность, благодаря использованию сотовых сетей 5G, 100-гигабитной сети Ethernet и Wi-Fi 802.11ax, так что скорости до 11 Гбит/с уже не за горами.

## НОВОЕ В ШЕСТОМ ИЗДАНИИ

Среди множества внесенных в эту книгу изменений важнейшим, конечно, является включение в число ее соавторов профессора Ника Фимстера (Nick Feamster). Ник Фимстер получил степень Ph.D. в Массачусетском технологическом институте и сейчас занимает должность профессора в Чикагском университете.

Еще одна важная доработка состоит в том, что профессор Херберт Бос (Herbert Bos) из Амстердамского свободного университета коренным образом переписал главу 8 (посвященную безопасности), сместив в ней акцент с криптографии на сетевую безопасность. Практически каждый день в новостях обсуждают компьютерный взлом, DoS-атаки и т. п., так что мы очень благодарны проф. Босу за переработку главы с упором на подробное обсуждение этих вопросов. Здесь описаны уязвимости, их исправление, реакция взломщиков на эти меры, ответная реакция защитников системы и далее до бесконечности. Посвященный криптографии материал был несколько сокращен, чтобы освободить место для новых материалов по сетевой безопасности.

Конечно, в эту книгу было внесено множество других изменений, отражающих постоянно меняющийся мир компьютерных сетей. Основные из них перечислены ниже, по главам.

**Глава 1** является вводной, как и в предыдущих изданиях, но ее содержимое было пересмотрено и актуализировано. Среди изменений: дополнительное обсуждение интернета вещей и современных архитектур сотовых сетей, включая сети 4G и 5G. Также был существенно обновлен раздел, посвященный политике в отношении интернета, особенно обсуждение сетевого нейтралитета.

Доработка **главы 2** включает обсуждение наиболее распространенных физических сред для сетей доступа, включая DOCSIS и различные оптоволоконные архитектуры. В этой главе были дополнительно освещены вопросы современных архитектур и технологий сотовых сетей, а также серьезно модифицирован раздел, посвященный спутниковым сетям. Появилось описание перспективной технологии виртуализации, в том числе обсуждение операторов мобильных виртуальных сетей и сегментации сотовых сетей. Раздел о нормативном регулировании переработан, добавлено обсуждение вопросов, связанных с беспроводными сетями (например, о диапазонах частот).

В **главе 3** в качестве примера протокола добавлена широко используемая технология доступа DOCSIS. Большинство корректирующих кодов, конечно, актуальности с течением времени не теряют.

**Глава 4** актуализирована и дополнена новыми материалами по 40- и 100-гигабитной сети Ethernet, протоколам 802.11ac, 802.11ad и 802.11ax. В нее вошли новые материалы по DOCSIS, рассказывающие о подуровне MAC кабельных сетей. Мы исключили материал о 802.16, так как эта технология постепенно уступает место 4G и 5G. Чтобы освободить место для новой информации, был также исключен раздел, посвященный RFID, как не связанный непосредственно с сетями.

**Глава 5** обновлена, чтобы внести ясность в вопросы перегруженности сети в соответствии с современным положением дел. Переработаны разделы, посвященные управлению трафиком, его формированию и регулированию. Кроме того, появился совершенно новый раздел о программно-конфигурируемых сетях (SDN), включая OpenFlow, и программируемом аппаратном обеспечении (например, Tofino). Данная глава включает обсуждение новейших сценариев применения SDN, например внутриполосной телеметрии сети. Также были внесены некоторые изменения в рассказ об IPv6.

**Глава 6** серьезно отредактирована, в нее вошел новый раздел по современным транспортным протоколам, включая TCP CUBIC, QUIC и BBR. Полностью переписан материал об измерениях производительности сети с упором на оценку пропускной способности компьютерных сетей. Добавлено развернутое обсуждение проблем измерения эффективности сетей доступа при росте скоростей, предоставляемых интернет-провайдерами. Также эта глава включает новый материал, посвященный передовому направлению измерения производительности — оценке QoE.

Существенно отредактирована и **глава 7**. Из нее исключено более 60 уже неактуальных страниц материала. Практически полностью переписана часть, касающаяся системы DNS, чтобы отразить новейшие разработки в этой области, включая текущую тенденцию к шифрованию DNS и усовершенствованию ее безопасности в целом. Описаны новейшие протоколы, например DNS поверх

HTTPS и другие методы защиты персональной информации для DNS. Значительно переработано обсуждение Всемирной паутины с учетом растущего применения в ней шифрования, а также распространения серьезных угроз приватности (например, отслеживание пользователей). В главу включен совершенно новый раздел, посвященный приватности во Всемирной паутине. Кроме того, расширен материал о современных технологиях (сетях) доставки контента и пиринговых сетях. Отредактирован раздел, посвященный эволюции интернета, с тем чтобы осветить тенденции к переходу на распределенные облачные сервисы.

**Глава 8**, посвященная безопасности, полностью переработана. В предыдущих изданиях основное внимание в ней было сосредоточено на защите информации криптографическими средствами. Но криптография — лишь один из аспектов безопасности сетей, причем на практике обычно не самый проблемный. Чтобы исправить это упущение, мы добавили новые материалы по принципам безопасности, главным методам сетевых атак, механизмам защиты и широкому спектру системных проблем защиты информации. Более того, мы обновили уже имеющиеся разделы, исключив из них некоторые устаревшие методики шифрования, и теперь знакомим читателя с более современными версиями протоколов и стандартов.

В главе 9 вы найдете обновленный список рекомендуемой литературы и обширную библиографию.

Кроме того, в книгу были включены десятки новых упражнений и библиографических ссылок.

## СПИСОК АББРЕВИАТУР

Компьютерные книги полны аббревиатур. И эта книга — не исключение. Когда вы закончите ее читать, вам должны быть знакомы следующие аббревиатуры: AES, AMI, ARP, ARQ, ASK, BGP, BSC, CCK, CDM, CDN, CRL, DCF, DES, DIS, DMT, DMZ, DNS, EAP, ECN, EPC, FDD, FDM, FEC, FSK, GEO, GSM, HFC, HLR, HLS, HSS, IAB, IDS, IGP, IKE, IPS, ISM, ISO, ISP, ITU, IXC, IXP, KDC, LAN, LCP, LEC, LEO, LER, LLD, LSR, LTE, MAN, MEO, MFJ, MGW, MIC, MME, MPD, MSC, MSS, MTU, NAP, NAT , NAV, NCP, NFC, NIC, NID, NRZ, ONF, OSI, PAR, PCF, PCM, PCS, PGP, PHP, PIM, PKI, PON, POP, PPP, PSK, RAS, RCP, RED, RIP, RMT, RNC, RPC, RPR, RTO, RTP, SCO, SDH, SDN, SIP, SLA, SNR, SPE, SSL, TCG, TCM, TCP, TDM, TLS, TPM, UDP, URL, USB, UTP, UWB, VLR, VPN, W3C, WAF, WAN, WDM, WEP, WFQ и WPA. Не беспокойтесь — каждая аббревиатура выделена **жирным шрифтом** и расшифрована. Ради забавы можно подсчитать количество известных вам аббревиатур *до* знакомства с данной книгой. Запишите результат на полях, а затем попробуйте повторить подсчет *после* прочтения.

## МАТЕРИАЛЫ ДЛЯ СТУДЕНТОВ

Авторы поддерживают веб-сайт с дополнительными ресурсами для студентов по адресу www.computernetworksbook.com.

# БЛАГОДАРНОСТИ

Во время подготовки шестого издания данной книги нам помогало множество людей. Мы хотели бы особо поблагодарить Филлис Дэвис (Phyllis Davis, Муниципальный колледж Сент-Луиса), Фара Канда (Farah Kandah, Университет Теннесси, Чаттануга), Джейсона Ливингуда (Jason Livingood) из компании Comcast, Луизу Мозер (Louise Moser, Калифорнийский университет, Санта-Барбара), Дженнифер Рексфорд (Jennifer Rexford, Принстонский университет), Пола Шмитта (Paul Schmitt, Принстонский университет), Дага Сикера (Doug Sicker, Университет Карнеги — Меллона), Вэнье Вана (Wenye Wang, Университет штата Северная Каролина) и Грэга Уайта (Greg White) из компании Cable Labs.

Ценные отзывы и замечания по рукописи и идеи мы получили от некоторых студентов профессора Таненбаума, в их числе: Эдже Додженер (Ece Doganer), Яэль Гуде (Yael Goede), Бруно Ховелакен (Bruno Hoevelaken), Элена Иби (Elena Ibi), Оскар Клоновски (Oskar Klonowski), Йоханна Сэнгер (Johanna Sänger), Тереза Шанц (Theresa Schantz), Карлис Свиланс (Karlis Svilans), Маша ван дер Марель (Mascha van der Marel), Энтони Уилкс (Anthony Wilkes).

Многие из новых упражнений в конце глав придуманы, на радость читателю, Джессе Донкервлитом (Jesse Donkervliet, Амстердамский свободный университет).

Слайды лекций в PowerPoint для преподавателей создал Пол Нэйджин (Paul Nagin) из издательства Chimborazo Publishing.

Наш редактор из издательства Pearson Трэйси Джонсон (Tracy Johnson), как обычно, помогала нам решать множество серьезных и мелких проблем. Без ее советов, энергичности и настойчивости это издание могло вообще не появиться на свет. Спасибо тебе, Трэйси. Мы очень ценим твою помощь.

И наконец, пришел черед самых важных для нас людей. Сьюзан пережила этот процесс уже 23 раза, всякий раз с неизменным терпением и любовью. Барбара и Марвин теперь уже знают разницу между хорошим и плохим учебником и всегда вдохновляют меня на написание хороших. Даниэл и Матильда — замечательное прибавление к нашему семейству. Арон, Нейтан, Оливия и Мирте, вероятно, это издание не прочитают, но они вдохновили меня в надежде на будущее (ЭТ). Маршини, Мила и Кира: моя любимая сеть — та, которую мы построили вместе. Спасибо вам за поддержку и любовь (НФ). Кэтрин и Люси не только всецело поддерживали меня, но и ухитрялись всегда обеспечить мне хорошее настроение. Спасибо вам (ДУ).

*Эндрю Таненбаум*
*Ник Фимстер*
*Дэвид Уэзеролл*

# Об авторах

**Эндрю Таненбаум** получил степень бакалавра естественных наук в Массачусетском технологическом институте и защитил докторскую диссертацию в Калифорнийском университете в Беркли. В настоящее время является почетным профессором компьютерных наук Амстердамского свободного университета, где преподает курсы по организации операционных систем, компьютерным сетям и смежным темам вот уже более 40 лет. Долгие годы Таненбаум изучал высоконадежные операционные системы, а также компиляторы, распределенные системы, безопасность и др. Результат его исследовательских проектов — более 200 рецензированных статей в журналах и докладов на конференциях.

Профессор Таненбаум является автором и соавтором пяти книг, которые были переизданы 24 раза и переведены на 21 язык, включая баскский, китайский, французский, немецкий, японский, корейский, румынский, сербский, испанский и тайский. Его книги изучают в университетах по всему миру.

Эндрю Таненбаум разработал Unix-подобную систему Minix, предназначенную для студенческих лабораторных работ по программированию. Она послужила вдохновением и платформой для создания операционной системы Linux.

Таненбаум является членом Ассоциации вычислительной техники (Association for Computing Machinery, ACM), Института инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers, IEEE), а также Королевской академии искусств и наук Нидерландов. Его достижения отмечены множеством научных премий от ACM, IEEE и Ассоциации USENIX (полный список вы найдете на его странице в Википедии). Кроме того, Таненбаум удостоен двух почетных докторских степеней.

Домашняя страница Эндрю Таненбаума находится по адресу www.cs.vu.nl/~ast.

**Ник Фимстер**, профессор компьютерных наук, возглавляет Центр данных и вычислений (Center for Data and Computing, CDAC) в Чикагском университете. Его исследования касаются многих вопросов компьютерных сетей и сетевых систем. Особое внимание он уделяет сетевым операциям, сетевой безопасности, цензуре в интернете и применению машинного обучения в компьютерных сетях.

Ник Фимстер окончил Массачусетский технологический институт: в 2000 и 2001 годах он получил степени бакалавра и магистра технических наук в области электротехники и компьютерных наук, а в 2005 году защитил докторскую диссертацию по компьютерным наукам. Свою карьеру Фимстер начал в компании Looksmart[1], для которой он создал первый поисковый модуль. Также он

[1] Поисковик, впоследствии ставший службой каталогов для поисковой системы AltaVista.

участвовал в разработке первого алгоритма мониторинга ботнетов компании Damballa.

Профессор Фимстер является членом ACM. За вклад в разработку подходов к сетевой безопасности, ориентированных на данные, он получил Президентскую премию для молодых ученых и инженеров (Presidential Early Career Award for Scientists and Engineers, PECASE). Одна из его ранних публикаций о платформе управления маршрутизацией была отмечена наградой ассоциации USENIX Test of Time («Испытание временем») за влияние на развитие программно-конфигурируемых сетей. Фимстер выпустил первый онлайн-курс на эту тему. Помимо этого, он стал учредителем и преподавателем Магистерской программы дистанционного обучения компьютерным наукам Технологического института Джорджии.

Ник Фимстер — заядлый бегун на длинные дистанции. Он пробежал 20 марафонов, в том числе Бостонский, Нью-Йоркский и Чикагский.

**Дэвид Уэзеролл** работает в компании Google. Ранее он был доцентом кафедры компьютерных наук и электротехники Вашингтонского университета, а также консультантом Intel Labs в Сиэтле. Будучи родом из Австралии, Уэзеролл получил степень инженерии в области электротехники в Университете Западной Австралии. Докторскую диссертацию в области компьютерных наук он защитил в Массачусетском технологическом институте.

Последние 20 лет доктор Уэзеролл работает в сфере компьютерных сетей. Его исследования направлены на сетевые системы, в особенности беспроводные сети и мобильные вычисления, разработку интернет-протоколов и измерение параметров сетей.

За исследования, которые положили начало разработке активных сетей (архитектуры для быстрого внедрения новых сетевых служб), Уэзеролл получил премию ACM SIGCOMM Test of Time. Также он был удостоен премии IEEE им. Уильяма Беннета за прорыв в области веб-картографии. В 2002 году его работа была отмечена наградой Национального научного фонда CAREER (National Science Foundation CAREER), а в 2004-м он стал стипендиатом Фонда Слоуна (Sloan Foundation).

Дэвид Уэзеролл — активный участник сообщества исследователей компьютерных сетей. Он является сопредседателем программных комитетов SIGCOMM, NSDI и MobiSys, а также одним из организаторов семинаров ACM HotNets. Уэзеролл был членом программных комитетов множества конференций, посвященных сетевым технологиям. Также он работает редактором журнала ACM Computer Communication.

# ОТ ИЗДАТЕЛЬСТВА

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

# ГЛАВА 1
# Введение

Каждое из прошлых трех столетий было отмечено своей господствующей технологией. Промышленная революция XVIII века положила начало развитию крупного машиностроения. XIX век стал эрой паровых двигателей. Ключевой технологией XX века стали сбор, обработка и распространение информации. В числе других достижений следует отметить создание всемирной телефонной сети, изобретение радио и телевидения, рождение и беспрецедентный рост компьютерной отрасли, запуск спутников связи и, конечно, появление интернета. Кто знает, какие чудеса ждут нас в XXI веке?

В результате стремительного научно-технического прогресса происходит слияние отраслей, и грань между сбором, передачей, хранением и обработкой информации стирается. Корпорации, насчитывающие сотни офисов по всему миру, должны иметь возможность получать информацию о своем даже самом удаленном представительстве одним нажатием клавиши. И как бы быстро ни росли возможности сбора, обработки и распространения информации, потребности во все более сложных технологиях растут еще быстрее.

## 1.1. ПРИМЕНЕНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

Хотя компьютерная индустрия еще очень молода по сравнению с другими отраслями промышленности (например, авиа- и автомобилестроением), ее эволюция за короткий промежуток времени поистине поразительна. В первые два десятилетия своего существования компьютерные системы были централизованными и, как правило, занимали целую комнату. Часто это были помещения со стеклянными окнами, через которые посетители могли полюбоваться на чудо электроники. Среднее предприятие или университет могли себе позволить один компьютер (иногда два), а крупная компания — до нескольких десятков. Сама идея о том, что через 50 лет будут произведены миллиарды куда более мощных компьютеров размером с почтовую марку, казалась научной фантастикой.

Слияние вычислительной техники и телекоммуникаций в корне изменило организацию компьютерных систем. Концепция «вычислительного центра» как помещения с одним большим компьютером, куда пользователи приносят свои задачи для обработки, безнадежно устарела (хотя вполне обыденными стали центры обработки данных, содержащие сотни тысяч интернет-серверов). На смену одному компьютеру, обслуживающему все вычислительные потребности

компании, пришла система множества отдельных, но связанных между собой компьютеров. Такие системы получили название **компьютерных сетей (computer networks)**[1]. Их архитектуре и организации и посвящена наша книга.

В этой книге термин «компьютерная (вычислительная) сеть» обозначает набор взаимосвязанных автономных вычислительных устройств. Компьютеры считаются взаимосвязанными, если могут обмениваться информацией. Соединение осуществляется с использованием разнообразных сред передачи данных. Это могут быть медные провода, оптоволоконные кабели и радиоволны (например, микроволны, инфракрасные волны, спутники связи). Нам предстоит исследовать компьютерные сети самых разных размеров, конфигураций и форм. Часто они объединяются в более крупные сети. Наиболее известный пример системы сетей — **интернет**.

### 1.1.1. Доступ к информации

Доступ к информации осуществляется разными способами. Веб-браузер — основной инструмент доступа к интернету. Он позволяет извлекать данные с различных сайтов, включая набирающие популярность соцсети. Сегодня мобильные приложения на смартфонах также предоставляют удаленный доступ к информации на всевозможные темы. Искусство, бизнес, кулинария, госуправление, здоровье, история, хобби, развлечения, наука, спорт, путешествия... Всего не перечислить (а некоторые темы и не стоит упоминать).

Большинство СМИ также мигрировали в интернет, а некоторые даже полностью отказались от бумажной версии. Доступ к информации, включая новости, все более персонализируется. Некоторые интернет-СМИ дают читателю возможность самому выбрать интересующие его темы. Например: коррумпированные политики, масштабные пожары, скандалы с участием знаменитостей и эпидемии, но не, скажем, футбол. Подобная тенденция определенно угрожает заработку 12-летних разносчиков газет — интернет позволяет распространить новости на гораздо более широкую аудиторию.

Отбор новостей также все чаще происходит в социальных сетях. Они позволяют публиковать новостной контент из самых разнообразных источников и делиться им с другими пользователями. Новости сортируются и персонализируются не только в соответствии с выбором конкретного пользователя, но и на основе сложных алгоритмов машинного обучения. Алгоритмы прогнозируют предпочтения на основе истории просмотра. Публикация в интернете и подбор контента в соцсетях порождают финансовую модель, которая во многом зависит от таргетированной поведенческой рекламы. Разумеется, это требует

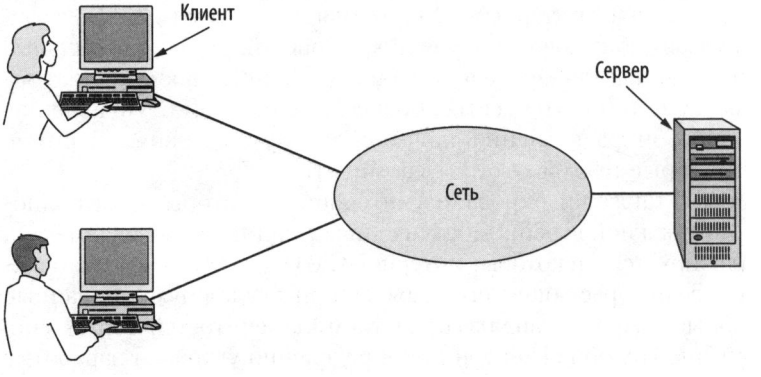
---

[1] Исторически сложилось так, что термины computer networks и computer systems чаще всего переводят на русский язык как «компьютерные сети» и «компьютерные системы», но на самом деле речь идет о вычислительных устройствах, которые являются компонентами сетей и систем. В этой книге мы будем придерживаться привычной терминологии, но имейте в виду, что под компьютером (computer) понимается вычислительное устройство. — *Примеч. науч. ред.*

сбора данных о поведении отдельных пользователей. Иногда такая информация используется неправомерно.

Сегодня онлайн-библиотеки и интернет-магазины содержат электронные версии изданий, от научных журналов до книг. Многие профессиональные объединения, такие как ACM (Association for Computing Machinery — Ассоциация по вычислительной технике; www.acm.org) и IEEE Computer Society (Общество специалистов по вычислительной технике IEEE; www.computer.org), уже давно оцифровали и выложили в интернет все свои журналы и труды конференций. В один прекрасный день бумажные книги могут стать архаизмом, уступив место электронным книгам и онлайн-библиотекам. Скептикам стоит сравнить этот процесс с эффектом, который оказало изобретение печатного станка на средневековые иллюстрированные рукописи.

Доступ к значительной доле информации в интернете производится посредством модели «клиент-сервер». Клиент явным образом запрашивает информацию с хранящего ее сервера, как показано на илл. 1.1.

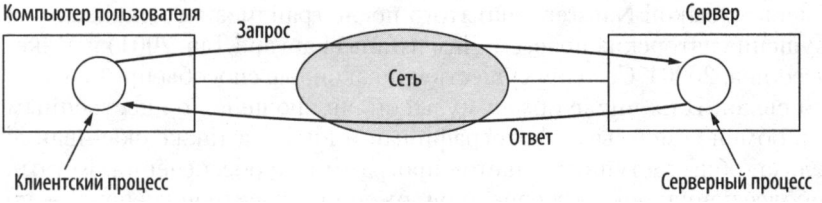



**Илл. 1.1.** Сеть, включающая два клиента и один сервер

**Модель «клиент-сервер»** используется очень широко, на ней основана большая часть сетевых приложений. Наиболее распространенная реализация этой модели — **веб-приложение**. Сервер генерирует веб-страницы на основе своей базы данных в ответ на запросы клиентов. Эти запросы, в свою очередь, пополняют базу данных сервера. Такая модель применима не только когда клиент и сервер физически находятся в одном здании (и принадлежат одной компании), но и когда они удалены на значительное расстояние. Например, пользователь у себя дома обращается к странице во Всемирной паутине. В этом случае его домашний компьютер играет роль клиента, а удаленный веб-сервер — сервера. Как правило, один сервер способен обслуживать большое число (сотни или тысячи) клиентов одновременно.

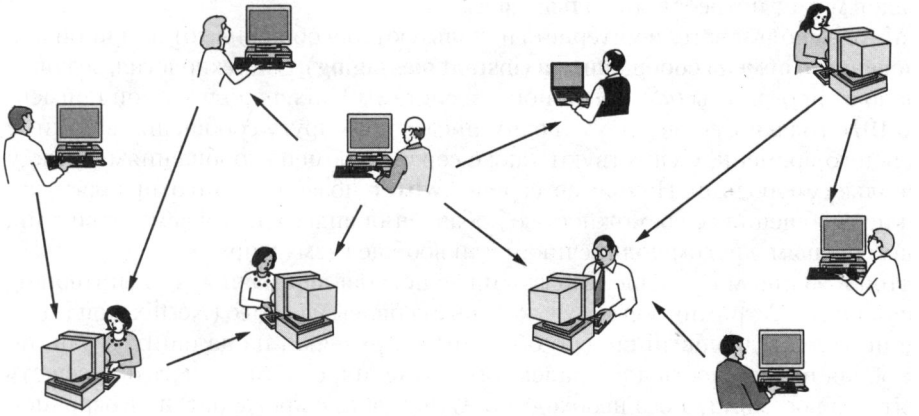
В первом приближении в модели «клиент-сервер» участвуют два процесса (работающие программы), один на компьютере пользователя, а второй — на сервере. Связь между ними происходит путем отправки клиентским процессом по сети сообщения серверному процессу. Далее клиентский процесс ожидает

ответного сообщения. При получении запроса серверный процесс производит требуемые действия или находит запрашиваемые данные, после чего отправляет ответ. Эти сообщения показаны на илл. 1.2.




**Илл. 1.2.** Модель «клиент-сервер» включает запросы и ответы

Еще одна популярная модель доступа к информации — **одноранговая**, или **пиринговая (peer-to-peer), связь** (Парамешваран и др.; Parameswaran et al., 2001[1]). При таком виде связи пользователи, образующие не слишком тесно связанную группу, могут обмениваться сообщениями с другими ее участниками, как показано на илл. 1.3. По сути, каждый из них может взаимодействовать с одним или несколькими людьми; никакого четкого деления на клиенты и серверы нет.



**Илл. 1.3.** В одноранговой системе отсутствует деление на клиенты и серверы

Во многих одноранговых системах, например BitTorrent (Коэн; Cohen, 2003), отсутствует централизованная база данных контента. Вместо этого каждый пользователь поддерживает свою локальную базу данных, а также список остальных участников системы. Новый пользователь может обратиться к любому участнику системы, чтобы получить его контент и имена остальных пользователей (для поиска дополнительного контента и прочих имен). Процесс поиска можно

[1] Список всех упоминаемых в тексте изданий вы найдете в конце книги в разделе «Алфавитный список литературы». — *Примеч. ред.*

повторять бесконечно, создавая обширную локальную базу данных. Для людей подобная деятельность утомительна, но компьютеры справляются с ней на ура.

Одноранговые системы связи часто применяются для распространения музыки и видео. Пик их популярности пришелся на 2000-е годы, с появлением сервиса обмена музыкой Napster, закрытого после грандиозного скандала по поводу нарушения авторских прав; см. Лам и Тань (Lam and Tan, 2001) и Македония (Macedonia, 2000). Сегодня существуют законные способы применения пиринговой связи. В их числе обмен музыкой, являющейся общественным достоянием, обмен семейными фотографиями и видео, а также скачивание пользователями общедоступных пакетов программного обеспечения. Кстати, одно из наиболее популярных интернет-приложений — электронная почта — по сути является одноранговой системой. Данный вид связи, вероятно, в будущем станет применяться еще более широко.

## 1.1.2. Общение

Общение онлайн — ответ XXI века на телефон XIX века. Электронная почта уже сейчас используется каждый день миллионами людей по всему миру, и ее популярность постоянно растет. Вложение в сообщения аудио- и видеофайлов наряду с текстом и рисунками — вполне обычное дело. Реализация отправки запахов может потребовать больше времени.

Многие пользователи интернета используют для общения тот или иной вид **мгновенного обмена сообщениями (instant messaging)**. Эта технология, ведущая начало от программы *talk* операционной системы Unix, используемой примерно с 1970 года, позволяет двум людям писать друг другу сообщения в режиме реального времени. Существуют также сервисы обмена сообщениями между несколькими людьми. Например, сервис **Twitter**, позволяющий отправлять короткие сообщения (с возможностью добавления видео), называемые твитами, своим друзьям, другим подписчикам или вообще всему миру.

Приложения могут использовать интернет для передачи аудио (интернет-радиостанции, стриминговые музыкальные сервисы) и видео (Netflix, YouTube). Это не только дешевый способ общения с друзьями из дальних стран, но и удобная возможность для удаленного обучения, с возможностью посещать занятия в восемь утра без необходимости вставать с кровати. В долгосрочной перспективе использование компьютерных сетей для расширения возможностей коммуникации будет иметь важнейшее значение. Благодаря им люди из далеких от цивилизации мест могут обрести такой же доступ к различным сервисам, что и жители мегаполиса.

**Социальные сети (social networks)** предоставляют и возможность общаться, и доступ к информации. Поток данных в них определяется публично заявленными взаимоотношениями между пользователями. Одна из наиболее популярных социальных сетей — **Facebook**. С его помощью пользователи могут создавать/обновлять свои личные профили и делиться обновлениями со своими друзьями. Другие приложения соцсетей предоставляют также возможности знакомства с друзьями друзей, отправки друзьям новостных сообщений (как в вышеупомянутом Twitter) и многое другое.

В еще более общем случае люди могут совместно генерировать контент. В качестве примера можно привести технологию **вики (wiki)** — совместно созданный и редактируемый членами сообщества веб-сайт. Наиболее известный пример использования технологии вики — **Википедия**, онлайн-энциклопедия, доступная всем для чтения и редактирования; но существуют тысячи других вики.

## 1.1.3. Электронная коммерция

Покупка товаров через интернет весьма популярна. Пользователи просматривают онлайн-каталоги товаров тысяч компаний и заказывают доставку прямо домой. А если покупатель приобрел товар через интернет, но не может разобраться, как им пользоваться, — к его услугам онлайн-техподдержка.

Еще одна сфера широкого применения электронной коммерции — доступ к финансовым услугам. Многие уже сейчас производят оплату, управляют банковскими счетами и даже инвестируют средства через интернет. Благодаря финансовым технологиям (или финтех-приложениям) пользователи осуществляют самые разнообразные денежные онлайн-операции, включая переводы между банковскими счетами или между друзьями.

Немалый размах приобрели онлайн-аукционы б/у товаров. В отличие от обычной электронной коммерции, основанной на модели «клиент-сервер», онлайн-аукционы производятся по принципу одноранговой сети. Это значит, что их участники могут быть как покупателями, так и продавцами одновременно, несмотря на наличие центрального сервера, на котором хранится база данных продаваемых товаров.

Некоторые формы электронной коммерции получили изящные короткие названия-аббревиатуры, в основе которых лежит тот факт, что в английском языке «to»[1] и «2» произносятся одинаково. Наиболее распространенные из них представлены на илл. 1.4.

| Аббревиатура | Полное название | Пример |
|---|---|---|
| B2C | Бизнес для потребителя (Business-to-consumer) | Заказ книг в интернете |
| B2B | Бизнес для бизнеса (Business-to-business) | Производитель автомобилей заказывает шины у поставщика |
| G2C | Правительство для потребителя (Government-to-consumer) | Правительство распространяет через интернет бланки налоговых деклараций |
| C2C | Потребитель для потребителя (Consumer-to-consumer) | Продажа на онлайн-аукционе б/у товаров |
| P2P | Пиринговые сети (Peer-to-peer) | Распространение музыки или файлов; Skype |

**Илл. 1.4.** Некоторые виды электронной коммерции

[1] Английский многозначный предлог, в данном случае обозначающий «для». — *Примеч. пер.*

## 1.1.4. Развлечения

Четвертая наша категория — развлечения. Индустрия домашних развлечений в последние годы растет семимильными шагами. Онлайн-распространение музыки, фильмов, радио- и телепередач конкурирует с традиционными механизмами потребления контента. Пользователи могут находить, покупать и скачивать песни в формате MP3 и фильмы в высоком качестве, а затем добавлять их в свою домашнюю коллекцию. Во многие дома телешоу сейчас попадают посредством систем **IPTV (IP-телевидение)**, в основе которых лежат IP-технологии (взамен кабельного телевидения или радио). С помощью приложений для потоковой передачи мультимедиа пользователи могут слушать интернет-радиостанции или смотреть фильмы или свежие эпизоды любимых телесериалов. Естественно, весь этот контент можно перемещать между различными устройствами и выводить на всевозможные экраны и динамики в пределах квартиры (обычно с помощью беспроводной сети).

Вероятно, скоро появится возможность моментально найти любой когда-либо снятый фильм или телепрограмму и вывести их на свой экран. В будущем фильмы станут интерактивными, и пользователи смогут выбрать сюжетную линию (убить ли Макбету короля сейчас или подождать более благоприятного момента?) из нескольких альтернативных сценариев для каждого случая. Прямой эфир на телевидении также может быть интерактивным: зрители могут участвовать в телевикторинах, выбирая участников, и т. д.

Еще один вид развлечений — игры. Уже сейчас существуют многопользовательские онлайн-симуляторы. Например, прятки в виртуальном подземелье или авиасимуляторы, в которых игроки одной команды пытаются сбить игроков из команды противника. Виртуальные миры служат сценой, на которой тысячи игроков сосуществуют в одной вселенной с трехмерной графикой.

## 1.1.5. Интернет вещей

Термин **повсеместные вычисления (ubiquitous computing)** означает, что вычисления неразрывно вплетены в повседневную жизнь согласно концепции Марка Вайзера (Mark Weiser, 1991). Сегодня многие дома обеспечиваются системами безопасности с датчиками на дверях и окнах. Кроме того, существует множество других видов датчиков, которые можно подключить к системе умного дома, например, для учета потребления электроэнергии. Интеллектуальные счетчики электроэнергии, газа и воды могут отправлять показания по сети. Это позволяет коммунальным компаниям сэкономить средства и не нанимать специальных людей для съема показаний. Датчики дыма могут отправлять сигнал непосредственно пожарным вместо запуска громкой сирены (от которой все равно будет мало толку, если дома никого нет). Умные холодильники могли бы сами, например, заказывать молоко, если оно почти закончилось. По мере снижения стоимости датчиков и передачи данных все больше измерений и отчетов будет осуществляться с помощью сетей. Эта непрерывная революция, получившая название **IoT (internet of things — интернет вещей)**, ведет к подключению практически всех приобретаемых нами электронных устройств к интернету.

Бытовые электроприборы все чаще подключаются к сети. Например, некоторые дорогие камеры подключаются к беспроводной сети и отправляют фотографии на ближайший экран для просмотра. Профессиональные фоторепортеры также пересылают снимки редакторам в режиме реального времени, сначала по беспроводной сети к точке доступа, а затем через интернет. Приборы, подключаемые к розетке (например, телевизоры), могут использовать **связь по ЛЭП (power-line networks)** для передачи информации по дому через электропроводку. Присутствием подобных устройств в сети трудно удивить. Однако объекты, которые обычно не ассоциируются с компьютерами, также могут считывать и передавать информацию. Например, душ может фиксировать расход воды (вы можете следить за ним прямо во время мытья), а по завершении отправить отчет в домашнее приложение экологического мониторинга, чтобы помочь вам сэкономить на воде.

## 1.2. ТИПЫ КОМПЬЮТЕРНЫХ СЕТЕЙ

Существует множество видов компьютерных сетей. В этом разделе перечислены некоторые из них. Это мобильные и широкополосные сети доступа (через них обычно осуществляется выход в интернет); сети дата-центров (они хранят ежедневно используемые нами данные и приложения); транзитные сети (соединяющие сети доступа с дата-центрами); корпоративные сети (внутренние сети университетов, компаний и других организаций).

### 1.2.1. Сети широкополосного доступа

В 1977 году президентом Digital Equipment Corporation (на тот момент второго по величине производителя компьютеров в мире после IBM) был Кен Олсен (Ken Olsen). Когда его спросили, почему Digital не стремится завоевать рынок персональных компьютеров, он сказал: «Не вижу никакой причины, зачем кому-то держать компьютер дома». Однако история доказала обратное, и Digital канула в Лету. Изначально люди покупали компьютеры для работы с документами и игр. Сегодня же основная причина покупки домашнего компьютера — доступ в интернет. Кроме того, многие бытовые электроприборы (например, ТВ-тюнеры, игровые консоли, телевизоры и даже дверные замки) содержат встроенные компьютеры с возможностью доступа к компьютерным сетям, в первую очередь беспроводным. Кроме того, домашние сети широко используются для развлечений, включая прослушивание музыки, просмотр фото и видео, а также создание контента.

Доступ в интернет предоставляет домашним пользователям возможность **подключения (connectivity)** к удаленным компьютерам. Как и в случае с компаниями, домашние пользователи могут просматривать информацию, общаться с другими людьми, а также покупать товары и услуги. Главное преимущество состоит в возможности подключения этих устройств к другим компьютерам, находящимся за пределами дома. Боб Меткалф (Bob Metcalfe), создатель Ethernet, выдвинул гипотезу, что полезность сети пропорциональна квадрату числа ее пользователей, поскольку оно приближенно равно числу возможных

соединений (Гилдер; Gilder, 1993). Эта гипотеза известна под названием «Закон Меткалфа». Она объясняет, как колоссальная популярность интернета связана с его размером.

Сегодня число широкополосных сетей доступа быстро растет. Во многих уголках мира домашний широкополосной доступ предоставляется с помощью медных проводов (например, линий телефонной связи), коаксиальных кабелей (в качестве примера можно привести кабельное ТВ/интернет) или оптоволоконной связи. Скорость широкополосного доступа в интернет также продолжает расти. Многие провайдеры в развитых странах обеспечивают для домашних пользователей скорость 1 Гбит/с. В некоторых регионах, особенно в развивающихся странах, основной вид доступа в интернет — через мобильные сети.

## 1.2.2. Мобильные и беспроводные сети

Мобильные компьютеры, такие как ноутбуки, планшеты и смартфоны, — один из наиболее активно развивающихся сегментов компьютерной индустрии. С точки зрения продаж они уже давно обогнали традиционные ПК. Почему они так востребованы? Люди часто используют мобильные устройства вне дома: для чтения и отправки электронной почты, твитов, просмотра фильмов, скачивания музыки, игр, доступа к картам или просто для веб-серфинга. Пользователи хотят иметь возможность делать все то же самое, что и дома или в офисе, причем везде — на суше, на море и в воздухе.

Почти для всех этих действий необходимо подключение к интернету. А поскольку проводное соединение в автомобилях, на кораблях и самолетах невозможно, растет интерес к беспроводным сетям. Примером таких сетей являются всем известные сотовые сети. Они предоставляются телефонными операторами и обеспечивают мобильную связь. Беспроводные **хот-споты (hotspots)** на основе стандарта 802.11 — еще одна разновидность беспроводной сети для мобильных компьютеров и переносных устройств (телефонов и планшетов). Они растут как грибы всюду, куда только ходят люди. В результате возникает «лоскутное одеяло» покрытия в кафе, отелях, аэропортах, школах, поездах и самолетах. При наличии мобильного устройства и беспроводного модема можно просто подключиться к интернету через беспроводную точку доступа, так же как через обычную проводную сеть.

Беспроводные сети необходимы водителям грузовиков, такси, автомобилей служб доставки, а также специалистам по ремонту — для связи с диспетчерской. Например, очень часто таксистами работают независимые предприниматели, а не служащие таксопарка. Обычно такси оборудовано специальным дисплеем для водителя. При поступлении заказа диспетчер указывает место, откуда надо забрать пассажира, и пункт назначения. Эта информация отображается на дисплеях водителей со звуковым сигналом. Заказ получает таксист, первым нажавший кнопку на дисплее. Взлет популярности мобильных и беспроводных сетей привел также к революции в самих наземных перевозках. Экономика совместного потребления позволяет водителям использовать свои собственные телефоны в качестве диспетчерского устройства, например, в случае таких агрегаторов такси, как Uber и Lyft.

Беспроводные сети играют также важную роль для военных. Если нужно в короткие сроки начать военные действия в любой точке земного шара, лучше не полагаться на локальную сетевую инфраструктуру, а развернуть свою собственную.

Хотя беспроводные и мобильные сети часто взаимосвязаны, это не одно и то же, как демонстрирует илл. 1.5. На нем показаны различия между **стационарными беспроводными (fixed wireless)** и **мобильными беспроводными (mobile wireless) сетями**. Даже ноутбуки зачастую подключаются к сети по проводам. Например, путешественник может получить доступ в интернет, подключив ноутбук к сетевой розетке в номере отеля. Конечно, вследствие повсеместного распространения беспроводных сетей подобная ситуация — редкость, хотя проводные сети обеспечивают большее быстродействие.

| Беспроводная | Мобильная | Типичные приложения |
|---|---|---|
| Нет | Нет | Стационарные компьютеры в офисах |
| Нет | Да | Ноутбук в номере отеля |
| Да | Нет | Сети в зданиях, где не проложены сетевые провода |
| Да | Да | Складской учет с использованием карманного компьютера |

**Илл. 1.5.** Варианты сочетания беспроводных сетей и мобильных вычислений

В свою очередь, некоторые беспроводные компьютеры не являются мобильными. Иногда в доме, в офисе или отеле может не быть соответствующего кабеля. Поэтому удобнее подключать стационарные компьютеры или проигрыватели мультимедиа по беспроводной связи, а не прокладывать провода. Для настройки беспроводной сети обычно достаточно купить маленькую коробочку с электроникой, распаковать ее и подключить. Это намного дешевле, чем нанимать рабочих для монтажа кабель-каналов и прокладки кабеля в здании.

Наконец, существуют по-настоящему мобильные беспроводные сетевые приложения. Например, когда кладовщик ходит по складу с карманным компьютером, фиксируя остатки товаров, он использует такое приложение. Во многих загруженных аэропортах клерки, занимающиеся возвратом арендованных машин, работают на парковке с беспроводными мобильными компьютерами. Они сканируют штрихкоды или RFID-метки возвращаемых автомобилей, а их мобильное устройство (со встроенным принтером) запрашивает данные из главного компьютера, получает информацию об аренде и печатает счет прямо на месте.

Ключевой стимул для развития приложений мобильной беспроводной связи — мобильные телефоны. Развитие таких приложений ускоряется вследствие слияния телефонных и интернет-технологий. **Смартфоны**, такие как iPhone компании Apple или Galaxy от Samsung, сочетают элементы мобильных телефонов и мобильных компьютеров. Эти телефоны тоже могут подключаться к беспроводным точкам доступа и автоматически переключаться между сетями, выбирая оптимальный для пользователя вариант. Ранее чрезвычайно популярным был

**обмен текстовыми сообщениями (text messaging, texting)** через сотовые сети. За пределами США эта технология называется **SMS (Short Message Service — сервис коротких сообщений)**. Пользователи мобильных телефонов могут набирать короткие сообщения и отправлять их по сотовой сети другому мобильному абоненту. Обмен текстовыми сообщениями — исключительно выгодная вещь для мобильного оператора. Передача сообщения обходится компании в ничтожную долю цента, а пользователи платят за этот сервис намного больше. Некоторое время SMS приносили бешеные прибыли операторам мобильной связи. Теперь же существует множество альтернатив, использующих либо сотовую сеть, либо Wi-Fi, включая WhatsApp, Signal и Facebook Messenger.

Использовать сотовые сети и точки доступа Wi-Fi для подключения к удаленным компьютерам может и другая бытовая электроника. Планшеты и устройства для чтения электронных книг могут скачать при подключении к интернету купленную книгу, последний выпуск журнала или свежую газету. А цифровые фоторамки могут выводить на экран новое изображение в нужный момент.

Мобильные телефоны обычно «знают», где находятся. **GPS (Global Positioning System — система глобального позиционирования)** может напрямую определить местоположение устройства. Кроме того, телефон может определить свое местоположение путем триангуляции по точкам доступа Wi-Fi с известными координатами. Работа некоторых приложений основана на этой информации. В первую очередь это мобильные карты, ведь ваш телефон или автомобиль с GPS, вероятно, лучше вас знает, где вы находитесь. Сюда входят также приложения для поиска ближайшего книжного магазина или китайского ресторана либо местный прогноз погоды. Регистрируют местоположение и другие сервисы, например, для снабжения фотографий и видео метками, соответствующими месту съемки. Проставление подобных меток называется **геотегированием (geo-tagging)**.

Мобильные телефоны все чаще используются в **мобильной коммерции (m-commerce)** (Сенн; Senn, 2000). Отправка SMS используется для авторизации платежей в торговых автоматах, оплаты билетов в кинотеатры и других мелких покупок (вместо наличных денег и платежных карт). Позже сумма покупки снимается со счета мобильного телефона. При наличии технологии **NFC (Near Field Communication — беспроводная связь малого радиуса действия)** мобильный телефон может играть роль бесконтактной (RFID) платежной карты. В этом случае платеж происходит путем взаимодействия со считывающим устройством. Движущими силами этого явления выступают производители мобильных устройств и провайдеры, которые отчаянно стремятся откусить кусок пирога электронной коммерции. С точки зрения магазина такая схема позволяет сэкономить большую часть банковской комиссии, которая обычно составляет несколько процентов. Конечно, у этого плана есть и недостатки. Покупатели могут воспользоваться RFID или сканером штрихкода на своих мобильных устройствах и получить подробную информацию о ценах на интересующий их товар в соседних магазинах.

Мобильная коммерция особенно привлекательна тем, что пользователи смартфонов привыкли за все платить. Это отличает их от интернет-пользователей, которые считают, что все должно быть бесплатно. Если бы какой-то веб-сайт

начал взимать плату с пользователей за возможность платить кредитной картой, они подняли бы страшный шум. При этом если мобильный оператор предоставит возможность оплатить товары в магазине, просто помахав перед кассой телефоном, а затем спишет за это удобство небольшую комиссию, то, вероятно, пользователи воспримут это как должное. Время покажет.

В будущем области применения мобильных и беспроводных компьютеров по мере уменьшения их размеров будут быстро расширяться, вероятно, совершенно неожиданным образом. Давайте рассмотрим некоторые из потенциальных вариантов. **Сенсорные сети (sensor networks)** состоят из узлов, собирающих и передающих информацию о состоянии окружающего мира. Эти узлы могут встраиваться в машины или телефоны, а могут представлять собой отдельные маленькие устройства. Например, автомобиль может собирать данные о своем местоположении, скорости, вибрации и КПД топлива из бортовой системы диагностики и загружать их в базу данных (Халл и др.; Hull et al., 2006). С помощью этой информации водитель сможет засечь ямы на дорогах, объехать пробки или выяснить, что он тратит слишком много бензина по сравнению с другими водителями на одном участке дороги.

Сенсорные сети производят настоящую революцию в науке, поскольку предоставляют массу данных о ранее не наблюдавшихся видах поведения. Один из примеров — отслеживание миграции отдельных зебр путем прикрепления к каждой из них маленького датчика (Цзюань и др.; Juang et al., 2002). Исследователи сумели уместить беспроводный компьютер в один-единственный кубический миллиметр (Варнеке и др.; Warneke et al., 2001). Благодаря таким крошечным компьютерам можно отслеживать перемещения даже маленьких птиц, грызунов и насекомых.

Беспроводные парковочные автоматы могут принимать платежи с помощью кредитных или дебетовых карт, с мгновенной верификацией по беспроводной линии связи. Они также сообщают, если место занято. Благодаря этому водители могут скачать актуальную карту парковок и с легкостью найти свободное место. Разумеется, когда оплаченное время заканчивается, автомат может проверить (с помощью датчика отраженного сигнала), на месте ли машина, и известить об этом охранника. По некоторым оценкам, только муниципальные власти США собирают подобным образом дополнительные $10 млрд штрафов (Харте и др.; Harte et al., 2000).

## 1.2.3. Сети доставки контента

В настоящее время многие интернет-сервисы подключены к «облаку», то есть **сетям дата-центров (data-center networks)**. Современные дата-центры состоят из сотен тысяч или даже миллионов серверов, находящихся в одном месте. Серверные стойки располагаются тесными рядами в помещениях, которые могут быть более километра в длину. Сети дата-центров обслуживают стремительно растущие потребности **облачных вычислений (cloud computing)**. Они проектируются в расчете на перемещение больших объемов данных между серверами дата-центра, а также между самим дата-центром и остальным интернетом.

Большинство используемых вами приложений и сервисов, начиная от посещаемых веб-сайтов и до облачных программ редактирования заметок, хранят данные в сетях дата-центров. Подобные сети сталкиваются с проблемой масштабирования, в плане как пропускной способности сети, так и использования электроэнергии. Одна из основных проблем, связанных с пропускной способностью сети, — так называемая пропускная способность сегмента сети, то есть скорость передачи данных между двумя серверами сети. В основе архитектуры ранних сетей дата-центров лежала простая топология типа «дерево» с тремя слоями коммутаторов: доступ, агрегирование и ядро. Эта простая архитектура плохо масштабировалась и была подвержена сбоям.

Многие популярные интернет-сервисы доставляют контент пользователям по всему миру. Для этого используются **сети доставки контента (CDN, Content Delivery Network)**. CDN представляет собой большой набор серверов, географически расположенных таким образом, что контент размещается как можно ближе к запрашивающему его пользователю. У крупных поставщиков контента, таких как Google, Facebook и Netflix, есть свои собственные CDN. Некоторые CDN, например Akamai и Cloudflare, предоставляют услуги хостинга более мелким сервисам, у которых нет своих собственных CDN.

Необходимый пользователям контент (от статических файлов до потокового видео) реплицируется по множеству различных мест в одной CDN. Когда пользователь запрашивает некий контент, CDN выбирает, какая реплика выдаст данные этому пользователю. Этот процесс должен учитывать расстояние до клиента от каждой из реплик, загруженность серверов CDN, а также интенсивность трафика и перегруженность самой сети.

## 1.2.4. Транзитные сети

Данные в интернете проходят через множество независимых сетей. Вряд ли контент посещаемых вами веб-сайтов содержится в сети, которую обслуживает ваш домашний провайдер. Обычно он располагается в сетях дата-центров, а пользователи могут обращаться к нему из сети доступа. При этом контент должен пройти через интернет от дата-центра до сети доступа, а затем до устройства пользователя.

Если поставщик контента и **интернет-провайдер (ISP, Internet Service Provider)** не связаны напрямую, трафик между ними зачастую передается через **транзитную сеть (transit network)**. Транзитные сети обычно взимают плату за сквозную передачу трафика как с ISP, так и с поставщика контента. Если содержащая контент сеть и сеть доступа обмениваются друг с другом достаточным количеством трафика, возможно, для них имеет смысл установить прямое взаимоподключение. Подобное прямое соединение характерно для серьезных ISP и крупных поставщиков контента, таких как Google и Netflix. Для этого ISP и поставщик контента должны создать и поддерживать сетевую инфраструктуру для прямого взаимного соединения, зачастую в нескольких географических точках.

Транзитные сети традиционно называют **опорными**, или **магистральными, сетями (backbone networks)**, поскольку их задача заключается в перемещении

трафика между двумя конечными точками. Много лет тому назад транзитные сети приносили огромный доход, поскольку все остальные сети нуждались в них (и платили им) для соединения с остальным интернетом.

В последнее десятилетие, впрочем, возникли две иные тенденции. Первая из них порождена изобилием сервисов и крупных сетей доставки содержимого, размещающих данные в облаке. Эта тенденция состоит в консолидации контента у нескольких крупных поставщиков. Вторая тенденция заключается в расширении зон обслуживания отдельных сетей доступа ISP. Ранее многие ISP были небольшими, региональными, однако сегодня многие из них охватывают целые страны (или даже несколько стран). Это расширяет как географию их возможного соединения с другими сетями, так и их абонентскую базу. А по мере того как размеры (и переговорные возможности) сетей доступа и сетей поставщиков контента растут, более крупные сети все меньше полагаются на транзитные сети при доставке трафика. Зачастую они предпочитают соединяться напрямую, оставляя транзитные сети в качестве резервного варианта.

## 1.2.5. Корпоративные сети

В большинстве организаций (компаний, университетов и т. д.) много компьютеров. Они могут понадобиться любому сотруднику для выполнения различных задач — от проектирования продукта до формирования платежной ведомости. Обычно эти компьютеры подключаются к общей сети, что позволяет сотрудникам совместно использовать данные, информацию и вычислительные ресурсы.

**Совместное использование ресурсов (resource sharing)** делает программы, оборудование и прежде всего данные доступными другим пользователям сети, независимо от того, где эти ресурсы и пользователи находятся физически. Самый простой пример — совместное использование принтера работниками офиса. Личный принтер многим сотрудникам ни к чему, а высокопроизводительный сетевой принтер намного экономичнее, быстрее и проще в обслуживании, чем большой парк отдельных принтеров.

Совместное использование информации, вероятно, играет даже более важную роль, чем совместное использование материальных ресурсов (таких, как принтеры и системы резервного копирования). Большинство компаний предоставляет сотрудникам онлайн-доступ к данным о клиентах и товарах, складской информации, финансовым и налоговым отчетам и многому другому. Банк не смог бы проработать и пяти минут, если бы все его компьютеры внезапно отказали. Современная фабрика, с управляемым компьютером конвейером, не продержалась бы и пяти секунд. Даже маленькому турагентству или юридической фирме из трех человек чрезвычайно необходимы компьютерные сети для быстрого доступа к нужной информации и документам.

В случае мелких компаний компьютеры могут располагаться в одном офисе отдельного здания. Что касается крупных корпораций, их компьютеры и сотрудники могут быть разбросаны по десяткам офисов и фабрик в нескольких странах. Тем не менее сотруднику отдела сбыта в Нью-Йорке может потребоваться доступ к базе данных склада товаров в Сингапуре. Для соединения географически удаленных сетей в одну логическую сеть применяются **виртуальные частные**

**сети (VPN, Virtual Private Networks)**. Пользователь, даже если он оказался за 15 000 км от нужных ему данных, должен иметь возможность обращаться к ним так, как будто он находится в том же офисе. Эту цель можно кратко сформулировать как попытку освободиться от «тирании географии».

Информационную систему компании можно представить как одну или несколько баз, содержащих все данные компании, и определенное количество сотрудников с удаленным доступом к этим базам. При такой модели данные хранятся на мощных компьютерах, называемых **серверами (servers)**. Чаще всего они размещаются в централизованной серверной комнате и обслуживаются системным администратором. Компьютеры сотрудников, с помощью которых они получают доступ к удаленным данным (например, для работы с электронными таблицами), — более простые, размещаются на их рабочих столах и называются **клиентами**, или **рабочими станциями (clients)**. Иногда мы будем называть клиентом человека, который использует рабочую станцию, но из контекста обычно понятно, что имеется в виду — компьютер или его пользователь. Рабочие станции и серверы объединяются сетью, как показано на илл. 1.1. Обратите внимание, что сеть на этом рисунке показана в виде простого овала, без каких-либо подробностей. Этот вариант мы будем использовать при обсуждении компьютерных сетей на наиболее абстрактном уровне. Когда же необходимы будут детали, мы их обозначим.

Вторая цель создания корпоративной сети связана скорее с человеческим фактором, а не с данными или даже с компьютерами. Компьютерная сеть обеспечивает прекрасную **среду обмена информацией (communication medium)** между сотрудниками. Практически в каждой компании, где есть хотя бы два компьютера, для повседневного обмена информацией чаще всего используется **электронная почта (email)**. При этом сотрудники любой компании жалуются в курилке на то, сколько писем им приходится читать. Просто начальство обнаружило, что достаточно нажать одну кнопку, чтобы отправить одно и то же сообщение (нередко лишенное всякого осмысленного содержания) всем подчиненным сразу.

Компьютерные сети могут вместо телефонной компании обеспечить и телефонную связь между сотрудниками. При использовании интернета эта технология называется **IP-телефонией (IP telephony)** или **VoIP (Voice over IP, передача голоса по IP,** или **по интернет-протоколу)**. При этом на обоих концах линии может использоваться телефон с поддержкой VoIP или компьютер сотрудника. Компании рассматривают эту технологию как замечательный способ сэкономить на счетах за телефон.

Компьютерные сети позволяют совершенствовать формы коммуникации. К аудио добавляется видео, чтобы несколько сотрудников вне офиса могли не только слышать, но и видеть друг друга во время совещания. Это отличный инструмент снижения затрат (денежных и временных) на путешествия. Благодаря **демонстрации рабочего стола (desktop sharing)** удаленные сотрудники могут видеть и взаимодействовать с экранами компьютеров своих коллег. Они могут читать общую информационную «доску» и писать на ней либо, например, совместно создавать отчет. При внесении одним из них правок в онлайн-документ остальные сразу же видят эти изменения, а не ждут письма в течение нескольких дней. Это ускоряет и значительно упрощает (и порой вообще делает возможным)

сотрудничество разбросанных на большие расстояния групп людей. Более амбициозные формы удаленного согласования действий, такие как телемедицина, лишь начинают появляться, но потенциально могут сыграть намного более важную роль (например, удаленный мониторинг пациентов). Иногда говорят, что в гонке между обменом информацией и перевозками победит что-то одно, а второе станет безнадежно устаревшим.

Третья цель для многих компаний — электронное ведение бизнеса, прежде всего с покупателями и поставщиками. Авиалинии, книжные магазины и другие представители ритейла обнаружили, что многим покупателям очень нравится совершать покупки, не выходя из дома. Поэтому многие компании предоставляют онлайн-каталоги товаров и сервисов и принимают заказы через интернет. Производители автомобилей, самолетов, компьютеров (среди прочих) закупают комплектующие у множества поставщиков, а затем собирают части воедино. С помощью компьютерных сетей производители при необходимости размещают заказы в электронной форме. Эта снижает потребность в больших складах и повышает производительность.

## 1.3. СЕТЕВЫЕ ТЕХНОЛОГИИ, ОТ ЛОКАЛЬНЫХ ДО ГЛОБАЛЬНЫХ

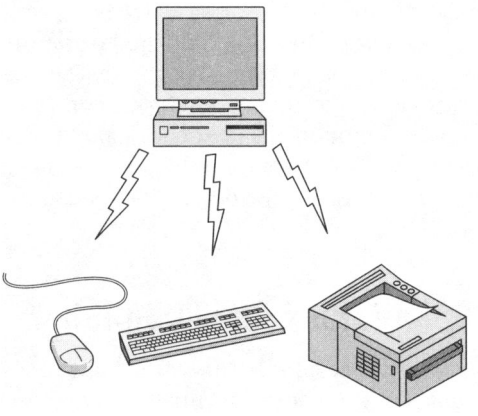
Сети варьируются от небольших и персональных до крупных и глобальных. В этом разделе представлены разнообразные сетевые технологии, позволяющие реализовывать сети различных размеров и масштабов.

### 1.3.1. Персональные сети

**Персональные сети (PAN, Personal Area Network)** обеспечивают обмен информацией между устройствами, используемыми одним человеком. Типичный пример — беспроводная сеть, связывающая компьютер с его периферийными устройствами. PAN используются для подключения беспроводных наушников, гарнитуры или часов к смартфону. Также они позволяют установить соединение между автомобилем и цифровым музыкальным плеером, как только устройство попадает в радиус действия сети.

Почти у каждого компьютера имеется несколько периферийных устройств: монитор, клавиатура, мышь и принтер. Если бы не беспроводные сети, все эти подключения пришлось бы выполнять при помощи кабелей. Для неопытного пользователя поиск нужных проводов и соответствующих им разъемов может стать настоящей проблемой (хотя обычно они различаются формой и цветом). По этой причине большинство поставщиков компьютеров предлагают услугу вызова мастера на дом. Чтобы помочь таким пользователям, несколько компаний объединились и разработали беспроводную сеть малого радиуса действия под названием **Bluetooth**. Идея в том, чтобы больше не нужно было возиться с проводами. Если все ваши устройства поддерживают Bluetooth, достаточно просто включить их, положить рядом, и они сами установят соединение друг с другом. Для многих людей такая простота в эксплуатации — большое преимущество.

В самом простом варианте Bluetooth-сети используют парадигму «главный — подчиненный» («master — slave»), приведенную на илл. 1.6. Системный блок (ПК) обычно играет роль главного узла и взаимодействует с мышью или клавиатурой, играющими подчиненную роль. Главный узел сообщает подчиненным, какие адреса использовать, когда и в течение какого промежутка времени осуществлять передачу данных, какие частоты использовать и т. д. Мы обсудим Bluetooth подробнее в главе 4.



**Илл. 1.6.** Конфигурация персональной сети на основе Bluetooth

Для создания PAN используется множество других технологий ближнего радиуса действия; они также представлены в главе 4.

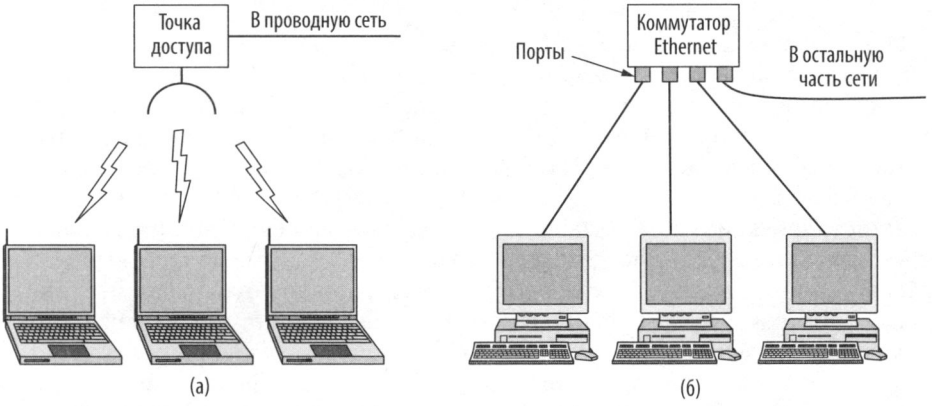
## 1.3.2. Локальные сети

**Локальная сеть (LAN, Local Area Network)** — частная сеть, функционирующая в отдельном здании и на прилегающей территории (это может быть дом, офис или завод). LAN широко применяется для соединения персональных компьютеров и бытовой электроники, позволяя совместно использовать различные ресурсы (например, принтеры) и обмениваться информацией.

На сегодняшний день беспроводные LAN применяются повсеместно. Изначально они были популярны в жилых помещениях, старых офисных зданиях, кафе и других местах, где прокладка кабелей обошлась бы слишком дорого. В подобных системах компьютеры обмениваются информацией с помощью встроенного радиомодема и антенны. Чаще всего компьютер обращается к специальному устройству, которое называется **точкой доступа (AP, Access Point)**, **беспроводным маршрутизатором (wireless router)** или **базовой станцией (base station)**, как показано на илл. 1.7. Это устройство осуществляет передачу пакетов данных между беспроводными компьютерами, а также между компьютером и интернетом. Точка доступа напоминает популярного ребенка в школе, поскольку все хотят с ней «поговорить». Еще один часто встречающийся сценарий — близко расположенные устройства обмениваются пакетами в конфигурации так называемой **ячеистой сети (mesh network)**. Иногда

конечные узлы выступают в роли передатчиков. Однако в большинстве случаев в ячеистую сеть входит специальный набор узлов, единственная функция которых — передача трафика. Ячеистые сети часто применяются в развивающихся регионах, где развертывать соединение по всей территории неудобно и дорого. Кроме того, растет их популярность в качестве решения для домашних сетей, особенно в больших зданиях.

Одним из самых популярных стандартов беспроводных LAN является **IEEE 802.11**, более известный как Wi-Fi. Он работает со скоростью от 11 Мбит/с (802.11b) до 7 Гбит/с (802.11ad). Обратите внимание, что в этой книге мы придерживаемся традиции и измеряем скорость передачи данных по линии в мегабитах в секунду, где 1 Мбит/с равен 1 000 000 бит/с, и гигабитах в секунду, где 1 Гбит/с равен 1 000 000 000 бит/с. Степени двойки мы будем использовать только при описании хранения информации, в этом случае 1 МБ памяти равен $2^{20}$, то есть 1 048 576 байт. Стандарт 802.11 подробнее обсуждается в главе 4.




**Илл. 1.7.** Беспроводные и проводные LAN. (а) 802.11. (б) Коммутируемая сеть Ethernet

В проводных LAN используется множество различных технологий передачи; наиболее распространенные физические среды — медные провода, коаксиальный кабель и оптоволокно. Размеры LAN ограниченны, а значит, наихудшее время прохождения сигнала имеет предел и известно заранее. Знание этих ограничений помогает разрабатывать сетевые протоколы. Обычно проводные LAN работают на скорости от 100 Мбит/с до 40 Гбит/с. Они отличаются низкой задержкой (не более десятков миллисекунд, а зачастую намного меньше), при этом ошибки передачи случаются редко. Проводные LAN обычно обладают меньшим значением задержки и процентом потери пакетов, а также более высокой пропускной способностью, чем беспроводные. И хотя с течением времени этот разрыв в показателях сократился, передавать сигналы по проводам или через оптоволокно намного проще, чем по воздуху.

Многие проводные LAN включают двухточечные проводные соединения. Стандарт IEEE 802.3, более известный как **Ethernet**, однозначно самая

популярная разновидность проводной LAN. На илл. 1.7 (б) показан пример **коммутируемой сети Ethernet (switched Ethernet)**. Любой компьютер может использовать протокол Ethernet и подключаться к устройству, называемому **коммутатором (switch)**, посредством двухточечного соединения. Задача этого устройства в том, чтобы передавать пакеты между связанными с ним компьютерами. Каждый пакет содержит адрес, по которому коммутатор определяет, на какой компьютер отправить данные.

Коммутатор содержит несколько **портов**, каждый из которых может быть подключен к одному устройству, например компьютеру или даже другому коммутатору. Для создания более крупных LAN коммутаторы можно подключать друг к другу через порты. Что произойдет, если случайно их зациклить? Сохранится ли работоспособность сети? К счастью, специалисты об этом уже подумали, и теперь все коммутаторы в мире используют соответствующий алгоритм против зацикливания (Перлман; Perlman, 1985). За выбор пути, по которому должен идти пакет, чтобы в целости и сохранности достичь нужного компьютера, отвечает протокол. В главе 4 мы увидим, как это происходит на деле.

Одну физически большую LAN можно разбить на две меньших логических. Наверное, вам интересно, зачем это может понадобиться. Иногда схема сетевого оборудования не соответствует структуре организации. Например, компьютеры инженерного и финансового отделов компании могут физически находиться в одной сети, поскольку они размещаются в одном крыле здания. Но если бы у каждого из этих отделов была своя **виртуальная LAN (VLAN, Virtual LAN)**, администрирование системы значительно упростилось бы. При такой архитектуре каждый порт будет помечен своим «цветом», скажем, финансовый отдел зеленым, а инженерный — красным. Коммутатор направляет пакеты таким образом, что компьютеры, подключенные к зеленым портам, отделены от подключенных к красным. Зеленый порт не будет получать широковещательные пакеты, отправляемые на красный порт, как будто это две физически отдельные LAN. VLAN подробнее рассматривается в конце главы 4.

Существуют и другие топологии проводных LAN. Известно, что коммутируемая сеть Ethernet представляет собой современный вариант первоначальной архитектуры Ethernet с широковещательной передачей всех пакетов по одному линейному кабелю. Успешно производить передачу сигнала могла только одна машина за раз, а для разрешения конфликтов использовался распределенный механизм арбитража. Применялся простой алгоритм: компьютеры могли осуществлять передачу в любое время, когда кабель не использовался. При конфликте двух или более пакетов каждый компьютер просто ожидал в течение случайного промежутка времени и повторял попытку отправки. Для ясности мы будем называть эту архитектуру **классической Ethernet (classic Ethernet)**. Как вы уже догадались, о ней можно прочитать в главе 4.

И беспроводные, и проводные широковещательные LAN могут выделять ресурсы как статически, так и динамически. Типичный вариант статического выделения ресурсов — разбиение времени на дискретные интервалы и использование циклического алгоритма, при котором каждая машина может передавать сигнал только в свой временной слот. При статическом выделении ресурсов

возможности канала тратятся впустую, если машине нечего передавать или получать в отведенный ей временной слот. Поэтому большинство систем стараются выделять каналы динамически (то есть по требованию).

Методы динамического выделения ресурсов для общего канала делятся на централизованные и децентрализованные. При централизованном методе за определение очередности вещания отвечает единая структура, например базовая станция в сотовых сетях. К примеру, при получении нескольких пакетов она расставляет их в порядке, соответствующем какому-либо внутреннему алгоритму. При децентрализованном методе выделения канала никакой центральной структуры нет; каждая машина сама определяет, когда передавать данные. Может показаться, что такой подход приведет к хаосу, но позднее мы изучим несколько алгоритмов, предназначенных для наведения порядка (при условии, конечно, что все машины соблюдают установленные правила).

## 1.3.3. Домашние сети

Имеет смысл уделить особое внимание LAN для домашних пользователей, то есть **домашним сетям (home networks)**. Домашние сети могут включать широкий диапазон подключенных к интернету устройств и должны быть чрезвычайно легкими в управлении, надежными и безопасными, особенно в руках неспециалистов.

Много лет назад домашняя сеть могла состоять из пары ноутбуков, соединенных беспроводной LAN. Сегодня домашняя сеть может включать смартфоны, беспроводные принтеры, термостаты, охранную и пожарную сигнализацию, лампочки, камеры, телевизоры, стереосистемы, умные колонки, холодильники и многое другое. Стремительное развитие интернета вещей позволяет подключить к сети практически любой бытовой прибор или электронное устройство (включая разнообразные датчики). Подобный размах и разнообразие подключаемых к интернету объектов ставит новые непростые задачи по проектированию, управлению и обеспечению безопасности домашней сети. Все чаще встречается удаленный мониторинг квартир для охраны помещений или для присмотра за пожилыми людьми. Многие люди готовы потратить определенную сумму денег, чтобы обезопасить жизнь своих престарелых родителей.

Несмотря на то что домашняя сеть — просто еще одна разновидность LAN, на практике она заметно отличается от других LAN по нескольким причинам. Во-первых, подключаемые к домашней сети устройства должны быть простыми в установке и обслуживании. Когда-то пользователи очень часто возвращали купленные беспроводные маршрутизаторы, поскольку ожидали, что сеть будет работать сразу же, как только они вытащат устройство из коробки и подключат. Вместо этого им приходилось подолгу общаться по телефону с техподдержкой. Оборудование для домашней сети должно снабжаться защитой от неправильного использования и не требовать от пользователя чтения и детального понимания 50-страничного руководства.

Во-вторых, безопасность и надежность домашних сетей имеет куда более важное значение, поскольку небезопасные устройства — прямая угроза здоровью и благополучию потребителя. Одно дело — потерять несколько файлов из-за

вируса и совсем другое — когда домушник отключает сигнализацию со своего телефона и грабит квартиру. За последние годы мы наблюдали бесчисленные примеры негативных последствий использования плохо защищенных или неправильно функционирующих устройств IoT: от замерзших труб до удаленного управления через вредоносные сторонние скрипты. Отсутствие надежной защиты большинства устройств позволяет злоумышленникам следить за активностью пользователей. Даже если трафик зашифрован, информация о типе передающего устройства, а также об объемах и времени отправки данных может раскрыть немало интересного о частной жизни человека.

В-третьих, домашние сети развиваются органично, по мере покупки и подключения различной бытовой электроники. В результате оборудование, подключенное к домашней сети, может быть очень разноплановым (в отличие от более однородных корпоративных LAN). Несмотря на такое разнообразие, пользователи ожидают, что все эти устройства смогут взаимодействовать друг с другом. Например, они хотят управлять лампами от одного производителя с помощью голосового помощника от другого. После установки устройство может оставаться подключенным на протяжении многих лет (или даже десятилетий). Это значит, что войны интерфейсов недопустимы. Нельзя продать потребителю периферийные устройства с интерфейсом IEEE 1394 (FireWire), а через несколько лет все переделать и объявить «интерфейсом месяца» USB 3.0, затем поменять его на 802.11g — ой, нет, мы имели в виду 802.11n, — нет, постойте, 802.11ac, — простите, мы имели в виду 802.11ax...

Наконец, поскольку размеры прибыли в сфере бытовой электроники невелики, ее производство стараются удешевлять. Многие пользователи выберут более дешевый вариант, если речь идет, скажем, о цифровой фоторамке. Необходимость снижать цены на бытовую электронику еще больше затрудняет достижение вышеуказанных целей. В конце концов, безопасность, надежность и совместимость стоят немалых денег. В ряде случаев и производителю, и потребителю может понадобиться серьезный стимул, чтобы установить и соблюсти общепризнанный стандарт.

Домашние сети обычно являются беспроводными — они удобнее и дешевле, поскольку не нужно прокладывать или, что хуже, перепрокладывать провода. По мере роста количества устройств все более затруднительно устанавливать сетевые порты везде, где есть розетки электропитания. Беспроводные сети удобнее и экономичнее. Впрочем, использование дома исключительно беспроводных сетей порождает специфические проблемы, связанные с производительностью и безопасностью. Во-первых, из-за увеличения объемов трафика и числа подключенных устройств страдает производительность сети. А когда домашняя сеть работает плохо, пользователи традиционно винят в этом интернет-провайдеров (которым обычно это не слишком нравится).

Во-вторых, беспроводные радиоволны могут проходить через стены (в основном это касается частотного диапазона 2,4 ГГц, в меньшей степени — 5 ГГц). И хотя безопасность беспроводных сетей за последнее десятилетие существенно выросла, они все еще подвергаются множеству атак с перехватом информации.

При этом определенные данные, например аппаратные адреса устройств и объем трафика, остаются незашифрованными. В главе 8 мы обсудим использование шифрования для защиты информации, но для неопытных пользователей эта задача не из простых.

**Сети на основе линий электропередач (powerline networks)** позволяют электрическим приборам транслировать информацию по всему дому. Телевизор так или иначе приходится включать в розетку, тогда он подключается и к интернету. Подобные сети способны одновременно передавать и электропитание, и информационный сигнал; это возможно в том числе благодаря использованию разных диапазонов частот.

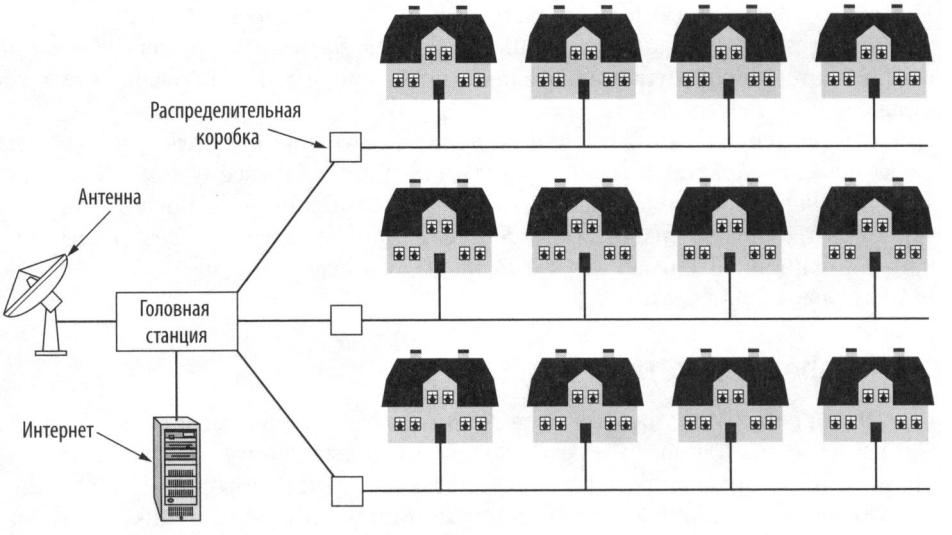
## 1.3.4. Городские сети

**Городская сеть (Metropolitan Area Network, MAN)** охватывает целый город. Самый известный пример — сети кабельного телевидения, которые эволюционировали из предшествующих ТВ-систем коллективного приема. Они использовалась в районах с плохим качеством приема телевизионного сигнала: на ближайшем холме размещалась большая антенна, а полученный сигнал направлялся в дома абонентов.

Сначала эти сети создавались локально, специально под конкретные условия. Далее за дело взялись крупные компании. Они стали получать контракты от местных властей на подключение целых городов. Следующим шагом стало создание телевизионных программ и даже целых каналов специально для кабельного телевидения. Зачастую это были узкоспециализированные каналы: новостные, спортивные, кулинарные, каналы по садоводству и т. д. С самого их появления и до второй половины 1990-х они предназначались исключительно для телевизионного приема.

Когда же аудитория интернета стала более массовой, операторы кабельных сетей поняли, что при небольшой модификации системы они могут предоставлять сервис двустороннего интернета в неиспользуемых частях спектра. На этом этапе кабельные ТВ-сети начали превращаться из простого способа телевещания в городские сети. В первом приближении MAN выглядит примерно так, как показано на илл. 1.8. На этой схеме видно, что как телевизионный сигнал, так и интернет поступают в централизованную **головную станцию кабельной сети (cable head-end)**, то есть **оконечную систему кабельных модемов (CMTS, Cable Modem Termination System)**, для последующего распределения по домам пользователей. Мы вернемся к этому вопросу подробнее в главе 2.

Кабельное телевидение не единственная разновидность MAN. Недавние разработки в области высокоскоростного беспроводного доступа в интернет привели к появлению еще одной MAN, описываемой стандартом 802.16 и широко известной под названием **WiMAX**. Впрочем, особенно популярной она не стала. В этой книге представлены и другие беспроводные технологии: **LTE (стандарт «долгосрочного развития», Long Term Evolution)** и 5G.




**Илл. 1.8.** Городская сеть на основе кабельного ТВ

## 1.3.5. Глобальные сети

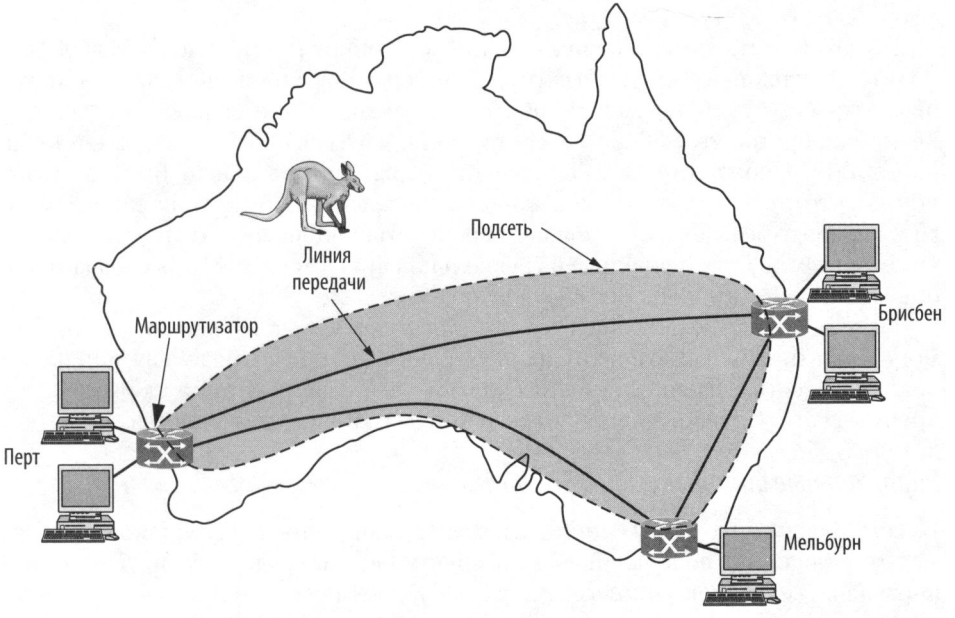
**Глобальная сеть (WAN, Wide Area Network)** охватывает значительные географические площади, зачастую целую страну, континент или даже несколько континентов. WAN может обслуживать частную компанию (корпоративная WAN) или предлагаться в качестве коммерческой услуги (транзитная сеть).

Для начала рассмотрим проводные WAN на примере компании, филиалы которой находятся в разных городах. WAN на илл. 1.9 соединяет филиалы в Перте, Мельбурне и Брисбене[1]. В каждом филиале имеется несколько компьютеров для выполнения пользовательских (то есть прикладных) программ. Согласно общепринятому употреблению, будем называть эти компьютеры **хостами (hosts)**. Оставшаяся часть сети, соединяющая хосты, называется **подсетью связи (communication subnet)**, или просто **подсетью (subnet)**. Подсеть осуществляет передачу сообщений между хостами, точно так же как телефонная система передает слова (на самом деле просто звуки) от говорящего слушающему.

В большинстве случаев подсеть состоит из двух отдельных компонентов: линий передачи и коммутирующих элементов. **Линии передачи (transmission lines)** отвечают за перемещения битов информации между устройствами. В их основе могут лежать медные провода, коаксиальный кабель, оптоволокно или каналы радиосвязи. У большинства организаций нет своих линий передачи, так что им приходится использовать уже существующие линии телекоммуникационных компаний. **Коммутирующие элементы (switching elements)**, или просто **коммутаторы (switches)**, представляют собой специализированные устройства,

[1] Города, расположенные на разных побережьях Австралии. — *Примеч. пер.*

соединяющие две или более линии передачи. При поступлении данных по входящей линии коммутатор выбирает исходящую линию для их отправки.




**Илл. 1.9.** WAN, соединяющая три филиала компании в Австралии

Большинство WAN состоит из множества линий передачи, каждая из которых связывает пару **маршрутизаторов (router)**. Два маршрутизатора, не связанные линией передачи, обязательно должны связываться через другие маршрутизаторы. В сети может быть множество путей, соединяющих два конкретных маршрутизатора. Для поиска наилучшего пути сеть использует **алгоритм маршрутизации (routing algorithm)**. При этом каждый маршрутизатор использует **алгоритм пересылки данных (forwarding algorithm)**, чтобы определить пункт назначения передачи пакета. Оба этих алгоритма подробнее обсуждаются в главе 5.

Здесь не помешает короткий комментарий относительно термина «подсеть». Изначально под ним понимался исключительно набор маршрутизаторов и линий передачи, производивших перемещение пакетов данных из хоста-источника в хост-получатель. Но читателям следует учесть, что недавно он приобрел и второе, более актуальное значение, связанное с сетевой адресацией. Мы поговорим об этом в главе 5, а до тех пор ограничимся исходным значением (набор линий и маршрутизаторов).

Может создаться впечатление, что WAN — это большая проводная LAN, но между ними существуют важные различия помимо длины проводов. Во-первых, в WAN за хосты и подсеть обычно отвечают разные люди. В нашем примере сотрудники компании отвечали бы за свои собственные компьютеры, а IT-отдел компании — за остальную часть сети. В приведенных ниже примерах границы

обязанностей проведены более четко: за обслуживание сети отвечает поставщик сетевых услуг или телефонная компания. Отделение чисто коммуникационных аспектов сети (подсети) от прикладных аспектов (хосты) существенно упрощает ее общую архитектуру.

Во-вторых, маршрутизаторы обычно соединяют различные (в смысле используемых технологий) части сети. Например, внутренняя офисная сеть может оказаться коммутируемым Ethernet, а магистральная линия передачи — соединением по протоколу SONET (представленным в главе 2). И тут какому-нибудь устройству необходимо к ним присоединиться... Внимательный читатель сразу поймет, что это выходит за рамки нашего определения сети. Это значит, что многие WAN на самом деле представляют собой **объединенные сети**, или **интерсети (internetworks)**, составленные из нескольких других сетей. Мы расскажем про объединенные сети подробнее в следующем разделе.

Главное различие определяется тем, *что именно* подключено к подсети. Это могут быть отдельные компьютеры, как в случае подключения к LAN, или целые локальные сети. Именно так из нескольких небольших сетей и формируются крупные сети. С точки зрения подсети при этом ничего не меняется.

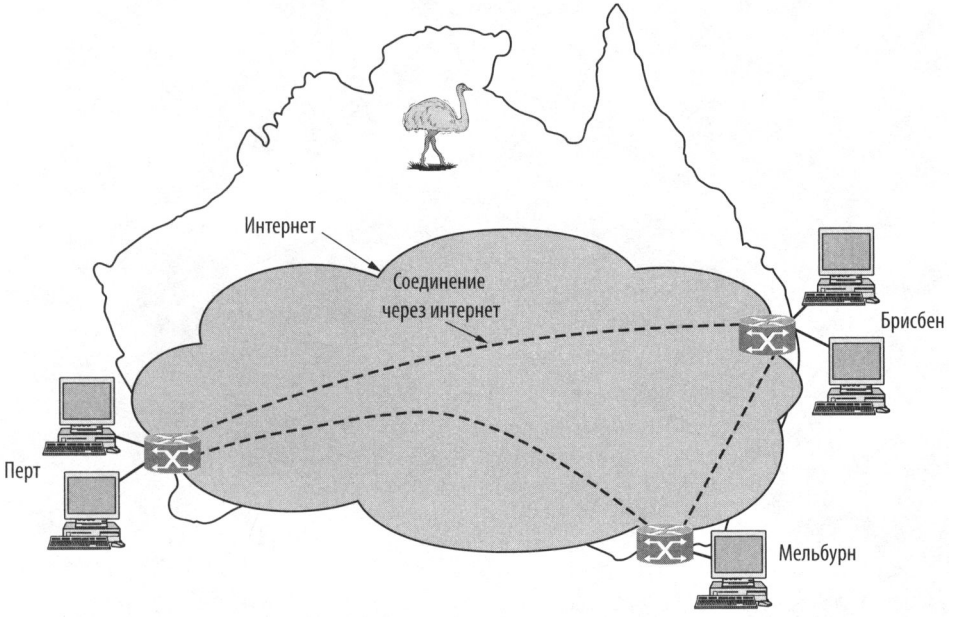
### *Виртуальные частные сети и SD-WAN*

Вместо аренды выделенных линий передачи организация может установить связь между филиалами, подключив их к интернету. Офисы соединяются посредством виртуальных каналов, использующих базовые возможности интернета. Как уже упоминалось ранее, эта схема, приведенная на илл. 1.10, называется **виртуальной частной сетью (VPN, virtual private network)**. В отличие от сетей с выделенными физическими соединениями, у VPN есть свойственное виртуализации преимущество — гибкое переиспользование ресурсов (интернет-соединений). Есть у VPN и столь же типичный для виртуализации недостаток — отсутствие контроля базовых ресурсов. Потенциальные возможности выделенного соединения четко понятны. В случае же VPN производительность может меняться в зависимости от быстродействия используемого интернет-соединения. Обслуживание самой сети может производить коммерческий интернет-провайдер (ISP). Эта схема соединения сайтов WAN друг с другом, а также с остальным интернетом показана на илл. 1.11.

Остальные виды WAN активно используют беспроводные технологии. В спутниковых системах у всех наземных компьютеров есть антенны, посредством которых они обмениваются данными с находящимся на орбите спутником. Все компьютеры «видят» исходящие от спутника данные, а в некоторых случаях также и данные, отправляемые на спутник другими компьютерами системы. Спутниковые сети по своей природе являются широковещательными. Они наиболее удобны при необходимости широкого вещания или при отсутствии наземной инфраструктуры (скажем, представьте себе нефтяников, исследующих удаленную пустыню).

Сеть сотовой связи — еще один пример WAN, использующей беспроводные технологии. Эта система уже насчитывает пять поколений. Первое поколение было аналоговым и предназначалось исключительно для голосовой связи. Второе

поколение было уже цифровым и также предназначалось только для передачи голоса. Третье поколение было цифровым и предназначалось для передачи как голоса, так и данных. Четвертое поколение — чисто цифровое, даже в случае передачи голоса. Пятое поколение (также чисто цифровое) отличается намного более высоким быстродействием по сравнению с четвертым, а также более короткими задержками.

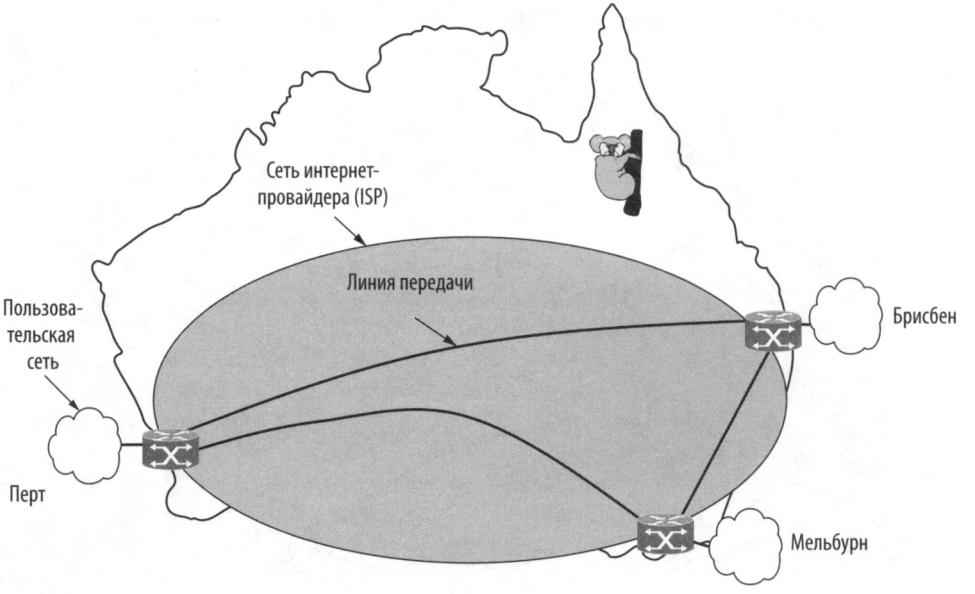



**Илл. 1.10.** WAN на основе виртуальной частной сети

Любая сотовая базовая станция покрывает гораздо большую площадь, чем беспроводная LAN; радиус ее действия исчисляется километрами, а не десятками метров. Базовые станции соединяются друг с другом опорной сетью, обычно проводной. Скорость передачи данных в сотовых сетях обычно составляет порядка 100 Мбит/с, то есть намного меньше, чем в беспроводных LAN, где она может достигать 7 Гбит/с. Подробная информация об этих сетях представлена в главе 2.

В последнее время географически распределенные организации, которым необходимо связывать друг с другом свои филиалы, проектируют и развертывают так называемые **программно-определяемые WAN (SD-WAN)**. Несмотря на использование различных дополняющих друг друга технологий, для всей сети обеспечивается единое **соглашение о качестве предоставления услуги (SLA, Service Level Agreement)**. Например, может использоваться сочетание дорогостоящих арендованных выделенных линий и вспомогательных, более дешевых, стандартных интернет-соединений. Заключенная в ПО логика перепрограммирует коммутирующие элементы в режиме реального времени для оптимизации стоимости и быстродействия сети. SD-WAN — пример

**программно-определяемой сети (SDN, Software-Defined Network)**, технологии, активно набирающей обороты в последнее десятилетие. Она представляет собой общее описание сетевых архитектур, в которых сеть контролируется путем сочетания программируемых коммутаторов с логикой управления, реализованной в виде отдельной программы.




**Илл. 1.11.** WAN на основе предоставляемой ISP сети

## 1.3.6. Объединенные сети

В мире существует множество сетей, нередко использующих разные технологии аппаратного и программного обеспечения. При этом люди, использующие различные сети, хотят общаться между собой. Для этого необходима возможность соединения зачастую несовместимых сетей. Набор взаимосвязанных между собой сетей называется **объединенной сетью (internetwork)**, или просто **интерсетью (internet)**. Обратите внимание, что глобальный интернет является лишь одной из многих интерсетей. Интернет соединяет поставщиков контента, сети доступа, корпоративные, домашние и многие другие сети между собой. Мы детально обсудим интернет далее в этой книге.

Сеть представляет собой сочетание подсети и хостов. Впрочем, термин «сеть» часто применяется также в более расплывчатом (и вводящем в заблуждение) смысле. Например, сетью можно считать подсеть, как в случае с сетью интернет-провайдера (ISP) на илл. 1.11. Интерсеть можно также считать сетью, как в случае WAN на илл. 1.9. Мы будем следовать этой логике, а там, где нужно отличать сеть от других архитектур, будем придерживаться определения сети

как набора вычислительных устройств, соединенных между собой при помощи одной и той же технологии.

Объединенная сеть подразумевает взаимосвязи отдельных, независимо функционирующих сетей. На наш взгляд, соединение LAN и WAN или соединение двух LAN — обычный способ формирования интерсети. Однако стоит уточнить терминологию. По большому счету, если две или более независимые сети платят за взаимное соединение либо две или более сети используют принципиально разные базовые технологии (например, широковещание и двухточечное соединение либо проводную и беспроводную технологии), то речь, вероятно, идет об интерсети.

Устройство, которое реализует соединение между двумя или более сетями и обеспечивает необходимые преобразования с точки зрения аппаратного и программного обеспечения, называется **шлюзом (gateway)**. Шлюзы различаются по уровню в иерархии протокола, на котором они функционируют. Более подробно уровни и иерархии протоколов рассмотрены в следующем разделе. Пока что примите к сведению, что высшие уровни сильнее связаны с приложениями (например, Web), а низшие — с каналами передачи (например, Ethernet). Весь смысл интерсети заключается в соединении компьютеров из разных сетей. Поэтому использовать слишком низкоуровневый шлюз нежелательно — он не сработает для различных типов сетей. Также не подойдет шлюз излишне высокого уровня, так как соединение будет работать только для определенных приложений. Оптимальный уровень, золотую середину, часто называют сетевым уровнем. Шлюзом, производящим коммутацию пакетов на сетевом уровне, служит маршрутизатор. Как правило, интерсети соединяются шлюзами сетевого уровня — маршрутизаторами; впрочем, даже отдельная крупная сеть обычно содержит много маршрутизаторов.

## 1.4. ПРИМЕРЫ СЕТЕЙ

Предмет изучения сетей включает множество их видов, крупных и небольших, популярных и малоизвестных. У них могут быть разные задачи, масштабы и технологии. В следующих разделах мы рассмотрим несколько примеров, чтобы составить представление о разнообразии сферы вычислительных сетей.

Мы начнем с интернета, вероятно, самой известной «сети», и расскажем о его истории, эволюции и используемых технологиях. Далее перейдем к мобильным сетям: в техническом отношении они довольно сильно отличаются от интернета. Затем мы познакомим вас с IEEE 802.11, главным стандартом беспроводных LAN.

### 1.4.1. Интернет

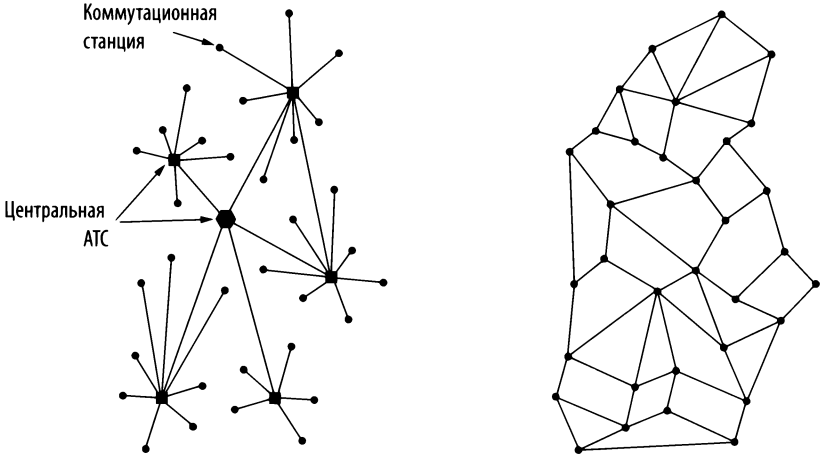
Интернет представляет собой обширное собрание сетей, использующих некоторые общие протоколы и предоставляющих определенные общие сервисы. Необычность этой системы в том, что она не была задумана и не контролируется какой-либо одной организацией. Чтобы лучше разобраться в устройстве

интернета, начнем с истоков и выясним, как он был создан и почему. История создания интернета замечательно описана в книге Джона Нотона (John Naughton, 2000). Это одна из редких книг, которые не только интересны простому читателю, но и содержат при этом 20 страниц ссылок на серьезные исследования. Некоторые материалы в данном разделе основаны на этой книге. Более актуальная история интернета представлена в книге Брайана Маккалоу (Brian McCullough, 2018).

Конечно, интернету, его истории и протоколам также посвящен гигантский массив технической литературы. Больше информации можно найти, например, в работе Северанса (Severance, 2015).

## *ARPANET*

История интернета начинается в конце 1950-х. В самый разгар холодной войны Министерству обороны США понадобилась система командования и управления, способная пережить ядерную войну. В это время все военные коммуникации осуществлялись через общественную телефонную сеть, которая считалась уязвимой. Причины этого ясны из илл. 1.12 (а). На нем черные точки соответствуют коммутационным станциям, к каждой из которых подключены тысячи телефонов. Эти коммутационные станции, в свою очередь, подключены к коммутационным станциям более высокого уровня (центральным АТС). В результате сформировалась общенациональная иерархия с очень незначительной избыточностью. Уязвимость этой системы состояла в том, что уничтожение всего нескольких ключевых центральных АТС раздробило бы ее на множество изолированных участков, так что генералы из Пентагона не смогли бы дозвониться до базы в Лос-Анджелесе.




**Илл. 1.12.** (а) Структура телефонной системы; (б) Предложение Бэрана

В начале 1960-х Минобороны заключило с корпорацией RAND контракт на поиск решения этой проблемы. Один из сотрудников компании, Пол Бэран (Paul Baran), разработал сильно распределенную и отказоустойчивую архитектуру, изображенную на илл. 1.12 (б). Длина пути между любыми двумя коммутационными станциями теперь значительно превышала расстояние, которое аналоговые сигналы могут проходить без искажений. Поэтому Бэран предложил использовать цифровую технологию коммутации пакетов. Бэран написал несколько отчетов для Минобороны, в которых подробно описал свои идеи (Baran, 1964). Представители Пентагона оценили его концепцию и предложили компании AT&T (на тот момент монополиста в сфере телефонных услуг в США) создать опытный образец системы. AT&T сразу же отмели идеи Бэрана. Крупнейшая и богатейшая корпорация в мире не собиралась позволять какому-то выскочке из Калифорнии[1] (AT&T тогда базировались на Восточном побережье США) указывать ей, как выстраивать телефонную систему. В компании заявили, что сеть Бэрана в принципе нереализуема, и идея была загублена на корню.

Прошло несколько лет, а у Минобороны все еще не было улучшенной системы командования и управления. Чтобы понять, что произошло далее, придется вернуться в октябрь 1957-го, когда СССР победил США в космической гонке, запустив первый искусственный спутник Земли. Когда президент США Дуайт Эйзенхауэр попытался выяснить, чей это был недосмотр, он был шокирован тем, как армия, ВМС и ВВС пререкались из-за бюджета Пентагона на исследования. Эйзенхауэр немедленно создал единую организацию для исследований в оборонной сфере, **ARPA (Advanced Research Projects Agency — Управление перспективных исследовательских проектов)**. У ARPA не было своих ученых или лабораторий; фактически оно представляло собой один офис с маленьким (по меркам Пентагона) финансированием. Его работа состояла в распределении грантов и контрактов университетам и компаниям, предлагавшим многообещающие идеи.

В первые несколько лет ARPA занималось поиском своей миссии. В 1967 году Ларри Робертс (Larry Roberts), руководитель проектов в ARPA, пытавшийся найти способ предоставления удаленного доступа к компьютерам, обратил свое внимание на сетевые технологии. Он связался с несколькими экспертами в этой области, чтобы определить порядок действий. Один из них, Уэсли Кларк (Wesley Clark), предложил построить подсеть с коммутацией пакетов, в которой каждый хост был бы связан со своим маршрутизатором.

Поначалу Робертс был настроен скептически, но в конце концов принял эту идею. Он представил несколько туманный доклад на симпозиуме по операционным системам ACM SIGOPS, проводившемся в Гатлинбурге, штат Теннесси, в конце 1967 года (Roberts, 1967). К большому удивлению Робертса, на конференции был представлен еще один доклад, описывающий аналогичную систему. Эта система была не только спроектирована, но и полностью реализована под руководством Дональда Дэвиса (Donald Davies) из Национальной физической лаборатории (NPL) Великобритании. Созданная в NPL система не охватывала

[1] Калифорния находится на Западном побережье. — *Примеч. пер.*

всю страну, а всего лишь соединяла несколько компьютеров на территории NPL. Тем не менее это убедило Робертса в принципиальной реализуемости идеи коммутации пакетов. Кроме того, в упомянутом докладе цитировалась более ранняя забракованная работа Бэрана. Робертс уехал из Гатлинбурга с твердым намерением создать то, что позднее получило название **ARPANET**.

Согласно разработанному им плану, подсеть состояла из мини-компьютеров **IMP (Interface Message Processors — обработчики сообщений интерфейсов)**, соединенных самыми современными на тот момент 56-килобитными линиями передачи. Для повышения надежности каждый IMP соединялся по крайней мере с двумя другими IMP. Все отправляемые через подсеть пакеты содержали полный адрес получателя, так что в случае уничтожения части линий связи и IMP следующие пакеты автоматически перенаправлялись бы по альтернативным путям.

Каждый узел сети представлял собой находящиеся в одном помещении IMP и хост, соединенные коротким кабелем. Хост мог отправлять своему IMP сообщения размером до 8063 бит. Затем IMP разбивал информацию на пакеты максимум по 1008 бит и по отдельности направлял их в пункт назначения. Перед дальнейшей отправкой каждый пакет нужно было получить полностью. Таким образом, эта подсеть стала первой электронной сетью с промежуточным хранением данных и коммутацией пакетов.

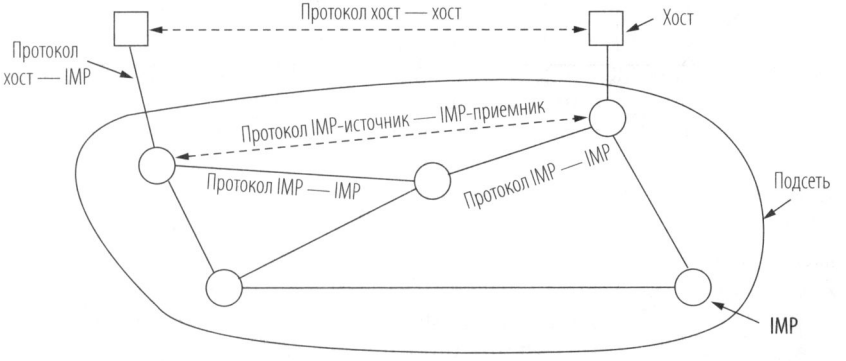
Далее ARPA объявило тендер на создание подсети и получило заявки от 12 компаний. После оценки всех предложений победила консалтинговая компания BBN (Кембридж, штат Массачусетс). В декабре 1968 года ARPA заключило с BBN контракт на разработку подсети и написание для нее программного обеспечения. В качестве IMP были выбраны специально модифицированные мини-компьютеры Honeywell DDP-316 с 12K 16-битных слов памяти на магнитных сердечниках. У этих IMP не было дисков, поскольку наличие движущихся частей сочли понижающим надежность. IMP соединялись между собой 56-килобитными линиями связи, арендованными у телефонных компаний. Сегодня скорость в 56 Кбит/с используется разве что в сельской местности, но тогда это было лучшее из возможного.

Программное обеспечение было разбито на две части: ПО подсети и ПО хоста. ПО подсети состояло из конечного IMP в соединении между хостом и IMP, протокола IMP — IMP и протокола для взаимодействия, передающего и принимающего IMP, созданного для повышения надежности. Первоначальная архитектура ARPANET приведена на илл. 1.13.

Вне подсети также требовалось программное обеспечение, а именно конечный хост в части соединения хоста с IMP, протокол хост — хост, а также прикладное ПО. Вскоре стало понятно, что BBN полагали, что их работа была выполнена, как только сообщение на линии хост — IMP было получено и передано в пункт назначения.

Однако для хостов также требовалось программное обеспечение. Для решения этой проблемы Робертс организовал встречу исследователей сетей (в основном аспирантов) в Сноуберде, штат Юта, летом 1969-го. Участники собрания ожидали, что какой-нибудь эксперт представит грандиозный проект по созданию сети, опишет нужное программное обеспечение, после чего распределит между ними

работу. Они с удивлением обнаружили, что никакого эксперта, как и проекта, нет. Им нужно было самим разобраться, что делать.



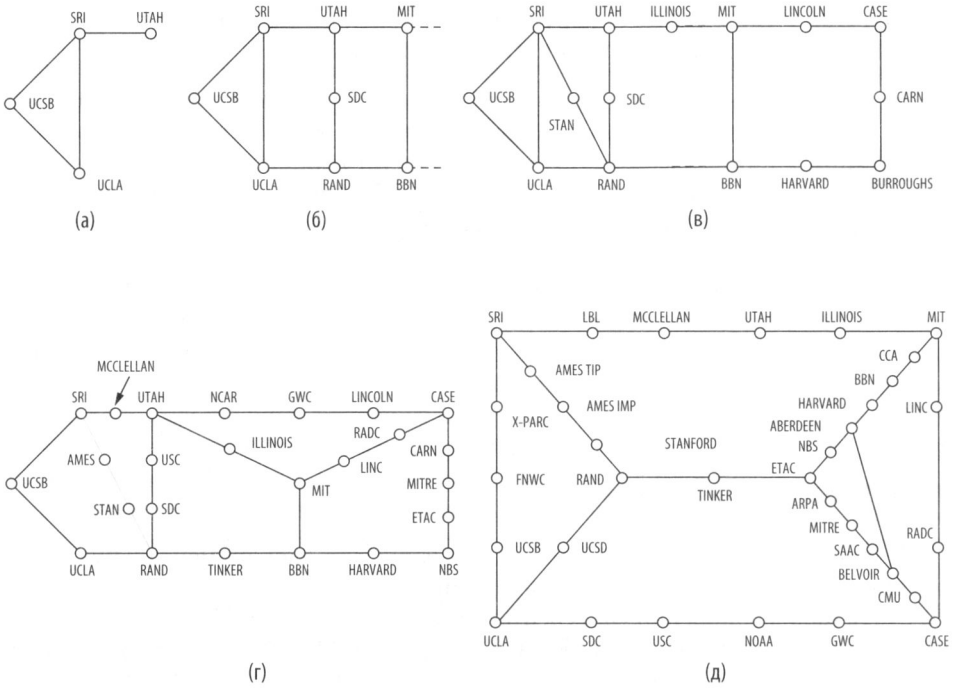

**Илл. 1.13.** Первоначальная архитектура ARPANET

Тем не менее в декабре 1969-го удалось запустить экспериментальную сеть, состоящую из четырех узлов: Калифорнийский университет в Лос-Анджелесе (UCLA), Калифорнийский университет в Санта-Барбаре (UCSB), Стэнфордский исследовательский институт (SRI) и Университет Юты. Эти четыре узла были выбраны, поскольку все они имели значительное количество контрактов с ARPA. Кроме того, их хост-компьютеры были совершенно несовместимы (что делало задачу более интересной). Первое сообщение между хостами было отправлено двумя месяцами ранее из узла UCLA в узел SRI группой под руководством Лена Клейнрока (Len Kleinrock), одного из первопроходцев теории коммутации пакетов. Сеть быстро росла по мере доставки и установки дополнительных IMP, и вскоре она полностью охватила Соединенные Штаты. На илл. 1.14 представлен стремительный рост ARPANET за первые три года.

ARPA не только помогала расти только что созданному ARPANET, но и спонсировала исследования в области спутниковых сетей и мобильных сетей пакетной радиосвязи. В знаменитом ныне эксперименте ехавший по Калифорнии грузовик с помощью пакетной радиосети передавал сообщения в SRI, которые отправлялись далее через ARPANET на Западное побережье, а затем в Университетский колледж Лондона по спутниковой сети. Таким образом находящийся в грузовике исследователь мог использовать лондонский компьютер во время поездки по Калифорнии.

Этот эксперимент также продемонстрировал, что существующие протоколы ARPANET не подходили для работы с различными сетями. Это наблюдение привело к дальнейшим исследованиям в сфере протоколов, завершившихся изобретением TCP/IP (Серф и Кан; Cerf & Kahn, 1974). Протокол TCP/IP был специально разработан для связи через интерсети, и его значение росло по мере подключения к ARPANET все новых сетей.

В качестве стимула для внедрения новых протоколов ARPA заключила несколько контрактов по реализации TCP/IP на различных компьютерных




**Илл. 1.14.** Разрастание ARPANET. (а) Декабрь 1969. (б) Июль 1970. (в) Март 1971. (г) Апрель 1972. (д) Сентябрь 1972

платформах, включая системы IBM, DEC и HP, а также Berkeley Unix. Исследователи из Калифорнийского университета в Беркли переписали TCP/IP на основе нового интерфейса программирования — **сокетов (sockets)** — для предстоящей версии 4.2BSD системы Berkeley Unix. Они также разработали множество приложений, утилит и программ управления, чтобы продемонстрировать, насколько удобно использовать сеть с сокетами.

Момент был выбран идеально. Во многих университетах как раз появился второй или третий компьютер VAX и соединяющая их LAN, но отсутствовало необходимое сетевое программное обеспечение. Как только появилась 4.2BSD — с TCP/IP, сокетами и множеством сетевых утилит, — ее немедленно приняли на вооружение в полном комплекте. Более того, благодаря TCP/IP значительно упрощалось соединение LAN с ARPANET, чем многие университеты и воспользовались. В результате объемы использования TCP/IP в середине 1970-х стремительно выросли.

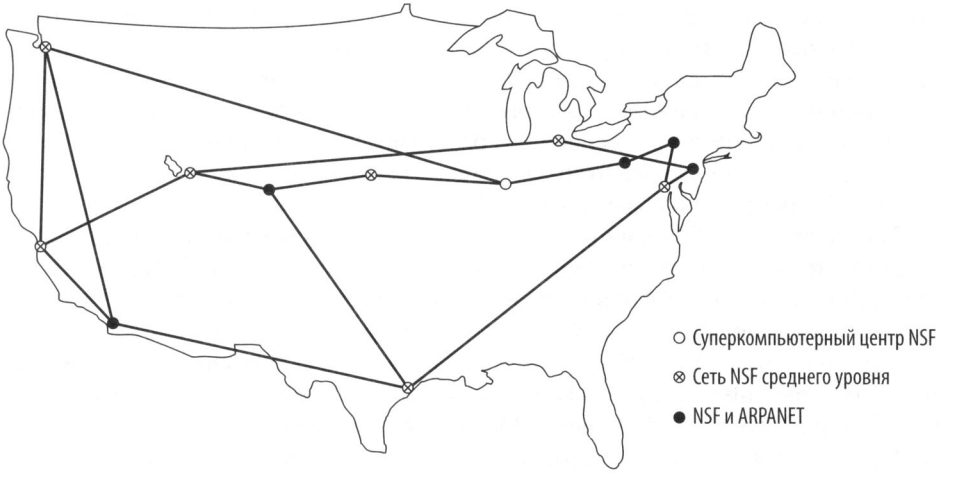
### *NSFNET*

В конце 1970-х в NSF (National Science Foundation, Национальный научный фонд США) осознали, какое колоссальное влияние ARPANET оказывает на университетские исследования, позволяя ученым со всей страны обмениваться

данными и сотрудничать в исследовательских проектах. Однако для подключения к ARPANET университету требовалось заключить научно-исследовательский контракт с Минобороны, что было доступно далеко не для всех. Первой реакцией NSF стало финансирование CSNET (Computer Science Network, сеть факультетов вычислительной техники) в 1981 году. Сеть связывала факультеты вычислительной техники и коммерческие исследовательские лаборатории с ARPANET через модемный доступ и арендованные линии. В конце 1980-х в NSF пошли еще дальше и решили спроектировать собственную систему, доступную для всех университетских исследовательских групп.

Для начала было решено создать опорную сеть, которая соединяла бы шесть суперкомпьютерных центров фонда: в Сан-Диего, Боулдере, Шампейне, Питтсбурге, Итаке и Принстоне. Каждый суперкомпьютер получил в качестве маленького «компаньона» микрокомпьютер LSI-11, так называемый **фаззбол (fuzzball)**. Фаззболы соединялись 56-килобитными арендованными линиями и образовывали подсеть аналогично аппаратному обеспечению ARPANET. Впрочем, программная составляющая отличалась: фаззболы с самого начала «общались» по протоколу TCP/IP, образуя, таким образом, первую WAN на основе TCP/IP.

NSF также финансировал несколько (в конечном счете около двадцати) региональных сетей, соединявшихся с опорной. Это позволяло пользователям из тысяч университетов, исследовательских лабораторий, библиотек и музеев получать доступ к любому из суперкомпьютеров фонда и обмениваться информацией друг с другом. Вся сеть в целом, включая опорную и региональные сети, называлась **NSFNET (National Science Foundation Network, сеть Национального научного фонда США)**. Она подключалась к ARPANET через соединение между IMP и фаззболом в машинном зале Университета Карнеги — Меллона. Первая опорная сеть NSFNET, наложенная на карту США, показана на илл. 1.15.




**Илл. 1.15.** Опорная сеть NSFNET в 1998 г.

NSFNET мгновенно стала популярной и была перегружена с самого запуска. NSF сразу приступил к планированию второй версии и заключил контракт с расположенным в штате Мичиган консорциумом MERIT. Для обеспечения ее работы у компании MCI (купленной Verizon в 2006 году) были арендованы 448-килобитные оптоволоконные каналы. В качестве маршрутизаторов использовались PC-RT компании IBM. Вскоре эта версия также оказалась перегруженной, и к 1990 году ее скорость нарастили до 1,5 Мбит/с.

Сеть продолжала расти, и в NSF осознали, что правительство не будет финансировать ее бесконечно. Кроме того, к проекту хотели присоединиться коммерческие организации, но это было запрещено уставом NSF. В результате NSF предложил MERIT, MCI и IBM сформировать некоммерческую корпорацию, ANS (Advanced Networks and Services) в качестве первого шага к выходу проекта на рынок. В 1990 году контроль над NSFNET перешел к ANS, которая модернизировала 1,5-мегабитные каналы связи до 45-мегабитных, сформировав **ANSNET**. Эта сеть функционировала в течение 5 лет, после чего была продана компании America Online. Но к тому времени коммерческие IP-услуги уже предлагало множество разнообразных компаний, и стало понятно, что правительству пора уходить из этой сферы.

Для упрощения перехода и гарантии взаимодействия всех региональных сетей между собой NSF заключили контракты с четырьмя сетевыми операторами на создание **NAP (Network Access Point, точка доступа в сеть)**. Этими операторами были PacBell (Сан-Франциско), Ameritech (Чикаго), MFS (Вашингтон) и Sprint (Нью-Йорк; для целей NAP город Пеннсокен, штат Нью-Джерси, считался относящимся к Нью-Йорку). Любой сетевой оператор, желающий предоставлять услуги опорной сети региональным сетям NSF, должен был подключаться ко всем NAP.

Такое соглашение означало, что пакет, исходящий из любой региональной сети, мог проходить различными маршрутами из NAP отправки в NAP назначения. В результате сетевые операторы были вынуждены конкурировать между собой с точки зрения цены и качества услуг, что, собственно, и было целью. Таким образом, на смену одной общей опорной сети пришла рыночная конкурентная инфраструктура. Многие критиковали правительство США за консерватизм. Однако в сфере компьютерных сетей именно Пентагон и NSF создали систему, ставшую впоследствии основой для интернета, а после передали ее коммерческим структурам для дальнейшего развития. Все это случилось потому, что компания AT&T, не увидев открывающихся перспектив, отказалась выполнить запрос Минобороны и создать ARPANET.

На протяжении 1990-х множество стран и регионов также создавали свои национальные исследовательские сети, зачастую по образцу ARPANET и NSFNET. В их числе EuropaNET и EBONE в Европе, вначале работающие на скорости 2 Мбит/с, а затем модернизированные до 34-мегабитных линий связи. В конце концов сетевая инфраструктура в Европе также стала коммерческой. С тех давних времен интернет сильно изменился. Он начал активно расти с появлением Всемирной паутины (World Wide Web, WWW) в начале 1990-х. Согласно свежей статистике от Internet Systems Consortium, число хостов в интернете составляет более 1 млрд[1]. Скорее

[1] На момент выхода книги. — *Примеч. ред.*

всего, это заниженная оценка, но в любом случае это количество намного превышает несколько миллионов хостов, насчитывавшихся в период первой конференции по WWW в Европейском центре ядерных исследований (CERN) в 1994 году.

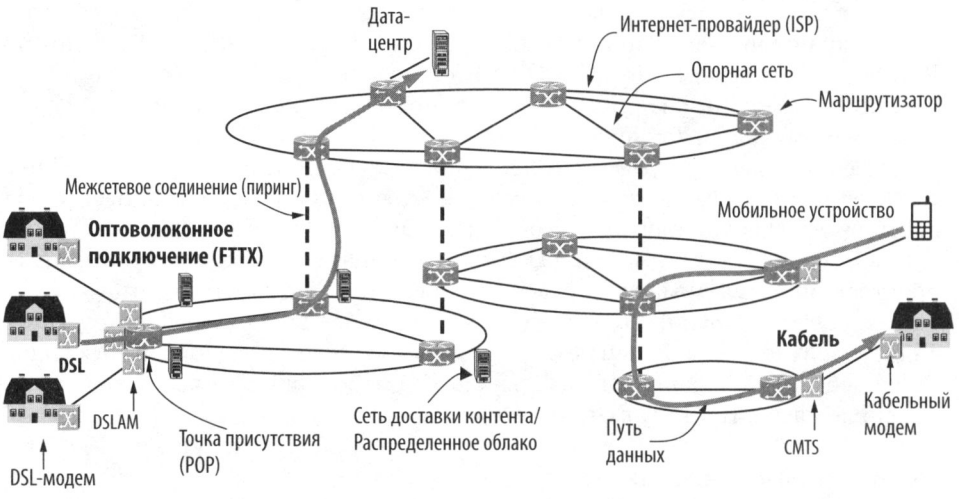
Способы использования интернета также радикально поменялись. Изначально преобладали такие приложения, как электронная почта для научных учреждений, новостные рассылки, удаленный вход в систему и передача файлов. Позднее электронная почта стала всеобщим достоянием, появилась Всемирная паутина и системы одноранговой распределения контента, такие как уже закрытый сервис Napster. Сегодня главными столпами интернета являются распространение мультимедиа в реальном времени и социальные сети (Twitter, Facebook). Основная форма трафика — потоковое видео (Netflix, YouTube). Такое развитие событий привлекло в интернет множество разнообразных видов мультимедиа, а следовательно, и большие объемы трафика. Это, в свою очередь, повлияло на архитектуру самого интернета.

### *Архитектура интернета*

Бурный рост интернета значительно повлиял на его архитектуру. В этом разделе мы попытаемся вкратце описать, как она выглядит сегодня. Вследствие постоянных изменений в работе телефонных компаний, кабельных компаний и интернет-провайдеров ситуация очень сильно запутана, и понять, кто чем занимается, непросто. Движущая сила этих изменений — непрерывное слияние в сфере телекоммуникаций, при котором сети становятся многоцелевыми. Например, при «тройных услугах» одна компания предоставляет услуги телефонии, ТВ и интернета при помощи одного и того же сетевого подключения (причем это обходится дешевле, чем если покупать эти услуги по отдельности). Приведенное здесь описание представляет собой лишь упрощенную версию реальности, а завтрашнее положение дел может отличаться от сегодняшнего.

На илл. 1.16 приведен общий обзор архитектуры интернета. Давайте рассмотрим этот рисунок по частям, начиная с домашних компьютеров (по краям рисунка). Для подключения к интернету компьютер связывается с поставщиком интернет-услуг, которые оплачивает пользователь. Это позволяет компьютеру обмениваться пакетами со всеми остальными доступными узлами. Существует множество разновидностей доступа в интернет, которые обычно отличаются шириной полосы пропускания и стоимостью, но главный признак — подключение.

Чтобы подключиться к интернету из дома, часто используют метод отправки сигналов через инфраструктуру кабельного телевидения. Кабельная сеть, иногда называемая **HFC (Hybrid Fiber-Coaxial — комбинированная оптико-коаксиальная сеть)**, представляет собой единую интегрированную инфраструктуру. Для предоставления разнообразных информационных сервисов (в том числе ТВ-каналов, высокоскоростной передачи данных и голоса) используется пакетный транспортный протокол **DOCSIS (Data Over Cable Service Interface Specification — стандарт передачи данных по телевизионному кабелю)**. На стороне домашнего пользователя располагается специальное устройство — **кабельный модем (cable modem)**, а в головном узле кабельной сети располагается **CMTS (Cable Modem Termination System — система оконечных кабельных**




**Илл. 1.16.** Обзор архитектуры интернета

**модемов)**. Термин **модем** (сокращение от «*мо*дулятор/*дем*одулятор») используется для любых устройств, предназначенных для преобразования между цифровым битовым представлением и аналоговыми сигналами.

Сети доступа ограничиваются пропускной способностью «последней мили» (последнего участка передачи сигнала). За прошедшее десятилетие развитие стандарта DOCSIS обеспечило намного более высокую пропускную способность для домашних сетей. В последней на текущий момент версии этого стандарта, полнодуплексном DOCSIS 3.1, появилась поддержка симметричной нисходящей/восходящей передачи данных с максимальной скоростью до 10 Гбит/с. Еще один вариант развертывания «последней мили» — прокладка оптоволоконных кабелей до жилых домов на основе технологии **FTTH (Fiber to the Home — «оптоволокно в дом»)**. Для предприятий, расположенных в промышленных районах, имеет смысл арендовать выделенную высокоскоростную линию передачи данных от офиса до ближайшего ISP. В некоторых крупных городах мира доступна аренда линий скоростью до 10 Гбит/с (или ниже). Например, линия T3 обеспечивает передачу данных со скоростью примерно в 45 Мбит/с. В других регионах, особенно в развивающихся странах, не проложены ни кабели, ни оптоволокно. Некоторые из них сразу стали использовать высокоскоростные беспроводные или мобильные сети в качестве основного вида интернет-доступа. В следующем разделе представлен обзор мобильного доступа в интернет.

Итак, мы можем передавать пакеты между домами конечных пользователей и ISP. Мы будем называть место входа пакетов пользователя в сеть ISP **точкой присутствия (POP, Point of Presence)** оператора связи. Далее мы расскажем, как пакеты перемещаются между точками присутствия различных ISP. С этого момента система является полностью цифровой и коммутируемой.

Сети ISP могут быть региональными, общенациональными или интернациональными. Их архитектура включает магистральные линии передачи,

связывающие между собой маршрутизаторы в точках присутствия в городах, обслуживаемых этим ISP. Это оборудование называется **опорной сетью (backbone)** ISP. Пакет, предназначенный для хоста, за который отвечает непосредственно данный ISP, проходит по маршруту через опорную сеть и доставляется в нужный хост. В противном случае он передается другому ISP.

ISP связывают свои сети для обмена трафиком в **точках обмена интернет-трафиком (IXP, Internet eXchange Points)**. В случае подключенных друг к другу ISP говорят о **пиринге (peering)**. Существует множество ISP в городах по всему миру. Они отображены на илл. 1.16 вертикально, потому что сети ISP географически пересекаются. Фактически точки обмена трафиком представляют собой здание, полное маршрутизаторов, как минимум по одному на каждый ISP. Все маршрутизаторы соединяются высокоскоростной оптической LAN, позволяющей перенаправлять пакеты из опорной сети одного ISP в опорную сеть любого другого ISP. IXP могут представлять собой крупные узлы, принадлежащие различным компаниям, конкурирующим друг с другом. Одна из крупнейших точек располагается в Амстердаме (Amsterdam Internet Exchange, AMS-IX); к ней подключено более 800 ISP, которые могут передавать через нее более чем 4000 гигабит (4 терабита) трафика *каждую секунду*.

Пиринг в IXP зависит от деловых отношений между ISP. Существует множество возможных видов таких отношений. Например, маленький ISP может платить более крупному за интернет-соединение для достижения удаленных хостов, аналогично тому как потребитель приобретает услуги ISP. В этом случае считается, что этот маленький ISP платит за **транзит**. Один из множества парадоксов интернета: провайдеры, публично конкурирующие за потребителей, зачастую тайно сотрудничают, чтобы осуществлять пиринг (Мец; Metz, 2001).

Путь прохождения пакетов через интернет зависит от выбранного ISP способа обмена трафиком. Если ISP, доставляющий пакет, обменивается трафиком с ISP, для которого этот пакет предназначается, он может доставить пакет напрямую. В противном случае пакет маршрутизируется в ближайшее место подключения к платному поставщику услуг транзита, который может доставить пакет. Два примера путей пакета через ISP показаны на илл. 1.16. Зачастую пакет проходит не по кратчайшему пути, а по наименее загруженному или самому дешевому.

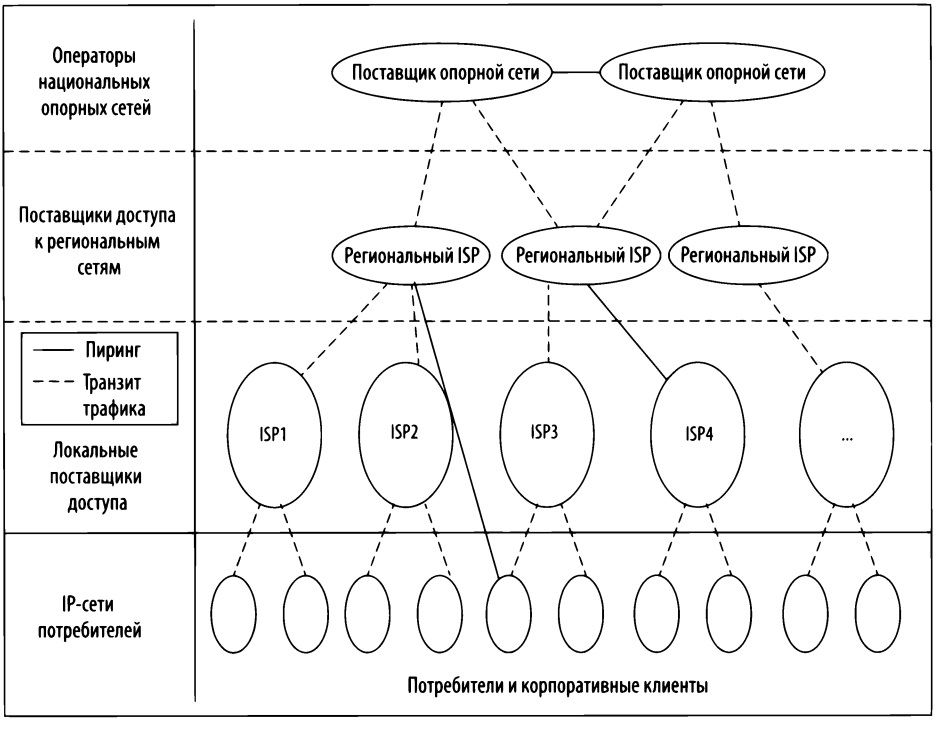
Небольшая горстка **поставщиков услуг транзита (transit providers)**, включая AT&T и Level 3, контролирует крупные международные опорные сети с тысячами маршрутизаторов, соединенных высокоскоростными оптоволоконными линиями связи. Их также называют **tier-1-операторами**[1] **(tier-1 operators)**. Они не платят за транзит трафика и формируют опорную сеть интернета, поскольку все остальные должны подключаться к ним для доступа ко всему интернету.

Серверы компаний, предоставляющих большое количество контента (например Facebook или Netflix), располагаются в **центрах обработки данных**, или **дата-центрах (data centers)**, с хорошим соединением со всем остальным интернетом. Эти центры проектируются в расчете на размещение компьютеров,

[1] Или «операторами верхнего уровня». — *Примеч. пер.*

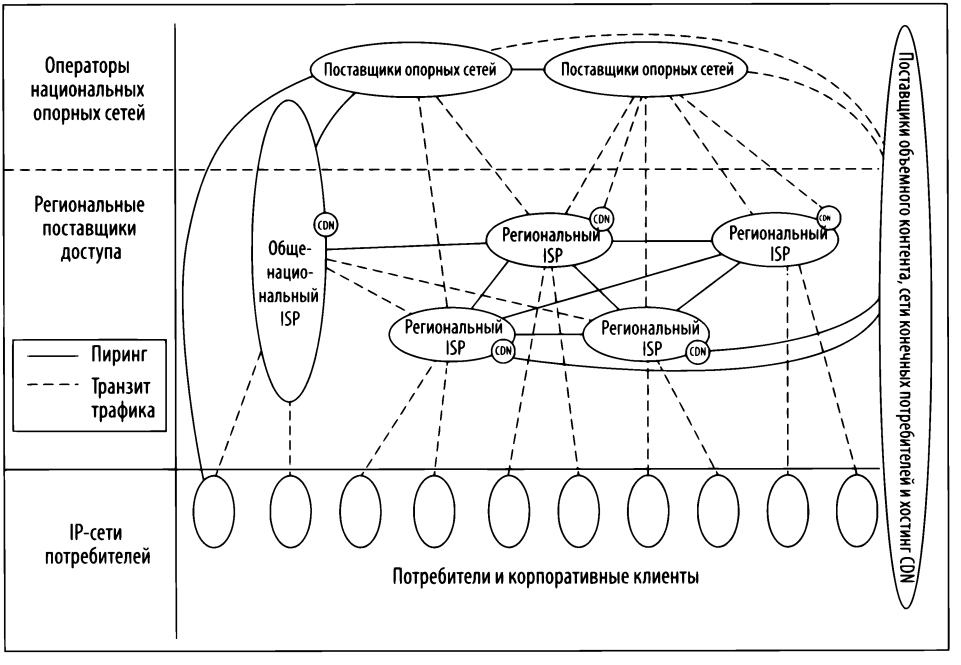
а не людей и могут содержать бесчисленные стойки с машинами. Подобные инженерные системы называются **фермой (или «парком») серверов (server farm)**. Предоставляемые дата-центрами услуги **колокейшн**, или **хостинга**, позволяют потребителям размещать оборудование (например, серверы) в точках присутствия ISP для обеспечения коротких и быстрых соединений между этими серверами и опорной сетью ISP. Индустрия интернет-хостинга все сильнее виртуализируется, так что аренда работающей на сервере виртуальной машины вместо установки реального компьютера стала обычным делом. Центры обработки данных настолько велики (сотни тысяч или миллионы машин), что основная статья их расходов — электричество. Поэтому иногда их специально строят в местах с дешевым электричеством. Например, компания Google построила дата-центр за $2 млрд в городке Те-Деллс (штат Орегон) из-за его близости к громадной гидроэлектростанции на могучей реке Колумбии, снабжающей его дешевым, экологически чистым электричеством.

Традиционно архитектура интернета считается иерархической; наверху находятся операторы tier-1, другие сети — ниже на один или несколько уровней, в зависимости от того, идет ли речь о больших региональных сетях или о меньших сетях доступа (как показано на илл. 1.17). Впрочем, за последнее десятилетие




**Илл. 1.17.** Архитектура интернета на протяжении 1990-х была иерархической

эта иерархия существенно эволюционировала и резко «схлопнулась», как видно на илл. 1.18. Импульсом для этой реорганизации послужило появление «сверхгигантских» поставщиков контента, включая Google, Netflix, Twitch и Amazon, а также крупных, распределенных по всему миру CDN, таких как Akamai, Limelight и Cloudflare. Они снова поменяли архитектуру интернета. В прошлом этим компаниям пришлось бы использовать транзитные сети для доставки контента местным ISP. Теперь поставщики интернет-услуг и контента стали настолько успешными и масштабными, что часто подключаются непосредственно друг к другу во множестве разных точек. Как правило, данные передаются напрямую от ISP поставщику контента. Иногда поставщик контента даже размещает серверы внутри сети ISP.



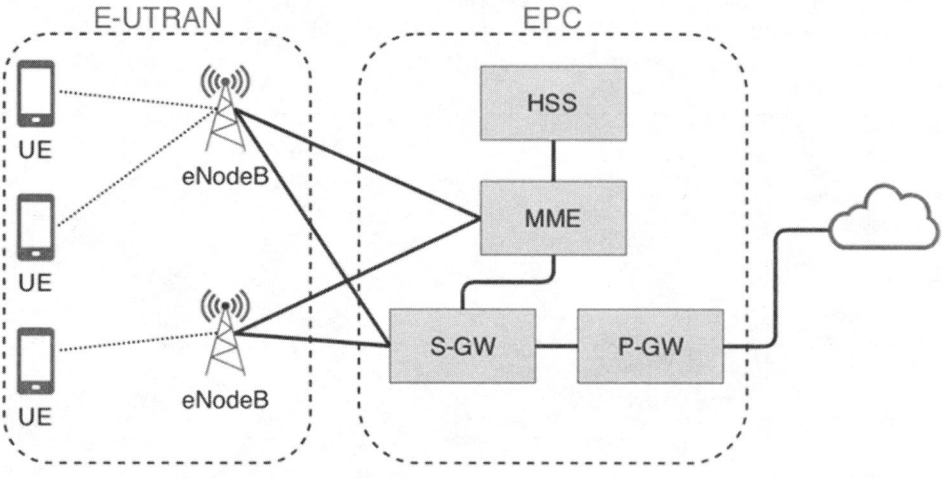

**Илл. 1.18.** «Схлопывание» иерархии интернета

## 1.4.2. Мобильные сети

Мобильные сети насчитывают сегодня более 5 млрд абонентов по всему миру — это примерно 65 % населения земного шара. Многие, если не большинство, из этих абонентов получают доступ в интернет с помощью своих мобильных устройств (ITU, 2016). В 2018 году мобильный интернет-трафик составил более половины глобального онлайн-трафика. Итак, далее у нас на очереди изучение мобильной телефонной системы.

### *Архитектура мобильной сети*

Архитектура мобильной телефонной сети весьма отличается от архитектуры интернета. Она состоит из нескольких частей, как видно на примере упрощенного варианта архитектуры 4G LTE (илл. 1.19). Этот стандарт мобильных сетей — один из самых распространенных и останется таковым вплоть до замены его 5G, сетью пятого поколения. Позже мы обсудим историю различных поколений мобильных сетей.




**Илл. 1.19.** Упрощенная архитектура сети 4G LTE

Начнем с сети **E-UTRAN (Evolved UMTS Terrestrial Radio Access Network — усовершенствованный беспроводной интерфейс 3GPP (LTE))**. Под этим хитрым названием скрывается протокол эфирной радиосвязи между мобильными устройствами (например, сотовыми телефонами) и **сотовой базовой станцией (cellular base station)**, называемой сегодня **eNodeB**. **UMTS (Universal Mobile Telecommunications System — универсальная мобильная телекоммуникационная система)** — официальное название сотовой телефонной сети. Достигнутый за последние десятилетия в сфере радиоинтерфейсов прогресс привел к существенному росту скоростей беспроводной передачи данных (и они продолжают расти). В основе этого радиоинтерфейса лежит **CDMA (Code Division Multiple Access — множественный доступ с кодовым разделением)**; эта методика представлена в главе 2.

Сотовая базовая станция в совокупности с ее контроллером образует **сеть радиодоступа (radio access network)**, которая составляет беспроводную часть мобильной телефонной сети. Узел-контроллер, он же **контроллер радиосети (Radio Network Controller, RNC)**, определяет использование полосы частот. Базовая станция реализует радиоинтерфейс.

Остальная часть мобильной телефонной сети отвечает за передачу трафика сети радиодоступа и называется **ядром сети (core network)**. В сетях 4G ядро

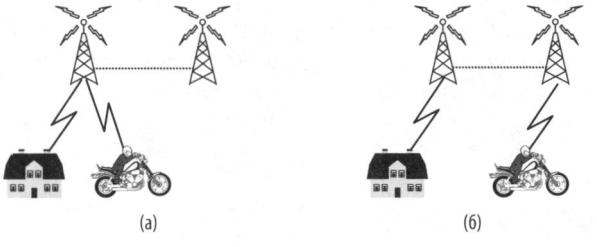
сети перешло на коммутацию пакетов и теперь называется **развитым пакетным ядром (Evolved Packet Core, EPC)**. Ядро сети 3G UMTS возникло на основе ядра предшествовавшей ей сети 2G GSM, а EPC 4G завершило переход к полностью пакетно-коммутируемому ядру сети. Система 5G также полностью цифровая. Пути назад больше нет: аналоговые системы вымерли, как птицы додо.

Сервисы данных стали куда более важной частью мобильной сети, чем раньше, во времена обмена текстовыми сообщениями и первых сервисов пакетного обмена данными, таких как **GPRS (General Packet Radio Service — пакетная радиосвязь общего пользования)** в системе GSM. Скорость первых сервисов данных составляла десятки килобит в секунду, но пользователям хотелось большего. Более современные мобильные сети поддерживают скорость в несколько мегабит в секунду. Для сравнения: при голосовом звонке данные передаются со скоростью 64 Кбит/с, а при использовании сжатия — в 3–4 раза меньше.

Для передачи всех этих данных узлы ядра сети UMTS подключаются непосредственно к сети с коммутацией пакетов. **Обслуживающий сетевой шлюз S-GW (Serving Network Gateway — обслуживающий сетевой шлюз)** и **P-GW (Packet Data Network Gateway — сетевой шлюз пакетного обмена данными)** доставляют пакеты данных на мобильные телефоны (и обратно) и подключаются к внешним пакетным сетям (например, к интернету).

Этот переход продолжится и в будущих мобильных телефонных сетях. Интернет-протоколы используются даже на мобильных телефонах для установки соединений при голосовых вызовах через сети пакетной передачи данных (IP-телефония). IP-протокол и пакеты используются на всем пути данных, от радиосвязи и до ядра сети. Конечно, архитектура IP-сетей также меняется для обеспечения лучшего качества обслуживания. Если бы не это, то пользователи, оплачивающие услуги, вряд ли обрадовались бы прерывистому звуку и подергивающемуся видео. Мы вернемся к этому вопросу в главе 5.

Еще одно различие между мобильными телефонными сетями и обычным интернетом — мобильность. Когда пользователь выходит из зоны приема одной базовой сотовой станции и входит в другую, необходимо перенаправить поток данных со старой станции на новую. Эта методика, проиллюстрированная на илл. 1.20, называется **передачей обслуживания (handover, handoff)**.




**Илл. 1.20.** Передача обслуживания мобильного телефона. (а) До. (б) После

При падении качества сигнала запросить передачу обслуживания может как мобильное устройство, так и базовая станция. В некоторых мобильных сетях, обычно использующих технологию CDMA, можно подключиться к новой

базовой станции, прежде чем отключиться от старой. Благодаря этому отсутствует перерыв в обслуживании, что повышает качество соединения. Такой подход называется **мягкой передачей обслуживания (soft handover)**, в отличие от **жесткой передачи обслуживания (hard handover)**, при которой мобильный телефон сначала отключается от старой базовой станции и только потом подключается к новой.

В связи с этим возникает проблема: необходимо найти мобильный телефон при поступлении входящего вызова. В ядре каждой мобильной сети находится **HSS (Home Subscriber Server — сервер абонентских данных)**, которому известно местоположение каждого абонента, равно как и прочая информация из профиля, необходимая для аутентификации и авторизации. Таким образом, любой мобильный телефон можно найти, обратившись за информацией в HSS.

Осталось обсудить только вопрос безопасности. Исторически телефонные компании относились к безопасности намного серьезнее, чем интернет-компании, поскольку хотели избежать мошенничества при оплате услуг пользователями. В процессе эволюции технологий от 1G к 5G сотовые компании сумели реализовать некоторые базовые механизмы безопасности для мобильных телефонов.

Начиная с системы 2G GSM, мобильные телефоны состоят из двух частей: переносного телефонного аппарата и съемного чипа, содержащего идентификационные данные и информацию о состоянии счета абонента. Неофициально этот чип называют **SIM-картой** — сокращение от Subscriber Identity Module (модуль идентификации абонента). SIM-карту можно переставить из одного мобильного телефона в другой и таким образом активировать его. Она предоставляет возможности для обеспечения безопасности. Когда абоненты GSM посещают другие страны в деловых или туристических целях, они обычно берут с собой свой телефон, но покупают за несколько долларов новую SIM-карту на месте, чтобы звонить внутри страны без дополнительной платы за роуминг.

В целях борьбы с мошенничеством содержащаяся на SIM-карте информация используется мобильной телефонной сетью для идентификации абонента и проверки, имеет ли он право использовать сеть. В случае UMTS мобильный телефон с помощью этих данных еще и проверяет легитимность сети подключения.

Еще один важный вопрос — защита персональной информации. Беспроводные сигналы отправляются на все приемные устройства, расположенные поблизости. Поэтому, во избежание подслушивания разговоров, для шифрования передаваемых данных используются криптографические ключи, расположенные на SIM-карте. Такой подход обеспечивает куда большую безопасность, чем в системах 1G, через которые можно было очень легко подслушать разговоры. Однако он не решает всех проблем, поскольку в схемах шифрования также встречаются слабые места.

### *Коммутация пакетов и коммутация каналов*

С самого появления вычислительных сетей шла непрерывная война между приверженцами сетей с коммутацией пакетов (без установления соединений) и сетей с коммутацией каналов (с ориентацией на соединения). Основные сторонники **коммутации пакетов (packet switching)** пришли из интернет-сообщества.

В архитектуре без установления соединений маршрутизация всех пакетов производится независимо друг от друга. В результате отказ части маршрутизаторов во время сеанса не несет никаких тяжелых последствий, поскольку система может динамически изменить конфигурацию и найти другой путь для последующих пакетов. Если один из маршрутизаторов получит слишком много пакетов за некоторый промежуток времени, он не сможет их все обработать и, вероятно, часть пакетов будет утеряна. Отправитель в конце концов заметит это и отправит данные повторно. При этом качество обслуживания все равно пострадает, если только используемое приложение не спроектировано с учетом возможной нестабильности.

Сторонники **коммутации каналов (circuit switching)** пришли из мира телефонных компаний. В телефонных системах вызывающий абонент должен набрать номер телефона и дождаться соединения, прежде чем говорить или отправлять данные. Это соединение формирует маршрут в телефонной системе, поддерживаемый вплоть до завершения разговора. Все произнесенные слова или пакеты следуют по одному пути. В случае сбоя линии или маршрутизатора звонок прерывается — эта архитектура явно менее отказоустойчивая, чем вариант без соединений.

При коммутации пакетов поддержание уровня качества обслуживания упрощается. Благодаря заблаговременному установлению соединения подсеть может зарезервировать полосу пропускания линии связи, место в буфере коммутатора и время CPU. Если при попытке позвонить оказывается, что ресурсов недостаточно, звонок отклоняется, а вызывающая сторона слышит сигнал «занято». Если соединение все-таки установлено, то уровень качества обслуживания будет хорошим.

На илл. 1.19 любопытно то, что в ядре сети присутствует оборудование как для коммутации пакетов, так и для коммутации каналов. Это значит, что мобильные сети находятся в переходной стадии и телефонные компании могут реализовывать любой из этих вариантов, а иногда и оба сразу. В более старых мобильных телефонных сетях для голосовых звонков используется ядро с коммутацией каналов как в обычных телефонных сетях. Эта технология до сих пор встречается в сети UMTS в таких элементах, как **MSC (Mobile Switching Center — мобильный коммутационный центр)**, **GMSC (Gateway Mobile Switching Center — шлюзовый коммутационный центр мобильной связи)** и **MGW (Media Gateway — медиашлюз)**. Они служат для установления соединения через опорную сеть с коммутацией каналов, например **PSTN (Public Switched Telephone Network — коммутируемая телефонная сеть общего пользования)**.

### *Первые поколения мобильных сетей: 1G, 2G и 3G*

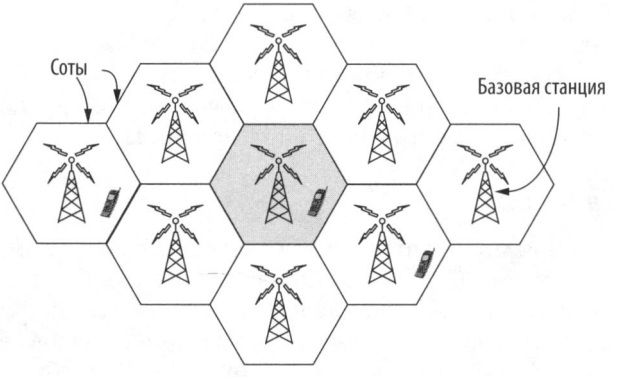
За последние 50 лет архитектура мобильных сетей невероятно разрослась и претерпела значительные изменения. Системы мобильной телефонной связи первого поколения передавали голосовые звонки в виде непрерывно меняющихся сигналов (аналоговые системы), а не последовательности битов (цифровые системы). Одной из широко используемых систем первого поколения была **AMPS (Advanced Mobile Phone System — продвинутая система мобильной**

**телефонной связи)**, развернутая в США в 1982 году. Системы второго поколения перешли на передачу голосовых звонков в цифровом виде. Это привело к увеличению мощности, повышению безопасности и обеспечению обмена текстовыми сообщениями. В 1991 году начала внедряться и широко использоваться по всему миру система второго поколения **GSM (Global System for Mobile communications — глобальная система мобильной связи)**.

Системы третьего поколения, 3G, появились в 2001 году. Они обеспечили цифровую передачу голосовых звонков и широкополосную цифровую передачу данных. В 3G-системах существует множество различных стандартов. **Международный союз электросвязи, МСЭ (International Telecommunication Union, ITU)**, который мы обсудим позже в этой главе, условно определил 3G как стандарт, обеспечивающий скорость как минимум 2 Мбит/с для неподвижных и идущих пользователей и 384 Кбит/с при передвижении на транспорте. UMTS — главная 3G-система, используемая по всему миру, а также основа для разнообразных последующих версий. Она способна обеспечить скорость входящей информации до 14 Мбит/с, а исходящей — около 6 Мбит/с. В будущих версиях планируется использование нескольких антенн и радиопередатчиков для дальнейшего повышения скорости.

Наиболее дефицитный ресурс в 3G-системах, как и в предшествующих им 2G и 1G, — диапазон радиочастот. Правительства дают лицензию на части диапазона радиочастот операторам мобильной связи. Как правило, это происходит в виде аукционов частот, на которых сетевые операторы представляют предложения. Доступ к части лицензированного диапазона упрощает проектирование и эксплуатацию системы, поскольку больше никто не сможет осуществлять передачу на этих частотах, но стоит немалых денег. Например, в Великобритании в 2000 году пять лицензий 3G ушли с аукциона примерно за $40 млрд.

Именно нехватка частот привела к созданию **сотовой сети (cellular network)**, применяемой в настоящее время для мобильных телефонных сетей (илл. 1.21). Для борьбы с радиопомехами, возникающими между пользователями, область покрытия делится на соты. В пределах одной соты пользователям выделяются каналы, не влияющие друг на друга и не вызывающие помех в соседних сотах.




**Илл. 1.21.** Сотовая архитектура мобильных телефонных сетей

Это дает возможность **повторного использования частот (frequency reuse)** в смежных сотах, что повышает производительность сети в целом. В системах 1G, где каждый голосовой звонок передавался в определенной полосе частот, частоты тщательно подбирались так, чтобы не конфликтовать с соседними сотами. При этом одну частоту можно было использовать только один раз в нескольких смежных сотах. В современных системах 3G каждая сота может использовать все частоты, но таким образом, что уровень помех в соседних сотах остается допустимым. Существует несколько вариантов сотовой архитектуры, включая использование направленных (секторных) антенн на сотовых вышках для дальнейшего снижения взаимных помех, но основной принцип остается неизменным.

### *Современные мобильные сети: 4G и 5G*

Мобильным телефонным сетям предстоит сыграть важную роль в развитии будущих сетей. Сегодня они скорее ориентированы на мобильные широкополосные приложения (например, доступ в интернет с телефона), чем на голосовые звонки. Это серьезно влияет на радиоинтерфейс, архитектуру ядра и безопасность будущих сетей. Технологии 4G и 4G LTE, появившиеся в конце 2000-х, обеспечивают более высокие скорости.

Сети 4G LTE очень быстро стали основным способом мобильного доступа в интернет, опередив своих конкурентов, таких как стандарт 802.16 (**WiMAX**). Технологии 5G обещают еще большие скорости — до 10 Гбит/с. Их широкомасштабное развертывание планируется в начале 2020-х. Эти технологии в основном различаются используемым диапазоном частот. Например, 4G использует полосы частот до 20 МГц; 5G разработан в расчете на полосы намного более высоких частот, до 6 ГГц. Проблема с переходом на более высокие частоты состоит в том, что высокочастотные сигналы не способны покрывать такое же расстояние, как низкочастотные. Система 5G должна предусматривать затухание сигнала, взаимные помехи и ошибки, используя новейшие алгоритмы и технологии, включая массивы антенн **MIMO («multiple input, multiple output» — «несколько входов, несколько выходов»)**. Кроме того, короткие микроволны на этих частотах легко поглощаются водой, поэтому нужны дополнительные усилия, чтобы обеспечить работу системы во время дождя.

## 1.4.3. Беспроводные сети (Wi-Fi)

С появлением ноутбуков люди стали мечтать о возможности волшебным образом подключать их к интернету, едва зайдя в офис. Множество различных организаций годами работали для достижения этой цели. Наконец было найдено наиболее разумное решение. Оно заключалось в том, чтобы оборудовать как офис, так и ноутбуки радиопередатчиками и радиоприемниками короткого радиуса действия для обмена информацией.

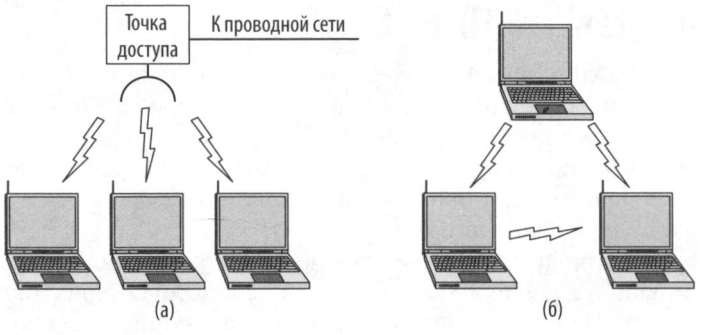
Работы в этой сфере быстро привели к появлению на рынке беспроводных LAN от различных компаний. Проблема была в том, что они были совершенно несовместимы друг с другом. Изобилие стандартов означало, что компьютер, оборудованный радиоприемником от бренда *X*, не сможет подключиться к интернету

в помещении с точкой доступа от бренда *Y*. В середине 1990-х было решено, что имеет смысл создать беспроводной стандарт LAN, и комитет IEEE, занимавшийся стандартизацией проводных LAN, получил такое задание.

Прежде всего нужно было ответить на самый простой вопрос: как его назвать? Все прочие стандарты для 802 LAN, созданные комитетом стандартизации IEEE, получали номера по порядку, от 802.1 и 802.2 до 802.10, поэтому беспроводной стандарт LAN получил название 802.11. Поистине гениально. На профессиональном жаргоне его называют **Wi-Fi**. Однако это важный стандарт, заслуживающий уважения, так что мы будем использовать для него более официальное название — 802.11. За прошедшие годы возникло множество вариантов и версий стандарта 802.11.

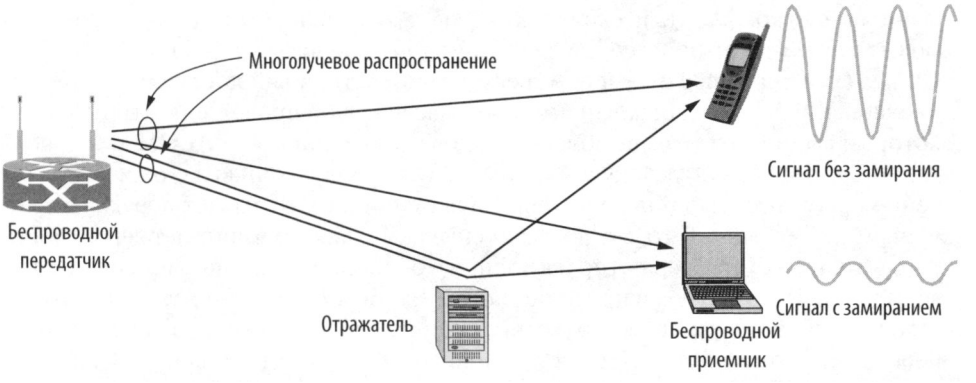
Дальнейшие задачи были посложнее. Необходимо было найти подходящую и свободную (причем желательно по всему миру) полосу частот. В результате был выбран подход, противоположный тому, который использовался в мобильных телефонных сетях. Вместо дорогостоящих лицензируемых частот системы 802.11 работают на нелицензируемых полосах частот, например **ISM («Industrial, Scientific, and Medical» — «промышленные, научные и медицинские»)**, устанавливаемых МСЭ-R (например, 902–928 МГц, 2,4–2,5 ГГц, 5,725–5,825 ГГц). Этот диапазон частот разрешено использовать любым устройствам, но мощность их излучения должна быть ограничена, чтобы различные устройства не мешали друг другу. Конечно, из-за этого 802.11-передатчики иногда начинают конкурировать за частоты с беспроводными телефонами, системами дистанционного открывания дверей гаража и микроволновками. Так что до тех пор, пока пользователям не понадобится позвонить гаражным дверям, важно все настроить правильно.

Сети 802.11 состоят из клиентских устройств, таких как ноутбуки и мобильные телефоны, а также **точек доступа (access points, AP)** — инфраструктур, располагаемых в зданиях. Точки доступа иногда называются **базовыми станциями (base stations)**. Они подключаются к проводной сети, через них осуществляется весь обмен данными между клиентами. Клиенты, находящиеся в зоне радиодоступа, могут также взаимодействовать напрямую: например, в случае с двумя компьютерами в офисе без точки доступа. Подобная схема называется **динамической (самоорганизующейся) сетью (ad hoc network)** и используется намного реже сети с точкой доступа. Оба варианта показаны на илл. 1.22.




**Илл. 1.22.** (а) Беспроводная сеть с точкой доступа. (б) Динамическая сеть

Передача данных по стандарту 802.11 осложняется условиями беспроводной передачи, которые меняются при малейших изменениях окружающей среды. На используемых для 802.11 частотах радиосигналы могут отражаться от твердых тел, так что приемник может регистрировать несколько отраженных сигналов, пришедших с различных направлений. Такие сигналы могут заглушать или усиливать друг друга, в результате чего итоговый сигнал сильно искажается. Этот феномен, показанный на илл. 1.23, называется **многолучевым замиранием (multipath fading)**.



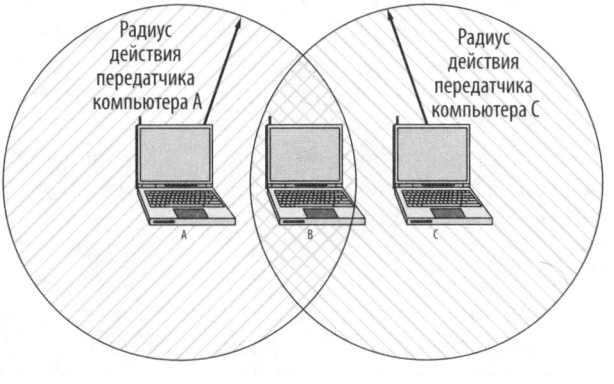

**Илл. 1.23.** Многолучевое замирание

Основной способ преодоления меняющихся условий беспроводной передачи — **разнесение путей (path diversity)**, то есть передача информации по различным независимым путям. В результате информация, скорее всего, попадет к получателю, даже если на одном из путей возникнут проблемы вследствие замирания. Эти независимые пути обычно встраиваются в используемую в аппаратном обеспечении схему цифровой модуляции. Для этого применяются всевозможные варианты: использование разных частот в пределах допустимой полосы, прокладывание различных путей между разными парами антенн и повтор битов через неравные промежутки времени.

Все эти методики использовались в различных версиях 802.11. В первоначальном стандарте (1997) описывалась работающая на скорости 1 или 2 Мбит/с беспроводная LAN, перепрыгивающая с частоты на частоту или распределяющая сигналы по разрешенному для нее диапазону частот. Практически сразу же начали поступать жалобы на слишком медленную скорость, и началась работа над более быстрыми стандартами. Архитектура с «размытием» спектра частот позднее была улучшена и стала стандартом 802.11b (1999), работающим на скорости до 11 Мбит/с. Позднее стандарты 802.11a (1999) и 802.11g (2003) были переведены на другую схему модуляции сигнала — **OFDM (Orthogonal Frequency Division Multiplexing — мультиплексирование с ортогональным частотным разделением каналов)**. При этом подходе широкая полоса частот делится на множество узких полос, через которые параллельно отправляются различные биты. Улучшенная схема (представленная в главе 2) позволила

повысить скорость передачи 802.11a/g до 54 Мбит/с. Это немалый прирост, но пользователям этого было недостаточно. Для обеспечения постоянно растущих нужд необходима бóльшая пропускная способность. Следующие версии стандарта предоставляют еще более высокие скорости передачи данных. Повсеместно развертываемый сейчас стандарт 802.11ac может работать на скорости 3,5 Гбит/с. А более новый 802.11ad способен достигать 7 Гбит/с, правда, только в пределах одной комнаты, поскольку радиоволны на используемых им частотах плохо проходят сквозь стены.

Беспроводные сети имеют широковещательную природу. Поэтому существует возможность конфликта нескольких сигналов, отправленных одновременно, что может помешать их приему. Для решения этой проблемы в 802.11 используется **CSMA (Carrier Sense Multiple Access — множественный доступ с контролем несущей)**. Этот метод основан на идеях классической проводной сети Ethernet, которые были взяты из еще более ранней беспроводной сети **ALOHA**, разработанной на Гавайях. Прежде чем отправлять сигнал, компьютер ожидает в течение короткого случайного интервала времени и откладывает передачу, если обнаруживает, что кто-то уже передает сигнал. Такая схема снижает вероятность того, что два компьютера отправят данные одновременно. Впрочем, она не так эффективна, как в случае проводной сети. Чтобы понять почему, взгляните на илл. 1.24. Допустим, компьютер A отправляет данные компьютеру B, но дальность передатчика компьютера A недостаточна, чтобы достичь компьютера C. Если C хочет передать что-то B, он может «прослушивать» эфир на предмет передачи, но это не гарантирует успеха его собственной передачи. То, что C не «слышит» передаваемый A сигнал, может привести к конфликтам. После любого конфликта отправитель должен подождать в течение случайного (но более длительного) промежутка времени и затем попытаться отправить пакет снова. Несмотря на эту и некоторые другие проблемы, данная схема неплохо работает на практике.




**Илл. 1.24.** Дальность действия одного радиопередатчика может не охватывать всей системы

Еще одна проблема связана с перемещением устройства в пространстве. Когда мобильный клиент удаляется от используемой точки доступа и попадает

в зону приема другой точки доступа, нужен какой-то способ при необходимости подключить его к новой точке. Сеть 802.11 может состоять из нескольких ячеек (каждая — со своей собственной точкой доступа) и системы распределения, соединяющей эти ячейки. Система распределения часто представляет собой коммутируемый Ethernet, но в ее основе может лежать и любая другая технология. Клиенты, перемещаясь, ищут точку доступа с лучшим качеством сигнала, чем у текущей, и если находят, могут переключиться на нее. Извне вся эта система напоминает единую проводную LAN.

Тем не менее для стандарта 802.11 мобильность пока что не имеет столь важного значения, как для мобильных телефонных сетей. 802.11 обычно применяется «кочующими» клиентами, меняющими место постоянного расположения, и не используется во время движения. При подобном стиле использования мобильность не так уж необходима. Даже если 802.11 и используется в движении, то только в пределах одной сети, которая охватывает максимум одно большое здание. Чтобы добиться возможности перемещения между различными сетями и технологиями, нужны новые схемы работы. Например, стандарт 802.21, обеспечивающий возможность передачи обслуживания между проводными и беспроводными сетями.

Наконец, остается проблема безопасности. В силу широковещательного характера беспроводной передачи данных ближайшие компьютеры могут с легкостью получать не предназначенные для них пакеты. Чтобы этого избежать, стандарт 802.11 содержит схему шифрования, известную под названием **WEP (Wired Equivalent Privacy)**. Ее цель — добиться безопасности беспроводной сети на уровне проводной. Идея неплохая, но, к сожалению, схема оказалась несовершенной и вскоре была взломана (Борисов и др.; Borisov et al., 2001). Позднее появились новые схемы шифрования с другими криптографическими особенностями, зафиксированные в стандарте 802.11i. Изначально он носил название **WPA (Wi-Fi Protected Access — защищенный доступ к Wi-Fi)**, сейчас используется версия **WPA2** и еще более хитроумные протоколы, например **802.1X**. Он обеспечивает аутентификацию точек доступа клиентами, а также множество различных способов аутентификации самого клиента точкой доступа.

Стандарт 802.11 произвел революцию в беспроводных сетях, и она продолжается до сих пор. Он широко применяется не только в зданиях, но и в поездах, самолетах, кораблях и автомобилях. Теперь люди могут пользоваться интернетом в дороге, куда и на чем бы они ни ехали. С помощью 802.11 информацией обмениваются и мобильные телефоны, и самые различные виды бытовых электроприборов, начиная от игровых консолей до цифровых видеокамер. К тому же 802.11 постепенно сливается с другими типами мобильных технологий; яркий тому пример — **LTE-Unlicensed (LTE-U — LTE на нелицензируемых частотах)**. Он представляет собой адаптацию сетевой сотовой технологии 4G LTE для работы с нелицензируемым диапазоном частот, в качестве альтернативы принадлежащим ISP точкам доступа Wi-Fi. Мы вернемся ко всем этим мобильным и сотовым сетевым технологиям в главе 4.

# 1.5. СЕТЕВЫЕ ПРОТОКОЛЫ

Мы начнем этот раздел с обсуждения целей разработки сетевых протоколов, затем изучим основную концепцию, применяемую при их создании, — разделение на уровни. Далее мы сравним службы, ориентированные на установление соединения, и службы без установления соединений, а также поговорим о примитивах служб, необходимых для их работы.

## 1.5.1. Цели проектирования

У разных сетевых протоколов зачастую схожие цели проектирования: надежность (способность восстанавливаться после ошибок, сбоев или отказов); распределение ресурсов (совместное использование общего ограниченного ресурса); способность к развитию (поэтапное развертывание усовершенствованных версий протокола с течением времени); безопасность (защита сети от различных типов атак). Рассмотрим в общих чертах все эти цели.

### *Надежность*

Некоторые ключевые задачи проектирования компьютерных сетей повторяются на разных уровнях. Ниже вкратце представлены наиболее важные.

Надежность — это свойство сети, работающей должным образом, даже если составляющие ее компоненты ненадежны. Представьте себе биты передаваемого по сети пакета. Вполне вероятно, что часть этих битов будет получена приемником в искаженном (инвертированном) виде вследствие скачка электричества, случайных беспроводных сигналов, изъянов в аппаратном обеспечении, ошибок программного обеспечения и т. д. Как же найти и исправить эти ошибки?

Один из механизмов поиска ошибок в полученной информации — использование кодов **обнаружения ошибок (error detection)**. Полученная в искаженном виде информация отправляется заново до тех пор, пока не будет принята в правильном виде. Существуют также более мощные коды **исправления ошибок (error correction)**. С их помощью правильное сообщение восстанавливается на основе полученных (возможно, неправильных) битов информации. Оба этих механизма основаны на добавлении избыточной информации в сообщение. Они применяются на низших уровнях (для защиты отправляемых по отдельным каналам пакетов) и на высших уровнях (для проверки корректности полученного содержимого).

Еще одна задача в обеспечении надежности — поиск работоспособного пути передачи данных по сети. Зачастую между источником и приемником существует несколько путей, а в большой сети некоторые каналы или маршрутизаторы могут не работать. К примеру, сеть в Берлине вышла из строя. Пакеты, отправленные из Лондона в Рим через Берлин, не пройдут, но вместо этого можно отправить пакеты из Лондона в Рим через Париж. Сеть должна принимать подобные решения автоматически. Это называется **маршрутизацией (routing)**.

### Распределение ресурсов

Вторая задача проектирования сетей — распределение ресурсов. С увеличением размеров сетей возникают новые проблемы. В крупных городах нередки транспортные пробки, нехватка телефонных номеров, а еще в них можно легко потеряться. Мало кто сталкивается с этими проблемами в родном дворе, но в масштабах всего города они могут оказаться очень серьезными. Архитектуры, сохраняющие должную работоспособность сети при ее росте, называются **масштабируемыми (scalable)**. Сети предоставляют хостам услуги на основе имеющихся ресурсов, например пропускной способности линий передачи. Чтобы один хост не слишком мешал другому, необходимы механизмы разделения ресурсов.

Часто пропускная полоса сети делится между хостами динамически, в соответствии с их потребностями. Это делается вместо того, чтобы выделять каждому хосту фиксированную часть полосы, которую он может и не использовать. Подобная архитектура называется **статистическим мультиплексированием (statistical multiplexing)** — ресурсы распределяются в соответствии со статистикой запросов на них. Она может применяться на низших уровнях для отдельных соединений, на высших уровнях для сети или даже для использующих сеть приложений.

Проблема, возникающая на всех уровнях: как не допустить, чтобы быстрый отправитель перегрузил данными медленного получателя. Для ее решения часто используется обратная связь от получателя отправителю. Это называется **управлением потоком (flow control)**. Иногда в сети возникает проблема **перегрузки (congestion)**, поскольку множество компьютеров отправляет слишком большой объем трафика и сеть не может обеспечить его доставку. Решить проблему можно путем сокращения всеми компьютерами запроса ресурсов (то есть полосы пропускания). Этот способ можно использовать на всех уровнях.

Стоит обратить внимание, что сети обладают и другими ресурсами помимо полосы пропускания. Например, при онлайн-трансляции видео важную роль играет своевременность доставки. Большинству сетей приходится одновременно предоставлять услуги и приложениям, требующим доставки **в реальном времени**, и приложениям, которым нужна высокая пропускная способность. Способ согласования подобных разнонаправленных требований носит название **QoS**.

### Способность к развитию

Еще одна задача проектирования связана с эволюцией сети. С течением времени сеть растет и возникают новые элементы, требующие подключения к ней. В последнее время применяется ключевой механизм поддержки изменений, работающий за счет разбиения общей задачи на составные части и сокрытия нюансов реализации, — **разделение протокола на уровни (protocol layering)**. Разработчикам сетей доступно также множество других стратегий.

Поскольку в сети содержится множество компьютеров, на каждом уровне необходим механизм идентификации отправителей и получателей конкретных сообщений. Этот механизм называется **адресацией (addressing)** и **именованием (naming)** на низшем и высшем уровне соответственно.

При росте сетей возникает проблема различных ограничений у разных сетевых технологий. Например, не все каналы связи сохраняют порядок отправляемых по ним сообщений, поэтому сообщения приходится нумеровать. Еще одна проблема — несовпадение максимального размера сообщений, которые могут быть переданы в разных сетях. Вследствие этого приходится создавать механизмы для разбиения сообщений на части, передачи и последующей их сборки. Это называется **организацией межсетевого взаимодействия (internetworking)**.

### *Безопасность*

Наконец, важная задача — обеспечить безопасность сети путем защиты ее от разнообразных угроз. Одна из таких угроз, перехват данных, уже была упомянута ранее. От этой угрозы защищают механизмы обеспечения **секретности информации (confidentiality)**, используемые на многих уровнях. Также существуют механизмы **аутентификации (authentication)**, гарантирующие, что никто не сможет выдать себя за кого-то другого. Они позволяют отличить поддельные сайты банков от настоящих, а сотовая сеть с их помощью может проверить, что звонок действительно поступил с вашего телефона, чтобы выставить вам счет. Механизмы обеспечения **целостности (integrity)** предотвращают внесение скрытых изменений в сообщения, например, замену сообщения «Списать с моего счета $10» на «Списать с моего счета $1000». Все эти возможности основаны на криптографии, которая будет рассмотрена в главе 8.

## 1.5.2. Разделение протокола на уровни

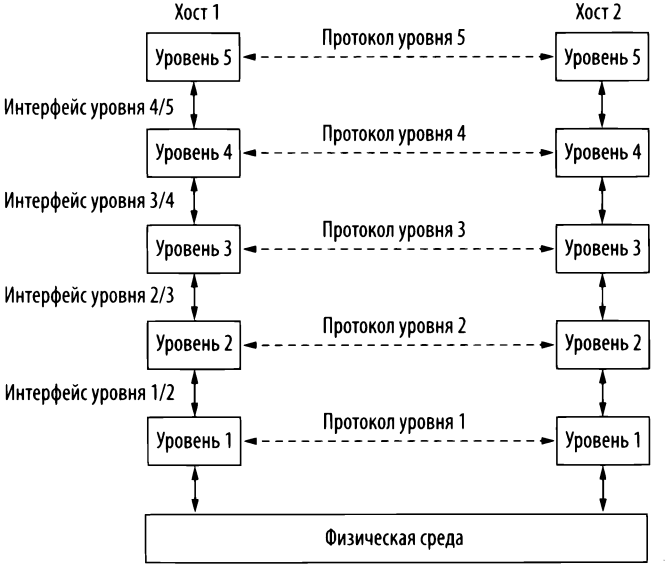
Для упрощения архитектуры большинство сетей организованы в виде пирамиды **уровней (levels)**, или **слоев (layers)**, каждый из которых построен на основе нижележащего. Количество, названия, содержимое и функции каждого уровня различаются в зависимости от сети. Задача каждого уровня состоит в предоставлении определенных служб вышележащим уровням, при одновременном сокрытии от них деталей фактической реализации служб. В определенном смысле каждый уровень представляет собой некую виртуальную машину, предлагающую службы расположенному выше уровню.

Эта концепция не нова и используется повсеместно под разными названиями: сокрытие информации, абстрактные типы данных, инкапсуляция данных и объектно-ориентированное программирование. Основная идея состоит в том, что конкретный элемент программного (или аппаратного) обеспечения предоставляет своим пользователям некую службу, но скрывает от них подробности относительно своего внутреннего состояния и нюансы реализации алгоритмов.

Правила и соглашения, касающиеся обмена информацией между уровнем $n$ одного устройства и уровнем $n$ другого, известны под общим названием «протокол уровня $n$». По сути, **протокол** представляет собой соглашение о том, как должно происходить взаимодействие между участниками обмена данными. Приведем аналогию. Когда женщину представляют мужчине, она может протянуть ему руку. Он же, в свою очередь, может эту руку либо пожать, либо поцеловать, в зависимости от того, является ли она американским юристом на

деловой встрече или европейской принцессой на официальном балу. Нарушение протокола затрудняет общение, а то и вовсе делает его невозможным.

На илл. 1.25 приведена пятиуровневая сеть. Сущности, образующие соответствующие уровни на различных компьютерах, называются **пирами (peers)**. Пиры могут быть программными процессами, аппаратными устройствами и даже людьми. Именно они взаимодействуют друг с другом с помощью протокола.




**Илл. 1.25.** Уровни, производители данных и интерфейсы

На самом деле никакие данные не передаются напрямую с уровня *n* одного устройства на уровень *n* другого. Вместо этого каждый уровень отправляет данные и управляющую информацию на уровень, лежащий непосредственно под ним, пока не будет достигнут самый низший уровень. Под уровнем 1 располагается **физическая среда (physical medium)**, через которую происходит фактический обмен информацией. На илл. 1.25 виртуальный обмен информацией показан пунктирными линиями, а фактический — сплошными.

Каждая пара смежных уровней связана **интерфейсом**. Он определяет, какие базовые операции и службы предоставляет низший уровень высшему. После того как архитекторы сети задали количество и функции уровней сети, одна из важнейших задач — создание понятных интерфейсов между ними. Для этого необходимо, чтобы каждый уровень выполнял конкретный набор четко определенных функций. Помимо минимизации объемов передаваемой между уровнями информации, четко заданные интерфейсы упрощают перевод уровня на совершенно другой протокол или реализацию. Например, можно представить себе замену всех телефонных линий спутниковыми каналами, поскольку единственное, что требуется от нового протокола (или реализации), — предоставление

«соседу» сверху прежнего набора служб. Хосты нередко используют разные реализации одного протокола (зачастую написанные различными компаниями). На самом деле протокол на каком-либо уровне может поменяться совершенно незаметно для уровней над и под ним.

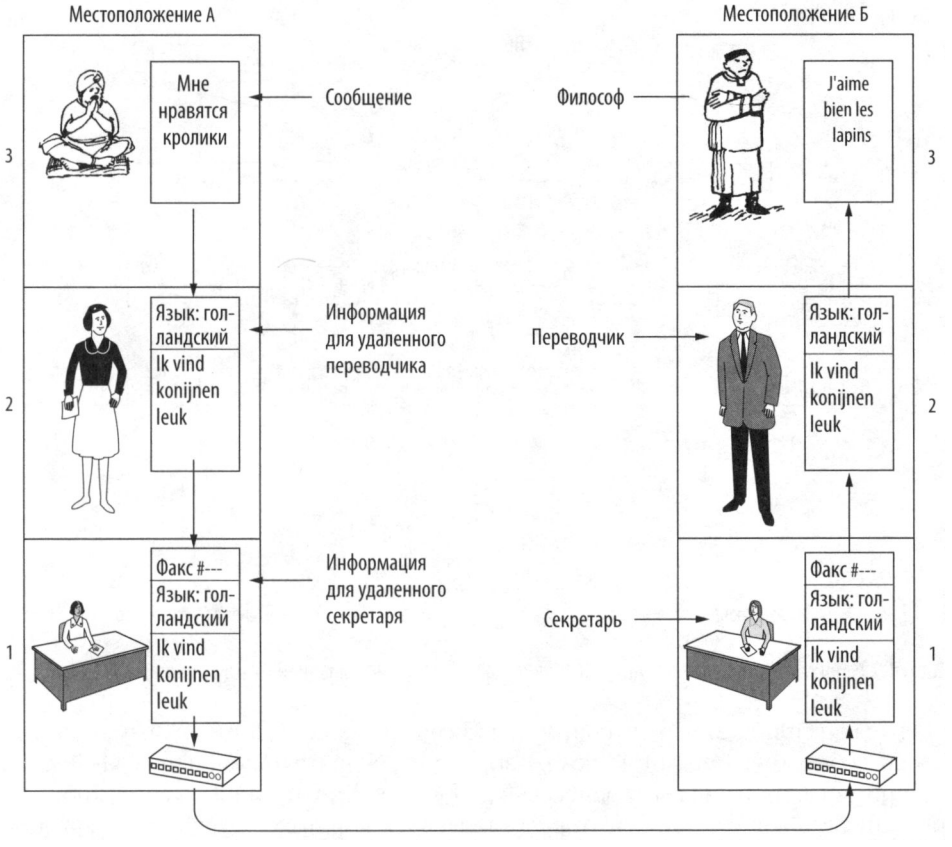
Набор уровней и протоколов называется **архитектурой сети**. Спецификация архитектуры должна включать достаточное количество информации. Тогда тот, кто будет заниматься внедрением, сможет написать программу или создать аппаратное обеспечение для каждого уровня, должным образом следующие соответствующему протоколу. Впрочем, ни нюансы реализации, ни спецификация интерфейсов не являются составными частями архитектуры, поскольку они скрыты внутри устройств и не видны извне. Интерфейсы на всех компьютерах сети даже могут быть разными, главное, чтобы каждый компьютер мог правильно использовать все протоколы. Список используемых конкретной системой протоколов, по одному на уровень, называется **стеком протоколов (protocol stack)**. Архитектуры сетей, стеки протоколов и сами протоколы являются основными темами данной книги.

Можно пояснить идею многоуровневой связи с помощью аналогии. Представьте себе двух философов (одноранговые процессы на уровне 3), один из которых разговаривает на урду и русском, а второй — на китайском и французском. Поскольку общего языка у них нет, они оба нанимают переводчиков (процесс на уровне 2), каждый из которых, в свою очередь, связывается с секретарем (процесс на уровне 1). Философ 1 хочет донести до коллеги свою любовь к *oryctolagus cuniculus*[1]. Для этого он передает своему переводчику сообщение (на русском языке) через интерфейс уровней 2/3: «Мне нравятся кролики» (илл. 1.26). Переводчики договорились насчет известного им обоим нейтрального языка, голландского, так что сообщение преобразуется в «Ik vind konijnen leuk». Выбор языка — протокол уровня 2, определяемый процессами уровня 2.

Далее переводчик отдает сообщение секретарю для дальнейшей передачи, например, по факсу (протокол уровня 1). Когда сообщение доходит до второго секретаря, он передает его второму переводчику, который переводит это сообщение на французский и передает через интерфейс 2/3 второму философу. Обратите внимание, что все протоколы совершенно не зависят от остальных, главное, чтобы интерфейсы не менялись. Переводчики могут в любой момент сменить голландский на, скажем, финский, если они оба согласны на это и ни один из них не меняет свои интерфейсы с уровнями 1 и 3. Аналогично секретари могут начать использовать вместо факса телефон, не затрагивая (и даже не оповещая) другие уровни. Любой процесс может добавлять какую-либо информацию, предназначенную только для одноуровневого с ним процесса. Эта информация не передается более высоким уровням.

Теперь рассмотрим более технический пример: как обеспечить связь для верхнего уровня пятиуровневой сети на илл. 1.27? Один из прикладных процессов, работающий на уровне 5, генерирует сообщение *M* и отдает его на уровень 4 для передачи. На уровне 4 перед сообщением вставляется **заголовок (header)** для его идентификации, а затем результат передается на уровень 3. Заголовок включает

[1] Дикий кролик (*лат.*). — *Примеч. ред.*

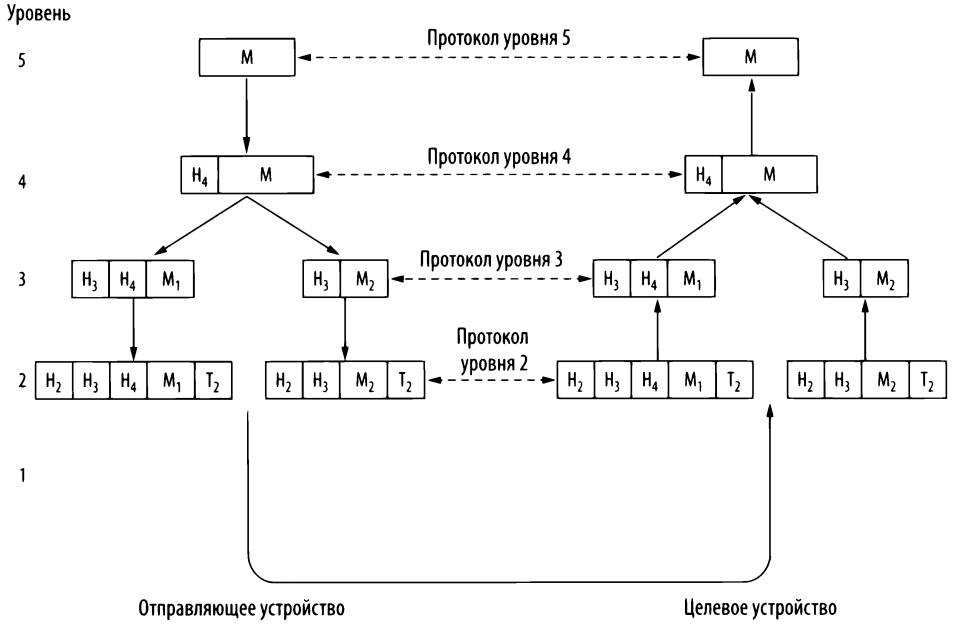



**Илл. 1.26.** Архитектура «философ — переводчик — секретарь»

управляющую информацию (например, адрес), чтобы уровень 4 на целевом устройстве смог доставить сообщение. Среди других примеров управляющей информации, используемой в некоторых уровнях, — последовательные числа (на случай, если низший уровень не сохраняет порядок сообщений), размер и метки времени.

Во многих сетях размер сообщения, передаваемого по протоколу уровня 4, не ограничивается, но протокол уровня 3 налагает ограничения практически всегда. Вследствие этого уровню 3 приходится разбивать входящие сообщения на меньшие куски (пакеты), добавляя перед каждым из них заголовок уровня 3. В нашем примере сообщение $M$ разбивается на две части, $M_1$ и $M_2$, пересылаемые отдельно.

Уровень 3 выбирает используемую исходящую линию связи и передает пакеты уровню 2. Уровень 2 добавляет в каждую часть сообщения не только заголовок, но и концевую метку, после чего передает полученный результат уровню 1 для физической передачи. На принимающем устройстве сообщение движется снизу вверх, с уровня на уровень, с удалением заголовков по мере продвижения. На уровень $n$ не попадает ни один заголовок расположенных ниже $n$ уровней.




**Илл. 1.27.** Пример потока данных при гипотетическом обмене информацией между уровнями 5

Глядя на илл. 1.27, важно понимать взаимосвязь между виртуальным и фактическим обменом данными, а также разницу между протоколами и интерфейсами. Например, одноранговые процессы на уровне 4 концептуально «считают», что обмениваются информацией «горизонтально», по протоколу уровня 4. В каждом из них, вероятно, есть процедуры с названиями наподобие *SendToOtherSide* (*ОтправитьНаДругуюСторону*) и *GetFromOherSide* (*ПолучитьОтДругойСтороны*), хотя на самом деле эти процедуры взаимодействуют с низшими уровнями через интерфейс 3/4, а не с другой стороной.

Абстракция однорангового процесса играет ключевую роль во всей сетевой архитектуре. С ее помощью неподъемную в целом задачу проектирования сети можно разбить на несколько меньших посильных задач проектирования отдельных уровней. Поэтому во всех реальных сетях применяется разделение на уровни или слои.

Стоит отметить, что низшие уровни иерархии протоколов часто реализуются в аппаратном обеспечении или его прошивках. Однако при этом используются сложные алгоритмы протоколов, пусть даже вшитые (полностью или частично) в аппаратное обеспечение.

## 1.5.3. Соединения и надежность

Нижележащие уровни предоставляют расположенным выше уровням два типа служб: с соединениями и без. Степень надежности также может отличаться.

### Службы, ориентированные на установление соединения

**Службы, ориентированные на установление соединения (connection-oriented service)**, строятся по принципу телефонных систем. Чтобы с кем-то пообщаться, необходимо поднять трубку телефона, набрать номер, поговорить, а затем повесить трубку. Аналогично работает и служба: сначала она устанавливает соединение, затем использует и, наконец, освобождает его. Соединение ведет себя подобно трубе: отправитель вставляет объекты (биты) с одного ее конца, а получатель вытаскивает их с другого. В большинстве случаев очередность сохраняется, так что биты приходят в том порядке, в каком были отправлены.

В некоторых случаях при установке соединения отправитель, получатель и подсеть проводят **согласование (negotiation)** используемых параметров, например максимального размера сообщения, требуемого QoS и других вопросов. Обычно при этом одна сторона выдвигает предложение, а вторая может его принять, отвергнуть или внести встречное предложение. Соединение вместе с соответствующими ему ресурсами также называется **каналом** или **линией (circuit)**. Это название пошло от телефонных сетей, где линией (каналом) связи назывался путь по медным проводам, по которым передавался телефонный разговор.

### Службы без установления соединений

В противоположность службам, ориентированным на установление соединения, службы **без установления соединений (connectionless)** строятся по принципу работы обычной почты. Каждое сообщение (письмо) содержит полный адрес назначения и проходит через промежуточные узлы внутри системы независимо от всех последующих сообщений. В различных контекстах такие сообщения называются по-разному; сообщение на сетевом уровне называется **пакетом**. Вариант, при котором промежуточные узлы получают сообщение полностью, прежде чем отправлять его следующему узлу, называется **коммутацией с промежуточным хранением данных (store-and-forward switching)**. Альтернативный вариант, при котором узел начинает передачу сообщения далее вплоть до полного его получения, называется **сквозной коммутацией (cut-through switching)**. Когда два сообщения отправляются в один пункт назначения, то обычно первое отправленное первым и прибывает. Впрочем, оно может задержаться в пути, тогда первым прибудет второе.

Устанавливать соединение нужно не для всех приложений. Например, спамеры отправляют нежелательную электронную почту сразу множеству адресатов. Ненадежные (то есть без подтверждения получения) службы называются **службами отправки дейтаграмм (datagram service)**, по аналогии с отправкой телеграмм, при которой отправитель тоже не получает уведомления о доставке.

### Надежность

Службы, ориентированные на установление соединений и без них, можно охарактеризовать по их параметру надежности. Некоторые службы надежны, поскольку никогда не теряют данные. Надежная служба обычно реализуется

так, что адресат подтверждает получение каждого сообщения и отправитель знает, что оно было доставлено. Процесс подтверждения приводит к накладным расходам и задержкам, которые зачастую оправданны, но иногда цена, которую необходимо заплатить за надежность, оказывается чрезмерной.

Типичный сценарий использования, для которого подходит надежная служба с установлением соединения, — пересылка файлов. Владелец файла хочет быть уверенным, что все биты достигли места назначения без искажений и в том порядке, в каком были отправлены. Очень немногие предпочтут службу, периодически меняющую местами или теряющую некоторые биты, даже если она работает намного быстрее.

Существуют два незначительно отличающихся варианта надежных служб с установлением соединения: последовательность сообщений (message sequences) и байтовые потоки (byte streams). В первом варианте сохраняются границы сообщений. Если было отправлено два сообщения по 1024 байта, они прибудут в место назначения в виде двух сообщений по 1024 байта, а не одного сообщения в 2048 байт. Во втором варианте соединение представляет собой просто поток байтов, без каких-либо границ сообщений. Когда в приемник поступает 2048 байт, нет никакой возможности определить, представляли ли они собой при отправке одно сообщение из 2048 байт, два сообщения по 1024 байта или 2048 сообщений по 1 байту. При отправке по сети страниц книги для фотонабора в виде отдельных сообщений сохранение границ может играть важную роль. С другой стороны, для скачивания фильма байтовый поток с сервера на компьютер пользователя — как раз то, что нужно. Границы сообщений (отдельных кадров) внутри фильма не имеют значения.

В некоторых случаях неудобно создавать соединение для отправки одного-единственного сообщения, но важна надежность. Для подобных сценариев подойдет **служба отправки дейтаграмм с подтверждением получения (acknowledged datagram service)**. Она напоминает отправку заказного письма с уведомлением о вручении. Получив уведомление, отправитель может быть полностью уверен в том, что письмо было доставлено по адресу и не потерялось по дороге. Один из примеров такой службы — доставка текстовых сообщений на мобильных телефонах.

Сама идея использования ненадежных коммуникаций может сначала привести в недоумение. В конце концов, как можно предпочесть ненадежную связь надежной? Прежде всего, надежная связь (в нашем понимании — с подтверждением получения) может оказаться недоступной на конкретном уровне. Например, Ethernet не обеспечивает надежного обмена данными. Пакеты периодически могут повреждаться в пути, а обеспечить их восстановление должны протоколы более высоких уровней. В частности, многие надежные службы строятся поверх ненадежных служб отправки дейтаграмм. Кроме того, присущие надежным службам задержки могут оказаться неприемлемыми, особенно для работающих в режиме реального времени приложений (например, мультимедийных). Поэтому и надежные, и ненадежные виды коммуникаций существуют параллельно.

Для некоторых приложений неприемлемы транзитные задержки, возникающие вследствие подтверждения получения. Одно из этих приложений — цифровой

голосовой трафик (VoIP). Возникающий время от времени небольшой шум на линии не так раздражает пользователей, как зависание разговора в ожидании подтверждений. Аналогично и при видеосвязи: небольшое число неправильно переданных пикселей не представляет собой проблемы, а вот подергивание изображения, когда поток данных останавливается для исправления ошибок, или долгое ожидание поступления идеального видеопотока раздражает пользователей.

Еще один вид служб — **запрос/ответ (request-reply)**. Отправитель передает одну дейтаграмму с запросом и получает дейтаграмму с ответом. С помощью схемы «запрос/ответ» часто реализуется связь в клиент-серверной модели: клиент отправляет запрос, а сервер на него реагирует. Например, клиент с мобильного телефона отправляет запрос картографическому серверу, чтобы получить список ближайших китайских ресторанов, а сервер присылает ему этот список.

На илл. 1.28 приведена краткая сводка представленных выше типов служб.

| | Служба | Пример |
|---|---|---|
| Ориентированные на установление соединения | Надежный поток сообщений | Последовательность страниц |
| | Надежный байтовый поток | Скачивание фильма |
| | Ненадежное соединение | Передача голоса по IP |
| Без установления соединений | Ненадежная дейтаграмма | Нежелательная электронная почта |
| | Дейтаграмма с подтверждением | Обмен текстовыми сообщениями |
| | Запрос/ответ | Запрос базы данных |

**Илл. 1.28.** Шесть различных типов служб

## 1.5.4. Примитивы служб

Службы формально описываются набором **примитивов (primitives)**, то есть операций, доступных обращающимся к ним пользовательским процессам. С помощью этих примитивов можно указать службе выполнить какое-либо действие или сообщить о действии, которое совершил объект на том же уровне. Если стек протоколов располагается в операционной системе (как это обычно и бывает), примитивы представляют собой системные вызовы. Такой вызов приводит к системному прерыванию в привилегированном режиме. Далее управление компьютером передается операционной системе для отправки нужных пакетов.

Набор доступных примитивов зависит от вида предоставляемой службы. Примитивы для ориентированных на соединения служб отличаются от примитивов служб без установления соединений. В качестве минимального набора примитивов, способного выдавать надежный байтовый поток, рассмотрим примитивы на илл. 1.29. Они хорошо знакомы всем приверженцам интерфейса сокетов Беркли, поскольку представляют собой его упрощенный вариант.

| Примитив | Значение |
|---|---|
| LISTEN (прослушивать) | Блокирующее ожидание входящего соединения |
| CONNECT (подключиться) | Установка соединения с находящимся в состоянии ожидания одноранговым процессом |
| ACCEPT (согласиться) | Принятие входящего соединения от однорангового процесса |
| RECEIVE (принять) | Блокирующее ожидание входящего сообщения |
| SEND (отправить) | Отправка сообщения одноранговому процессу |
| DISCONNECT (отключиться) | Завершение соединения |

**Илл. 1.29.** Простая, ориентированная на установление соединения служба с шестью примитивами

Эти примитивы можно использовать для механизма «запрос/ответ» в клиент-серверной среде. Для иллюстрации их работы покажем схему работы простого протокола, реализующего службу получения дейтаграмм с подтверждением.

Прежде всего сервер выполняет примитив LISTEN, чтобы оповестить о готовности к приему входящих соединений. Примитив LISTEN обычно реализуют при помощи блокирующего системного вызова. После выполнения этого примитива серверный процесс блокируется (приостанавливается) до тех пор, пока не появится запрос на соединение.

Далее клиентский процесс выполняет CONNECT, чтобы установить соединение с сервером. В вызове CONNECT должно быть указано, с кем соединяться, поэтому у него может быть параметр для адреса сервера. После этого операционная система отправляет пакет одноуровневому процессу с запросом на соединение, как показано в (1) на илл. 1.30. Клиентский процесс приостанавливается до получения ответа.

**Илл. 1.30.** Простое взаимодействие типа «клиент-сервер» получения дейтаграмм с подтверждением

Когда пакет прибывает на сервер, операционная система видит, что он запрашивает соединение. Она проверяет наличие прослушивающего процесса и разблокирует его, если он есть. Далее серверный процесс может устанавливать

соединение при помощи вызова ACCEPT, в результате которого клиентскому процессу отправляется ответ (2) с согласием на соединение. Поступление этого ответа приводит к снятию блокировки с клиента. И сервер, и клиент в этот момент уже работают, и между ними установлено соединение.

Очевидна аналогия между этим протоколом и звонком покупателя в службу поддержки какой-либо компании. В начале дня менеджер по работе с клиентами сидит возле своего телефона в ожидании звонков. Затем звонит клиент. Когда менеджер поднимает трубку — устанавливается соединение.

Следующий шаг: выполнение сервером операции RECEIVE для подготовки к получению первого запроса. Обычно сервер делает это сразу же после снятия блокировки примитивом LISTEN, прежде чем клиент получит подтверждение. Вызов RECEIVE блокирует сервер.

Далее клиент выполняет примитив SEND для передачи своего запроса (3) с последующим RECEIVE для получения ответа. Поступление пакета с запросом разблокирует сервер, и он может обработать запрос. После завершения необходимых действий сервер отправляет ответ клиенту (4) с помощью примитива SEND. Получение этого пакета разблокирует клиента, который теперь может обработать ответ и отправить дополнительные запросы, если таковые у него есть.

По окончании работы с сервером клиент выполняет DISCONNECT для завершения соединения (5). Обычно первый DISCONNECT представляет собой блокирующий вызов, который приостанавливает клиента, а серверу отправляется пакет с сообщением, что соединение больше не нужно. При получении этого пакета сервер также выполняет свою операцию DISCONNECT, подтверждая клиенту получение, и освобождает соединение (6). При поступлении серверного пакета на компьютер пользователя клиентский процесс освобождается и соединение разрывается. Такова краткая схема работы связи с использованием соединений.

Конечно, на практике не все так просто. Многое может пойти не так. Могут возникать проблемы с очередностью происходящего (например, выполнение CONNECT перед LISTEN), теряться пакеты и многое другое. Эти проблемы будут подробнее рассмотрены позже, а пока что илл. 1.30 вкратце резюмирует взаимодействие «клиент-сервер» для дейтаграмм с подтверждением (мы можем игнорировать потери пакетов).

Учитывая, что для одного цикла этого протокола требуется шесть пакетов, может показаться странным, что взамен него не используется протокол без установления соединения. Конечно, в идеальном мире так бы и было. Понадобились бы только два пакета: один для запроса, а второй для ответа. Впрочем, если учесть возможность передачи сообщений большого размера в обе стороны (например, файла размером в мегабайт), потенциальные ошибки передачи и возможную потерю пакетов, ситуация меняется. Если ответ состоит из сотен пакетов, клиенту необходимо знать, не потерялась ли часть из них при передаче. Как он узнает, что последний полученный пакет был последним отправленным? Или представим, что клиент запросил второй файл. Как он сможет отличить пакет 1 второго файла от потерянного и внезапно найденного пакета 1 из первого файла? Короче говоря, на практике простого протокола запрос/ответ при связи по ненадежной сети обычно недостаточно. В главе 3 мы подробно изучим множество протоколов, позволяющих решить эти и другие проблемы. А пока что отметим

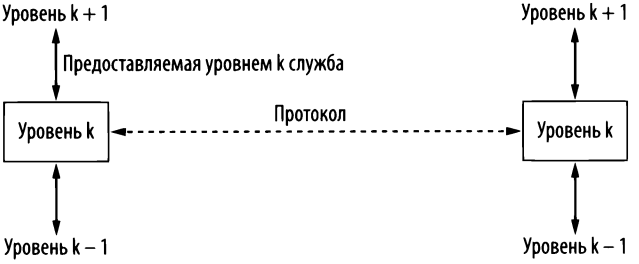
только, что надежный упорядоченный байтовый поток между процессами иногда может очень пригодиться.

## 1.5.5. Службы и протоколы

Службы и протоколы не одно и то же. Различать эти понятия столь важно, что мы еще раз подчеркнем их различия. *Службы* — это набор примитивов (операций), которые нижележащий уровень может делать для вышележащего. Служба описывает операции, которые уровень может выполнять для своих пользователей, но при этом не упоминается о том, как эти операции реализуются. Служба описывает интерфейс между двумя уровнями, один из которых (расположенный ниже) является поставщиком службы, а другой (расположенный непосредственно над первым) — ее потребителем.

*Протокол*, напротив, представляет собой набор правил, определяющих формат и смысл пакетов (сообщений), которыми обмениваются сущности внутри одного уровня. Протоколы используются такими сущностями для реализации описаний служб. Они могут менять протоколы, как им заблагорассудится, главное — не менять видимую пользователям службу. Таким образом, служба и протокол совершенно независимы друг от друга. Это ключевая концепция, которую должен хорошо понимать любой архитектор сетей.

Еще раз повторим: службы имеют непосредственное отношение к интерфейсам между уровнями (как показано на илл. 1.31). Протоколы же имеют непосредственное отношение к пакетам, пересылаемым между одноранговыми сущностями на различных компьютерах. Очень важно не путать эти понятия.




**Илл. 1.31.** Соотношение между службой и протоколом

Уместно будет провести аналогию с языками программирования. Служба подобна абстрактному типу данных или объекту в объектно-ориентированном языке. Она описывает, какие операции можно выполнять над объектом, но не уточняет, как они должны быть реализованы. А протокол относится к *реализации* службы и сам по себе пользователю службы не виден.

Во многих старых протоколах службы и протоколы не различались. В сущности, типичный уровень мог включать примитив службы SEND PACKET[1],

[1] «Отправить пакет». — *Примеч. пер.*

а пользователь передавал ссылку на полностью готовый пакет. Такое соглашение означало, что все изменения протокола сразу же становились видимы пользователям. Большинство разработчиков сетей считают подобную архитектуру серьезной ошибкой.

## 1.6. ЭТАЛОННЫЕ МОДЕЛИ

Многоуровневая архитектура протоколов — одна из ключевых абстракций в сетевой архитектуре. Важнейшей задачей при этом является описание функциональности уровней и взаимодействий между ними. Ниже рассматриваются две основные эталонные модели — TCP/IP и OSI, а также модель, которая представляет собой компромисс между ними (ее мы и будем использовать далее в книге).

### 1.6.1. Эталонная модель OSI

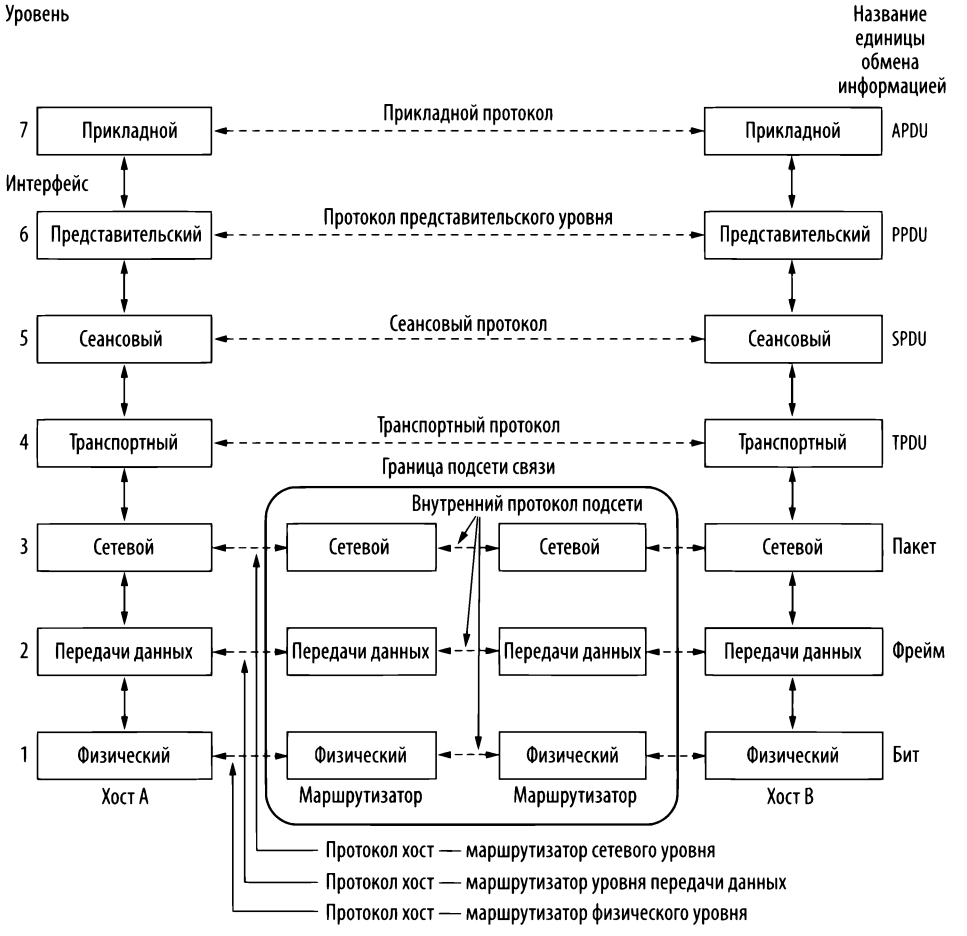
Модель OSI (за исключением физической среды) показана на илл. 1.32. В основе этой модели лежит проект, разработанный **Международной организацией по стандартизации (International Standards Organization, ISO)**. Это был первый шаг к международной стандартизации протоколов, используемых на различных уровнях (Дэй и Циммерман; Day & Zimmermann, 1983). Модель была пересмотрена в 1995 году (Дэй; Day, 1995) и получила название **эталонной модели взаимодействия открытых систем ISO (ISO OSI (Open Systems Interconnection) Reference Model)**. Она описывает вопросы соединения открытых систем, то есть таких, которые открыты для обмена информацией с другими системами. Для краткости мы будем называть ее **моделью OSI**.

Модель OSI включает 7 уровней, выделенных согласно следующим принципам:

1. Каждый уровень соответствуют отдельной абстракции.
2. Все уровни выполняют четко определенные функции.
3. Функция каждого уровня выбирается с учетом создания в дальнейшем международных стандартизированных протоколов.
4. Границы уровней должны выбираться так, чтобы минимизировать поток информации через интерфейсы.
5. Количество уровней не должно быть слишком низким, чтобы не приходилось собирать разные функции в одном уровне; но нельзя, чтобы их было чересчур много, иначе архитектура будет громоздкой.

Центральными для модели OSI являются три понятия:

1. Службы.
2. Интерфейсы.
3. Протоколы.




**Илл. 1.32.** Эталонная модель OSI

Вероятно, главная ценность модели OSI — четкое разграничение этих понятий. Каждый уровень предоставляет вышележащему уровню какие-либо службы. Определение службы говорит лишь о том, что делает данный уровень, но не о том, как вышележащие уровни взаимодействуют с ним или как он работает.

В модели TCP/IP изначально отсутствовало четкое разграничение служб, интерфейсов и протоколов; хотя позже были попытки сделать ее более похожей на модель OSI.

## 1.6.2. Эталонная модель TCP/IP

Эталонная модель TCP/IP использовалась в прародителе всех глобальных вычислительных сетей — ARPANET и его наследнике — всемирной сети интернет. Как мы рассказывали выше, ARPANET была исследовательской сетью,

которую финансировало Минобороны США. В конечном счете она объединила сотни университетов и правительственных объектов при помощи выделенных телефонных линий. С появлением радио- и спутниковых сетей оказалось, что существующие протоколы не подходили для межсетевого взаимодействия, так что появилась необходимость в новой эталонной архитектуре. Таким образом, практически с самого начала одной из основных целей ее разработки было обеспечение бесперебойного соединения множества сетей. Эта архитектура позднее стала известна под названием **эталонной модели TCP/IP (TCP/IP Reference Model**; TCP и IP — два ее основных протокола). Первыми ее описали Серф и Кан (Cerf & Kahn, 1974), а позднее она была пересмотрена и зафиксирована в виде интернет-стандарта (Брейден; Braden, 1989). Основные принципы проектирования этой модели обсуждаются в работе Кларка (Clark, 1988).

Вследствие опасений Пентагона потерять в один миг часть своих драгоценных хостов, маршрутизаторов и межсетевых шлюзов в результате атаки Советского Союза, была поставлена еще одна важная цель. Сеть должна продолжать нормальную работу, без разрыва текущих разговоров, даже в случае потери аппаратного обеспечения подсети. Другими словами, Пентагон хотел, чтобы соединения поддерживались до тех пор, пока работают отправляющее и целевое устройства, даже если некоторые узлы или линии передачи между ними внезапно вышли из строя. Более того, архитектура должна была быть гибкой, поскольку предполагалось использование приложений с нестандартными требованиями, от передачи файлов до передачи голоса в режиме реального времени.

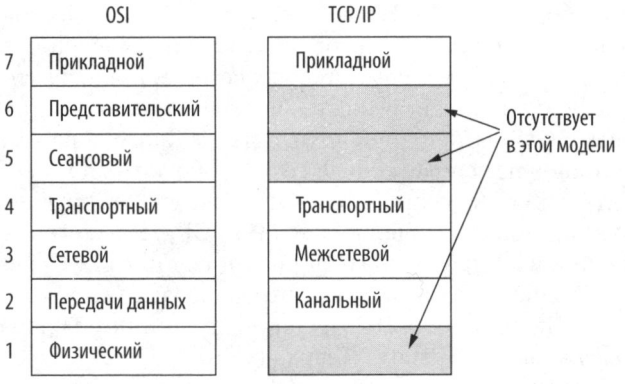
### *Канальный уровень*

С учетом этих требований была выбрана сеть с коммутацией пакетов, основанная на уровне без соединений, работающем в различных сетях. Низший уровень модели, **канальный уровень (link layer)**, описывает, какими должны быть каналы связи (например, последовательные линии связи и традиционный Ethernet) для удовлетворения потребностей межсетевого уровня без соединений. На самом деле это даже не уровень в обычном смысле этого слова, а скорее интерфейс между хостами и линиями передачи. В первых описаниях модели TCP/IP он игнорировался.

### *Межсетевой уровень*

**Межсетевой уровень (internet layer)** — краеугольный камень всей архитектуры. Он показан на илл. 1.33. Его задача — сделать так, чтобы хосты могли внедрять пакеты в произвольную сеть для последующего движения к пункту назначения (возможно, расположенного в другой сети) независимо друг от друга. Эти пакеты даже могут прибывать совершенно не в том порядке, в каком они были отправлены. Если необходимо соблюдение очередности, высшие уровни должны упорядочить пакеты.

Здесь можно провести аналогию с обычной почтой. Бросаем пачку международных писем в почтовый ящик в одной стране, и, если повезет, большинство из них попадут по нужным адресам в стране назначения. Скорее всего, эти письма




**Илл. 1.33.** Эталонная модель TCP/IP

по дороге пройдут через один или несколько международных сортировочных центров, но пользователи об этом не узнают. Более того, от пользователей скрыт тот факт, что в каждой стране (то есть сети) свои почтовые марки, размеры конвертов и правила доставки.

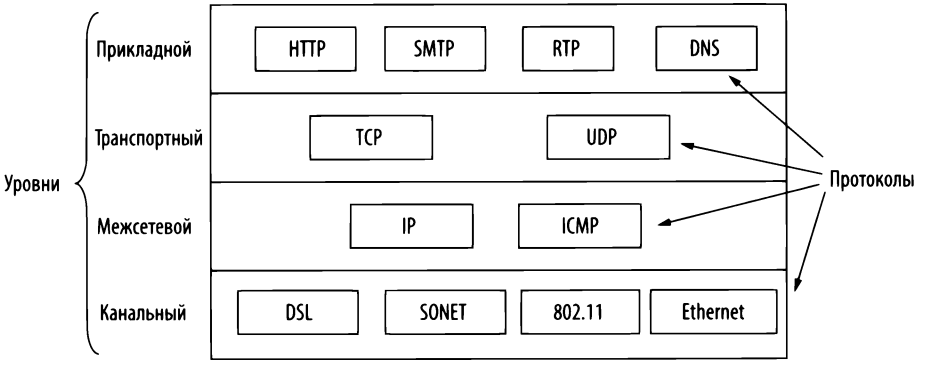
Межсетевой уровень определяет официальный формат пакетов, **IP (Internet Protocol — межсетевой протокол)**, а также вспомогательный протокол **ICMP (Internet Control Message Protocol — протокол управления межсетевым обменом сообщениями)**. Задача межсетевого уровня заключается в доставке IP-пакетов по месту назначения. Главная проблема состоит в маршрутизации пакетов, а также в контроле перегруженности сети. Задача маршрутизации в основном уже решена, но справиться с перегруженностью нельзя без помощи вышележащих уровней.

### *Транспортный уровень*

Непосредственно над межсетевым уровнем в модели TCP/IP располагается уровень, обычно называемый **транспортным (transport layer)**. Он был создан для обеспечения связи объектов одного ранга, расположенных на разных хостах, аналогично транспортному уровню OSI. В нем определено два сквозных транспортных протокола. Первый, **TCP (Transmission Control Protocol — протокол управления передачей данных)**, — надежный, ориентированный на установление соединения протокол, с помощью которого можно доставить без ошибок байтовый поток с одного компьютера на любой другой. Он разбивает входящий поток на отдельные сообщения и передает их по одному в межсетевой уровень. В пункте назначения получающий TCP-процесс собирает полученные сообщения в выходной поток. TCP также управляет движением данных, чтобы быстрый отправитель не перегрузил медленного получателя чрезмерным числом сообщений.

Второй протокол этого уровня, **UDP (User Datagram Protocol — протокол пользовательских дейтаграмм)**, представляет собой ненадежный протокол

без соединений. Он предназначен для приложений, которым не требуется TCP для разбиения на сообщения и управления потоками; они выполняют данную задачу сами (или обходятся без этого). **UDP** также широко применяется для однократных клиент-серверных запросов и приложений, работающих по принципу «запрос/ответ». Для таких приложений важнее быстрота доставки, чем ее точность, например, в случае передачи голоса или видео. Взаимосвязи IP, TCP и UDP приведены на илл. 1.34. IP был реализован во множестве сетей с момента разработки модели TCP/IP.




**Илл. 1.34.** Модели TCP/IP и некоторые протоколы, которые мы изучим позднее

### *Прикладной уровень*

В модели TCP/IP нет сеансового и представительского[1] уровней — в них нет необходимости. Вместо этого приложения просто включают все необходимые для них функции, связанные с сеансами или представлением. Опыт доказал правильность такого подхода: эти уровни бесполезны для большинства приложений; в результате они практически исчезли.

Над транспортным уровнем располагается **прикладной уровень (application layer)**, он же **уровень приложений**. Он включает все высокоуровневые протоколы. Самые первые протоколы подобного рода — протокол виртуального терминала TELNET (Teletype network), протокол передачи файлов FTP (File transfer protocol) и протокол электронной почты SMTP (Simple mail transfer protocol). В дальнейшем к ним добавилось множество других. Наиболее важные протоколы (приведенные на илл. 1.34) мы рассмотрим далее. В их числе система доменных имен, DNS (Domain Name System), предназначенная для установления соответствий между именами хостов и их сетевыми адресами; HTTP (Hypertext transfer protocol), протокол для получения страниц из Всемирной паутины, и RTP (Real-time transport protocol), протокол доставки мультимедиа (например, голоса или фильмов) в режиме реального времени.

[1] Или уровня представления. — *Примеч. ред.*

## 1.6.3. Критика модели и протоколов OSI

Ни модель OSI, ни модель TCP/IP с их протоколами не являются идеальными. Обе они нередко подвергались критике. В этом и следующем разделах рассмотрены некоторые из этих критических замечаний. Мы начнем с OSI, а затем поговорим о TCP/IP.

На момент выхода в печать второго издания данной книги (1989 год) многим экспертам в сфере сетевых технологий представлялось, что будущее — за моделью OSI и ее протоколами, а все остальное обречено на забвение. Но все обернулось иначе. Почему? Имеет смысл проанализировать причины, по которым OSI не стала успешной. Их можно резюмировать так: неподходящий момент времени, плохая архитектура, неудачная реализация и неудачная политика.

### *Неподходящий момент времени*

Рассмотрим первую причину: неподходящий момент времени. Для успеха стандарта время его принятия играет критическую роль. Дэвид Кларк (David Clark) из MIT разработал теорию стандартов, которую он назвал *«апокалипсис двух слонов»* (илл. 1.35).

**Илл. 1.35.** «Апокалипсис двух слонов»

На этом графике отражена активность, сопутствующая любой новой разработке. Возникновение новой темы приводит к колоссальному взрыву исследовательской деятельности в виде научных работ, обсуждений, статей и конференций. Через какое-то время ажиотаж понемногу угасает, однако разработку замечают крупные корпорации, и в нее вливаются миллиардные инвестиции.

Стандарты должны быть написаны в «углублении» между двумя «слонами». Если они создаются слишком рано (до появления общепризнанных результатов исследований), то вопрос, скорее всего, еще недостаточно изучен; в результате получается плохой стандарт. Если же слишком поздно, то к тому моменту множество компаний уже инвестируют огромные средства в свои варианты, так что

стандарты будут просто проигнорированы. Если промежуток между слонами слишком мал (поскольку все торопятся начать эксплуатацию), то они могут «раздавить» создателей стандартов.

Похоже, что стандартные протоколы OSI действительно оказались «раздавлены». К моменту появления протоколов OSI конкурирующие с ними протоколы TCP/IP уже широко использовались университетами. И хотя миллиардная волна инвестиций еще не пришла, академический рынок оказался достаточно обширным для того, чтобы многие компании начали осторожно предлагать продукты на TCP/IP. Когда же появилась модель OSI, никто не хотел поддерживать второй стек протоколов (разве что принудительно), так что на рынке не возникло никаких первоначальных предложений. Все компании ждали, пока кто-нибудь сделает первый шаг в этом направлении, но этого так и не произошло, и поддержки OSI не появилось.

### Плохая архитектура

Вторая причина того, что OSI так и не завоевала популярность: и модель, и ее протоколы несовершенны по своей сути. Выбор семи уровней носил скорее политический, а не технический характер. Два уровня (сеансовый и уровень представления) были практически пусты, в то время как два других (сетевой и уровень передачи данных) были переполнены.

Модель OSI, вместе с соответствующими определениями служб и протоколами, чрезвычайно сложна. В распечатанном виде стандарты представляют собой стопку бумаги высотой почти в метр. Плюс еще сложность реализации и неэффективность эксплуатации. В этой связи вспоминается загадка, придуманная Полом Мокапетрисом (Paul Mockapetris), цитируемая в работе Роуза (Rose, 1993):

*Вопрос:* Что получится, если скрестить гангстера с международным стандартом?

*Ответ:* Человек, делающий вам предложение, которое вы не способны понять.

Еще одна проблема OSI, помимо сложности, состоит в том, что некоторые функции (например, адресация, управление потоком и обработка ошибок) встречаются на всех уровнях модели. Зальцер и др. (Saltzer et al., 1984) отмечают, что эффективная обработка ошибок должна производиться на самом верхнем уровне, так что повторять ее на каждом из нижних уровней не нужно, к тому же это бесполезно.

### Неудачная реализация

Учитывая запредельную сложность модели и ее протоколов, ничего удивительного, что первые реализации были громоздкими и медленными. Все, кто пробовал с ними работать, потерпели неудачу. Очень скоро понятие «OSI» стало ассоциироваться у всех с плохим качеством. И хотя реализация со временем улучшилась, образ закрепился. Стоит людям решить, что продукт неудачный, его песенка спета.

И напротив, одна из первых реализаций TCP/IP в составе Berkeley UNIX была довольно неплохой (и к тому же бесплатной). Она быстро вошла в употребление, и в результате образовалось большое пользовательское сообщество. Это влекло за собой совершенствование продукта, что, в свою очередь, приводило к дальнейшему расширению сообщества. В случае с OSI все было иначе.

### *Неудачная политика*

Благодаря первой реализации многие люди, особенно в академической среде, считали TCP/IP частью Unix, а к Unix в 1980-х в университетах испытывали нечто среднее между родительскими чувствами и любовью к яблочному пирогу.

OSI, напротив, многими считался детищем европейских министерств телекоммуникаций, Европейского сообщества и позднее правительства США. Эти убеждения имели под собой основания лишь частично. Однако сама идея навязывания правительственными бюрократами второсортного стандарта несчастным исследователям и программистам, в поте лица разрабатывающим сети, не слишком помогала продвижению OSI. Некоторые даже провели аналогию с ситуацией вокруг языка программирования ПЛ/1. В 1960-х IBM назвала ПЛ/1 языком будущего, однако позднее Пентагон объявил, что им станет Ada.

## 1.6.4. Критика модели и протоколов TCP/IP

У модели и протоколов TCP/IP тоже есть свои проблемы. Во-первых, в этой модели нет достаточно явного разграничения понятий служб, интерфейсов и протоколов. Рекомендуемые практики разработки ПО требуют четко различать спецификацию и реализацию, что модель OSI делает очень тщательно, а TCP/IP — нет. Вследствие этого модель TCP/IP не может применяться в качестве основы для проектирования сетей с применением новых технологий.

Во-вторых, модель TCP/IP не слишком универсальна и не подходит для описания любых других стеков протоколов, кроме TCP/IP. Например, описать Bluetooth с помощью модели TCP/IP невозможно.

В-третьих, канальный уровень вообще не является уровнем в обычном понимании этого термина в контексте многоуровневых протоколов. Это интерфейс (между сетевым и уровнем передачи данных). Различие между интерфейсом и уровнем критически важно, небрежности тут недопустимы.

В-четвертых, в TCP/IP не различаются физический уровень и уровень передачи данных. Между тем они представляют собой совершенно разные вещи. Физический уровень связан с характеристиками передачи информации по медному кабелю, оптоволокну или беспроводными средствами. Задача уровня передачи данных состоит в определении начала и конца фреймов (или кадров) и отправка их с одной стороны на другую с надлежащей степенью надежности. Корректная модель должна содержать оба упомянутых уровня по отдельности. В случае с TCP/IP это не так.

И последнее. IP и TCP были тщательно продуманы и хорошо реализованы. При этом многие ранние протоколы зачастую были сделаны на скорую руку парой аспирантов, работавших до изнеможения. Реализации протоколов тогда

распространялись бесплатно — это привело к их повсеместному использованию и глубокому внедрению в повседневную практику. В результате они плохо поддаются замене. Сегодня некоторые из них выглядят как сущее недоразумение. Например, протокол виртуального терминала, TELNET, был разработан для механического терминала Teletype, рассчитанного на 10 символов в секунду. Ему неведомы графические интерфейсы и мышь. Тем не менее 53 года спустя его все еще используют.

## 1.6.5. Модель, используемая в этой книге

Как уже упоминалось ранее, сильная сторона эталонной модели OSI — сама *модель* (за вычетом сеансового уровня и уровня представления), оказавшаяся исключительно удобной для описания сетей. И напротив, сильная сторона эталонной модели TCP/IP — *протоколы*, повсеместно используемые на протяжении многих лет. В данной книге мы будем применять гибридную модель, чтобы совместить эти преимущества (илл. 1.36).

**Илл. 1.36.** Эталонная модель, используемая в этой книге

Эта модель состоит из пяти уровней: физического, канального, сетевого, транспортного и, наконец, прикладного. Физический уровень определяет способ передачи битов по различным средам в виде электрических (или прочих аналоговых) сигналов. Канальный уровень имеет дело с пересылкой сообщений конечной длины между непосредственно соединенными устройствами с заданной степенью надежности. Примеры протоколов канального уровня — Ethernet и 802.11.

Сетевой уровень занимается объединением каналов связи в сети и интерсети для пересылки пакетов между удаленными компьютерами. Сюда входит поиск пути пересылки пакетов. Основной пример протокола этого уровня, который мы изучим далее, — IP. Транспортный уровень повышает предоставляемые сетевым уровнем гарантии доставки. Чаще всего это выражается в увеличении надежности и предоставлении абстракций доставки (например, надежного байтового потока), подходящих для нужд различных приложений. Важный пример протокола транспортного уровня — TCP.

Наконец, на прикладном уровне располагаются программы, подключающиеся к сети. У большинства сетевых приложений есть пользовательский интерфейс (например, у веб-браузеров). Впрочем, нас больше интересует та часть программы, которая непосредственно использует сеть. В случае веб-браузера это HTTP.

На прикладном уровне работают также важные вспомогательные программы (например, DNS), используемые многими приложениями. Все это обеспечивает функционирование сети.

Последовательность глав нашей книги основана на этой модели. Таким образом, мы не упускаем из виду значение модели OSI для понимания архитектур сетей, но в основном сосредоточиваемся на протоколах, играющих важную практическую роль, начиная с TCP/IP и заканчивая более новыми – 802.11, SONET и Bluetooth.

## 1.7. СТАНДАРТИЗАЦИЯ

Внедрение инноваций в интернет-технологиях зависит как от самих технологий, так и от политических и юридических аспектов. Развитие интернет-протоколов обычно происходит посредством процесса стандартизации, о котором пойдет речь ниже.

### 1.7.1. Стандартизация и открытый исходный код

Существует множество разработчиков и поставщиков сетевых технологий, и каждый имеет свое представление о том, что и как делать. Без согласования наступил бы полный хаос и пользователи не смогли бы работать. Единственный способ решения этой проблемы – прийти к соглашению относительно сетевых стандартов. Хорошие стандарты не только позволяют различным компьютерам обмениваться информацией, но и расширяют рынок для соответствующих им продуктов. Наращивание рынка ведет к массовому производству, масштабной экономии при разработке, улучшению реализаций и другим преимуществам, снижающим цену и повышающим популярность продукта.

В этом разделе мы вкратце обсудим важный, но мало кому известный мир международной стандартизации. Но сначала поговорим о том, что содержит стандарт. Любой здравомыслящий человек предположил бы, что стандарт описывает, как должен работать протокол, чтобы можно было осуществить хорошую реализацию. И ошибся бы.

Стандарты описывают, что требуется для совместимости с другими продуктами – ни больше ни меньше. Это способствует расширению рынка и росту конкуренции между компаниями на основе качества программных продуктов. Например, стандарт 802.11 описывает несколько скоростей передачи данных. При этом он не указывает, при каких обстоятельствах отправитель должен использовать конкретную скорость, что является ключевым фактором для высокой производительности. Это решение отдается на откуп создателю продукта. Зачастую обеспечить совместимость при таком подходе непросто, ведь существуют разные стандарты, в которых описано множество способов реализации. Например, в случае 802.11 существует столько проблем, что отраслевая группа **Wi-Fi Alliance** приступила к работе над совместимостью внутри стандарта 802.11. Что касается программно-определяемых сетей, **ONF (Open Networking Foundation)** пытается разработать как стандарты, так и их реализации с открытым исходным

кодом, чтобы гарантировать совместимость протоколов управления с программируемыми сетевыми коммутаторами.

Стандарт задает протокол передачи данных, но не внутренний интерфейс службы (разве что в качестве пояснений к протоколу). Существующие интерфейсы служб обычно защищены патентами. Например, для обмена информацией с удаленным хостом интерфейс между TCP и IP не имеет значения. Важно только то, что удаленный хост понимает TCP/IP. На самом деле TCP и IP часто реализуются вместе, без какого-либо четкого интерфейса между ними. Тем не менее хорошие интерфейсы служб, как и продуманные **API (Application Programming Interfaces — программные интерфейсы приложений)**, важны для использования протоколов, а лучшие из них (например, сокеты Беркли) приобретают большую популярность.

Стандарты делятся на две категории: де-факто и де-юре. Стандарты **де-факто** (от лат. «фактически») просто возникают сами собой, без какого-либо предварительного плана. HTTP, составляющий основу интернета, появился в качестве стандарта де-факто. Он был включен в первые веб-браузеры, разработанные Тимом Бернерсом-Ли (Tim Berners-Lee) в ЦЕРН, и широко распространился с ростом интернета. Еще один пример — Bluetooth, разработанный компанией Ericsson и теперь используемый повсеместно.

Внедрение стандартов **де-юре** (от лат. «юридически», «согласно праву»), напротив, происходит на основе правил, описанных неким официальным комитетом стандартизации. Органы международной стандартизации делятся на два класса. Они либо создаются на основе межправительственного договора, либо входят в общественные организации, не связанные государственными соглашениями. В сфере стандартов сетей существует несколько организаций каждого типа, в частности ITU, ISO, IETF и IEEE, которые мы обсудим ниже.

На практике взаимосвязи между стандартами, компаниями и комитетами по стандартизации сильно запутаны. Стандарты де-факто часто превращаются в стандарты де-юре, особенно если они успешны. Так было в случае с HTTP, который был быстро подхвачен IETF. Комитеты по стандартизации часто ратифицируют стандарты друг друга, чтобы увеличить долю рынка для конкретной технологии. Сегодня важную роль в разработке и уточнении сетевых стандартов играют многочисленные коммерческие союзы, сформировавшиеся вокруг конкретных технологий. Например, консорциум **3GPP (Third Generation Partnership Project)** начался с сотрудничества ассоциаций по телекоммуникации, занимавшихся внедрением стандартов подвижной телефонной связи UMTS 3G.

## 1.7.2. Кто есть кто в мире телекоммуникаций

Правовой статус мировых телефонных компаний существенно различается в разных странах. На одном конце спектра располагаются США, где существует множество (в основном очень мелких) частных телефонных компаний. Еще несколько фирм добавилось с распадом AT&T в 1984 году (на тот момент крупнейшей корпорации в мире, предоставляющей услуги телефонной связи примерно 80 % абонентов США) и с принятием Акта о телекоммуникациях от

1996 года, который усовершенствовал законодательство в этой сфере с целью стимулирования конкуренции. Впрочем, эта идея не сработала так, как планировалось. Крупные компании выкупали более мелкие до тех пор, пока в большинстве регионов не остались одна-две корпорации.

На другом конце спектра располагаются страны с полной правительственной монополией на все средства связи, включая почту, телеграф, телефон, а часто и радио с телевидением. К этой категории относится большая часть мира. В некоторых случаях регулятор телекоммуникаций представляет собой национализированную компанию, в других — просто часть правительственных структур, обычно называемую **Управлением почтово-телеграфной и телефонной связи (Post, Telegraph & Telephone administration, PTT)**. Во всем мире наблюдается тенденция к переходу от правительственной монополии к либерализации и свободной конкуренции. PTT в большинстве европейских стран уже частично приватизированы, в других странах этот процесс только начинает набирать обороты.

При таком разнообразии поставщиков услуг необходимо обеспечить совместимость в мировом масштабе, чтобы люди (и компьютеры) из разных стран могли связываться друг с другом. На самом деле такая потребность существует уже длительное время. В 1865 году представители европейских стран встретились, чтобы создать союз, ставший предшественником нынешнего **Международного союза электросвязи**[1], **МСЭ (International Telecommunication Union, ITU)**. Его задачей стала стандартизация международных телекоммуникаций (в те годы это был телеграф).

Уже тогда было ясно, что если половина стран будет использовать азбуку Морзе, а вторая половина — любой другой код, возникнет проблема. Когда же в международных масштабах начали использовать телефон, МСЭ взял на себя и стандартизацию телефонии. В 1947 году МСЭ стал одним из агентств ООН.

В МСЭ входят почти 200 государств[2], включая практически всех членов ООН. А поскольку в США нет PTT, представлять США в МСЭ должна какая-то другая организация. Эта задача была возложена на Государственный департамент (вероятно, потому, что МСЭ имеет отношение к иностранным государствам, а это как раз специализация Госдепа). МСЭ насчитывает также более 700 представителей секторов и ассоциированных членов. В их числе телефонные компании (например, AT&T, Vodafone, Sprint), производители телекоммуникационного оборудования (например, Cisco, Nokia, Nortel), поставщики компьютеров (например, Microsoft, Dell, Toshiba), производители микросхем (например, Intel, Motorola, TI) и другие заинтересованные компании (например, Boeing, CBS, VeriSign).

МСЭ включает три основных сектора. Мы сосредоточим внимание в основном на **МСЭ-T, Секторе стандартизации электросвязи**, который занимается телефонными системами и системами передачи данных. До 1993 года этот сектор

---

[1] Здесь и далее мы будем иногда использовать термины «телекоммуникации» и «электросвязь» как синонимы, хотя, строго говоря, первое понятие иногда считается несколько более широким, включая в себя неэлектрические виды связи, например обычную почту. — *Примеч. пер.*

[2] Точнее, 193 (на июнь 2022 года). — *Примеч. пер.*

назывался **CCITT**, по акрониму его французского названия Comité Consultatif International Téléphonique et Télégraphique. **МСЭ-R, Сектор радиосвязи**, занимается глобальной координацией использования радиочастот заинтересованными группами. И последний сектор — **МСЭ-D, Сектор развития**. Он занимается созданием благоприятных условий для развития информационных технологий и технологий связи. Его цель — сокращение «цифровых барьеров» между странами с эффективным доступом к информационным технологиям и странами с ограниченным доступом к ним.

Задача МСЭ-Т состоит в выпуске технических рекомендаций относительно интерфейсов для телефонной и телеграфной связи, а также обмена данными. Эти рекомендации нередко становятся стандартами, признанными на международном уровне. Формально они являются лишь предложениями, которые правительства могут принять или проигнорировать (поскольку правительства напоминают тринадцатилетних мальчишек, которым не нравится, когда им указывают, что делать). На практике любая страна может принять телефонный стандарт, который отличается от мирового. Но тем самым она отрежет себя от всего мира, ведь ее граждане не смогут совершать международные звонки. Возможно, подобный вариант подойдет Северной Корее, но для остальных стран это станет серьезной проблемой.

Вся настоящая работа МСЭ-Т выполняется в его исследовательских комиссиях (Study Group, SG). В настоящее время в МСЭ-Т есть 11 исследовательских комиссий, насчитывающих порядка 400 человек. Они изучают широкий спектр вопросов, от тарификации телефонов и мультимедийных сервисов до безопасности. SG 15, например, занимается стандартизацией оптоволоконных домашних подключений. Благодаря этому производители могут создать продукты, которые будут работать повсеместно. Чтобы добиться хоть каких-то результатов, исследовательские комиссии подразделяются на рабочие группы (Working Parties), состоящие из групп экспертов (Expert Teams), которые, в свою очередь, делятся на группы по решению узких задач. Бюрократия — всегда бюрократия.

Несмотря на все это, МСЭ-Т действительно решает поставленные задачи. С начала своей деятельности он выработал более 3000 рекомендаций, многие из которых часто используются на практике. Например, рекомендация H.264 (она же стандарт ISO, известный под названием MPEG-4 AVC) широко применяется для сжатия видеоданных, а сертификаты открытых ключей X.509 — для безопасного пользования интернетом и цифровых подписей сообщений электронной почты.

Важность стандартов растет по мере завершения начатого в 1980-х перехода отрасли телекоммуникаций от чисто внутригосударственной к глобальной, и все больше организаций будут стремиться участвовать в их создании. Больше информации о МСЭ вы можете найти в работе Ирмера (Irmer, 1994).

## 1.7.3. Кто есть кто в мире международных стандартов

Международные стандарты разрабатываются и публикуются **Международной организацией по стандартизации (International Standards Organization, ISO)**. Это общественная неправительственная организация, основанная в 1946 году.

В нее входят национальные организации по стандартизации 161 страны. В их числе ANSI (США), BSI (Великобритания), AFNOR (Франция), DIN (Германия) и 157 других.

ISO публикует стандарты, затрагивающие огромный спектр вопросов — от болтов и гаек до покраски телефонных столбов (не говоря уже о какао-бобах (ISO 2451), рыболовных сетях (ISO 1530), женском нижнем белье (ISO 4416) и множестве других вещей, казалось бы, не требующих стандартизации). Что же касается вопросов телекоммуникаций, ISO и МСЭ-Т часто сотрудничают (ISO состоит в МСЭ-Т) во избежание конфликта двух официальных и взаимно несовместимых международных стандартов.

За все время было выпущено 24 316 стандартов (по состоянию на июнь 2022 года. — *Примеч. ред.*), включая стандарты OSI. ISO включает более 200 технических комитетов (Technical Committees, TC), пронумерованных в порядке их создания, каждый из которых занимается какой-либо конкретной темой. TC1, например, действительно занимается болтами и гайками (стандартизация винтовой резьбы). JTC1 работает с информационными технологиями, включая сети, компьютеры и программное обеспечение. Это первый (и пока что единственный) объединенный технический комитет (Joint Technical Committee), созданный в 1987 году путем слияния TC97 и одного из технических комитетов IEC (еще одной организации по стандартизации). Каждый TC включает множество подкомитетов (SC), разбитых на рабочие группы (WG).

Основная работа выполняется, как правило, в рабочих группах, куда входит более сотни тысяч добровольцев со всего мира. Правда, многие из этих «добровольцев» работают в ISO по указанию их работодателей, чьи продукты стандартизируются. В числе прочих — госслужащие, заинтересованные в том, чтобы стандарт, принятый в их стране, стал международным. Во многих рабочих группах также принимают активное участие научные специалисты.

Процедура принятия стандартов ISO была разработана с расчетом на максимально широкий консенсус. Процесс запускает одна из национальных организаций по стандартизации, заявляющая о потребности в международном стандарте в какой-либо сфере. Затем формируется рабочая группа для создания **проекта комитета (Committee Draft, CD)**. В течение 6 месяцев члены ISO должны внести в этот проект критические замечания. В случае одобрения значительным большинством переработанный документ выпускается и распространяется для дальнейших комментариев и голосования. Теперь он носит название **проекта международного стандарта (Draft International Standard, DIS)**. На основе результатов этой стадии готовится, одобряется и публикуется окончательный текст **международного стандарта (International Standard, IS)**. В случаях, вызывающих особенно жаркие споры, может потребоваться несколько версий DIS, прежде чем он наберет достаточно голосов. Весь процесс может занимать годы.

**Национальный институт стандартов и технологий (National Institute of Standards and Technology, NIST)** является подразделением Министерства торговли США. Ранее он назывался Национальным бюро стандартов. Он выпускает стандарты, обязательные для закупок правительства США. Исключение составляют закупки Министерства обороны, выпускающего свои собственные стандарты.

Еще один важный игрок в мире стандартов — **Институт инженеров электротехники и электроники (Institute of Electrical and Electronics Engineers, IEEE)** — крупнейшее профессиональное объединение в мире. IEEE публикует десятки журналов и ежегодно проводит сотни конференций. Он также разрабатывает стандарты в области электротехники и вычислительной техники. Комитет 802 IEEE стандартизовал множество видов LAN. Мы обсудим некоторые результаты его деятельности далее в этой книге. Фактическую работу выполняет множество рабочих групп, перечисленных на илл. 1.37. Деятельность

| Номер | Тематика |
|---|---|
| 802.1 | Обзор и архитектура LAN |
| 802.2 | Управление логической связью |
| 802.3 * | Ethernet |
| 802.4 † | Маркерная шина (недолго использовалась на производстве) |
| 802.5 † | Token ring (первый шаг IBM в мире LAN) |
| 802.6 † | Двойная шина с распределенной очередью (одна из первых городских сетей) |
| 802.7 † | Техническая консультативная группа по широкополосным технологиям |
| 802.8 † | Техническая консультативная группа по оптоволоконным технологиям |
| 802.9 † | Изохронные LAN (для приложений, работающих в режиме реального времени) |
| 802.10 † | Виртуальные LAN и безопасность |
| 802.11 * | Беспроводные LAN (Wi-Fi) |
| 802.12 † | Обработка запросов с учетом приоритетов (AnyLAN компании Hewlett-Packard) |
| 802.13 | Несчастливый номер; никто не захотел его брать |
| 802.14 † | Кабельные модемы (закрылась: промышленный консорциум опередил их) |
| 802.15 * | Персональные сети PAN (Bluetooth, Zigbee) |
| 802.16 † | Широкополосные беспроводные сети (WiMAX) |
| 802.17 † | Отказоустойчивое пакетное кольцо |
| 802.18 | Техническая консультативная группа по вопросам нормативного регулирования радиосвязи |
| 802.19 | Техническая консультативная группа по вопросам совместимости стандартов |
| 802.20 | Мобильные широкополосные беспроводные сети (аналогично 802.16e) |
| 802.21 | Независимая от физической среды передача обслуживания (для перехода между технологиями) |
| 802.22 | Региональные беспроводные сети |

**Илл. 1.37.** Рабочие группы комитета 802. Наиболее важные отмечены знаком *. Отмеченные знаком † прекратили работу

этих групп не слишком результативна, а присвоение стандарту номера 802.x не гарантирует успеха. Тем не менее наиболее удачные стандарты (особенно 802.3 и 802.11) оказали колоссальное влияние на экономику и мир в целом.

### 1.7.4. Кто есть кто в мире интернет-стандартов

У всемирной сети интернет есть свои механизмы стандартизации, сильно отличающиеся от механизмов МСЭ-Т и ISO. Грубо говоря, разница в том, что на совещания по стандартизации в ITU и ISO люди приходят в костюмах, а участники собраний по стандартизации интернета носят джинсы (за исключением совещаний в Сан-Диего — туда они приходят в шортах и футболках).

Совещания МСЭ и ISO посещают должностные лица компаний и государственные чиновники, для которых стандартизация является работой. Они считают стандартизацию благим делом и посвящают ей свою жизнь. Люди интернета же, напротив, принципиально предпочитают анархию. Но когда сотни миллионов людей делают то, что им заблагорассудится, организовать связь очень непросто. Так что, увы, без стандартов не обойтись. Дэвид Кларк из MIT однажды высказал ставшее знаменитым замечание о том, что стандартизация интернета заключается в «приблизительном консенсусе и работающем коде».

При создании сети ARPANET Пентагон сформировал неофициальный комитет для надзора за ней. В 1983 году он получил название **IAB (Internet Activities Board — Совет по деятельности в сфере интернета)** и его функции были несколько расширены. Его задачей стало обеспечение более или менее единого направления исследований ARPANET и интернета (занятие, чем-то напоминающее ловлю разбегающихся кошек). Позднее аббревиатура IAB стала расшифровываться как **Internet Architecture Board (Совет по архитектуре интернета)**.

В IAB входили примерно десять участников, и каждый из них возглавлял тематическую рабочую группу по какому-либо важному вопросу. IAB собирался несколько раз в год для обсуждения результатов и отчета перед Пентагоном и NSF, которые в основном финансировали его деятельность. Когда требовался новый стандарт (например, алгоритм маршрутизации), члены IAB тщательно прорабатывали этот вопрос, а затем сообщали об изменениях аспирантам (выполнявшим главный объем работ по созданию программ), чтобы они могли его реализовать. Информационное взаимодействие производилось с помощью ряда технических отчетов — **RFC (Request For Comments)**. RFC доступны онлайн для всех желающих на сайте www.ietf.org/rfc. Они пронумерованы в порядке, соответствующем хронологии их создания. На сегодняшний день существует более 8000 RFC. Многие из них будут упоминаться в этой книге.

К 1989 году интернет разросся настолько, что подобный неформальный подход больше не работал. Тогда многие производители предлагали продукты на основе TCP/IP и не хотели менять их лишь потому, что какие-то десять исследователей придумали нечто получше. Летом 1989-го IAB снова реорганизовали. Специалистов перевели в **IRTF (Internet Research Task Force — Исследовательская группа интернет-технологий)**, подчиненную IAB, а также в **IETF (Internet Engineering Task Force — Инженерный совет интернета)**. Одновременно в IAB

вошли люди из более широкого спектра организаций, а не только сообщества исследователей. Изначально эта группа была самовозобновляемой: ее члены назначались на два года, причем прежние участники выбирали новых. Позже было создано **Общество интернета (Internet Society)**, в котором состояли люди, заинтересованные в развитии интернета (этим оно походило на ACM или IEEE). Во главе Общества интернета стоит выборный совет попечителей, назначающий членов IAB.

Основной смысл такого разделения был в том, чтобы IRTF сосредоточилась на долгосрочных исследованиях, а IETF занимался краткосрочными инженерными проектами. Теперь они не мешали друг другу. IETF разделился на рабочие группы, каждая из которых занималась решением конкретной задачи. Координационный комитет, куда входили председатели групп, определял общее направление деятельности. Тематика исследований включала новые приложения, информацию о пользователях, интеграцию OSI, маршрутизацию и адресацию, безопасность, управление сетями и стандарты. В итоге было сформировано так много рабочих групп (более 70), что они были объединены по сферам деятельности, и координационный комитет формировался из председателей этих объединений.

Кроме того, был взят на вооружение более формальный процесс стандартизации, по образцу ISO. Прежде чем стать **предлагаемым стандартом (Proposed Standard)**, идея должна быть изложена в RFC. В случае заинтересованности сообщества она получает дальнейшее рассмотрение. Для перехода рабочей реализации в стадию **проекта стандарта (Draft Standard)** она тщательно тестируется минимум на двух независимых площадках в течение хотя бы 4 месяцев[1]. Если IAB приходит к выводу, что идея оказалась здравой и программное обеспечение работает, он объявляет RFC **стандартом интернета (Internet Standard)**. Часть стандартов интернета стала стандартами Минобороны США (MIL-STD), обязательными для поставщиков Пентагона.

Что касается веб-стандартов, протоколы и руководства, обеспечивающие долгосрочное развитие Всемирной паутины, разрабатывает **Консорциум Всемирной паутины (World Wide Web Consortium, W3C)**. Это промышленный консорциум под руководством Тима Бернерса-Ли, созданный в 1994 году, когда Всемирная паутина начала по-настоящему обретать популярность. Сегодня в W3C состоят почти 500 компаний, университетов и других организаций. W3C издал более ста рекомендаций W3C (так называются его стандарты), охватывающих такие темы, как HTML и защита персональной информации в интернете.

---

[1] Авторы описывают этот процесс не совсем точно. Согласно RFC 2026 (https://tools.ietf.org/html/rfc2026), для перехода спецификации в стадию проекта стандарта должны существовать как минимум две независимые и функционально эквивалентные ее реализации на основе различных баз кода, причем должен быть накоплен достаточный опыт успешной практической эксплуатации. Также спецификация должна оставаться на стадии предлагаемого стандарта по крайней мере в течение 6 (а не 4 месяцев). — *Примеч. пер.*

## 1.8. ПОЛИТИЧЕСКИЕ, ПРАВОВЫЕ И СОЦИАЛЬНЫЕ ПРОБЛЕМЫ

Как и печатный станок пятьсот с лишним лет назад, компьютерные сети предоставили рядовым гражданам новые возможности получения и распространения информации. Но есть и обратная сторона медали — новым возможностям сопутствует множество нерешенных социальных, политических и этических проблем. Мы кратко обсудим их в этой главе. Позже, при рассмотрении конкретных технологий, мы коснемся (где это уместно) связанных с ними политических, юридических и социальных вопросов. Здесь представлены некоторые наиболее общие политические и правовые проблемы, влияющие на многие сферы интернет-технологий, включая установку приоритетов трафика, сбор данных и защиту персональной информации, а также ограничение свободы слова в интернете.

### 1.8.1. Свобода слова в интернете

Социальные сети, форумы, сайты обмена контентом и множество других ресурсов позволяют людям делиться своими взглядами с единомышленниками. До тех пор пока тематика общения ограничивается техническими вопросами или увлечением садоводством, особых проблем не возникает.

Неприятности начинаются при обсуждении действительно острых вопросов — политики, религии или секса. Мнение, публично высказанное одним человеком, может быть крайне оскорбительным для другого. Кроме того, различные платформы позволяют обмениваться не только текстом, но и цветными фотографиями в высоком разрешении и видеороликами. Некоторые публикации нарушают закон, например, когда речь идет о детской порнографии или подстрекательстве к терроризму.

Возможность публикации незаконных или оскорбительных материалов в соцсетях и на так называемых платформах **пользовательского контента** вызвала серьезный вопрос: ответственность этих площадок за модерацию. В течение долгого времени Facebook, Twitter, YouTube и другие платформы пользовались относительным иммунитетом от судебного преследования за размещенный контент. В США, например, такую защиту обеспечивал раздел 230 **Закона «О соблюдении приличий в коммуникациях» (Communications Decency Act)**. Многие годы социальные сети называли себя всего лишь инструментом для обмена информацией подобно печатному станку и утверждали, что они не должны отвечать за опубликованные материалы. Впрочем, поскольку эти платформы все более активно отбирают, ранжируют и персонализируют контент для пользователей, этот аргумент постепенно теряет силу.

Как в США, так и в Европе чаша весов постепенно склоняется в другую сторону. Принимаются законы, признающие ответственность сайтов за определенные виды незаконного контента, например связанного с предложением секс-услуг. Совершенствуются автоматические алгоритмы классификации публикаций на основе машинного обучения. Это приводит к попыткам возложить на социальные медиаплатформы ответственность за более широкий спектр контента, поскольку эти алгоритмы обеспечивают автоматическое обнаружение нежелательных материалов (от нарушений авторского права до

разжигания ненависти). На практике, однако, все не так просто, поскольку алгоритмы могут выдавать ложные результаты. Иногда алгоритм ошибочно распознает контент как оскорбительный или незаконный и автоматически его удаляет. Это воспринимается как цензура и посягательство на свободу слова. Законы, требующие от платформ вводить автоматическую модерацию, в конечном счете автоматизируют цензуру.

Звукозаписывающие компании и киностудии часто продвигают законы, предписывающие применение технологий автоматической модерации контента. В США их представители регулярно рассылают уведомления о нарушениях **DMCA (Digital Millennium Copyright Act — Закон об авторском праве в цифровую эпоху)** с угрозами обратиться в суд, если получатель не предпримет требуемых действий и не удалит материалы. Важно отметить, что интернет-провайдер (ISP) или поставщик контента не несут ответственности за нарушение авторских прав, если они перенаправили уведомление об этом лицу, допустившему нарушение. Также они не обязаны активно искать материалы, нарушающие авторские права, — это бремя возлагается на правообладателя (например, на студию звукозаписи или кинопродюсера). Найти и идентифицировать защищенный авторскими правами контент непросто, поэтому неудивительно, что держатели авторских прав продвигают законы, перекладывающие эту ответственность на ISP и поставщиков контента.

## 1.8.2. Сетевой нейтралитет

Один из главных юридических и политических вопросов за последние 15 лет — в какой мере ISP может блокировать или ранжировать контент в собственной сети. Принцип, согласно которому ISP обеспечивает равное качество сервиса для трафика одного типа (вне зависимости от источника контента), часто называют **сетевым нейтралитетом (network neutrality)** (У; Wu, 2003).

Основные принципы сетевого нейтралитета можно свести к следующим четырем правилам: (1) никакой блокировки; (2) никакого ограничения трафика; (3) никакой платной приоритизации; (4) прозрачность обоснованных методов сетевого управления, которые потенциально могут рассматриваться как нарушение любого из первых трех правил. Обратите внимание, что сетевой нейтралитет не означает, что ISP не может выбирать приоритеты трафика. В следующих главах представлены случаи, в которых это оправданно. Например, интернет-провайдер присваивает более высокий приоритет трафику в реальном времени (сетевым играм или видеоконференциям) относительно неинтерактивного трафика (резервного копирования больших файлов). Для подобных «обоснованных методов сетевого управления» в вышеуказанных правилах обычно делаются исключения. Безусловно, «обоснованность» методов — вопрос спорный. Перечисленные правила предназначены скорее для предотвращения попыток со стороны ISP установить монополию путем блокировки или ограничения трафика конкурентов. Такое случается, если конкурент предлагает аналогичные услуги: по этой причине AT&T заблокировала приложение FaceTime компании Apple. Также бывают ситуации, когда видеосервис (например, Netflix) конкурирует с собственной системой видео по запросу.

Хотя на первый взгляд идея сетевого нейтралитета может показаться простой и понятной, ее правовые и политические нюансы намного сложнее, особенно с учетом различий законодательства и сетей в разных странах. Например, в США пытаются решить вопрос о том, кто должен обеспечивать выполнение правил сетевого нейтралитета. Многочисленные судебные решения, вынесенные за последнее десятилетие, последовательно давали и снова отбирали у Федеральной комиссии по связи (Federal Communications Commission, FCC) полномочия по контролю за соблюдением сетевого нейтралитета различными ISP. Особенно жаркие споры ведутся о статусе ISP: либо это служба «общего пользования» (по аналогии с коммунальной службой), либо это информационный сервис (как Google и Facebook). Поскольку многие ISP предлагают все более разнообразные продукты, становится сложнее четко отнести их к той или иной категории. 11 июня 2018 года FCC упразднила сетевой нейтралитет на всей территории США, сохранив, впрочем, за отдельными штатами право вводить свои собственные правила.

Один из связанных с сетевым нейтралитетом вопросов, актуальный для многих стран, — практика **нулевого тарифа (zero rating)**. При такой системе пользователи оплачивают услуги ISP в соответствии с объемами трафика, но для отдельных сервисов делается исключение (то есть установлен «нулевой тариф»). Например, ISP может брать плату за просмотр Netflix, но предоставлять бесплатный доступ к другим видеосервисам, тем самым продвигая их. В некоторых странах операторы мобильной связи используют нулевой тариф в качестве конкурентного преимущества. Например, оператор может выставить нулевой тариф на Twitter в качестве промоакции, чтобы переманить абонентов других компаний. Еще один пример — сервис «Free Basics» компании Facebook, предоставляющий абонентам ISP бесплатный и неограниченный доступ к набору сайтов и сервисов, включенных Facebook в акционное предложение. Многие воспринимают подобные предложения как нарушение сетевого нейтралитета, ведь они подразумевают избирательный доступ к одним сервисам и приложениям в ущерб другим.

## 1.8.3. Безопасность

Интернет спроектирован так, что кто угодно может легко подключиться к нему и начать передавать трафик. Открытая архитектура не только стимулировала волну инноваций, но и сделала интернет платформой для атак неслыханного масштаба и размаха. Мы обсудим безопасность подробнее в главе 8.

Одна из самых распространенных и разрушительных атак — **DDoS (Distributed Denial of Service — «отказ в обслуживании»)**, при которой множество компьютеров в сети посылают трафик на компьютер-жертву в попытке истощить ее ресурсы. Существует множество различных видов DDoS-атак. Простейшая форма DDoS-атаки представляет собой отправку трафика большим числом компьютеров (в совокупности называемых **ботнетом**) на компьютер-жертву. DDoS-атаки обычно проводятся со взломанных компьютеров (например, ноутбуков или серверов), но изобилие плохо защищенных устройств IoT привело к возникновению совершенно нового вектора DDoS-атак. Может

ли скоординированная атака миллиона подключенных к интернету «умных» тостеров обрушить Google? К сожалению, IoT-производители не слишком озабочены безопасностью программного обеспечения. На сегодняшний день защита от атак со стороны крайне плохо защищенных IoT-устройств ложится на плечи сетевых операторов. Чтобы отбить у пользователей охоту подключать к сети ненадежные устройства, необходимы новые стимулы или нормативная база. В целом многие проблемы безопасности в интернете сводятся к наличию/отсутствию мотивации.

**Спам** (нежелательные сообщения) сегодня составляет более 90 % всего трафика электронной почты, поскольку спамеры накопили базы из миллионов адресов, а горе-маркетологи могут без особых затрат рассылать по ним сгенерированные сообщения. К счастью, существует ПО для фильтрации таких писем, способное анализировать и отбрасывать созданный другими компьютерами спам. Ранние варианты фильтрующего ПО отличали спам от настоящих писем в основном по содержанию сообщений. Однако спамеры быстро научились обходить эти фильтры (ведь можно легко сгенерировать 100 разных написаний слова «виагра»). Зато проверка IP-адресов отправителей и получателей, а также шаблонов отправки электронной почты оказалась намного эффективнее.

Чаще всего спам в электронной почте просто раздражает. Но порой такие сообщения могут оказаться частью мошеннической схемы или попыткой украсть персональные данные — пароли или информацию о банковских счетах. **Фишинговые (phishing)** сообщения маскируются под письма от надежного источника (например, банка). С их помощью злоумышленники пытаются обманом заставить пользователя раскрыть конфиденциальную информацию (к примеру, номера банковских карт). Хищение персональных данных становится серьезной проблемой, если мошенники собирают достаточно данных для оформления кредитных карт и других документов на имя жертвы обмана.

## 1.8.4. Защита персональной информации

По мере разрастания сетей и роста числа подключенных к ним устройств данные о действиях пользователей становятся доступнее для заинтересованных лиц. Компьютерные сети сильно упрощают обмен информацией, но одновременно дают возможность людям, управляющим сетями, шпионить за пользователями, просматривая их трафик. Собирать данные об использовании интернета могут провайдеры, операторы мобильной связи, приложения, веб-сайты, сервисы облачного хостинга, сети доставки контента, производители различных устройств, рекламодатели и производители программного обеспечения для веб-отслеживания.

Одна из самых распространенных практик, применяемых многими веб-сайтами и поставщиками приложений, — **профилирование (profiling)** и **отслеживание (tracking)** пользователей путем сбора информации об их поведении в сети за определенный промежуток времени. В частности, рекламодатели для отслеживания используют **cookie-файлы**, которые сохраняются веб-браузерами на компьютерах пользователей. Cookie-файлы позволяют рекламодателям и компаниям, занимающимся отслеживанием, следить за поведением пользователей

и их действиями на конкретных сайтах. В последние годы были разработаны и более изощренные механизмы отслеживания, например **сигнатуры браузеров (browser fingerprinting)**. Оказалось, что конфигурация браузера пользователя достаточно уникальна для его идентификации с высокой степенью достоверности. Для извлечения настроек браузеров на сайтах используется соответствующий скрипт. Компании, предоставляющие веб-услуги, хранят большие объемы персональной информации о своих клиентах для непосредственного изучения их поведения. Например, Google может читать электронную почту пользователей своего сервиса Gmail и показывать им рекламу в соответствии с их интересами.

Вследствие роста мобильных сервисов все большее значение приобретает проблема **защиты информации о местоположении (location privacy)** (см. работу Бересфорда и Стаяно (Beresford & Stajano, 2003)). Поставщик мобильной операционной системы обладает доступом к точной информации о местонахождении пользователя, включая географические координаты и даже высоту над уровнем моря (благодаря датчику атмосферного давления в вашем телефоне). Компания Google (поставщик ОС Android для мобильных телефонов) может определять местоположение пользователя внутри здания или торгового центра, чтобы показывать ему рекламу в зависимости от того, мимо какого магазина он проходит. Операторы мобильной связи могут определить текущее местонахождение клиента исходя из того, к какой базовой станции подключен его телефон.

Чтобы повысить конфиденциальность пользователей путем сокрытия источника пользовательского трафика, были созданы различные технологии, от VPN до средств анонимного просмотра информации в интернете (например, браузера Tor). Уровень защиты, предоставляемый этими системами, зависит от их свойств. Например, VPN может защитить незашифрованный трафик пользователя от просмотра интернет-провайдером. Но эти данные все равно будут доступны оператору VPN. Дополнительный слой защиты может обеспечить Tor, но его эффективность по-разному оценивается исследователями. Многие отмечают его слабые стороны, особенно в случае, когда значительные части инфраструктуры контролирует кто-то один. Благодаря анонимной связи студенты, сотрудники предприятий и простые граждане могут пожаловаться на незаконные действия, не опасаясь преследования. С другой стороны, в США и большинстве других демократических стран закон дает обвиняемому право на очную ставку в суде с тем, кто выдвигает против него обвинения, так что анонимные обращения не могут использоваться в качестве доказательств. Новые юридические проблемы возникают при применении устаревших законов к деятельности в сетях. На сегодняшний день один из любопытных правовых вопросов касается доступа к данным. В каких случаях у правительства должен быть доступ к информации о его гражданах? Должны ли отображаться в результатах поиска данные, физически расположенные в другой стране? Если данные передаются по территории страны, то до какой степени на них распространяются ее законы? Компания Microsoft столкнулась с этими вопросами в деле, рассмотренном Верховным судом США. Правительство США пыталось получить доступ к информации о гражданах США, хранившейся на серверах Microsoft в Ирландии. Весьма вероятно, что сама сущность интернета, не признающего границ, в будущем приведет к возникновению новых вопросов на стыке закона и технологий.

## 1.8.5. Дезинформация

Интернет дает возможность быстрого поиска информации, но немалая ее часть плохо проверена, приводит к неверным умозаключениям или вообще не соответствует действительности. Найденные в интернете медицинские рекомендации относительно боли в груди могут быть написаны как лауреатом Нобелевской премии, так и двоечником, не окончившим даже средней школы. Вопрос о том, как людям со всего мира находить достоверную информацию о текущих событиях и новостях, вызывает все большую озабоченность. К примеру, во время президентских выборов 2016 года в США отмечалась волна фейковых новостей. Определенные заинтересованные стороны фабриковали фальшивые истории, чтобы заставить читателей поверить в то, чего не было. Кампания по дезинформации поставила перед операторами сетей и платформ новые непростые вопросы. Что именно считать дезинформацией? Как ее обнаружить? И наконец, какие действия должен предпринять оператор сети или платформы при ее обнаружении?

# 1.9. ЕДИНИЦЫ ИЗМЕРЕНИЯ

Во избежание недоразумений имеет смысл уточнить, что в этой книге (и вообще в вычислительной технике) вместо традиционных английских единиц измерения используются метрические единицы. На илл. 1.38 перечислены основные метрические префиксы. Обычно они сокращаются до первых букв. Префиксы пишутся с заглавной буквы при числе больше единицы (Кбайт, Мбайт и т. д.). Итак, 1-Мбит/с линия связи передает $10^6$ бит в секунду, а шаг 100-пикосекундных часов равен $10^{-10}$ с. Также отметим, что префиксы милли- и микро- начинаются

| Степень 10 | Десятичное выражение | Префикс | Степень 10 | Десятичное выражение | Префикс |
|---|---|---|---|---|---|
| $10^{-3}$ | 0,001 | милли | $10^{3}$ | 1000 | кило |
| $10^{-6}$ | 0,000001 | микро | $10^{6}$ | 1000000 | мега |
| $10^{-9}$ | 0,000000001 | нано | $10^{9}$ | 1000000000 | гига |
| $10^{-12}$ | 0,000000000001 | пико | $10^{12}$ | 1000000000000 | тера |
| $10^{-15}$ | 0,000000000000001 | фемто | $10^{15}$ | 1000000000000000 | пета |
| $10^{-18}$ | 0,000000000000000001 | атто | $10^{18}$ | 1000000000000000000 | экса |
| $10^{-21}$ | 0,000000000000000000001 | зепто | $10^{21}$ | 1000000000000000000000 | зетта |
| $10^{-24}$ | 0,000000000000000000000001 | иокто | $10^{24}$ | 1000000000000000000000000 | иотта |

**Илл. 1.38.** Основные метрические префиксы

с «м». Чтобы не возникло путаницы, для них используются сокращения «м» и «мк»[1] соответственно.

При описании размеров дисков, файлов, баз данных и объемов оперативной памяти смысл вышеупомянутых единиц обычно несколько отличается. В этом случае префикс «кило» означает $2^{10}$ (1024), а не $10^3$ (1000), поскольку объем памяти всегда кратен степени двойки. Таким образом, 1 Кбайт памяти равен 1024 байта, а не 1000. Аналогично 1 Мбайт памяти содержит $2^{20}$ (1 048 576) байт, 1 Гбайт – $2^{30}$ (1 073 741 824) байт, а 1 Тбайт – $2^{40}$ (1 099 511 627 776) байт. В то же время по 1 Кбит/с линии связи передается 1000 бит в секунду, а 10 Мбит/с LAN работает на скорости 10 000 000 бит/с, поскольку эти скорости не являются степенями двойки. К сожалению, многие путают эти системы измерения, особенно в случае объемов дисков. В этой книге мы будем использовать Кбайт, Мбайт, Гбайт и Тбайт для $2^{10}$, $2^{20}$, $2^{30}$ и $2^{40}$ байт соответственно, а Кбит/с, Мбит/с, Гбит/с и Тбит/с для $10^3$, $10^6$, $10^9$ и $10^{12}$ бит/с соответственно.

## 1.10. КРАТКИЙ ОБЗОР СЛЕДУЮЩИХ ГЛАВ

В этой книге обсуждаются как общие принципы, так и практическое применение вычислительных сетей. Большинство глав начинается с обсуждения основ, а затем приводится множество иллюстрирующих их примеров. Обычно примеры связаны с интернетом или беспроводными сетями (например, мобильными), поскольку обе эти сферы важны и существенно отличаются друг от друга. Там, где это уместно, мы будем приводить и другие примеры.

Книга организована в соответствии с гибридной моделью, приведенной на илл. 1.36. Начиная с **главы 2**, мы будем продвигаться снизу вверх по иерархии протоколов. Мы приведем основные сведения из области обмена данными, охватывающие как проводные, так и беспроводные системы передачи. Основное внимание будет уделено доставке информации по физическим каналам, хотя мы будем изучать в основном вопросы архитектуры, а не аппаратного обеспечения. Также мы рассмотрим несколько примеров физического уровня: общественные коммутируемые и мобильные телефонные сети, а также сети кабельного телевидения.

В главах 3 и 4 обсуждается канальный уровень. В **главе 3** мы разберем задачу пересылки пакетов по линии связи, включая обнаружение и исправление ошибок. В качестве реального примера протокола канального уровня приведена технология DSL (используемая для широкополосного доступа в интернет по телефонным линиям).

В **главе 4** мы изучим подуровень доступа к среде передачи – составную часть канального уровня, отвечающую за совместное использование канала несколькими компьютерами. Мы рассмотрим несколько примеров, в том числе беспроводные (802.11) и проводные (Ethernet) LAN. Далее обсудим коммутаторы

[1] В международных обозначениях вместо «мк» используется «μ» (греческая буква «мю»). – *Примеч. пер.*

канального уровня, соединяющие различные LAN (например, в случае коммутируемой сети Ethernet).

**Глава 5** посвящена сетевому уровню, а именно маршрутизации. В ней описывается множество алгоритмов маршрутизации, как статических, так и динамических. Даже при хорошем алгоритме, если трафик превышает возможности сети, некоторые пакеты будут доставлены с задержкой или вообще пропадут. Мы обсудим способы предотвращения перегруженности сети и обеспечения качественного обслуживания. В этой главе затронуты многочисленные проблемы, связанные с объединением однородных сетей в интерсети. Кроме того, подробно описан сетевой уровень интернета.

**Глава 6** описывает транспортный уровень. Многие приложения нуждаются в протоколах, ориентированных на установление соединения, а также в обеспечении надежности. Этим вопросам уделено основное внимание. Подробно описаны оба транспортных протокола интернета, TCP и UDP, а также их проблемы с производительностью (особенно это касается TCP, одного из ключевых протоколов интернета).

**Глава 7** посвящена прикладному уровню, его протоколам и приложениям. Первая тема этой главы — DNS, «телефонная книга» интернета, вторая — электронная почта и ее протоколы. Далее мы поговорим о Всемирной паутине, подробно обсудим статический и динамический контент, а также процессы на клиентской и серверной сторонах. Затем коснемся темы мультимедийного сетевого контента, включая потоковые аудио и видео. И наконец, мы рассмотрим сети доставки контента, в том числе технологию пиринга.

**Глава 8** рассказывает о сетевой безопасности. Некоторые ее составляющие относятся ко всем уровням сетевой модели. Поэтому разумно обсуждать эту тему после подробного изучения уровней. Глава начинается со знакомства с криптографией. Далее рассказывается, как с ее помощью обеспечить безопасность связи, электронной почты и интернета вообще. Завершается глава обсуждением некоторых сфер, в которых безопасность переплетается с защитой персональной информации, свободой слова, цензурой и другими социальными вопросами.

**Глава 9** содержит аннотированный список рекомендуемой литературы по главам. Он предназначен для тех читателей, которые хотели бы продолжить знакомство с сетями. Эта глава также включает алфавитную библиографию всех упомянутых в книге источников.

Дополнительную информацию, которая может вас заинтересовать, вы найдете на веб-сайтах авторов:

https://www.pearsonhighered.com/tanenbaum

https://computernetworksbook.com

## 1.11. РЕЗЮМЕ

Компьютерные сети используются для множества целей, компаниями и частными лицами, дома и в дороге. В компаниях с их помощью осуществляется совместный доступ к корпоративной информации, как правило, при помощи

модели «клиент-сервер». Компьютеры сотрудников играют роль рабочих станций, которые обращаются к мощным серверам, расположенным в серверной комнате. Частным лицам сети дают доступ к разнообразной информации и развлекательным ресурсам, а также возможность покупать и продавать товары и услуги. Домашние пользователи подключаются к интернету через поставщиков телефонных или кабельных услуг (хотя для ноутбуков и телефонов все чаще применяется беспроводное подключение). Совершенствование технологий рождает новые виды мобильных приложений и сетей с помощью компьютеров, встроенных в различные бытовые приборы и другие устройства. Однако это способствует возникновению социальных проблем, например, связанных с защитой персональной информации.

Грубо говоря, сети можно разделить на LAN, MAN, WAN и интерсети. LAN обычно охватывает одно здание и работает на достаточно высокой скорости. MAN обычно охватывает целый город, например системы кабельного телевидения, широко используемые для доступа в интернет. WAN могут покрывать страну или целый континент. Для построения таких сетей используются как соединения «точка-точка» (в случае кабельных сетей), так и широковещательные технологии (в случае беспроводных сетей). Сети могут объединяться при помощи маршрутизаторов в интерсети, наиболее крупный и значимый пример — интернет. Также огромную популярность получили беспроводные сети, например LAN на основе стандарта 802.11 и мобильная телефония 4G.

В основе компьютерных сетей лежат протоколы, представляющие собой правила взаимодействия процессов. Большинство сетей поддерживает иерархии протоколов, в которых каждый уровень предоставляет службы вышележащему уровню, скрывая от него детали протоколов нижележащих уровней. В основе стека протоколов обычно лежат модели OSI или TCP/IP. В обеих моделях есть канальный, сетевой, транспортный и прикладной уровни; различаются они наличием/отсутствием остальных уровней. Главные вопросы, которые учитываются при разработке, — надежность, выделение ресурсов, способность к развитию, безопасность и пр. Значительная часть этой книги посвящена именно протоколам и их архитектуре.

Сети предоставляют своим пользователям разнообразные службы: от доставки пакетов без установления соединений по принципу «лучшее из возможного» до гарантированной доставки на основе соединений. В некоторых сетях на одном уровне предоставляются службы без соединений, а на уровне выше — ориентированные на установление соединения.

Среди наиболее известных сетей — интернет, мобильные телефонные сети и LAN на основе стандарта 802.11. Прародителем интернета стала сеть ARPANET, к которой подключалось множество сетей для формирования объединенной сети. Современный интернет фактически представляет собой набор из многих тысяч сетей, использующих стек протоколов TCP/IP. Мобильные телефонные сети предоставляют беспроводной и мобильный доступ в интернет на скорости в несколько мегабит в секунду и, разумеется, поддерживают также и голосовые звонки. Беспроводные LAN на основе стандарта IEEE 802.11 развертываются во множестве домов, отелей, аэропортов и ресторанов и могут обеспечить скорость в 1 Гбит/с и более. Различные виды беспроводных сетей постепенно сливаются.

Это можно наблюдать на примере LTE-U, который позволяет протоколам сотовых сетей работать в нелицензируемом диапазоне частот вместе с 802.11.

Чтобы миллионы устройств могли взаимодействовать друг с другом, необходима серьезная стандартизация как аппаратного, так программного обеспечения. За различные стороны процесса стандартизации отвечают такие организации, как МСЭ-Т, ISO, IEEE и IAB.

## ВОПРОСЫ И ЗАДАЧИ

1. Вы установили канал связи между двумя средневековыми замками в виде обученного ворона, переносящего свиток из замка-отправителя в замок-получатель, находящийся за 160 км. Ворон летит со средней скоростью в 40 км/ч и переносит один свиток за раз. Каждый свиток содержит 1,8 ТБ данных. Вычислите скорость передачи данных по этому каналу при отправке (а) 1,8 ТБ данных; (б) 3,6 ТБ данных; (в) бесконечного потока данных.
2. Каждый день все больше устройств подключается к IoT-сетям. Помимо прочего, IoT упрощает наблюдение за собственностью и контроль потребления коммунальных услуг. Но любую технологию можно использовать как во благо, так и во зло. Обсудите возможные недостатки IoT.
3. Беспроводные сети уже обогнали проводные по популярности, несмотря на то что их пропускная способность обычно меньше. Приведите две причины этого явления.
4. Вместо покупки своего собственного аппаратного обеспечения небольшие компании часто размещают оборудование в дата-центрах. Обсудите достоинства и недостатки этого подхода с точки зрения как компании, так и ее пользователей.
5. Альтернативой LAN может служить большая система с разделением времени, с терминалами для всех пользователей. Приведите два преимущества клиент-серверной системы, использующей LAN.
6. На быстродействие клиент-серверной системы очень сильно влияют две основные характеристики сети: пропускная способность сети (сколько битов в секунду она может передавать) и время задержки (через сколько секунд первый бит, отправленный с клиента, попадет на сервер). Приведите пример сети с высокой пропускной способностью и большим временем задержки. А затем — пример сети с низкой пропускной способностью и низким временем задержки.
7. Одним из факторов, влияющих на задержку при коммутации пакетов с промежуточным хранением данных, является время, требуемое для сохранения и отправки пакета далее через коммутатор. Окажет ли время коммутации в 20 мкс существенное влияние на задержку при ответе клиент-серверной системы, в которой клиент находится в Нью-Йорке, а сервер — в Калифорнии? При этом скорость распространения сигнала по медному и оптическому кабелю составляет 2/3 скорости света в вакууме.

8. Сервер отправляет пакеты клиенту через спутник. Прежде чем достичь места назначения, пакеты должны пройти через один или несколько спутников. В спутниках используется коммутация пакетов с промежуточным хранением данных со временем коммутации в 100 мкс. Если полное расстояние, проходимое пакетами, равно 29 700 км, то через сколько спутников должны пройти пакеты, учитывая, что на коммутацию пакетов приходится 1 % задержки?
9. Клиент-серверная система использует спутниковую сеть, причем спутник находится на высоте 40 000 км. Чему равна минимально возможная задержка ответа на запрос?
10. Сигнал движется со скоростью, составляющей 2/3 скорости света в вакууме, и попадает в место назначения через 100 мс. Какое расстояние он прошел?
11. Сегодня, когда практически у всех есть домашние компьютеры или мобильные устройства, подключенные к сети, стали возможны мгновенные голосования по важным законопроектам. Когда-нибудь законодательные органы вообще можно будет упразднить, а люди будут выражать свою волю непосредственно. Положительные стороны прямой демократии очевидны; подумайте, какие у нее могут быть недостатки.
12. Необходимо подключить пять маршрутизаторов к двухточечной сети. Каждую пару маршрутизаторов можно соединить линией с высокой, средней, низкой скоростью или вообще не соединять. Если генерация и оценка каждой топологии занимает у компьютера 50 мс, то сколько займет оценка их всех?
13. Пусть дана группа из $2^n - 1$ маршрутизаторов, связанных между собой в централизованном бинарном дереве, по одному маршрутизатору в каждом узле дерева. Маршрутизатор $i$ взаимодействует с маршрутизатором $j$, отправляя сообщение в корневой узел дерева. Далее корневой узел пересылает сообщение вниз, в узел $j$. Выведите приближенное выражение для среднего количества транзитных участков сети, проходимых сообщением, для больших значений $n$. Предполагается, что все пары маршрутизаторов равновероятны.
14. Недостаток широковещательной подсети — бесполезная трата ресурсов при одновременной попытке нескольких хостов получить доступ к каналу связи. В качестве упрощенного примера представьте себе, что время поделено на дискретные интервалы и каждый из $n$ хостов пытается воспользоваться каналом с вероятностью $p$ в каждом интервале времени. Какая доля интервалов будет расходоваться впустую из-за конфликтов?
15. В вычислительных сетях и других сложных системах, вследствие большого количества взаимодействий между компонентами, зачастую невозможно уверенно предсказать, когда случится что-то плохое. Как это учитывается при разработке сетей?
16. Объясните, почему канальный, сетевой и транспортный уровни добавляют к полезным данным информацию об источнике и месте назначения.

17. Сравните канальный, сетевой и транспортный уровни на предмет возможных гарантий, предоставляемых ими высшим уровням.

| ГАРАНТИЯ | УРОВЕНЬ |
|---|---|
| Доставка в режиме «лучшее из возможного» | Сетевой |
| Надежная доставка | Транспортный |
| Доставка с сохранением порядка | Транспортный |
| Абстракция байтового потока | Транспортный |
| Абстракция двухточечных соединений | Канальный |

18. Все уровни сети взаимодействуют с расположенным ниже уровнем через интерфейс. Укажите, интерфейсу какого уровня принадлежит каждая из приведенных ниже функций:

| ФУНКЦИЯ | ИНТЕРФЕЙС |
|---|---|
| send_bits_over_link(bits) | |
| send_bytes_to_process(dst, src, bytes) | |
| send_bytes_over_link(dst, src, bytes) | |
| send_bytes_to_machine(dst, src, bytes) | |

19. Пусть время прохода туда и обратно между двумя конечными точками сети составляет 100 мс, а отправитель посылает по пять пакетов во время каждого цикла. Какова будет скорость передачи отправителя, если размер пакетов — 1500 байт? Укажите ответ в байтах в секунду.
20. Президент Specialty Paint Corp. решил совместно с местной пивоварней разработать невидимую пивную бутылку (в качестве меры борьбы с мусором). Он поручает юридическому отделу изучить эту идею. Юристы, в свою очередь, обращаются за помощью к инженерам. Главный инженер звонит своему коллеге, работающему на аналогичной должности в пивоварне, чтобы обсудить техническую сторону проекта. Инженеры информируют свои юридические отделы, которые далее созваниваются между собой для согласования правовых вопросов. Наконец, два президента корпораций обсуждают финансовую сторону сделки. Какой из принципов модели OSI нарушает эта схема взаимодействия?
21. Каждая из двух сетей предоставляет надежную, ориентированную на установление соединения службу. Но одна из них предлагает стабильный байтовый поток, а другая — бесперебойный поток сообщений. Идентичны ли они? Если да, то почему мы их различаем? Если нет, проиллюстрируйте на примере их отличия.
22. Что означает термин «согласование» в контексте сетевых протоколов? Приведите пример.

23. На илл. 1.31 представлена служба. Подразумеваются ли на этом рисунке другие службы? Если да, то где? Если нет, то почему?
24. В некоторых сетях канальный уровень передачи данных отвечает за обработку ошибок передачи, запрашивая повторную отправку поврежденных фреймов. Если вероятность того, что фрейм поврежден, равна $p$, то чему в среднем равно число необходимых операций передачи для отправки фрейма? Предполагается, что подтверждения отправки никогда не теряются.
25. Какие из уровней OSI и TCP/IP отвечают за следующее:

    а) разбиение отправленных битовых потоков на фреймы;

    б) выбор пути через подсеть.
26. Если на канальном уровне обмениваются фреймами, а на сетевом уровне — пакетами, то фреймы инкапсулируют пакеты или наоборот? Поясните свой ответ.
27. Рассмотрим шестиуровневую иерархию протоколов, в которой низшим уровнем является уровень 1, а высшим — уровень 6. Приложение отправляет сообщение $M$ уровню 6. Каждый уровень с четным номером присоединяет к содержимому сообщения по концевой метке, а каждый с нечетным — по заголовку. Изобразите графически заголовки, концевые метки и исходное сообщение $M$ в порядке их отправки по сети.
28. Система включает иерархию протоколов из $n$ уровней. Приложения генерируют сообщения длиной $M$ байт. На каждом из уровней добавляется заголовок размером $h$ байт. Какая доля полосы пропускания сети заполнена заголовками?
29. Приведите пять примеров устройств, подключенных к двум сетям одновременно, и объясните, почему это может быть удобно.
30. Подсеть на илл. 1.12 (б) была разработана в расчете на возможную ядерную войну. Сколько бомб потребуется, чтобы разбить ее узлы на два не связанных между собой множества? Предполагается, что любая бомба уничтожает узел и все ведущие к нему каналы связи.
31. Интернет удваивается в размере примерно каждые 18 месяцев. Хотя никто не может сказать точно, но, согласно оценке 2018 года, число хостов было равно 1 млрд. Вычислите на основе этого ожидаемое число хостов в интернете в 2027 году. Кажется ли вам эта оценка правдоподобной? Поясните почему.
32. При передаче файла между двумя устройствами возможны две стратегии подтверждения получения. В первом варианте файл разбивается на пакеты и получатель отдельно подтверждает доставку каждого из них, но передача всего файла не подтверждается. Во втором случае подтверждается только получение файла целиком. Обсудите эти два подхода.
33. Операторам мобильных телефонных сетей необходимо знать, где находятся телефоны их абонентов (а значит, и сами эти абоненты). Объясните, чем это плохо для пользователей. В каких ситуациях это может быть полезно?

34. Какова была длина одного бита в исходном стандарте 802.3 в метрах? При этом скорость передачи составляет 10 Мбит/с, а скорость распространения сигнала в коаксиальном кабеле равна 2/3 скорости света в вакууме.
35. Пусть дано изображение размером 3840 × 2160 пикселей, причем каждый пиксель занимает 3 байта. Допустим, что оно не сжато. Сколько времени займет его передача по 56-килобитному модемному каналу? Через 1-мегабитный кабельный модем? По 10-мегабитной сети Ethernet? Через гигабитную сеть Ethernet?
36. Между Ethernet и беспроводными сетями есть определенные сходства и различия. Одно из свойств Ethernet заключается в передаче одновременно только одного фрейма. Разделяет ли стандарт 802.11 это свойство с Ethernet? Раскройте эту тему.
37. Беспроводные сети очень просты в установке, а потому и дешевы (ведь затраты на установку в других случаях намного превосходят стоимость оборудования). Тем не менее у них есть и недостатки. Назовите два из них.
38. Приведите два достоинства и два недостатка существования международных стандартов сетевых протоколов.
39. Если система состоит из постоянной и съемной частей (например, как привод CD-ROM и компакт-диск), важна ее стандартизация, чтобы компоненты от разных производителей работали друг с другом. Приведите три примера подобных международных стандартов в других сферах помимо вычислительной техники. Теперь назовите три области (не относящиеся к компьютерам), где таких стандартов нет.
40. На илл. 1.34 приведено несколько различных протоколов из сетевого стека TCP/IP. Объясните, зачем могут пригодиться несколько протоколов на одном уровне. Приведите пример.
41. Предположим, что поменялись алгоритмы, используемые для реализации операций на уровне $k$. Как это повлияет на операции на уровнях $k - 1$ и $k + 1$?
42. Предположим, что изменилась служба (набор операций), предоставляемая уровнем $k$. Как это повлияет на операции на уровнях $k - 1$ и $k + 1$?
43. Выясните, как открыть встроенный сетевой монитор вашего браузера. Откройте его и перейдите на какую-нибудь веб-страницу (например, https://www.cs.vu.nl/~ast/). Сколько запросов ваш браузер (клиент) отправил на сервер? Какие это запросы? Почему запросы отправляются по отдельности, а не как один большой запрос?
44. Составьте список ваших ежедневных дел, в которых задействованы компьютерные сети.
45. Программа *ping* дает возможность отправить тестовый пакет по заданному адресу и узнать, сколько времени занимает его путь туда и обратно. Попробуйте воспользоваться *ping*, чтобы узнать, сколько времени занимает прохождение пакетов от вашего компьютера до нескольких конкретных мест. Исходя из этих данных постройте график времени прохождения пакета в одну сторону как функцию расстояния. Лучше использовать для этой цели

университеты, поскольку местоположение их серверов известно с большой точностью. Например, berkeley.edu находится в Беркли, штат Калифорния; mit.edu — в Кембридже, штат Массачусетс; vu.nl — в Амстердаме, Нидерланды; www.usyd.edu.au — в Сиднее, Австралия; и www.uct.ac.za — в Кейптауне, ЮАР.

46. Перейдите на веб-сайт IETF, www.ietf.org, и ознакомьтесь с его деятельностью. Выберите проект по своему вкусу и напишите отчет на полстраницы о решаемой им задаче и предлагаемом решении.
47. Стандартизация играет важную роль в мире сетей. Основные официальные организации, занимающиеся стандартизацией, — МСЭ и ISO. Перейдите на их веб-сайты, www.itu.org и www.iso.org соответственно, и почитайте об их работе. Напишите короткий отчет о стандартизируемых ими вещах.
48. Интернет состоит из огромного количества сетей. Их расположение определяет топологию интернета. В Сети доступно немало информации на эту тему. Воспользуйтесь поисковыми системами, чтобы узнать больше о топологии интернета, и напишите короткий отчет по итогам найденного.
49. Поищите в интернете информацию о важных пиринговых точках, используемых в настоящее время для маршрутизации пакетов в интернете.
50. Напишите программу, которая осуществляет поток сообщений с верхнего на нижний уровень 7-уровневой модели протоколов. Программа должна включать по отдельной функции протокола для каждого уровня. Заголовки протокола представляют собой последовательности, содержащие до 64 символов. У каждой функции протокола есть два параметра: сообщение, передаваемое от протокола более высокого уровня (символьный буфер), и размер этого сообщения. Функция присоединяет заголовок перед сообщением, выводит получившееся новое сообщение в стандартный поток вывода, после чего вызывает функцию протокола нижележащего уровня. Входными данными для программы служит сообщение приложения.

# ГЛАВА 2
# Физический уровень

В этой главе мы рассмотрим низший уровень нашей эталонной модели — физический. Он задает электрические, синхронизационные и прочие интерфейсы, посредством которых биты пересылаются по каналам в виде сигналов. Физический уровень — фундамент сети. Свойства различных видов физических каналов определяют быстродействие (например, пропускную способность, время задержки и частоту ошибок), так что это идеальная стартовая точка для путешествия в мир сетей.

Начнем со знакомства с тремя видами сред передачи данных: проводные/направляемые (например, медные провода, коаксиальный кабель, оптоволокно), беспроводные (наземная радиосвязь) и спутниковые. Все эти технологии обладают различными свойствами, влияющими на архитектуру и быстродействие сетей. Мы дадим базовую информацию о ключевых технологиях передачи данных в современных сетях.

Далее представим теоретический анализ передачи данных и обнаружим, что природа наложила определенные ограничения на работу каналов связи (то есть физических сред, используемых для отправки битов). Затем обсудим цифровую модуляцию — преобразование аналоговых сигналов в цифровые биты и обратно. После этого рассмотрим схемы мультиплексирования и изучим возможности передачи в одной среде нескольких сеансов связи одновременно, без каких-либо помех друг для друга.

Наконец, рассмотрим три системы связи, применяемые для глобальных сетей: стационарную и мобильную телефонные системы, а также систему кабельного телевидения. Все они имеют важное практическое значение и заслуживают внимания.

## 2.1. ПРОВОДНЫЕ СРЕДЫ ПЕРЕДАЧИ ДАННЫХ

Задача физического уровня состоит в переносе битов с одного устройства на другое. Для передачи данных могут использоваться различные физические среды. Среды передачи с использованием физического кабеля или провода часто называются **проводными**, или **направляемыми (guided transmission media)**, поскольку в них сигнал направлен по физическому пути. Наиболее распространенные

проводные среды передачи — медные кабели (в виде коаксиального кабеля или витой пары) и оптоволокно. Все они имеют свои достоинства и недостатки в том, что касается частот, ширины полосы пропускания, задержки, стоимости и удобства установки, а также технического обслуживания. Ширина полосы пропускания — мера пропускной способности среды передачи. Она измеряется в **герцах** (Гц) (МГц, ГГц). Эта единица измерения названа в честь немецкого физика Генриха Герца. Мы обсудим ее подробнее далее в этой главе.

## 2.1.1. Запоминающее устройство

Стандартный способ переноса информации с одного устройства на другое — записать на носитель, магнитный или твердотельный (например, записываемый DVD), физически доставить его к целевому устройству и считать данные. Это не так современно, как использование геостационарного спутника связи, но зачастую более рентабельно, особенно в ситуациях, где ключевым фактором является высокая скорость передачи данных или стоимость в расчете на переданный бит.

Это ясно из простого расчета. Стандартный магнитный картридж в формате Ultrium может содержать до 30 Тбайт данных. В коробке размером 60 × 60 × 60 см помещается около 1000 таких накопителей общей емкостью 30 000 ТБ, то есть 240 000 Тбит (240 Пбит). Federal Express или другая логистическая компания может доставить эту коробку в любую точку США за 24 часа. Фактическая пропускная способность при таком варианте передачи данных составит 240 000 Тбит/86 400 с, то есть более 2700 Гбит/с. А если место назначения всего в часе езды, то пропускная способность превысит 66 Тбит/с. Ни одна сеть не способна даже приблизиться к подобным показателям. Конечно, скорость сетей растет, но растет и плотность записи на магнитную ленту.

Если взглянуть на стоимость, картина будет аналогичной. Оптовая цена картриджа Ultrium — $40. Учитывая, что его можно повторно использовать минимум 10 раз, коробка обойдется в $4000. Добавим к этому $1000 за услуги доставки (скорее всего, намного меньше), и получится примерно $5000 за передачу 30 000 Тбайт данных. Стоимость пересылки одного гигабайта составит чуть более половины цента. Сети не могут с этим соперничать. Мораль истории такова:

*Никогда не недооценивайте пропускную способность несущегося по шоссе грузовика, набитого магнитными картриджами.*

Зачастую это наилучшее решение при перемещении очень больших объемов данных. Компания Amazon предоставляет сервис Snowmobile. Это большой грузовик, набитый тысячами жестких дисков, подключенных к высокоскоростной сети внутри грузовика. Его общая емкость составляет 100 Пбайт (100 000 Тбайт или 100 млн Гбайт). Если компания нуждается в перемещении огромного объема данных, такой грузовик приезжает на ее территорию, подключается к оптоволоконной сети компании, а затем извлекает нужную информацию. По завершении грузовик едет в место назначения и выгружает данные. Этот сервис может пригодиться компании, желающей использовать

облако Amazon вместо собственного огромного дата-центра. Остальные методы передачи и близко не сравнятся с этим сервисом, если речь идет о гигантских объемах данных.

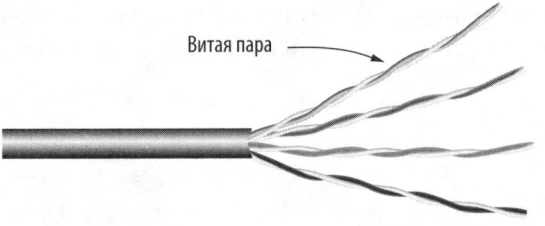
## 2.1.2. Витая пара

При использовании запоминающих устройств можно получить отличную пропускную способность. Однако с показателями задержки все обстоит иначе: время передачи данных измеряется часами или днями, а не миллисекундами. Для многих приложений, включая веб-приложения, видеоконференции и онлайн-игры, важна низкая задержка при передаче данных. **Витая пара (twisted pair)** — одна из старейших, но все еще наиболее популярная среда передачи данных. Витая пара состоит из двух изолированных медных проводов, обычно толщиной около 1 мм. Провода скручены в спираль подобно молекуле ДНК. Два параллельных провода образуют отличную антенну, а когда они скручены, волны от различных витков взаимно гасятся, так что провод в целом распространяет излучение гораздо слабее. Сигнал обычно передается в виде разности потенциалов между двумя проводами пары, что обеспечивает (в отличие от абсолютного напряжения) лучшую устойчивость к внешнему шуму. Как правило, шум одинаково влияет на напряжение в обоих проводах, таким образом, разность потенциалов остается практически неизменной.

Чаще всего витая пара используется в телефонных системах. Практически все телефоны соединяются с АТС посредством такого кабеля. По этим линиям связи осуществляются и телефонные звонки, и доступ в интернет по технологии ADSL. Витые пары длиной до нескольких километров могут обходиться без усиления, но на больших расстояниях сигнал ослабляется и необходимы повторители. Чтобы протянуть множество витых пар параллельно на большое расстояние (например, от многоквартирного дома до АТС), их связывают вместе и заключают в защитную оболочку. Если бы не скручивание, пары проводов создавали бы помехи друг на друга. В некоторых регионах телефонные линии прокладываются над землей, на столбах, и связки проводов достигают диаметра в несколько сантиметров.

Витая пара используется для передачи как аналоговых, так и цифровых данных. Пропускная способность зависит от диаметра провода и расстояния. В большинстве случаев при расстоянии в несколько километров она может достигать сотен мегабит в секунду (и даже больше — при использовании различных хитростей). Благодаря достаточному быстродействию, широкой доступности и низкой стоимости витая пара очень популярна, и в ближайшем будущем эта ситуация вряд ли изменится.

Кабели на основе витой пары бывают нескольких видов. Повсеместно используемая сегодня разновидность витой пары называется кабелем **категории 5е**, или «Cat. 5e». Витая пара категории 5е состоит из двух аккуратно скрученных проводов. Четыре такие пары обычно заключаются в ПВХ-оболочку, которая защищает провода и держит их вместе. Эта компоновка проводов показана на илл. 2.1.




**Илл. 2.1.** Кабель категории 5e из четырех витых пар. Подобные кабели применяются для локальных сетей

В различных стандартах LAN витая пара используется по-разному. Например, 100-мегабитная сеть Ethernet использует две (из четырех) пар, по одной для каждого направления. Для повышения скорости 1-гигабитная Ethernet использует все четыре пары в обоих направлениях одновременно, поэтому получатель должен факторизовать передаваемый сигнал.

Немного общей терминологии. Каналы связи, позволяющие передавать сигнал в обе стороны одновременно (подобно двухполосной дороге), называются **полнодуплексными (full-duplex)**. Линии, которые можно использовать в конкретный момент времени только в одном направлении (подобно одноколейной железной дороге), называют **полудуплексными (half-duplex)**. Третья категория — **симплексные (simplex)** каналы связи, по которым трафик всегда движется лишь в одном направлении, как по односторонним улицам.

Вернемся к витой паре. Более раннюю **категорию 3** сменила категория 5, с аналогичным кабелем и таким же разъемом, но с увеличенным количеством скручиваний на метр. Большее число скручиваний уменьшает перекрестные помехи и улучшает качество сигнала при передаче на дальние расстояния. Благодаря этому подобные кабели лучше подходят для высокоскоростного обмена данными между устройствами, особенно для 100-мегабитных и 1-гигабитных сетей Ethernet.

Новым стандартом, вероятно, станет **категория 6** или даже **7**. Еще более строгие спецификации этих категорий обеспечивают передачу сигналов с большей шириной полосы пропускания. Некоторые кабели категории 6 поддерживают 10-гигабитные каналы связи. Сегодня такие каналы широко развертываются во многих сетях, например в новых офисных зданиях. Кабели **категории 8** работают на более высоких скоростях, чем витые пары более низких категорий, но только на коротких расстояниях (около 30 м). Поэтому они подходят только для датацентров. У стандарта категории 8 есть две разновидности: Класс I, совместимый с категорией 6A, и Класс II, совместимый с категорией 7A.

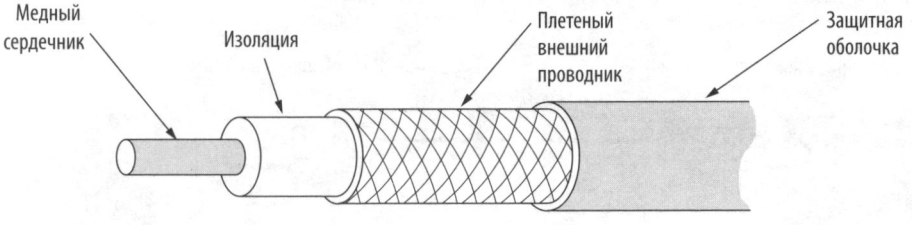
Кабели вплоть до категории 6 носят название **неэкранированной витой пары (Unshielded Twisted Pair, UTP)**, поскольку состоят только из проводов и изоляции. В отличие от них, в кабеле категории 7 экранирована каждая витая пара и весь пучок в целом (под защитной оболочкой). Экранирование снижает чувствительность к внешним и перекрестным помехам с расположенными рядом кабелями, благодаря чему кабель может удовлетворять самым высоким требованиям к производительности. Эти кабели напоминают высококачественные, но громоздкие и дорогие экранированные кабели на основе витых пар, выпущенные IBM в начале 1980-х. Особой популярности, помимо

использования в системах самой IBM, они не приобрели. Видимо, пришло время повторить попытку.

## 2.1.3. Коаксиальный кабель

Еще одна распространенная среда передачи — **коаксиальный кабель (coaxial cable)**. Он лучше экранирован и обладает более широкой полосой пропускания, чем неэкранированные витые пары, так что подходит для передачи на более далекие расстояния и с более высокой скоростью. Широко используются два типа коаксиального кабеля. Один из них, 50-омный, обычно применяется, когда изначально планируется передача данных в цифровом виде. Другой, 75-омный кабель, часто используется для передачи аналоговых данных и кабельного телевидения. Это разделение возникло скорее по историческим, чем по техническим причинам. Например, полное сопротивление первых дипольных антенн[1] составляло 300 Ом, так что удобно было использовать уже существующие согласующие трансформаторы полного сопротивления 4:1. С середины 1990-х операторы кабельного телевидения начали предоставлять доступ в интернет, после чего возросла значимость 75-омных кабелей.

Коаксиальный кабель состоит из жесткой медной жилы, покрытой изоляцией. Изоляция, в свою очередь, заключена в цилиндрический проводник — обычно в виде тесно переплетенной сетки. Внешний проводник покрыт защитной оболочкой. Коаксиальный кабель в разрезе показан на илл. 2.2.




**Илл. 2.2.** Коаксиальный кабель

Конструкция и экранирование коаксиального кабеля обеспечивает удачное сочетание высокой пропускной способности и отличной защиты от помех (например, пультов дистанционного управления гаражными дверями, микроволновых печей и т. д.). Коаксиальный кабель имеет чрезвычайно широкую полосу пропускания (она зависит от качества и длины кабеля). У современных кабелей она достигает 6 ГГц, что позволяет передавать по одному кабелю множество сеансов связи параллельно (одна телевизионная программа занимает приблизительно 3,5 МГц). Коаксиальные кабели когда-то широко применялись в междугородних телефонных системах, но сегодня на замену им пришло оптоволокно. Коаксиальный кабель все еще широко используется для кабельного телевидения и городских сетей, а также для высокоскоростного домашнего интернета.

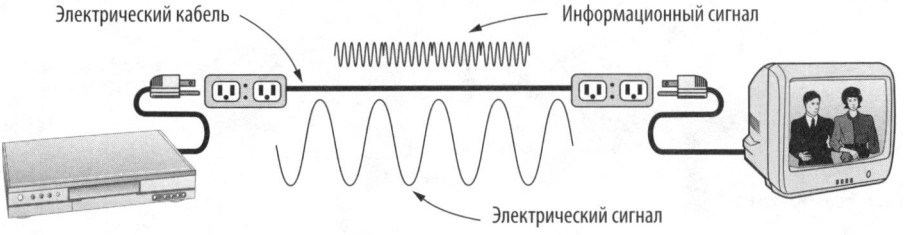
[1] Они же «антенны Герца». — *Примеч. пер.*

## 2.1.4. Линии электропередачи

Телефонные сети и сети кабельного телевидения не единственные проводные линии, которые можно дополнительно использовать для обмена данными. Существует еще более распространенный вид проводов — линии электропередачи (ЛЭП). ЛЭП служат для поставки электроэнергии в дома, а электропроводка внутри жилищ — для распределения энергии по электрическим розеткам.

Идея передачи данных по ЛЭП возникла давно. Долгие годы электроэнергетические компании использовали ЛЭП для низкоскоростного обмена данными, чтобы удаленно снимать показания счетчиков. Кроме того, данная технология позволяет управлять различными домашними устройствами (например, по стандарту X10). В последние годы возродился интерес к высокоскоростному обмену данными по таким линиям, как внутри жилых зданий в качестве LAN, так и снаружи — для широкополосного доступа в интернет. Мы рассмотрим наиболее распространенный сценарий — использование электропроводки в жилых домах.

Преимущества использования электропроводки для вычислительных сетей очевидны. Достаточно включить телевизор и тюнер в розетку — это придется сделать в любом случае, поскольку им требуется питание, — и они сразу получают возможность отправлять и принимать фильмы по электропроводке. Такая конфигурация представлена на илл. 2.3. Никаких других подключений или радиоустройств не требуется. Информационный сигнал накладывается на низкочастотный электрический сигнал (по активным, находящимся под напряжением проводам): оба сигнала используют проводку одновременно.




**Илл. 2.3.** Сеть на основе домашней электропроводки

Проблема использования домашней электропроводки для организации сети состоит в том, что она была предназначена для подачи электроэнергии. Эта задача коренным образом отличается от распространения информационных сигналов, и домашняя электропроводка справляется с ней очень плохо. Электрические сигналы передаются на частоте 50–60 Гц, при этом более высокочастотные сигналы (начиная от 1 МГц), необходимые для высокоскоростного обмена данными, затухают. Электрические свойства проводов различны в разных домах и меняются по мере включения/выключения бытовых электроприборов, что приводит к резким скачкам информационных сигналов в проводах. Возникающие при включении/выключении бытовых электроприборов переходные токи создают электрический шум в широком диапазоне частот. А без аккуратного скручивания как у витой

пары электропроводка ведет себя как антенна, подхватывая внешние сигналы и излучая в пространство свои собственные. Следовательно, для удовлетворения нормативных требований информационный сигнал должен избегать лицензируемых частот (например, диапазонов, выделенных для радиолюбителей).

Несмотря на эти сложности, по обычной электропроводке вполне можно передавать данные на короткие расстояния со скоростью до 500 Мбит/с с помощью схем связи, избегающих проблемных частот и устойчивых к всплескам количества ошибок. Для многих продуктов применяются защищенные патентами стандарты по организации сетей на основе ЛЭП; разрабатываются также открытые стандарты.

## 2.1.5. Оптоволокно

Многие представители компьютерной индустрии невероятно гордятся быстротой развития вычислительных технологий в соответствии с законом Мура, по которому число транзисторов в микросхеме удваивается каждые два года (см. работу Кушика и Хаммудеха; Kuszyk and Hammoudeh, 2018). Первый ПК IBM (1981) работал на тактовой частоте 4,77 МГц. Сорок лет спустя ПК содержат четырехъядерный CPU, работающий на частоте 3 ГГц. Ускорение примерно в 2500 раз. Впечатляет.

В то же время скорость глобальных линий связи выросла от 45 Мбит/с (линия T3 в телефонных системах) до 100 Гбит/с (современная междугородняя линия). Ничуть не менее впечатляющий рост — более чем в 2000 раз; частота ошибок при этом упала с $10^{-5}$ практически до нуля. За прошлое десятилетие был достигнут предел возможностей отдельного CPU, вследствие чего начало расти число ядер CPU на чип. А потенциальная пропускная способность оптоволокна превышает 50 000 Гбит/с (50 Тбит/с), и мы еще очень далеки от этих пределов. На сегодняшний день мы достигли «потолка» (около 100 Гбит/с) лишь из-за нашей неспособности быстрее преобразовывать электрические сигналы в оптические и обратно. Для достижения более высокой производительности по одному оптоволоконному кабелю параллельно передаются данные нескольких каналов связи.

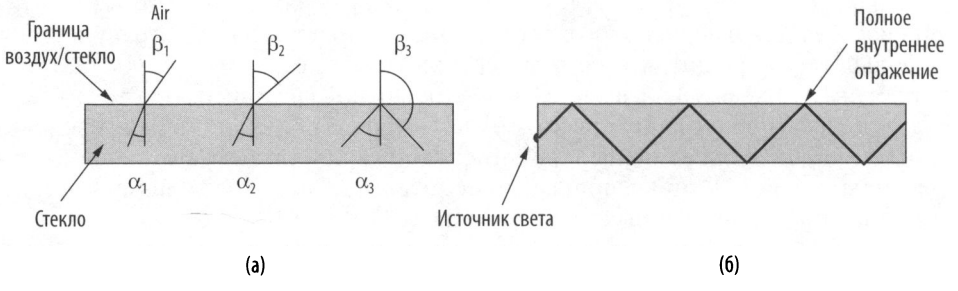
В этом разделе мы разберемся, как работает оптоволоконная технология передачи данных. В непрерывном состязании компьютерных систем и средств связи последние вполне могут выиграть за счет оптоволоконных сетей. Если это произойдет, пропускная способность окажется практически неограниченной, а большинство решит, что компьютеры работают безнадежно медленно, так что сетям лучше избегать вычислений любой ценой (независимо от того, какая часть полосы пропускания при этом будет потеряна). Пройдет немало времени, прежде чем подобное мнение станет господствующим в среде специалистов по вычислительной технике, привыкших смотреть на все через призму жестких ограничений, свойственных медным проводам.

Конечно, подобный сценарий не учитывает стоимость. Затраты на прокладку оптоволоконного кабеля до каждого потребителя (во избежание низкой пропускной способности проводов и ограниченного диапазона частот) колоссальны. Кроме того, расход электроэнергии на передачу битов больше,

чем на вычисления. Всегда будут появляться «островки неравенства», где либо вычисления, либо передача данных практически бесплатны. Например, при входе в интернет применяются средства вычисления и хранения, чтобы сжать и кэшировать контент, — все для оптимального использования каналов доступа. Внутри интернета же может происходить обратное. Такие компании, как Google, перемещают по Сети колоссальные массивы данных туда, где их хранение или обработка обойдутся дешевле.

Оптоволокно используется для передачи данных на большие расстояния в опорных сетях, высокоскоростных LAN (хотя медные провода нередко успешно с ними в этом соперничают), а также для высокоскоростного доступа в интернет по технологии FTTH («оптоволокно в дом»). Оптические системы передачи данных состоят из трех основных компонентов: источника (генератора) света, среды передачи и приемника. Принято считать, что световой импульс означает 1, а отсутствие света — 0. Среда передачи представляет собой сверхтонкое стекловолокно. При попадании света приемник генерирует электрический импульс. Установив генератор света на одном конце оптоволоконного кабеля, а приемник — на другом, мы получаем однонаправленную (то есть симплексную) систему передачи данных, которая принимает входной электрический сигнал, преобразует его, передает в виде световых импульсов, после чего преобразует выходной сигнал обратно в электрический на принимающей стороне.

Подобная система передачи данных была бы бесполезной на практике из-за утечек света, если бы не один интересный физический принцип. Когда луч света переходит из одной среды в другую, например из кварцевого стекла в воздух, он преломляется на границе стекло/воздух. На илл. 2.4 (а) показано, как луч света падает на границу под углом $\alpha_1$ и отражается под углом $\beta_1$. Сила преломления зависит от свойств обеих сред (в частности, их коэффициентов преломления). Если углы падения превышают определенное критическое значение, свет отражается обратно в стекло и не попадает в воздух вообще. Следовательно, луч света, падающий под критическим (или превышающим его) углом, оказывается «пойман» внутри оптоволокна, как показано на илл. 2.4 (б). Этот луч может распространяться на многие километры практически без потерь.




**Илл. 2.4.** (а) Три примера попадания луча света изнутри кварцевого волокна на границу воздух/стекло под разными углами. (б) Свет, удерживаемый внутри вследствие полного внутреннего отражения

На илл. 2.4 показан только один «пойманный» внутри волокна луч. Но поскольку любой луч с углом падения выше критического отразится внутрь, множество лучей будет отражаться внутри волокна под разными углами. В этом случае говорят, что у лучей различные моды, а волокно при этом называется **многомодовым (multimode fiber)**. Если же уменьшить диаметр волокна до нескольких длин волны света (менее 10 мкм (микрометров); при этом диаметр многомодового волокна превышает 50 мкм), волокно становится волноводом. Это значит, что свет может распространяться в нем только по прямой, без отражений. Такое волокно называется **одномодовым (single-mode fiber)**. Несмотря на высокую стоимость, оно широко используется для передачи данных на большие расстояния. Одномодовое волокно способно передавать сигналы на расстояние примерно в 50 раз больше, чем многомодовое. Современные одномодовые оптоволоконные кабели работают со скоростью 100 Гбит/с на расстоянии до 100 км без усиления. В лабораторных условиях были достигнуты еще большие скорости, правда, для коротких дистанций. Выбор одномодового или многомодового волокна зависит от сценария применения. Многомодовые оптоволоконные кабели используются для передачи данных на расстояние до 15 км и позволяют применять более дешевое оптоволоконное оборудование. Однако их пропускная способность уменьшается по мере увеличения расстояния.

### *Передача света через оптоволокно*

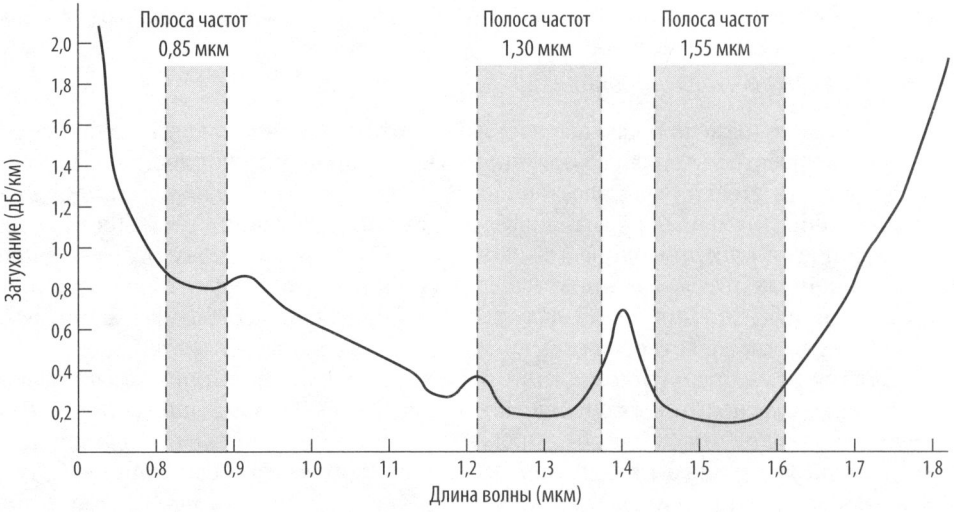
Оптическое волокно производится из стекла, которое, в свою очередь, делается из песка — дешевого материала, доступного в неограниченных количествах. Секрет изготовления стекла был известен еще древним египтянам, но стекло должно быть толщиной не более 1 мм, иначе через него не проходит свет. Достаточно прозрачное стекло, пригодное для окон, появилось лишь в эпоху Возрождения. В современных оптоволоконных кабелях используется невероятно прозрачное стекло. Если бы оно заполняло океаны вместо воды, можно было бы рассмотреть морское дно так же четко, как землю из самолета в ясный день.

Затухание света при прохождении через стекло зависит от длины волны света (равно как и от некоторых физических свойств стекла). Оно определяется как отношение мощности входного сигнала к мощности выходного. На илл. 2.5 приведен график затухания света для стекла, применяемого в оптоволоконных кабелях, в децибелах (дБ) на километр длины кабеля. В качестве примера: двукратное ослабление мощности сигнала соответствует затуханию в $10 \log_{10} 2 = 3$ дБ. Мы обсудим децибелы чуть позже. Если кратко, это логарифмическая мера отношения мощностей, где 3 дБ соответствуют двукратному отношению мощностей. На илл. 2.5 представлена часть спектра, близкая к инфракрасной, применяемая на практике. Длина волны видимого света чуть короче, от 0,4 до 0,7 мкм (1 мкм = $10^{-6}$ м). Настоящий борец за чистоту метрической системы описал бы эти длины волн как «от 400 до 700 нанометров», но мы будем придерживаться более традиционного написания.

В настоящее время для оптоволоконной связи наиболее широко используются три диапазона длин волн. Центры их находятся в точках 0,85, 1,30 и 1,55 мкм. Ширина всех трех диапазонов — от 25 000 до 30 000 ГГц. Сначала использовался

0,85-микрометровый диапазон. Он отличался более быстрым затуханием и потому применялся для передачи на меньшие расстояния, но при такой длине волны лазеры и электроника могут быть из одного материала (арсенид галлия). Остальные два диапазона характеризуются хорошими показателями затухания: потери составляют менее 5 % на километр. Сегодня 1,55-микрометровый диапазон широко применяется с усилителями, легированными ионами эрбия. Эти усилители работают непосредственно в оптической зоне.

Световые импульсы растягиваются в длину по мере их движения по оптоволокну. Это явление называется **хроматической дисперсией (chromatic dispersion)**. Ее степень зависит от длины волны. Можно избежать наложения растянутых импульсов, увеличив расстояние между ними, но это снижает частоту передачи. К счастью, было обнаружено, что если придать импульсам специальную форму (соответствующую обратной величине гиперболического косинуса), то дисперсионные эффекты практически сойдут на нет. Поэтому теперь можно посылать сигналы на тысячи километров без заметного искажения их формы. Такие сигналы, именуемые **солитонами (soliton)**, применяются все чаще.




**Илл. 2.5.** Затухание света при прохождении по оптоволокну в инфракрасном диапазоне

### *Оптоволоконные кабели*

Оптоволоконные кабели аналогичны коаксиальным, за исключением оплетки. На илл. 2.6 (а) показано одиночное волокно сбоку. В центре расположен стеклянный сердечник, через который распространяется свет. В многомодовых оптоволоконных кабелях диаметр сердечника обычно составляет около 50 мкм — это примерная толщина человеческого волоса. В одномодовых оптоволоконных кабелях диаметр сердечника составляет от 8 до 10 мкм.

**Илл. 2.6.** (а) Вид одиночного волокна сбоку. (б) Трехжильный кабель с торца

Сердечник окружен стеклянным покрытием с более низким, чем у сердечника, коэффициентом преломления. Таким образом, свет не выходит за пределы сердечника. Далее следует тонкая пластиковая оболочка, защищающая стеклянное покрытие. Оптические волокна обычно группируются по несколько штук и защищаются внешней оболочкой. На илл. 2.6 (б) представлен кабель с тремя волокнами.

Наземные линии оптоволоконных кабелей обычно укладываются в земле на глубине до метра, где их иногда повреждают экскаваторы или грызуны. У побережья трансокеанские оптоволоконные кабели укладываются в специальные желоба с помощью своего рода морского плуга. На глубоководье они просто лежат на дне, где иногда получают повреждения от рыболовных траулеров или подвергаются атакам гигантских кальмаров.

Оптоволоконные кабели могут соединяться тремя различными способами. Во-первых, они могут оканчиваться коннекторами и включаться в оптические розетки. На коннекторах теряется от 10 до 20 % света, зато упрощается изменение конфигурации системы. Во-вторых, они могут сращиваться механически: два кабеля с аккуратными срезами укладываются вместе в специальную соединительную втулку и фиксируются на месте. Для лучшего выравнивания через точку сопряжения пропускается свет и производятся небольшие сдвиги для поиска максимально сильного сигнала. Механическое сращивание занимает у квалифицированного специалиста примерно 5 минут, в результате чего потери света составляют около 10 %. В-третьих, можно произвести сварку (сплавление) двух кусков оптоволокна в один. Сваренный кабель почти ничем не хуже целого, однако небольшое затухание происходит даже в этом случае. При всех трех видах соединений в точке стыковки свет может отражаться, а отраженная энергия создает помехи сигналу.

Для генерации световых сигналов обычно используются две разновидности источников света: светодиоды (Light Emitting Diodes, LED) и полупроводниковые лазеры. Их свойства, как показано на илл. 2.7, различны. Длину волны можно варьировать путем вставки между источником света и оптоволокном интерферометра Фабри — Перо (Fabry — Perot) или интерферометра Маха — Цендера (Mach — Zehnder). Интерферометр Фабри — Перо представляет собой простой резонатор, состоящий из двух параллельных зеркал. Свет падает

перпендикулярно зеркалам. Длины волн, которые укладываются внутри резонирующей полости целое число раз, исключаются. Интерферометр Маха – Цендера разделяет свет на два луча, которые проходят немного разное расстояние. На выходе они снова объединяются, причем в фазе окажутся только лучи с определенными длинами волн.

| Характеристика | LED | Полупроводниковые лазеры |
|---|---|---|
| Скорость передачи данных | Низкая | Высокая |
| Тип оптоволокна | Многомодовое | Многомодовое или одномодовое |
| Расстояние | Короткое | Длинное |
| Срок службы | Долгий | Короткий |
| Чувствительность к температуре | Незначительная | Существенная |
| Стоимость | Низкая | Высокая |

**Илл. 2.7.** Сравнение полупроводниковых диодов и светодиодов как источников света

Принимающая сторона оптоволоконного кабеля представляет собой фотодиод, генерирующий электрический импульс, когда на него попадает свет. Время реакции фотодиодов, преобразующих оптический сигнал в электрический, ограничивает скорость передачи данных примерно до 100 Гбит/с. Тепловые помехи также являются проблемой, поэтому световой импульс должен быть достаточно мощным, чтобы его можно было уловить. Усиливая мощность излучения световых импульсов, можно радикально снизить количество ошибок передачи данных.

### *Сравнение оптоволокна и медных проводов*

Любопытно сравнить оптоволокно и медные провода. Преимуществ у оптоволокна немало. Для начала, полоса пропускания у него намного шире, чем у медного кабеля. Одного этого достаточно, чтобы оправдать его использование в высокоскоростных сетях. Благодаря слабому затуханию требуется только один повторитель на каждые 50 км междугородних линий, что позволяет сэкономить немалые средства, в то время как для медных проводов повторитель необходим каждые 5 км. На оптоволоконные кабели не влияют скачки напряжения, электромагнитные помехи и перебои в подаче электроэнергии. Также они не боятся коррозионных химических примесей в воздухе, что играет важную роль в суровых условиях на производстве.

Но что удивительно, телефонные компании любят оптоволоконные кабели совсем по другим причинам — они легкие и тонкие. Многие кабель-каналы давно заполнены, и новые провода туда не помещаются. Замена всех медных проводов на оптоволокно позволила бы освободить место, а медные провода можно выгодно сдать на переработку — медь в них отличного качества. Кроме

того, оптоволокно намного легче меди. Тысяча витых пар длиной в 1 км весит 8000 кг. Пара оптоволоконных кабелей с большей пропускной способностью весят всего 100 кг, что дает возможность отказаться от дорогостоящих систем механических опор. При построении новых маршрутов оптоволокно с легкостью выигрывает у медных проводов за счет гораздо более низкой стоимости прокладки. И наконец, оптоволокно не дает утечек света, а значит, затрудняет несанкционированные подключения. Это дает хорошую защиту от перехвата информации.

С другой стороны, оптоволокно — менее привычная технология, требующая специальных навыков, которыми обладают не все инженеры. При этом его очень легко повредить, просто слишком сильно изогнув. Кроме того, поскольку оптическая передача данных по своей природе является однонаправленной, то для двустороннего обмена данными необходимы два кабеля или две полосы частот в одном кабеле. Наконец, оптоволоконные блоки сопряжения дороже электрических. Тем не менее будущее всего стационарного обмена данными на длинных расстояниях, безусловно, за оптоволокном. Подробную информацию об оптоволоконных кабелях и сетях на их основе можно найти в работе Пирсона (Pearson, 2015).

## 2.2. БЕСПРОВОДНАЯ ПЕРЕДАЧА ДАННЫХ

На сегодняшний день множество людей использует беспроводную связь при работе с разными устройствами, от ноутбуков и смартфонов до «умных» часов и холодильников. Все эти устройства передают данные друг другу и конечным точкам сети по беспроводным каналам.

Следующие разделы посвящены общим вопросам беспроводной передачи данных. Для нее существует множество важных сценариев применения (помимо выхода в интернет для пользователей, желающих побродить по Всемирной паутине, лежа на пляже). Иногда беспроводная связь удобнее даже для стационарных устройств, например, если из-за рельефа местности (горы, джунгли, болота и т. п.) провести оптоволоконный кабель к зданию затруднительно. Современная беспроводная связь возникла в 1970-х благодаря проекту профессора Нормана Абрамсона (Norman Abramson) из Гавайского университета. Стоит отметить, что гавайских пользователей от вычислительных центров отделял Тихий океан, а телефонная система в то время оставляла желать лучшего. Мы обсудим проект Абрамсона, ALOHA, в главе 4.

### 2.2.1. Спектр электромагнитных волн

Электроны при движении создают электромагнитные волны, способные распространяться в пространстве (даже в вакууме). В 1865 году британский физик Джеймс Клерк Максвелл (James Clerk Maxwell) высказал гипотезу о существовании этих волн, а в 1887 году они впервые были зафиксированы немецким физиком Генрихом Герцем (Heinrich Hertz). Число колебаний волны в секунду, измеряемое в герцах, называется ее **частотой (frequency)**, *f*.

Расстояние между двумя последовательными максимумами (или минимумами) называется **длиной волны (wavelength)** и традиционно обозначается греческой буквой λ (лямбда).

Если к электрической цепи подключить антенну нужного размера, можно с успехом передавать электромагнитные волны на приемник, расположенный на некотором расстоянии. На этом принципе основана вся беспроводная связь.

В вакууме все электромагнитные волны перемещаются с одной скоростью, вне зависимости от их частоты. Эту скорость называют **скоростью света**, *c*. Она равна приблизительно $3\times10^8$ м/с, то есть около 1 фута (30 см) в наносекунду. (Можно было бы пересмотреть традицию и определить фут как расстояние, проходимое светом в вакууме за наносекунду, вместо того чтобы опираться на размер обуви давно умершего короля.) В медном проводе или оптоволокне скорость волн снижается до 2/3 этого значения и начинает в некоторой степени зависеть от частоты. Скорость света — абсолютный предел скорости во Вселенной. Никакой объект или сигнал не может перемещаться быстрее.

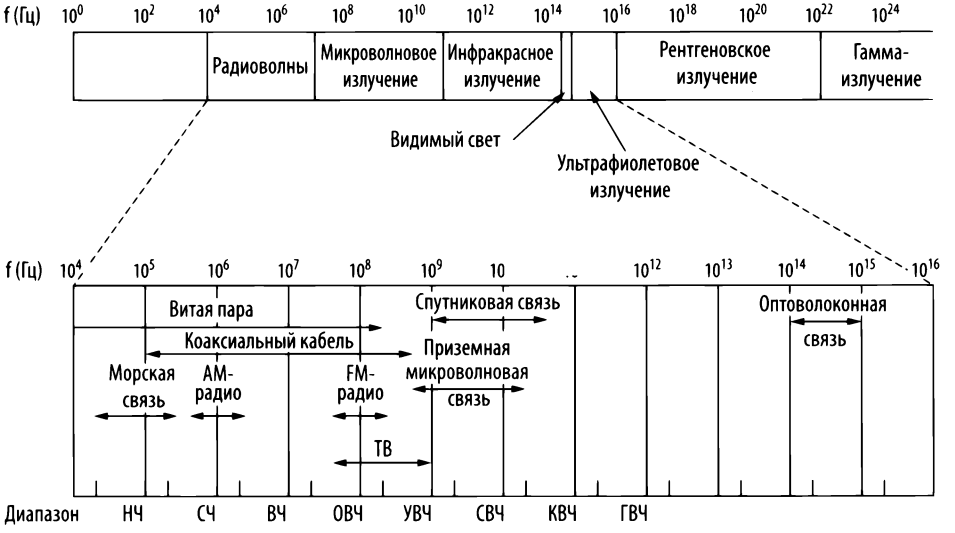
*f*, λ и *c* (в вакууме) связаны фундаментальным соотношением:

$$\lambda f = c. \tag{2.1}$$

Поскольку *c* — постоянная, то, зная *f*, можно найти λ, и наоборот. Эмпирическое правило: если λ измеряется в метрах, а *f* — в мегагерцах, то $\lambda f \approx 300$. Например, длина волн частотой 100 МГц составляет около 3 м, 1000 МГц — 0,3 м, а 3000 МГц — 0,1 м.

На илл. 2.8 приведен спектр электромагнитных волн. Для передачи данных путем модуляции амплитуды, частоты или фазы волн можно использовать радиоволны, микроволновое и инфракрасное излучение, а также видимый свет. Еще лучше для этой цели подошло бы ультрафиолетовое, рентгеновское и гамма-излучение благодаря более высокой частоте, но генерировать и модулировать их сложнее: они плохо проходят сквозь здания и, кроме того, опасны для всего живого.

Названия частотных диапазонов, представленные в нижней части илл. 2.8, являются официальными наименованиями Международного союза электросвязи (ITU). Они соответствуют длине волны: к примеру, низкочастотный диапазон охватывает длины волн от 1 до 10 км (примерно от 30 до 300 КГц). Сокращения НЧ (LF), СЧ (MF) и ВЧ (HF) означают низкие (low), средние (medium) и высокие частоты (high frequency) соответственно. Разумеется, при выборе названий никто не ожидал, что будут использоваться частоты выше 10 МГц. Поэтому последующие диапазоны получили названия ОВЧ, очень высокие частоты (VHF, very high frequency); УВЧ, ультравысокие частоты (UHF, ultra high frequency); СВЧ, сверхвысокие частоты (SHF, super high frequency); КВЧ, крайне высокие частоты (EHF, extremely high frequency), и ГВЧ, гипервысокие частоты (THF, tremendously high frequency). Наименования следующих диапазонов пока не придуманы, но как нам кажется, отлично подойдут «невероятно высокие», «поразительно высокие» и «чудовищно высокие» частоты (НВЧ, ПВЧ и ЧВЧ). Выше $10^{12}$ Гц начинается инфракрасное излучение, которое имеет смысл сравнивать с видимым светом, а не с радиоволнами.




**Илл. 2.8.** Спектр электромагнитных волн и их применение для электросвязи

Из теоретических основ электросвязи (изложенных далее в этой главе) известно, что количество информации, переносимой сигналом (например, электромагнитной волной), зависит от принимаемой мощности и пропорционально ширине полосы пропускания. Из илл. 2.8 становится ясно, почему разработчики сетей так любят оптоволокно. В микроволновом диапазоне для передачи данных доступна полоса пропускания на много гигагерц, но оптоволокно находится правее на логарифмической шкале, поэтому его показатели еще лучше. В качестве примера рассмотрим 1,30-микрометровый диапазон на илл. 2.5; ширина диапазона составляет 0,17 мкм. Воспользуемся уравнением (2.1) и вычислим начальную и конечную частоты на основе соответствующих длин волн. Диапазон составляет около 30 000 ГГц. При допустимом соотношении «сигнал/шум» в 10 дБ скорость будет равна 300 Тбит/с.

Большая часть данных передается в относительно узком диапазоне частот, другими словами, $\Delta f / f << 1$. Сигналы сосредоточиваются в узком диапазоне для более эффективного использования спектра и достижения хороших скоростей передачи за счет достаточной мощности. Далее мы опишем три типа передачи, при которых используются более широкие диапазоны частот.

## 2.2.2. Псевдослучайная перестройка рабочей частоты

При расширении спектра сигнала с **псевдослучайной перестройкой рабочей частоты (frequency hopping spread spectrum)** передатчик меняет частоту сотни раз в секунду. Этот метод широко используется в военной связи: такую передачу труднее засечь и практически невозможно заглушить. Помимо этого он снижает замирание сигналов, поскольку они движутся от источника к приемнику

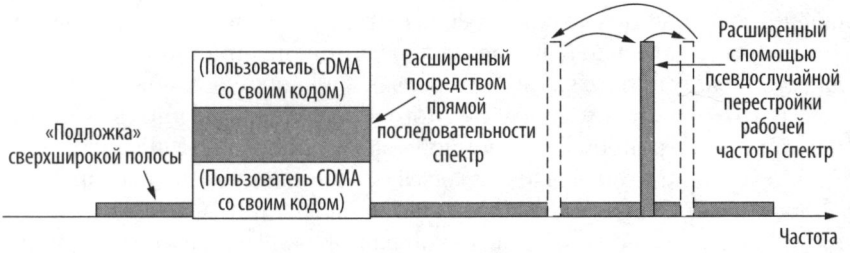
различными путями. Также он устойчив к узкополосным помехам, поскольку приемник не задерживается долго на проблемной частоте и связь не разрывается. Благодаря такой ошибкоустойчивости данный метод хорошо подходит для перегруженных частей спектра, таких как диапазоны ISM (мы расскажем о них чуть позднее). Псевдослучайная перестройка рабочей частоты также применяется в коммерческих системах, например в Bluetooth и в старых версиях стандарта 802.11.

Любопытно, что одним из изобретателей этой технологии стала австрийская и американская киноактриса Хеди Ламарр (Hedy Lamarr), снискавшая славу ролями в европейских фильмах в 1930-х годах под своим настоящим именем Хедвиг (Хеди) Кислер (Hedwig (Hedy) Kiesler). Ее первый супруг — богатый владелец оружейной фабрики — однажды рассказал ей, насколько легко блокировать радиосигналы управления торпедами. Обнаружив, что он продает вооружение Гитлеру, Хеди пришла в ужас. Переодевшись горничной, она сбежала в Голливуд, где продолжила карьеру актрисы. А в перерыве между съемками Хеди создала технологию перестройки рабочей частоты, чтобы помочь антигитлеровской коалиции.

В ее схеме использовалось 88 частот — по числу клавиш (и частот) фортепиано. Хеди Ламарр и ее друг, композитор Джордж Антейл (George Antheil), запатентовали изобретение (патент U.S. 2292387). Впрочем, им не удалось убедить ВМС США в практической ценности этой технологии, так что никаких выплат они так никогда и не получили. Лишь спустя многие годы после того, как патент утратил силу, эта методика была вновь открыта. Теперь она используется в мобильных электронных устройствах (вместо того, чтобы блокировать сигналы для торпед).

## 2.2.3. Метод прямой последовательности для расширения спектра

При расширении спектра **методом прямой последовательности (direct sequence spread spectrum)** информационный сигнал распределяется по более широкому диапазону частот с помощью кодовой последовательности. Данный метод обеспечивает эффективное совместное использование одной полосы частот несколькими сигналами, а потому широко применяется в промышленности. Сигналам присваиваются разные коды **методом множественного доступа с кодовым разделением каналов (code division multiple access, CDMA)**; его мы обсудим позже. На илл. 2.9 приводится сравнение двух методов расширения спектра сигнала (прямой последовательности и псевдослучайной перестройки рабочей частоты). На методе прямой последовательности основываются мобильные телефонные сети 3G. Кроме того, он применяется в GPS. Даже без различных кодов метод прямой последовательности не боится помех и замирания, поскольку теряется лишь небольшая доля полезного сигнала. В таком виде он применяется в старых версиях протокола беспроводных LAN 802.11b. Захватывающая история связи на основе расширения спектра подробно описана в работе Уолтерса (Walters, 2013).




**Илл. 2.9.** Расширение спектра и сверхширокополосная связь (UWB)

## 2.2.4. Сверхширокополосная связь

При использовании **сверхширокополосной связи (ultra-wideband, UWB)** происходит отправка ряда быстрых сигналов низкой мощности, а передача данных происходит за счет варьирования несущих частот. В результате быстрых переходов сигнал распределяется по очень широкой частотной полосе. К UWB относятся сигналы с полосой частот не менее 500 МГц либо занимающие как минимум 20 % от средней частоты их частотного диапазона. UWB-связь также показана на илл. 2.9. При такой ширине полосы частот скорость UWB-связи потенциально может достигать нескольких сотен мегабит в секунду. А поскольку сигнал распределен по широкому диапазону, ему не страшны довольно сильные помехи со стороны других сигналов с узкой полосой частот. Важный нюанс: поскольку при передаче данных на короткие расстояния мощность UWB-сигналов очень невелика, они не генерируют помех для вышеупомянутых узкополосных сигналов. В отличие от передачи данных при расширенном спектре, UWB-сигналы не мешают несущим сигналам в той же полосе частот. Их можно также использовать для просвечивания твердых объектов (земли, стен и тел) или в качестве составной части систем точного позиционирования. Эта технология нередко применяется для связи на коротких расстояниях в помещениях, а также для получения точных координат и отслеживания местоположения.

# 2.3. ПРИМЕНЕНИЕ СПЕКТРА ЭЛЕКТРОМАГНИТНЫХ ВОЛН ДЛЯ ПЕРЕДАЧИ ДАННЫХ

В этом разделе мы поговорим об использовании различных частей спектра электромагнитных волн, представленных на илл. 2.8, и начнем с радиоволн. Будем считать, что все передаваемые сигналы — узкополосные (если не указано иное).

## 2.3.1. Радиосвязь

Радиоволны легко генерировать, они способны преодолевать большие расстояния и с легкостью проходить сквозь стены. Поэтому их повсеместно используют для связи как в помещениях, так и на открытом пространстве. Радиоволны являются

всенаправленными, то есть расходятся во все стороны от источника, а значит, нет необходимости тщательно нацеливать передатчик на приемник.

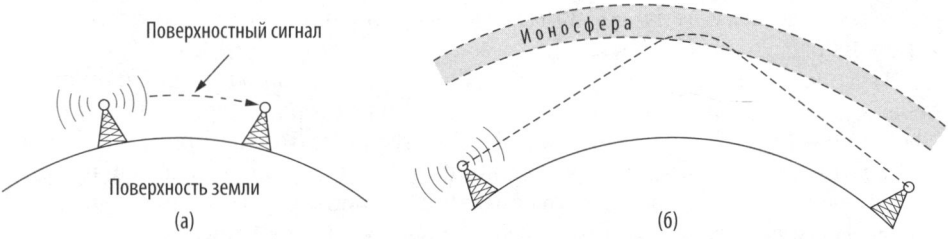
Иногда всенаправленность радиоволн полезна, но порой она может сыграть злую шутку. В 1970-х компания General Motors решила оборудовать все свои новые кадиллаки электронной антиблокировочной системой. При нажатии на педаль тормоза устройство подавало сигналы включения/выключения тормозов вместо их блокировки. Однажды дорожный патрульный штата Огайо попытался связаться с управлением по своей новенькой рации, и внезапно проезжающий мимо кадиллак стал вести себя как необъезженный жеребец. Когда офицер наконец остановил эту машину, водитель заявил, что ничего не делал и что автомобиль вдруг будто взбесился.

В конце концов начала прослеживаться закономерность: иногда кадиллаки «бунтовали», но только на крупных шоссе Огайо и только когда за ними наблюдал дорожный патруль. Долгое время в General Motors не могли понять, почему эта проблема не возникает во всех остальных штатах, а также на небольших дорогах Огайо. Только после тщательных исследований они обнаружили, что электропроводка кадиллака служит прекрасной антенной для частот, используемых новой радиосистемой дорожного патруля штата Огайо.

Свойства радиоволн зависят от частоты. Низкочастотные радиоволны легко проходят сквозь препятствия, но их мощность резко падает с удалением от источника — со скоростью минимум $1/r^2$ в воздухе, — поскольку энергия сигнала распределяется более тонким слоем по большей поверхности. Подобное затухание называется **потерями в тракте передачи (path loss)**. Высокочастотные радиоволны движутся по прямой и отражаются от препятствий. Эти отражения сильно влияют на мощность сигнала помимо уже упомянутых потерь в тракте передачи. Высокочастотные радиоволны сильнее поглощаются дождем и остальными препятствиями, чем низкочастотные. При этом радиоволны любой частоты подвержены помехам от моторов и прочего электрического оборудования.

Интересно сравнить затухание радиоволн и сигналов в направляющих средах передачи. В оптоволокне, коаксиальном кабеле и витой паре мощность сигнала падает на одинаковую долю за единицу расстояния, например, на 20 дБ за 100 м для витой пары. В случае радиоволн мощность сигнала падает на одинаковую долю при удвоении расстояния, например, в вакууме эта доля равна 6 дБ. Это означает, что радиоволны могут проходить большие расстояния, при этом основной проблемой являются взаимные помехи между пользователями. Поэтому правительства всех стран жестко регулируют использование радиопередатчиков (за несколькими исключениями, которые мы обсудим далее).

В диапазонах ОНЧ, НЧ и СЧ радиоволны следуют вдоль земной поверхности, как показано на илл. 2.10 (а). Прием этих волн возможен на расстоянии до 1000 км для низких частот и на меньшем — для чуть более высоких. Для АМ-радиовещания используется диапазон СЧ, именно поэтому поверхностный сигнал бостонских АМ-радиостанций не так-то просто услышать в Нью-Йорке. Радиоволны в этих диапазонах легко проникают сквозь здания, поэтому радио прекрасно работает в помещении. Основная проблема с использованием этих диапазонов частот для передачи данных — низкая полоса пропускания.




**Илл. 2.10.** (а) Радиоволны ОНЧ, НЧ и СЧ следуют вдоль земной поверхности. (б) Радиоволны ВЧ отражаются от ионосферы

В диапазонах ВЧ и ОВЧ поверхностный сигнал поглощается почвой. Впрочем, те волны, что достигают ионосферы — слоя заряженных частиц, окружающих нашу планету на высоте от 100 до 500 км, — отражаются от нее и попадают обратно на землю, как показано на илл. 2.10 (б). При определенных атмосферных условиях сигнал даже может отразиться туда и обратно несколько раз. Радиолюбители применяют диапазоны ВЧ и ОВЧ для переговоров на больших расстояниях. Эти диапазоны также используются военными.

## 2.3.2. Микроволновая связь

На частоте выше 100 МГц волны движутся практически по прямой, а значит, их можно сфокусировать в узкий пучок с помощью параболической антенны наподобие всем привычной спутниковой тарелки. Концентрация энергии сигнала в виде узкого луча позволяет добиться гораздо лучшего соотношения сигнал/шум. Но для этого нужно тщательно выровнять передающую и принимающую антенны относительно друг друга. Этот подход позволяет выстроить в ряд несколько передатчиков и направить их на расположенные друг за другом приемники. Направленность позволяет осуществлять связь без взаимных помех, конечно, при соблюдении минимального расстояния между объектами. До появления оптоволоконных кабелей именно на таких микроволнах десятилетиями основывалась вся система междугородних телефонных разговоров. На самом деле вся система MCI (одного из первых конкурентов AT&T, основанного после распада последней) была построена на базе микроволновой связи между разнесенными на десятки километров вышками. Этот факт отражен в самом названии упомянутой компании: MCI расшифровывается как Microwave Communications, Inc. — Корпорация «Микроволновая связь». С тех пор MCI уже успела перейти на оптоволокно и, пройдя через ряд корпоративных слияний и банкротств, стала частью Verizon.

Микроволны являются **направленными (directional)**. Они движутся по прямой, так что если вышки находятся слишком далеко друг от друга, на пути сигнала может оказаться земная поверхность (представьте себе канал связи Сиэтл — Амстердам). Поэтому необходимо периодически устанавливать повторители. Чем больше высота вышек, тем реже их можно ставить. Расстояние между повторителями примерно пропорционально квадратному корню их

высоты. К примеру, в случае 100-метровых вышек повторители могут располагаться на расстоянии в 80 км.

В отличие от радиоволн более низкой частоты, микроволны плохо проходят через здания. Кроме того, даже если пучок волн на стороне передатчика хорошо сфокусирован, он вполне может разойтись в пути. Некоторые волны могут отразиться от низших слоев атмосферы и поступить в приемник позже, чем прямые. Подобные волны могут попадать в приемник со сдвигом по фазе с прямой волной и тем самым гасить сигнал. Этот эффект носит название **многолучевого замирания (multipath fading)** и зачастую представляет собой серьезную проблему. Его степень зависит от погодных условий и частот волн. Некоторые операторы связи держат около 10 % каналов в качестве резерва на случай, если многолучевое замирание приведет к потере какой-нибудь полосы частот.

Потребность в высокоскоростной передаче данных заставляет операторов беспроводных сетей переходить на все более высокие частоты. Сейчас повсеместно применяются полосы частот до 10 ГГц, но в районе 4 ГГц возникает новая проблема: поглощение волн водой. Длина этих волн всего несколько сантиметров, и они гасятся дождем. Этот эффект пригодился бы, если бы кто-то решил построить под открытым небом огромную микроволновую печь для жарки пролетающих мимо птиц. Но для связи он представляет серьезную проблему. Как и в случае многолучевого замирания, единственное решение — отключать каналы, на пути которых идет дождь, и использовать запасные варианты.

Микроволновая связь так широко применяется — для междугородней и мобильной телефонной связи, телевидения и других целей, — что возникла серьезная нехватка частот. У нее есть несколько важных преимуществ перед оптоволокном. Главное, для этого вида связи не требуются права на прокладку кабелей. Достаточно купить по одному крошечному клочку земли на каждые 50 км и расставить на них микроволновые вышки — и можно полностью отказаться от использования телефонной системы. Именно благодаря этому MCI смогла так быстро выйти на рынок компаний междугородней телефонной связи. Компания Sprint (еще один конкурент распавшейся AT&T) пошла совсем другим путем. Ее основателем стала Южная Тихоокеанская железнодорожная компания, уже владевшая правами на значительные участки земли. Оптоволоконный кабель просто прокладывался вдоль железнодорожного полотна.

Стоимость микроволновой связи относительно невелика. Возведение двух примитивных вышек (это могут быть просто большие столбы с четырьмя тросами) и установка антенн может обойтись намного дешевле, чем проведение 50 км оптоволокна под землей в густонаселенном городе или, скажем, в горах. Даже аренда оптоволоконной сети у телефонной компании может сильнее ударить по карману, особенно если эта компания еще не полностью расплатилась за медный провод, от которого отказалась в пользу оптоволокна.

## 2.3.3. Передача данных в инфракрасном диапазоне

Инфракрасные волны без использования кабеля широко применяются для связи на короткие расстояния. Пульты дистанционного управления телевизорами, Blue-ray плеерами и стереосистемами используют связь в ИК-диапазоне. Они

относительно направленные, дешевые, а их производство не представляет сложности. Впрочем, есть у них и очень крупный недостаток: сигнал не проходит сквозь плотные объекты (попробуйте встать между пультом дистанционного управления и телевизором и проверьте, будет ли он работать). В целом по мере перехода от длинноволнового радио к видимому свету волны все больше напоминают световые и все меньше — радиоволны.

С другой стороны, неспособность инфракрасных волн проникать через сплошные стены — положительное качество. Ведь инфракрасная система в одной комнате не будет мешать аналогичной системе в соседних помещениях: вы не сможете управлять соседским телевизором с помощью своего пульта. Более того, благодаря этому свойству инфракрасные системы защищены от перехвата информации лучше, чем радио. Следовательно, для их работы не нужна государственная лицензия, в отличие от радиосистем, работающих за пределами ISM-диапазонов частот. В ограниченном объеме инфракрасная связь применяется и в настольных ПК, например, для подключения ноутбуков и принтеров по стандарту **IrDA (Infrared Data Association)**, но на рынке связи ее роль второстепенна.

## 2.3.4. Передача данных в видимом диапазоне

Беспроводная оптическая коммуникация, или **оптика свободного пространства (free-space optics)**, применялась на протяжении столетий. Пол Ревир (Paul Revere)[1] накануне своей знаменитой «Скачки» посылал двоичные световые сигналы (с помощью фонаря. — *Примеч. ред.*) из бостонской Старой Северной церкви. Сегодня оптическая коммуникация используется для соединения двух LAN: на крышах зданий, в которых они развернуты, устанавливаются лазеры. Лазерная связь по своей природе является однонаправленной, так что на каждой стороне нужен свой лазер и свой фотодатчик. Такая архитектура обеспечивает очень широкую полосу пропускания за небольшие деньги. При этом она достаточно хорошо защищена, ведь перехват узкого лазерного луча — непростая задача. Установка оборудования проста и, в отличие от передачи в микроволновом диапазоне, не требует лицензии от **Федеральной комиссии по связи (Federal Communications Commission, FCC)** в США и аналогичных правительственных органов в других странах.

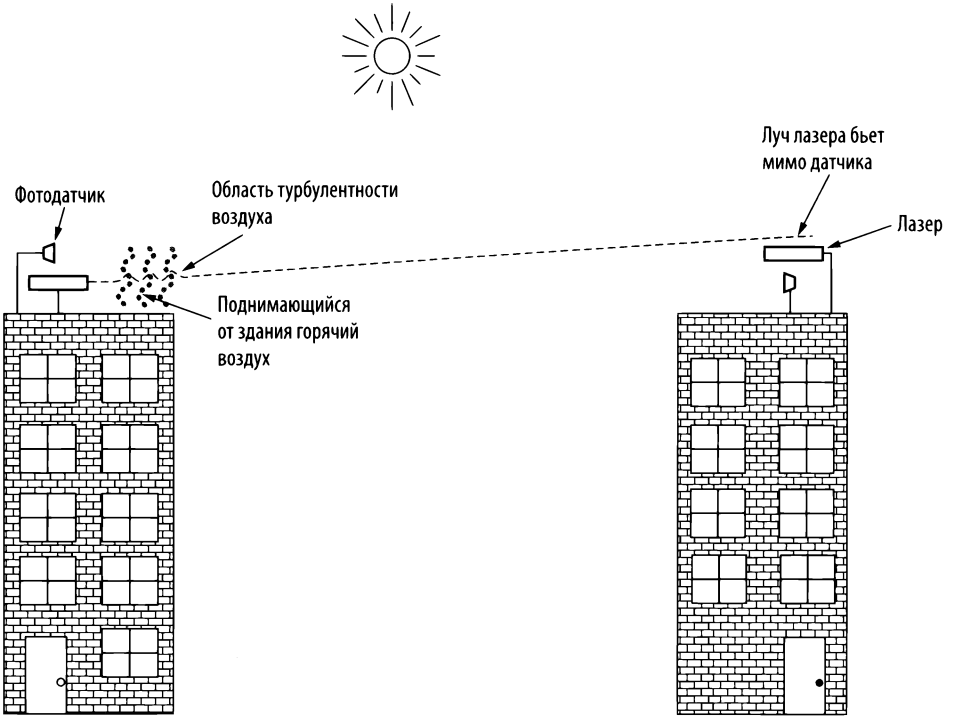
Главное преимущество лазера, узкий луч, одновременно является его недостатком. Попасть лучом миллиметровой ширины в цель размером с булавочную головку на расстоянии 500 м может разве что современная Энни Оукли (Annie Oakley)[2]. Обычно в систему включаются специальные линзы для небольшой расфокусировки луча. Проблема усугубляется тем, что луч подвержен искажению из-за ветра и температурных изменений, а дождь или сильный туман становятся непреодолимой преградой. При этом лазер прекрасно работает в солнечный

[1] Герой Войны за независимость США 1775–1783 гг. Прославился тем, что оповестил повстанцев о наступлении британских отрядов. — *Примеч. ред.*

[2] Легендарная американская женщина-стрелок, способная пулей погасить пламя свечи. — *Примеч. ред.*

день. Впрочем, эти факторы не имеют значения, если речь идет о соединении двух космических аппаратов.

Однажды в 1990-х один из авторов книги, Эндрю Таненбаум, посетил конференцию, проходившую в современном европейском отеле. Организаторы заботливо предоставили конференц-зал с терминалами, чтобы участники могли проверять свою электронную почту во время нудных докладов. Местная телефонная компания отказалась проводить множество телефонных линий ради трех дней конференции. Тогда организаторы поставили на крыше лазер и нацелили его на здание факультета вычислительной техники своего университета, расположенное в нескольких километрах. В ночь перед мероприятием связь работала отлично. Однако в 9 утра следующего дня — ясного и солнечного — канал полностью отказал и не работал до вечера. В оставшиеся два дня повторилась та же картина. Только после конференции организаторы выяснили причину: в дневное время из-за солнечного тепла с крыши здания поднимались конвекционные потоки воздуха (илл. 2.11). Эти турбулентные потоки отклоняли лазерный луч, в результате чего он плясал вокруг датчика. Подобный эффект можно наблюдать над шоссе в жаркий день. Мораль истории такова: чтобы беспроводные оптические каналы связи хорошо работали не только в идеальных, но и в сложных условиях, необходимо проектировать их с учетом возможных погрешностей.




**Илл. 2.11.** Конвекционные потоки воздуха мешают работе систем лазерной связи. На рисунке приведена двунаправленная система с двумя лазерами

Использование беспроводной оптической связи для построения сетей может показаться странной идеей, но у нее есть потенциал. В повседневной жизни нас окружают светочувствительные камеры и дисплеи, излучающие свет при помощи LED и других технологий. Эти устройства можно усовершенствовать, добавив возможность обмена данными. Информация будет зашифрована в паттернах мигания светодиодов за рамками восприятия человека. Передача данных при помощи видимого света вполне безопасна и создает низкоскоростную сеть в непосредственной близости от дисплея. Она позволяет повсеместно реализовывать самые разнообразные вычислительные сценарии. Проблесковые маячки на автомобилях экстренных служб могут сигнализировать ближайшим светофорам и машинам о необходимости уступить дорогу. Информационные табло могут транслировать карты. Даже праздничные гирлянды можно использовать для проигрывания музыки синхронно с миганием огоньков.

## 2.4. ОТ ФОРМ ВОЛН К БИТАМ

В этом разделе мы обсудим передачу сигналов по физическим средам, описанным выше. Начнем с теоретических основ обмена данными, а затем расскажем о модуляции (процессе преобразования аналоговых форм волн в биты) и мультиплексировании (с помощью которого одна физическая среда может служить проводником для передачи нескольких сигналов одновременно).

### 2.4.1. Теоретические основы обмена данными

Информацию можно передавать по проводам путем варьирования какой-либо физической величины, например напряжения или силы тока. Если представить значение напряжения или силы тока в виде однозначной функции времени $f(t)$, можно смоделировать поведение сигнала и проанализировать его математически. Этому анализу и посвящены следующие разделы.

#### *Гармонический анализ*

В начале XIX столетия французский математик Жан-Батист Фурье (Jean-Baptiste Fourier) доказал, что любую обычную периодическую функцию $g(t)$ с периодом $T$ можно представить в виде суммы ряда (возможно, бесконечного) синусов и косинусов:

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft), \qquad (2.2)$$

где $f = 1/T$ — частота основной гармоники, $a_n$ и $b_n$ — амплитуды $n$-х **гармоник** (членов ряда), а $c$ — константа, определяющая среднее значение функции. Подобное разложение называется **рядом Фурье (Fourier series)**. Функцию можно восстановить по ее ряду Фурье. Другими словами, зная период $T$ и амплитуды, можно восстановить исходную функцию времени путем вычисления сумм уравнения (2.2).

Можно представить, что информационный сигнал конечной длительности (а все информационные сигналы именно такие) повторяет весь паттерн снова и снова до бесконечности (то есть интервал от $T$ до $2T$ идентичен интервалу от 0 до $T$ и т. д.).

Вычислить амплитуды $a_n$ для любой заданной функции $g(t)$ можно путем умножения обеих сторон уравнения (2.2) на $\sin(2\pi kft)$ и взятия интеграла по отрезку от 0 до $T$. А поскольку

$$\int_0^T \sin(2\pi kft)\sin(2\pi nft)dt = \begin{cases} 0 \text{ при } k \neq n \\ T/2 \text{ при } k = n, \end{cases}$$

то остается только один из членов суммы: $a_n$. Сумма с коэффициентами $b_n$ исчезает полностью. Аналогично, умножив уравнение (2.2) на $\cos(2\pi kft)$ и взяв интеграл по отрезку от 0 до $T$, можно определить $b_n$. Чтобы найти $c$, достаточно проинтегрировать обе половины уравнения в его первоначальном виде. В результате этих операций получаем:

$$a_n = \frac{2}{T}\int_0^T g(t)\sin(2\pi nft)dt \quad b_n = \frac{2}{T}\int_0^T g(t)\cos(2\pi nft)dt \quad c = \frac{2}{T}\int_0^T g(t)dt.$$

### Сигналы с ограниченным диапазоном частот

Гармонический анализ можно применить к обмену данными, поскольку на практике каналы влияют на сигналы различной частоты по-разному. Рассмотрим конкретный пример: передачу ASCII-символа «b», закодированного в виде 8-битного числа. Передаваемая комбинация битов имеет вид 01100010. Слева на илл. 2.12 (а) показан выходной сигнал передающего устройства в виде напряжения электрического тока. При гармоническом разложении этого сигнала получаем следующие коэффициенты:

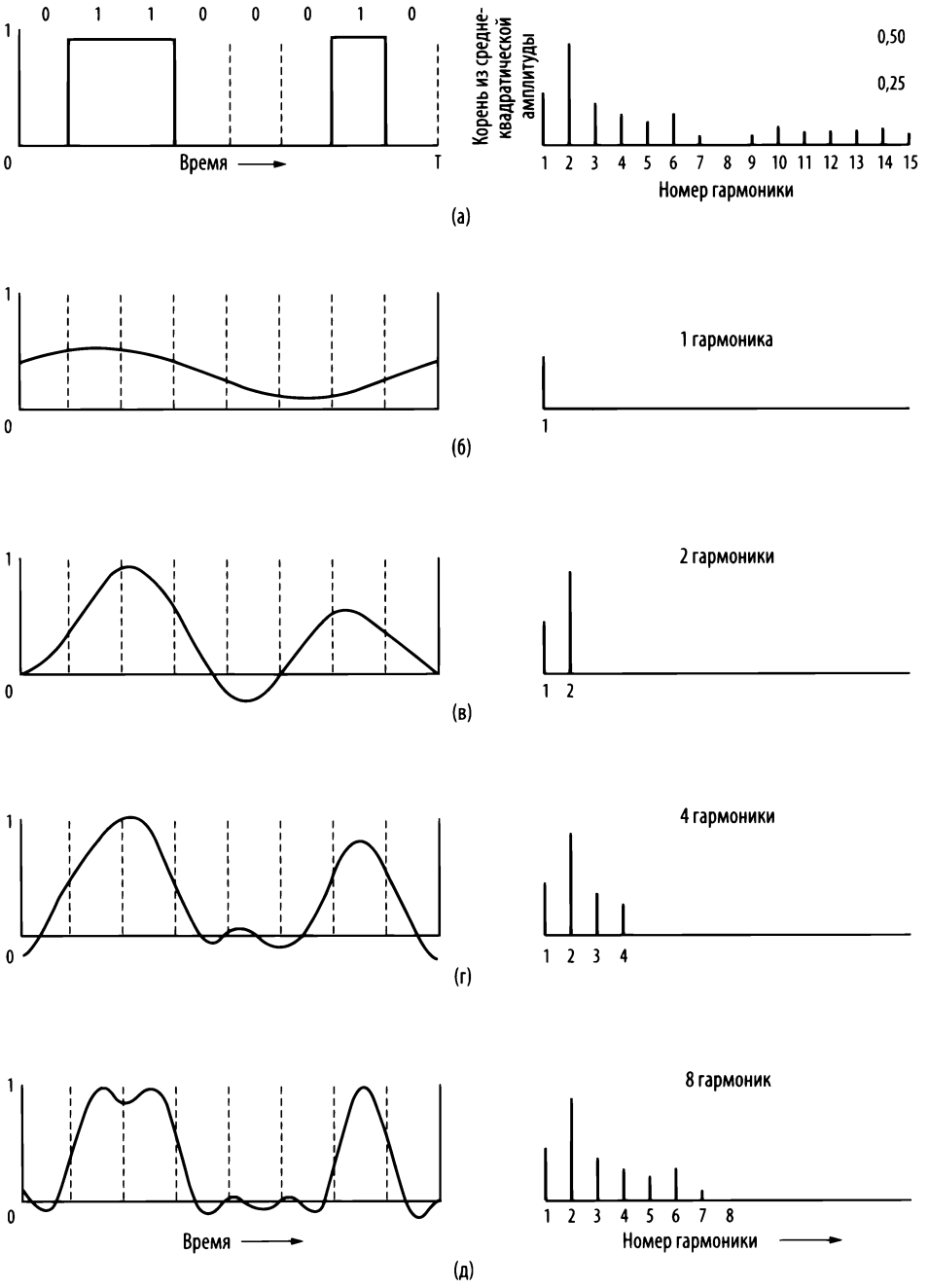
$$a_n = \frac{1}{\pi n}\left[\cos(\pi n/4) - \cos(3\pi n/4) + \cos(6\pi n/4) - \cos(7\pi n/4)\right];$$

$$b_n = \frac{1}{\pi n}\left[\sin(3\pi n/4) - \sin(\pi n/4) + \sin(7\pi n/4) - \sin(6\pi n/4)\right];$$

$$c = 3/4.$$

Корень из среднеквадратической амплитуды, $\sqrt{a_n^2 + b_n^2}$, для нескольких первых членов разложения приведен в правой части илл. 2.12 (а). Эти значения интересны тем, что их квадраты пропорциональны передаваемой на соответствующей частоте энергии.

Ни одно средство связи не может передавать сигналы без потери в процессе хотя бы небольшой доли мощности. Если уменьшить все гармоники Фурье в равной степени, амплитуда итогового сигнала уменьшится, но он не исказится; то есть он по-прежнему будет иметь аккуратную прямоугольную форму, как на илл. 2.12 (а). К сожалению, любое передающее оборудование уменьшает различные гармоники в разной степени, вследствие чего возникает искажение сигнала.




**Илл. 2.12.** (а) Бинарный сигнал и его среднеквадратичные амплитуды Фурье.
(б)–(д) Последовательные аппроксимации исходного сигнала

Обычно амплитуды передаются по проводам практически в неизменном виде от нуля до некой частоты $f_c$ (измеряемой в герцах), а все частоты сверх этой частоты среза ослабляются. Ширина диапазона частот, передаваемых практически без затухания, называется **шириной полосы пропускания**, или просто **пропускной способностью (bandwidth)**. На практике частота среза не настолько четко выражена, так что нередко упомянутая частота указывается в виде диапазона от 0 до частоты, на которой мощность полученного сигнала падает вдвое.

Пропускная способность — физическое свойство среды передачи, зависящее от конструкции, толщины, длины и материала провода или оптоволокна, а также других факторов. Для ее дальнейшего ограничения нередко применяются фильтры. Например, в беспроводных каналах 802.11 обычно используется диапазон в 20 МГц, поэтому радиоустройства, работающие по стандарту 802.11, фильтруют ширину полосы пропускания сигнала, чтобы привести ее к этим рамкам (хотя в некоторых случаях применяется диапазон в 80 МГц).

Приведем еще один пример: традиционные (аналоговые) телевизионные каналы (как проводные, так и беспроводные) занимают полосу в 6 МГц каждый. Такая фильтрация позволяет большему числу сигналов совместно использовать одну область спектра, что повышает общую эффективность системы. Это значит, что диапазон частот для некоторых сигналов начинается не с нуля, а с более высокого значения. Впрочем, это не важно. Полоса пропускания остается шириной диапазона переданных частот, а передаваемая информация зависит только от нее, а не от начальной и конечной частот. Сигналы, охватывающие частоты от 0 до максимальной частоты, называются **немодулированными (baseband signals)**[1]. А сигналы, смещенные по спектру на более широкий диапазон частот, как в случае всех проводных передач данных, называются **полосовыми сигналами (passband signals)**.

Теперь представим, как выглядел бы сигнал с илл. 2.12 (а), если бы его полоса пропускания была настолько узкой, что передавались бы только самые низкие частоты. Точнее, если бы функция аппроксимировалась первыми несколькими членами уравнения (2.2). На илл. 2.12 (б) показан сигнал, пришедший по каналу, пропускающему далее только первую гармонику (основную, $f$). Аналогично на илл. 2.12 (в)–(д) представлены спектр и восстановленные функции для каналов с большей полосой пропускания. В случае передачи цифровых данных нужно получить сигнал, достаточно достоверный для восстановления отправленной последовательности битов. Мы уже можем с легкостью это сделать (см. илл. 2.12 (д)), так что не имеет смысла использовать больше гармоник для повышения точности.

В нашем случае для отправки 8 бит (по 1 биту за раз) при скорости передачи данных в $b$ бит/с понадобится $8/b$ с. Таким образом, частота первой гармоники этого сигнала равна $b/8$ Гц. В обычных телефонных линиях, часто называемых **каналами передачи голоса (voice-grade line)**, искусственно производится срез на частоте чуть выше 3000 Гц. В результате этого ограничения номер высшей гармоники, проходящей по линии, приблизительно равен $3000/(b/8)$, то есть $24\,000/b$ (срез — плавный).

[1] Или «передаваемыми в основной полосе частот». — *Примеч. пер.*

Конкретные значения для некоторых скоростей передачи данных приведены на илл. 2.13. Из этих чисел понятно, что если отправить 9600 бит/с по каналу передачи голоса, график на илл. 2.12 (а) станет ближе к графику на илл. 2.12 (в). В итоге будет довольно сложно обеспечить четкий прием обычного бинарного битового потока. Очевидно, что при скоростях выше 38,4 Кбит/с использовать *бинарные* сигналы невозможно, даже если канал полностью свободен от шума. Другими словами, установление предела пропускной способности ограничивает скорость передачи данных, даже если каналы идеальны. Впрочем, существуют схемы кодирования с использованием нескольких уровней напряжения тока, за счет чего можно добиться более высоких скоростей. Мы обсудим эти схемы позже.

| Бит/с | T (мс) | Первая гармоника (Гц) | Число отправляемых гармоник |
|---|---|---|---|
| 300 | 26,67 | 37,5 | 80 |
| 600 | 13,33 | 75 | 40 |
| 1200 | 6,67 | 150 | 20 |
| 2400 | 3,33 | 300 | 10 |
| 4800 | 1,67 | 600 | 5 |
| 9600 | 0,83 | 1200 | 2 |
| 19 200 | 0,42 | 2400 | 1 |
| 38 400 | 0,21 | 4800 | 0 |

**Илл. 2.13.** Соотношение скорости передачи данных и гармоник в нашем простом примере

Термин «полоса пропускания» (bandwidth) часто вызывает путаницу, поскольку означает различные вещи для разных специалистов; она может быть аналоговой (для инженеров-электриков) и цифровой (для специалистов по вычислительной технике). Аналоговая полоса пропускания — это, как мы описывали выше, величина, измеряемая в герцах. Цифровая полоса пропускания — это максимальная скорость передачи данных канала, измеряемая в битах в секунду. Эта скорость — конечный результат использования аналоговой полосы пропускания физического канала для передачи цифровых данных. Таким образом, два этих понятия связаны между собой (как мы обсудим далее). В этой книге из контекста всегда будет ясно, о какой полосе пропускания идет речь: об аналоговой (Гц) или цифровой (бит/с).

## 2.4.2. Максимальная скорость передачи данных по каналу

Еще в 1924 году инженер компании AT&T Гарри Найквист (Harry Nyquist) осознал, что возможности передачи данных даже для идеального канала ограничены. Он вывел уравнение максимальной скорости передачи данных свободного от шумов канала с ограниченной полосой пропускания. В 1948 году Клод Шеннон

(Claude Shannon) развил идеи Найквиста и применил их к каналу со случайным (то есть термодинамическим) шумом (Shannon, 1948). Его исследование стало важнейшей научной работой в теории информации. Мы лишь кратко обобщим полученные Найквистом и Шенноном результаты, уже ставшие классическими.

Найквист доказал, что если произвольный сигнал проходит через низкочастотный фильтр с полосой пропускания $B$, то для его полного восстановления понадобится произвести $2B$ дискретных измерений в секунду. Нет смысла замерять сигнал чаще чем $2B$ раз в секунду, поскольку более высокочастотные компоненты, которые можно было бы восстановить на основе подобных измерений, были отфильтрованы ранее. Если сигнал состоит из $V$ дискретных уровней, то теорема Найквиста гласит:

$$\text{Максимальная скорость передачи данных} = 2B \log_2 V \text{ бит/с.} \quad (2.3)$$

Например, по свободному от шумов каналу с полосой пропускания 3 кГц нельзя передавать двоичные (то есть двухуровневые) сигналы со скоростью выше 6000 бит/с.

До сих пор мы говорили только о свободных от шумов каналах. При наличии случайного шума ситуация резко ухудшается. Из-за движения молекул в системе случайный (тепловой) шум присутствует всегда. Объем теплового шума измеряется в виде отношения мощности сигнала к мощности шума и называется **отношением сигнал/шум (Signal-to-Noise Ratio, SNR)**. Если обозначить мощность сигнала $S$, а мощность шума — $N$, то отношение сигнал/шум равно $S/N$. Обычно эта величина указывается на логарифмической шкале в виде $10 \log_{10} S/N$, поскольку может варьироваться в очень широких пределах. Единицы этой логарифмической шкалы названы **децибелами (дБ)** в честь Александра Грэхема Белла (Alexander Graham Bell), который первым запатентовал телефон. Отношение $S/N$ для 10 равно 10 дБ, 100 – 20 дБ, 1000 – 30 дБ и т. д. Производители стереоусилителей часто указывают полосу пропускания (частотный диапазон), в которой их аппаратура имеет линейную амплитудно-частотную характеристику с допуском в 3 дБ на каждой стороне. Этот допуск соответствует падению коэффициента усиления примерно в 2 раза (поскольку $10 \log_{10} 0{,}5 \approx -3$).

Основной результат, полученный Шенноном: максимальная скорость передачи данных (пропускная способность) зашумленного канала с полосой пропускания $B$ Гц и отношением сигнал/шум $S/N$ равна:

$$\text{Максимальная скорость передачи данных} = B \log_2 (1 + S/N) \text{ бит/с.} \quad (2.4)$$

С помощью этого уравнения можно вычислить наилучшую пропускную способность реального канала. Например, полоса пропускания ADSL (Asymmetric Digital Subscriber Line — асимметричная цифровая абонентская линия), используемой для доступа в интернет по обычной телефонной линии, равна примерно 1 МГц. На SNR сильно влияет расстояние от квартиры до АТС; для коротких каналов связи (от 1 до 2 км) SNR около 40 дБ считается очень неплохим. При таких характеристиках канал связи не может передавать со скоростью выше 13 Мбит/с, вне зависимости от того, сколько уровней сигнала используется, и вне

зависимости от частоты измерений. Первоначальные ADSL были рассчитаны на скорость до 12 Мбит/с, хотя пользователи иногда наблюдали более низкую скорость передачи. Подобная скорость передачи данных для того времени была очень неплохой, более чем 60 лет развития методик электросвязи существенно сократили разрыв между пропускной способностью по Шеннону и пропускной способностью реальных систем.

Полученный Шенноном результат основан на доказательствах теории информации и применим к любому каналу, подверженному тепловому шуму. К попыткам доказать обратное стоит относиться с тем же скепсисом, что и к вечным двигателям. Чтобы выйти за рамки 12 Мбит/с, ADSL необходимо либо улучшить SNR (например, с помощью установки цифровых ретрансляторов на линиях, поближе к пользователям), либо использовать большую полосу пропускания. Это и было сделано в ходе развития технологии в ADSL2+.

## 2.4.3. Цифровая модуляция

Изучив свойства проводных и беспроводных каналов, перейдем к отправке цифровой информации. Каналы связи передают аналоговые сигналы в виде непрерывно меняющегося напряжения, интенсивности света или звука. Чтобы переслать цифровую информацию, нужно придумать соответствующие битам аналоговые сигналы. Процесс преобразования битов в сигналы и наоборот называется **цифровой модуляцией (digital modulation)**.

Начнем со схем непосредственного преобразования битов в сигнал. При их использовании возникает **передача сигналов в базовой полосе (baseband transmission)**: сигнал занимает частоты от нуля до максимума, возможного при данной скорости передачи. Такой вариант часто применяется при проводной передаче данных. Далее рассмотрим схемы, изменяющие амплитуду, фазу или частоту несущего сигнала. Они приводят к **передаче сигналов в полосе пропускания (passband transmission)**: сигнал занимает полосу частот, близкую к частоте несущего сигнала. Обычно они применяются в беспроводных и оптоволоконных каналах, в которых сигналам отводятся конкретные полосы частот.

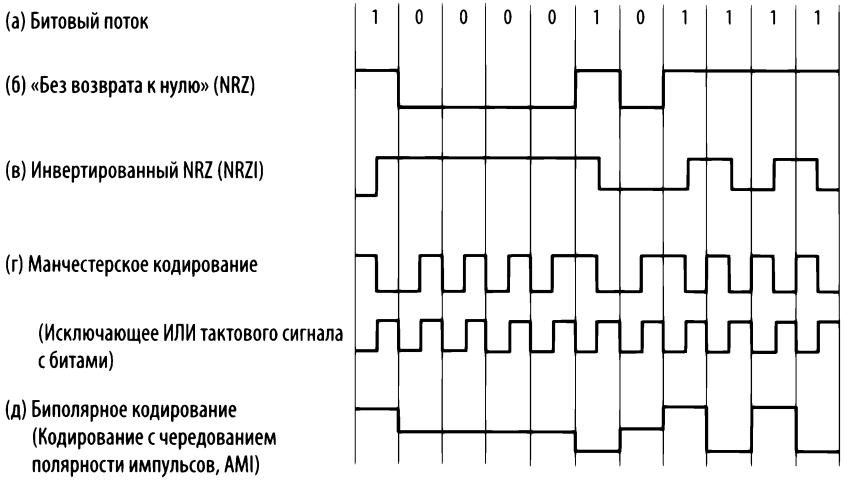
Часто по каналу передается ряд сигналов одновременно. В конце концов, намного удобнее использовать один провод для передачи нескольких сигналов, чем прокладывать отдельный провод для каждого. Этот подход называется **мультиплексированием (multiplexing)**. Реализовать его можно несколькими способами. Мы обсудим методы мультиплексирования по времени, по частоте и с кодовым разделением.

Методики модуляции и мультиплексирования, описанные в этом разделе, широко применяются для проводных, оптоволоконных, приземных беспроводных и спутниковых каналов связи.

### *Передача в базовой полосе*

В простейшем варианте цифровой модуляции 1 бит выражается положительным напряжением, а 0 бит – отрицательным, как показано на илл. 2.14 (а). В случае оптоволокна наличие светового импульса соответствует «1», а его

отсутствие — «0». Схема носит название **NRZ (Non-Return-to-Zero, «без возврата к нулю»)**. Это странное название возникло чисто по историческим причинам и всего лишь означает, что сигнал формируется в соответствии с данными. Пример приведен на илл. 2.14 (б).




**Илл. 2.14.** Линейные коды. (а) Биты. (б) NRZ. (в) NRZI. (г) Манчестерское кодирование. (д) Биполярное кодирование (AMI)

Отправленный сигнал NRZ проходит по проводу. На другой стороне приемник преобразует его в биты путем дискретизации сигнала через равномерные промежутки времени. Этот сигнал несколько отличается от исходного. Он ослабляется и искажается каналом и шумом на стороне приемника. Для декодирования битов приемник сопоставляет импульсные сигналы с ближайшими символами. Для NRZ положительное напряжение означает, что была отправлена «1», отрицательное — «0».

Метод NRZ — отличная отправная точка для изучения методов кодирования, поскольку он прост. Но на практике он применяется редко. Существуют более сложные схемы преобразования битов в сигналы, лучше отвечающие инженерным соображениям, — **линейные коды (line codes)**. Ниже описаны линейные коды, которые повышают эффективность полосы пропускания, а также обеспечивают восстановление синхронизации и баланс постоянного тока.

### *Эффективность полосы пропускания*

При использовании NRZ сигнал может перескакивать между положительным и отрицательным уровнями чуть ли не каждые 2 бита (когда единицы и нули меняют друг друга). Поэтому для скорости передачи данных в $B$ бит/с необходима полоса пропускания минимум $B/2$ Гц, как следует из уравнения Найквиста (2.3). Это фундаментальное ограничение, так что без дополнительной полосы пропускания NRZ не способна обеспечить большую скорость. Полоса

пропускания — зачастую ограниченный ресурс, даже в случае проводных каналов. Чем выше частота сигнала, тем сильнее его затухание и ниже эффективность. Кроме того, высокочастотные сигналы требуют более быстрой электроники.

Для более эффективного использования ограниченной полосы пропускания повышается число уровней сигнала (больше двух). Например, при четырех уровнях вольтажа можно отправлять два бита сразу в виде одного **символа**. Такая архитектура вполне работоспособна, если поступающий сигнал достаточно мощный для различения всех четырех уровней. Скорость изменения сигнала составляет половину битрейта, так что требуется меньшая полоса пропускания.

Скорость, с которой меняется сигнал, — это **скорость передачи символов (symbol rate)**. Необходимо отличать ее от **скорости передачи в битах**, или **битрейта (bit rate)**. Битрейт равен скорости передачи символов, умноженной на количество битов в символе. Ранее скорость передачи символов называлась **скоростью передачи в бодах**, или **бодрейтом (baud rate)**. Это понятие применялось в отношении работы телефонных модемов, передающих цифровые данные по телефонным линиям. В литературе термины «битрейт» и «бодрейт» часто путают.

Обратите внимание, что количество уровней сигнала не обязательно должно равняться степени двойки. Во многих случаях это не так, при этом часть уровней используется для защиты от ошибок и упрощения архитектуры приемника.

### *Восстановление тактового (синхронизационного) сигнала*

В любой схеме преобразования битов в символы приемник должен знать, где кончается один символ и начинается следующий, чтобы правильно декодировать биты. В NRZ символы представляют собой уровни напряжения, поэтому при длинной последовательности нулей или единиц сигнал остается неизменным. Рано или поздно становится сложно различать биты (ведь 15 нулей очень похожи на 16), разве что ваш синхросигнал чрезвычайно точен.

Точный синхросигнал позволяет решить проблему, но это слишком затратно для серийного производства. Учтите, что речь идет о синхронизации битов в каналах связи, работающих на скорости во много мегабит в секунду. Отклонение тактового сигнала более чем на долю микросекунды недопустимо даже на максимально длинном отрезке. Такое решение подходит только для медленных каналов связи или коротких сообщений.

Одна из возможных стратегий — отправка на приемник отдельного синхросигнала. Еще одна тактовая линия не проблема для компьютерных шин или коротких кабелей, и так содержащих множество параллельных линий связи. Но в большинстве сетевых подключений она станет напрасной тратой ресурсов — по дополнительной линии разумнее отправлять данные. Чтобы обойтись без нее, можно воспользоваться хитростью: соединить синхросигнал с информационным, применив к ним операцию XOR («исключающее ИЛИ»). Результат представлен на илл. 2.14 (г). Уровень тактового сигнала меняется при каждой передаче бита, поэтому тактовый генератор должен работать со скоростью, вдвое превышающей битрейт. Логический «0» кодируется (с помощью XOR) тактовым переходом с низкого уровня на высокий, то есть просто самим тактовым сигналом. А при

операции XOR с высоким уровнем он меняется на противоположный и происходит тактовый переход с высокого уровня на низкий. Этот переход соответствует логической «1». Такая схема, применявшаяся в классических сетях Ethernet, называется **манчестерским кодированием (Manchester encoding)**.

Недостаток этой схемы в том, что из-за тактового генератора полоса пропускания должна быть в два раза больше по сравнению с NRZ (а мы помним, что полоса пропускания — ценный ресурс). Еще одна стратегия — закодировать данные, обеспечив достаточное количество тактовых переходов в сигнале. Ведь у схемы NRZ возникают проблемы с восстановлением тактового сигнала только в случае длинных цепочек нулей и единиц. При частых тактовых переходах синхронизировать приемник с поступающим потоком символов будет несложно.

Для начала можно упростить задачу, закодировав «1» в виде тактового перехода, а «0» — в виде его отсутствия, либо наоборот. Это вариация NRZ — **инвертированный NRZ (Non-Return-to-Zero Inverted, NRZI)**. Пример NRZI приведен на илл. 2.14 (в). Он используется в популярном стандарте подключения периферийных устройств — USB (Universal Serial Bus, универсальная последовательная шина). При такой схеме длинные последовательности единиц не проблема.

Остается решить вопрос с длинными цепочками нулей. Телефонная компания могла бы потребовать от абонента не отправлять слишком много нулей подряд. В США старые цифровые телефонные каналы T1 (мы обсудим их далее) имели ограничение в 15 последовательных нулей. Чтобы действительно решить эту проблему, можно разбить очереди нулей. Для этого небольшие группы передаваемых битов сопоставляются так, чтобы цепочки нулей были представлены в виде чуть более длинных паттернов, в которых не так много последовательных нулей.

Популярный код, предназначенный для этой цели, — **4B/5B**. Каждой группе из 4 бит соответствует 5-битный паттерн на основе фиксированной таблицы преобразования. 5-битные паттерны подобраны таким образом, что последовательности из более чем 3 нулей невозможны. Таблица соответствий приведена на илл. 2.15. Избыточность схемы составляет 25 % — это намного лучше,

| Данные (4B) | Кодовое сочетание (5B) | Данные (4B) | Кодовое сочетание (5B) |
|---|---|---|---|
| 0000 | 11110 | 1000 | 10010 |
| 0001 | 01001 | 1001 | 10011 |
| 0010 | 10100 | 1010 | 10110 |
| 0011 | 10101 | 1011 | 10111 |
| 0100 | 01010 | 1100 | 11010 |
| 0101 | 01011 | 1101 | 11011 |
| 0110 | 01110 | 1110 | 11100 |
| 0111 | 01111 | 1111 | 11101 |

**Илл. 2.15.** Соответствие 4B/5B

чем в случае с манчестерским кодированием (100 %). А поскольку имеется 16 входных комбинаций битов и 32 выходных, часть выходных комбинаций не используется. Даже если отбросить сочетания со слишком длинными цепочками нулей, их останется немало. В качестве бонуса можно использовать эти коды в виде управляющих сигналов физического уровня. Например, иногда «11111» обозначает свободную линию, а «11000» — начало фрейма.

Еще один метод — **скремблирование (scrambling)** — состоит в видимой рандомизации данных. При этом подходе вероятность появления частых тактовых переходов очень высока. В основе работы **скремблера** лежит операция XOR с псевдослучайной последовательностью, которая применяется к данным до их передачи. В результате данные становятся столь же случайными, как эта последовательность (при этом они не зависят от нее). Для восстановления настоящих данных приемник применяет к входящему потоку XOR с той же псевдослучайной последовательностью. Чтобы все сработало, создание последовательности должно быть очень простым. Обычно ее задают в виде начального значения простого генератора случайных чисел.

Метод скремблирования хорош тем, что не требует избыточной полосы пропускания и дополнительного времени. К тому же он часто предотвращает попадание энергии сигнала в преобладающие гармоники, излучающие электромагнитные помехи (такие гармоники возникают из-за повторяющихся паттернов данных). Скремблирование полезно тем, что случайные сигналы обычно являются «белыми», то есть их энергия распределена по всем гармоникам.

Впрочем, скремблирование не гарантирует отсутствия длинных цепочек. Иногда просто не везет. Если данные полностью совпадают со случайной последовательностью, в результате операции XOR получаются сплошные нули. Такой исход маловероятен при длинной псевдослучайной последовательности, которую трудно предсказать. Но в случае коротких или легко предсказуемых последовательностей злоумышленники могут отправлять битовые паттерны, образующие после скремблирования длинные цепочки нулей. В результате происходит сбой связи. Подобным недостатком страдали ранние версии стандартов отправки IP-пакетов по каналам SONET в телефонной системе (см. Малис и Симпсон; Malis and Simpson, 1999). Пользователи могли отправлять определенные «пакеты-убийцы», которые гарантированно вызывали проблемы.

### *Симметричные сигналы*

Сигналы, в которых доля положительного напряжения равна доле отрицательного даже за короткий промежуток времени, называются **симметричными (balanced signals)**[1]. Их среднее значение равно нулю, а значит, в них отсутствует составляющая постоянного тока. Это является преимуществом, поскольку некоторые каналы связи (например, коаксиальный кабель и линии с трансформаторами) сильно ослабляют составляющую постоянного тока из-за их физических свойств. Кроме того, при подключении приемника к каналу связи методом **емкостного**

[1] В русскоязычной литературе встречается также название «балансный сигнал», особенно применительно к аудиоаппаратуре. — *Примеч. пер.*

**соединения (capacitive coupling)** передается только переменная составляющая тока. В любом случае при отправке сигнала с ненулевым средним значением только впустую тратится энергия, ведь составляющая постоянного тока будет отфильтрована.

Симметрирование кабеля обеспечивает тактовые переходы для синхросигналов благодаря сочетанию положительного и отрицательного напряжения. Также оно позволяет легко настраивать приемники, ведь среднее значение сигнала всегда можно измерить и использовать как порог решения для декодирования символов. Если сигналы несимметричны, среднее значение может отклоняться от истинного уровня принятия решения, например, из-за плотности единиц. Таким образом, большее число символов будет декодировано с ошибками.

Простейший способ реализации симметричного кода — использовать в качестве логической «1» и логического «0» два разных уровня напряжения. Например, +1 В для бита 1 и −1 В для бита 0. Для отправки «1» передатчик чередует уровни +1 В и −1 В, чтобы среднее значение всегда было нулевым. Это **биполярное кодирование (bipolar encoding)**. В телефонных сетях оно называется **кодированием с чередованием полярности (Alternate Mark Inversion, AMI)** в соответствии со старой терминологией, в которой «1» называлась «отметка» («mark»), а «0» — «пробел» («space»). Пример приведен на илл. 2.14 (д).

При биполярном кодировании добавляется еще один уровень напряжения, чтобы достигнуть баланса. Для этой цели также можно воспользоваться кодом, аналогичным 4B/5B (как и для получения тактовых переходов при восстановлении синхросигналов). Пример подобного симметричного кода — линейный код **8B/10B**. В нем 8 бит входного сигнала соотносится с 10 битами выходного, так что его КПД составляет 80 % (как и в случае 4B/5B). 8 бит разбиваются на две группы: из 5 бит (которые сопоставляются с 6 битами) и из 3 бит (сопоставляются с 4 битами). Далее 6-битный и 4-битный символы объединяются. В каждой группе некоторые входные паттерны можно соотнести с симметричными выходными паттернами с тем же числом нулей и единиц. Например, «001» соответствует симметричный паттерн «1001». Впрочем, возможных сочетаний недостаточно, чтобы все выходные паттерны были симметричными. В подобных случаях входной паттерн сопоставляется с двумя выходными, у одного из которых будет лишняя единица, а у второго — лишний ноль. Например, паттерн «000» ассоциируется с паттерном «1011» и дополнительным к нему паттерном «0100». При сопоставлении входных битов с выходными, кодировщик запоминает **дисбаланс (disparity)** предыдущего символа. Этот дисбаланс равен общему количеству нулей или единиц, которых сигналу не хватает до симметричности. Далее кодировщик выбирает либо выходной паттерн, либо дополнительный к нему для снижения дисбаланса. В случае кода 8B/10B максимальный дисбаланс равен 2 битам. Следовательно, сигнал никогда не будет сильно отличаться от симметричного. Также в нем не будет более пяти последовательных единиц или нулей, что удобно для восстановления синхросигнала.

### *Передача в полосе пропускания*

Передача сигналов в базовой полосе частот лучше всего подходит для проводной связи: по витой паре, коаксиальному или оптоволоконному кабелю. В других случаях (особенно в беспроводных сетях или радиосвязи) для передачи информации используется диапазон частот, не начинающийся с нуля. Отправлять сигналы очень низкой частоты по беспроводным каналам не имеет смысла, поскольку длина антенны должна составлять определенную долю длины волны сигнала. При низкочастотной передаче она окажется довольно большой. В любом случае выбор частот обычно диктуется нормативными ограничениями и желанием избежать помех. Даже при проводной передаче данных ограничение сигнала определенной полосой частот позволяет различным видам сигналов одновременно проходить по каналу. Этот процесс называется передачей в полосе пропускания, поскольку для нее используются произвольные полосы частот.

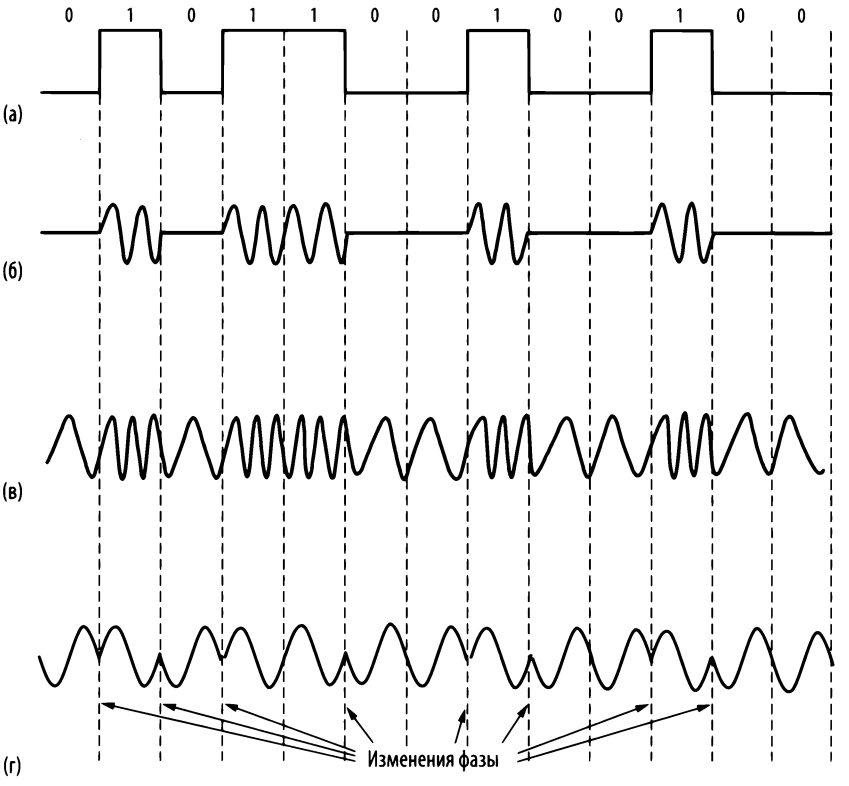
К счастью, все основные результаты, представленные в этой главе, сформулированы на языке пропускной способности, то есть *ширины* диапазона частот. Абсолютные величины частот не влияют на производительность. Это значит, что при сдвиге сигнала, занимающего основную полосу частот от 0 до $B$ Гц, на полосу частот от $S$ до $S+B$ Гц объем информации в сигнале не поменяется, хотя сам он будет выглядеть иначе. А чтобы обработать сигнал на приемнике, его можно сдвинуть назад до основной полосы частот, где удобнее находить символы.

При полосовой передаче сигналов цифровая модуляция производится путем модуляции несущего сигнала таким образом, чтобы он располагался в нужной полосе частот. Модулировать можно амплитуду, частоту и фазу несущего сигнала. У всех этих методов есть названия.

При **кодировании со сдвигом амплитуды (Amplitude Shift Keying, ASK)** «0» и «1» соответствуют две различные амплитуды. Пример с нулевым и ненулевым уровнями приведен на илл. 2.16 (б). Для кодирования символов из нескольких битов можно использовать более двух уровней.

Аналогично при **кодировании со сдвигом частоты (Frequency Shift Keying, FSK)** используется две или более различные тональности. В примере на илл. 2.16 (в) используются лишь две частоты. В простейшем варианте **кодирования со сдвигом фазы (Phase Shift Keying, PSK)** фаза несущей волны периодически смещается на 0 или 180 градусов на границе каждого символа. Этот вариант называется **двоичным кодированием со сдвигом фазы (Binary Phase Shift Keying, BPSK)**, поскольку фаз две. «Двоичный» тут относится к двум символам, а не к тому, что каждый символ соответствует двум битам. Пример этого кодирования приведен на илл. 2.16 (г). В усовершенствованной схеме, более эффективно использующей полосу пропускания, для передачи 2 бит информации на символ применяется четыре сдвига, например, на 45, 135, 225 и 315 градусов. Такая версия называется **квадратурным кодированием со сдвигом фазы (Quadrature Phase Shift Keying, QPSK)**.

Эти схемы можно сочетать между собой и использовать больше уровней для передачи большего числа битов на символ. Частоту и фазу нельзя модулировать одновременно, поскольку они связаны между собой: частота соответствует скорости изменения фазы по времени. Как правило, совместно модулируют

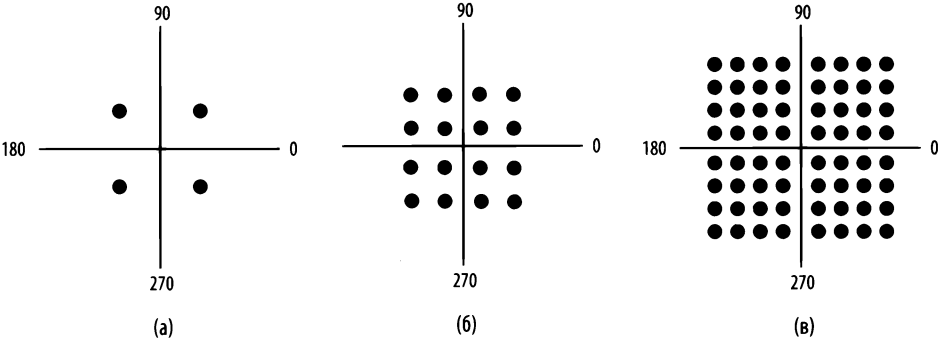



**Илл. 2.16.** (а) Бинарный сигнал. (б) Кодирование со сдвигом амплитуды. (в) Кодирование со сдвигом частоты. (г) Кодирование со сдвигом фазы

амплитуду и фазу. На илл. 2.17 приведены три примера. В каждом из них точки обозначают допустимые сочетания амплитуды и фазы для каждого символа. На илл. 2.17 (а) показаны равноудаленные точки на углах в 45, 135, 225 и 315 градусов. Фаза каждой точки соответствует углу между положительной частью оси $x$ и прямой, проведенной в эту точку из начала координат. Амплитуда каждой точки равна расстоянию от начала координат. Этот рисунок наглядно иллюстрирует схему QPSK.

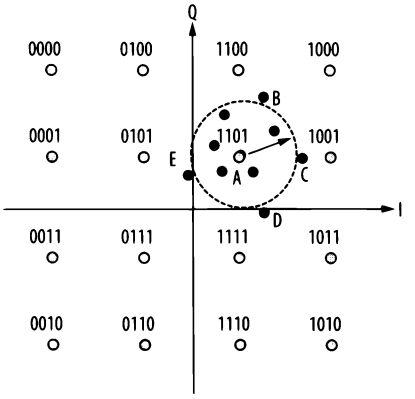
Подобные диаграммы называются **квадратурными (constellation diagram)**. На илл. 2.17 (б) показана схема модуляции с более плотным расположением точек. В ней используется 16 сочетаний амплитуд и фаз, так что эта схема модуляции пригодна для передачи 4 бит на символ. Она называется **QAM-16**, где QAM — **квадратурная модуляция амплитуды (Quadrature Amplitude Modulation)**. На илл. 2.17 (в) приведена еще более плотная схема модуляции, с 64 различными комбинациями, позволяющая передавать 6-битные символы — **QAM-64**. Существуют QAM еще более высокого порядка. Глядя на эти структуры, можно догадаться, что проще создать электронные схемы для генерации

символов в виде сочетания значений на различных осях координат, чем в виде комбинаций амплитуд и фаз. Именно поэтому приведенные паттерны напоминают квадраты, а не концентрические круги.




**Илл. 2.17.** (а) QPSK. (б) QAM-16. (в) QAM-64

Из приведенных выше диаграмм неясно, как именно биты распределяются по символам. При распределении важно позаботиться, чтобы небольшой всплеск шума на стороне приемника не привел к большому количеству ошибок в битах. Такое может случиться, если присвоить последовательные значения битов смежным символам. Допустим, в QAM-16 один символ означает 0111, а соседний — 1000; если приемник по ошибке выберет этот соседний символ, неправильными окажутся все биты. Лучше задать такие соответствия битов символам, чтобы соседние символы отличались только на 1 бит. Этот метод называется **кодом Грея (Gray code)**. На илл. 2.18 показан результат кодирования QAM-16 кодом Грея. Теперь если приемник ошибочно декодирует какой-либо символ, то ошибка будет только в одном бите, в случае если декодированный символ близок к переданному.




При отправке паттерна 1101:

| Точка | Декодируется как | Ошибки в битах |
|---|---|---|
| A | 1101 | 0 |
| B | 1100 | 1 |
| C | 1001 | 1 |
| D | 1111 | 1 |
| E | 0101 | 1 |

**Илл. 2.18.** QAM-16, закодированная кодом Грея

## 2.4.4. Мультиплексирование

Рассмотренные нами схемы модуляции позволяют отправлять цифровой сигнал по проводному или беспроводному каналу связи, но они описывают только передачу одного битового потока за раз. На практике при использовании сетей важную роль играет экономия ресурсов. Прокладка и сопровождение канала связи с широкой полосой пропускания между двумя офисами стоит столько же, сколько и с низкой (то есть затраты определяются стоимостью выкапывания желоба, а не тем, какой кабель в него укладывается). Поэтому были разработаны схемы мультиплексирования, позволяющие одновременно передавать по одному каналу связи много сигналов. Существует три главных способа мультиплексирования физического канала связи: по времени, по частоте и с кодовым разделением, а также мультиплексирование по длинам волн (по сути это оптическая разновидность мультиплексирования с частотным разделением каналов).

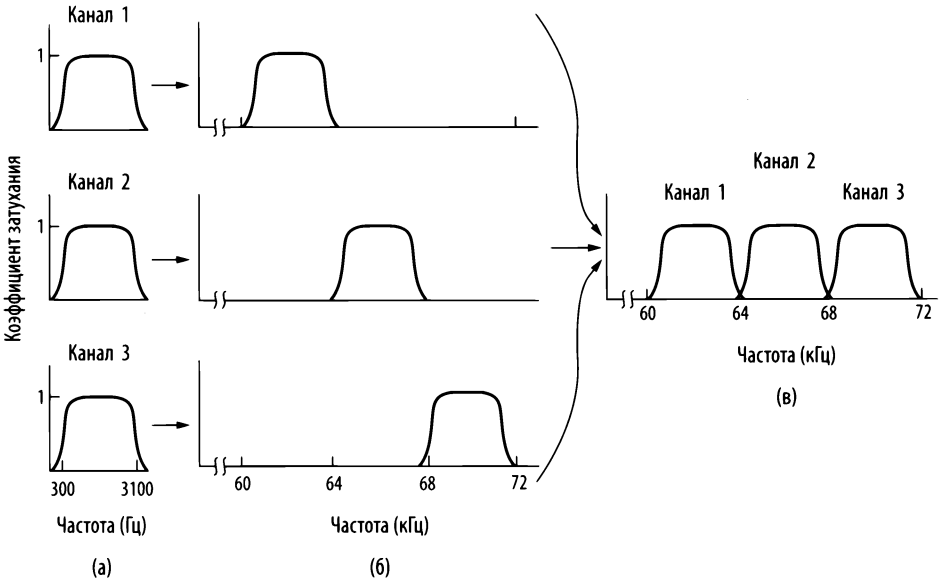
### *Мультиплексирование с частотным разделением каналов*

**Мультиплексирование с частотным разделением каналов (Frequency Division Multiplexing, FDM)** использует преимущества передачи в полосе пропускания. Спектр делится на диапазоны частот, и каждый пользователь получает эксклюзивный доступ к определенной полосе для отправки сигналов. AM-радиовещание хорошо иллюстрирует FDM. Выделенный на него спектр частот составляет около 1 МГц (примерно от 500 до 1500 кГц). Логическим каналам (станциям) выделяются разные частоты, и каждый из них работает только в своей части спектра. При этом каналы достаточно разделены между собой для предотвращения взаимных помех.

Приведем более подробный пример. На илл. 2.19 показаны три голосовых телефонных канала, мультиплексированных при помощи FDM. Доступная полоса пропускания ограничена фильтрами: примерно до 3100 Гц на каждый голосовой канал. При мультиплексировании нескольких каналов вместе на каждый выделяется полоса 4000 Гц. Дополнительная полоса пропускания называется **защитной полосой частот (guard band)** и служит для более надежного разделения каналов.

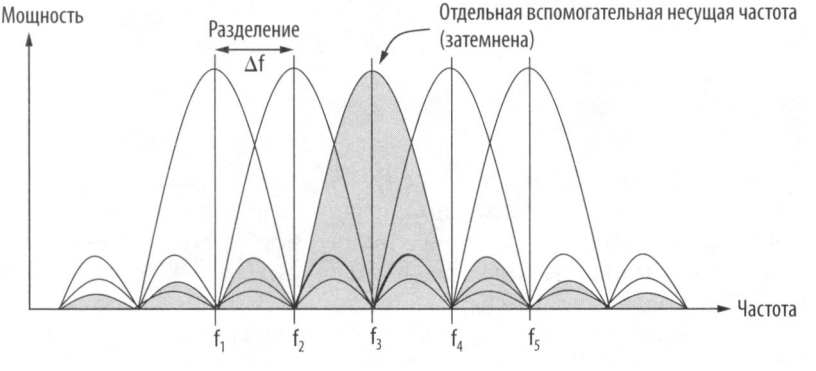
Сначала голосовые каналы поднимаются по частоте, каждый — на свою величину. Затем их можно объединить, поскольку теперь они все занимают различные части спектра частот. Несмотря на наличие промежутков между каналами благодаря защитным полосам, смежные каналы немного пересекаются. Это происходит потому, что на практике фильтры не производят четкого среза частот. А значит, сильный всплеск на границе одного канала будет ощущаться как нетепловой шум в смежном с ним.

Долгие годы подобная схема применялась для мультиплексирования телефонных звонков, но сейчас для этого чаще используется мультиплексирование по времени. Однако FDM по-прежнему встречается в телефонных системах, а также сотовых, приземных беспроводных и спутниковых сетях на более высоком уровне детализации.




**Илл. 2.19.** Мультиплексирование с частотным разделением каналов (FDM). (а) Исходные полосы частот. (б) Полосы, сдвинутые по частоте. (в) Мультиплексированный канал

При передаче цифровых данных спектр частот можно эффективно разбивать и без защитных полос. При **мультиплексировании с ортогональным частотным разделением каналов (Orthogonal Frequency Division Multiplexing, OFDM)** полоса делится на множество вспомогательных несущих частот с независимой передачей данных (например, с помощью схемы QAM). Эти частоты плотно упаковываются в диапазоне, поэтому их сигналы могут распространяться на смежные вспомогательные несущие. Впрочем, частотная характеристика каждой из них разработана так, чтобы в центре соседней частоты равняться нулю (илл. 2.20). Таким образом, вспомогательные несущие можно измерять в их



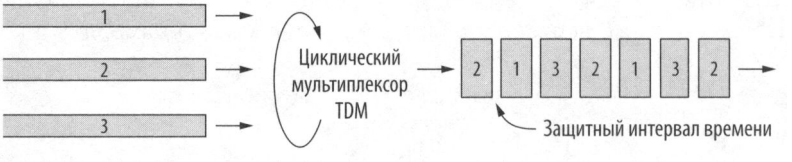

**Илл. 2.20.** Мультиплексирование с ортогональным частотным разделением каналов (OFDM)

центральных частотах, без опасения каких-либо помех. Чтобы метод сработал, необходим **защитный интервал времени (guard time)**. Нужно успеть повторить часть посылаемых символьных сигналов и добиться желаемой частотной характеристики. Однако эти издержки намного меньше, чем при большом числе защитных полос частот.

Метод OFDM существует уже давно, но начал активно применяться только в начале 2000-х. Тогда стало понятно, что можно эффективно реализовать OFDM в виде преобразования Фурье цифровых данных по всем вспомогательным несущим частотам (вместо того, чтобы модулировать по отдельности каждую такую частоту). OFDM используется в 802.11, сетях кабельного телевидения, сетях на основе ЛЭП и сотовых системах четвертого поколения (4G). Чаще всего один высокоскоростной поток цифровой информации разбивается на несколько низкоскоростных, и они параллельно передаются на вспомогательных несущих частотах. Такое разделение полезно, поскольку проблемы с ухудшением характеристик канала проще решать на уровне вспомогательных несущих: их можно заменить на более эффективные.

### *Мультиплексирование по времени*

Одна из альтернатив FDM — **мультиплексирование по времени (Time Division Multiplexing, TDM)**. Пользователям по очереди (циклически) предоставляется полная полоса пропускания на определенный интервал времени. Пример трех потоков данных, мультиплексированных при помощи TDM, приведен на илл. 2.21. Биты входных потоков забираются в фиксированный **временной слот** и выводятся в агрегирующий поток. Скорость этого потока равна сумме скоростей отдельных потоков. Для этого все потоки должны быть синхронизированы по времени. Чтобы приспособиться к небольшим временным флуктуациям, можно добавить маленькие защитные интервалы времени (аналогичные защитным полосам частот в FDM).




**Илл. 2.21.** Мультиплексирование по времени (TDM)

TDM широко применяется в качестве ключевого метода работы телефонных и сотовых сетей. Во избежание возможной путаницы сразу уточним, что он коренным образом отличается от альтернативного метода **мультиплексирования со статистическим разделением по времени (Statistical Time Division Multiplexing, STDM)**. Слово «статистический» здесь указывает, что отдельные потоки вносят свой вклад в общий поток *не* по фиксированному расписанию, а согласно статистике их потребностей. По сути, STDM представляет собой коммутацию пакетов, только под другим названием.

### *Мультиплексирование с кодовым разделением каналов*

Существует и третий вид мультиплексирования, работающий совершенно иначе, чем FDM и TDM. **Мультиплексирование с кодовым разделением каналов (Code Division Multiplexing, CDM)** представляет собой разновидность связи с расширением спектра, при которой узкополосный сигнал «размывается» по более широкой полосе частот. Благодаря этому он становится устойчивее к помехам. Кроме того, несколько сигналов от разных пользователей могут совместно использовать одну полосу частот. Именно для этого CDM чаще всего и используется, поэтому обычно его называют **множественным доступом с кодовым разделением каналов (Code Division Multiple Access, CDMA)**.

С помощью CDMA каждая из станций может все время вещать на всем спектре частот. Для разделения нескольких одновременных трансляций используется теория кодирования. Прежде чем углубиться в подробности алгоритма, рассмотрим аналогию: зал ожидания в аэропорту, множество беседующих между собой пар людей. Метод TDM соответствует тому, что пары разговаривают по очереди. FDM – пары разговаривают в разных тональностях: некоторые высоким голосом, некоторые – низким, благодаря чему все пары могут общаться независимо друг от друга. CDMA скорее напоминает вариант, когда все пары разговаривают одновременно, но на разных языках. Франкоговорящая пара болтает на французском, игнорируя звуки на прочих языках. Таким образом, главное в CDMA – извлечь нужный сигнал и отбросить все прочее как случайный шум. Ниже приведено несколько упрощенное описание CDMA.

В CDMA каждый интервал передачи бита разбивается на $m$ коротких интервалов, **элементарных сигналов (chips)**, комбинируемых с исходной последовательностью данных. Эти интервалы являются битовой последовательностью, но называются элементарными сигналами, чтобы не возникало путаницы с битами самого сообщения. Обычно на один бит приходится 64 или 128 элементарных сигналов, но в приведенном ниже примере используется 8 элементарных сигналов/бит, для простоты. Всем станциям назначаются уникальные $m$-битные коды – **последовательности элементарных сигналов (chip sequences)**. Чисто для удобства запишем эти коды в виде последовательностей –1 и +1. Мы будем указывать последовательности элементарных сигналов в круглых скобках.

Чтобы передать бит 1, станция передает свою последовательность элементарных сигналов. Для передачи бита 0 отправляется инвертированная последовательность элементарных сигналов. Другие паттерны не допускаются. При $m = 8$, если станции была назначена последовательность элементарных сигналов (–1 –1 –1 +1 +1 –1 +1 +1), она может отправить бит 1, передав свою последовательность элементарных сигналов или бит 0 путем передачи ее дополнения: (+1 +1 +1 –1 –1 +1 –1 –1). На самом деле отправляются уровни напряжения, но можно рассматривать их просто как последовательности.

Увеличение объема отправляемой каждой станцией информации с $b$ бит/с до $mb$ элементарных сигналов в секунду означает, что необходимая для CDMA полоса пропускания в $m$ раз больше, чем полоса, нужная для станции, не использующей CDMA (если считать, что схемы модуляции и кодирования одинаковы). Если 100 станциям доступна полоса частот в 1 МГц, при использовании

FDM каждая из них получит 10 кГц и сможет отправлять данные на скорости в 10 Кбит/с (из расчета 1 бит на 1 Гц). В CDMA каждая станция использует весь диапазон в 1 МГц, так что скорость передачи элементарных сигналов составит 100 сигналов на бит и доступная скорость передачи в 10 Кбит/с распределяется на весь канал.

На илл. 2.22 (а) и (б) показаны последовательности элементарных сигналов для четырех станций и соответствующих им сигналов. У каждой станции — своя уникальная последовательность. Будем использовать обозначение для вектора $m$ элементарных сигналов станции $S$ и $\bar{\mathbf{S}}$ для обратного к нему. Все последовательности попарно **ортогональны**, то есть нормализованное внутреннее произведение любых двух последовательностей **S** и **T** (записываемое в виде $\mathbf{S} \cdot \mathbf{T}$) равно нулю. Для генерации ортогональных последовательностей элементарных сигналов существует **код Уолша (Walsh code)**[1]. На более строгом математическом языке их ортогональность выражается следующим образом:

$$\mathbf{S} \cdot \mathbf{T} \equiv \frac{1}{m} \sum_{i=1}^{m} S_i T_i = 0. \tag{2.5}$$

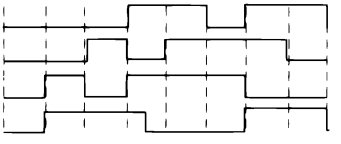
То есть все пары различны. Свойство ортогональности сыграет важную роль в дальнейшем. Обратите внимание, что если $\mathbf{S} \cdot \mathbf{T} = 0$, то и $\mathbf{S} \cdot \bar{\mathbf{T}} = 0$. Нормализованное внутреннее произведение любой последовательности элементарных сигналов с самой собой равно 1:

$$\mathbf{S} \cdot \mathbf{S} = \frac{1}{m} \sum_{i=1}^{m} S_i S_i = \frac{1}{m} \sum_{i=1}^{m} S_i^2 = \frac{1}{m} \sum_{i=1}^{m} (\pm 1)^2 = 1.$$

Это свойство следует из того, что поскольку каждый из $m$ членов внутреннего произведения равен 1, то их сумма равна $m$. Отметим, что $\mathbf{S} \cdot \bar{\mathbf{S}} = -1$.

A = (–1 –1 –1 +1 +1 –1 +1 +1)
B = (–1 –1 +1 –1 +1 +1 +1 –1)
C = (–1 +1 –1 +1 +1 +1 –1 –1)
D = (–1 +1 –1 –1 –1 –1 +1 –1)

(а)



(б)

$S_1$ = C = (–1 +1 –1 +1 +1 +1 –1 –1)
$S_2$ = B+C = (–2 0 0 0 +2 +2 0 –2)
$S_3$ = A+$\bar{B}$ = ( 0 0 –2 +2 0 –2 0 +2)
$S_4$ = A+$\bar{B}$+C = (–1 +1 –3 +3 +1 –1 –1 +1)
$S_5$ = A+B+C+D = (–4 0 –2 0 +2 0 +2 –2)
$S_6$ = A+B+$\bar{C}$+D = (–2 –2 0 –2 0 –2 +4 0)

(в)

$S_1 \cdot C$ = [1+1+1+1+1+1+1+1]/8 = 1
$S_2 \cdot C$ = [2+0+0+0+2+2+0+2]/8 = 1
$S_3 \cdot C$ = [0+0+2+2+0–2+0–2]/8 = 0
$S_4 \cdot C$ = [1+1+3+3+1–1+1–1]/8 = 1
$S_5 \cdot C$ = [4+0+2+0+2+0–2+2]/8 = 1
$S_6 \cdot C$ = [2–2+0–2+0–2–4+0]/8 = –1

(г)

**Илл. 2.22.** (а) Последовательности элементарных сигналов для четырех станций. (б) Соответствующие этим последовательностям сигналы. (в) Шесть примеров передачи данных. (г) Восстановление сигнала станции C

[1] Известен также под названием кода Адамара. — *Примеч. пер.*

На каждом интервале передачи бита станция может отправлять «1» (то есть свою последовательность элементарных сигналов) или «0» (обратную ей последовательность) либо может «промолчать» и не посылать ничего. Предположим, что все станции синхронизированы по времени, поэтому все последовательности элементарных сигналов отправляются в один момент. В случае одновременной передачи данных двумя или более станциями происходит линейное сложение их биполярных последовательностей. Например, если в одном интервале передачи бита три станции выдают на выходе +1, а одна выдает –1, будет получено +2. Можно рассматривать это как суперпозицию напряжений тока в канале для сигналов: три станции выдают на выходе +1 В, а одна выдает –1, так что приемник получит +2 В. Например, на илл. 2.22 (в) приведено шесть примеров одновременной передачи бита 1 одной или несколькими станциями. На первом примере станция *C* передает бит 1, так что мы получим только последовательность элементарных сигналов станции *C*. Во втором примере станции *B* и *C* передают бит 1, так что получается сумма их биполярных последовательностей элементарных сигналов, а именно:

$$(-1\ -1\ +1\ -1\ +1\ +1\ +1\ -1) + (-1\ +1\ -1\ +1\ +1\ +1\ -1\ -1) =$$
$$= (-2\ 0\ 0\ 0\ +2\ +2\ 0\ -2).$$

Чтобы восстановить битовый поток отдельной станции, приемник должен заранее знать последовательность ее элементарных сигналов. Приемник вычисляет нормализованное внутреннее произведение полученной последовательности элементарных сигналов и последовательности элементарных сигналов станции, чей битовый поток он пытается восстановить. Если получена последовательность элементарных сигналов **S**, а приемник настроен на прием станции с последовательностью элементарных сигналов **C**, ему достаточно будет вычислить нормализованное внутреннее произведение $\mathbf{S} \cdot \mathbf{C}$.

Чтобы понять работу этого метода, рассмотрим две станции, **A** и **C**. Они передают бит 1 в то же время, когда станция **B** передает бит 0, как в третьем примере. Приемник видит сумму $\mathbf{S} = \mathbf{A} + \bar{\mathbf{B}} + \mathbf{C}$ и вычисляет:

$$\mathbf{S} \cdot \mathbf{C} = \left(\mathbf{A} + \bar{\mathbf{B}} + \mathbf{C}\right) \cdot \mathbf{C} = \mathbf{A} \cdot \mathbf{C} + \bar{\mathbf{B}} \cdot \mathbf{C} + \mathbf{C} \cdot \mathbf{C} = 0 + 0 + 1 = 1.$$

Первые два члена уравнения равны нулю, поскольку заранее были тщательно отобраны ортогональные пары последовательностей элементарных сигналов, как показано в уравнении (2.5). Теперь вам должно быть ясно, зачем на последовательности элементарных сигналов было наложено такое ограничение.

На илл. 2.22 (г) представлены шесть примеров для наглядной демонстрации процесса декодирования. Допустим, приемник хочет извлечь из каждого сигнала с $S_1$ по $S_6$ бит, отправленный станцией *C*. Для этого он попарно суммирует произведения полученной последовательности **S** и вектора **C** из илл. 2.22 (а), а затем вычисляет 1/8 результата (поскольку в данном случае $m = 8$). Шесть примеров на илл. 2.22 (г) включают следующие случаи: станция *C* «молчит», отправляет бит 1 или бит 0, отдельно или в сочетании с отправкой других сигналов. Как видите, каждый раз декодируется правильный бит. Все равно что разговаривать по-французски.

Теоретически при достаточных вычислительных ресурсах приемник может «слушать» всех отправителей и выполнять алгоритм декодирования сигналов всех станций одновременно. В реальности это не так просто, к тому же важно знать, какая именно станция в данный момент передает информацию.

В вышеописанной идеальной (без помех) CDMA-системе число станций, параллельно отправляющих сигналы, может быть произвольно большим — просто за счет более длинных последовательностей элементарных сигналов. Для $2^n$ станций код Уолша позволяет сгенерировать $2^n$ ортогональных последовательностей элементарных сигналов длиной $2^n$. Но существует одно немаловажное ограничение: предполагается, что все элементарные сигналы синхронизируются по времени на приемнике. В некоторых приложениях, например сотовых сетях (в которых CDMA широко применялся с 1990-х), такой синхронизации быть не может. Поэтому разрабатываются различные архитектуры.

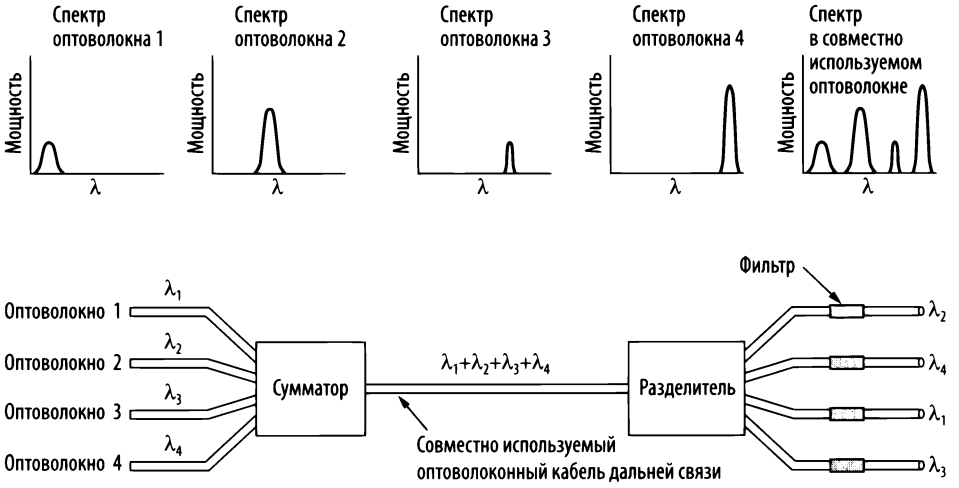
CDMA используется не только в сотовых, но также в спутниковых и кабельных сетях. В этом кратком обзоре мы обошли стороной множество его проблемных мест. Тем, кто хотел бы разобраться в технологии CDMA глубже, рекомендуем работы Витерби (Viterbi, 1995) и Харте и др. (Harte et al., 2012), правда, для чтения этих книг необходим изрядный опыт в сфере инженерии связи.

### *Мультиплексирование по длинам волн*

**Мультиплексирование по длинам волн (Wavelength Division Multiplexing, WDM)** — разновидность FDM, при которой несколько сигналов мультиплексируется в одном оптоволокне при помощи различных длин волн света. На илл. 2.23 четыре оптоволокна объединяются в оптическом сумматоре; энергия в каждом из них транслируется на своей длине волны. Четыре пучка света объединяются в одном общем оптоволокне для передачи в некую удаленную точку. На дальнем конце системы луч разделяется на исходное число оптических волокон. Сердечник каждого из них на выходе специально подбирается так, чтобы отфильтровывать все длины волн, кроме одной. Полученные в итоге сигналы можно направить в точку назначения или объединять различными способами для дальнейшей передачи с мультиплексированием.

В этом методе ничего нового для нас нет. Это просто FDM на очень высоких частотах, а термин WDM описывает оптоволоконные каналы через длины волн («цвета»), а не частоты. Для мультиплексирования каналов в оптоволоконном кабеле дальней связи достаточно выделить каждому каналу свой диапазон частот (то есть длин волн). При этом диапазоны не должны пересекаться. Единственное отличие от электрического FDM — в оптических системах используется полностью пассивная, а потому чрезвычайно надежная дифракционная решетка.

Причина популярности WDM в том, что энергия отдельного канала обычно распределяется по диапазону всего в несколько гигагерц, поскольку скорость преобразования электрических и оптических сигналов на сегодняшний день ограниченна. Благодаря параллельной работе нескольких каналов на различных длинах волн суммарная полоса пропускания растет линейно относительно числа каналов. А поскольку полоса пропускания отдельного оптоволокна составляет




**Илл. 2.23.** Мультиплексирование по длинам волн

около 25 000 ГГц (см. илл. 2.5), то теоретически в нем возможны 2500 10-гигабитных каналов, даже при 1 бит/Гц (возможна и большая скорость передачи данных).

Технология WDM развивалась с такой быстротой, что компьютерные технологии могли ей только позавидовать. Этот метод был изобретен около 1990 года. Первые доступные на рынке системы включали восемь каналов, по 2,5 Гбит/с на канал; к 1998 году появились и быстро нашли применение системы с 40 каналами по 2,5 Гбит/с; к 2006 году уже были продукты с 192 каналами по 10 Гбит/с и 64 каналами по 40 Гбит/с, способные передавать до 2,56 Тбит/с; в 2019 году существуют системы, работающие с 160 каналами и поддерживающие пропускную способность более чем 16 Тбит/с для отдельной волоконной пары. Это в 800 раз больше, чем пропускная способность систем 1990 года. Кроме того, каналы очень плотно размещаются в оптоволокне, их разделяет 200, 100 или даже всего 50 ГГц.

Сужение расстояния между ними до 12,5 ГГц позволяет использовать 320 каналов в одном оптоволокне, тем самым дополнительно повышая пропускную способность. Подобные системы с большим числом каналов и маленьким расстоянием между ними называются **плотными WDM (Dense WDM, DWDM)**. DWDM-системы обходятся дороже, поскольку из-за малых промежутков между каналами приходится поддерживать фиксированные длины волн и частоты. В результате подобные системы жестко контролируют параметры, чтобы гарантировать точность частот.

Одна из движущих сил технологии WDM — разработка полностью оптических компонентов. Ранее приходилось через каждые 100 км разделять каналы, по отдельности превращать оптические сигналы в электрические, чтобы усилить, а затем выполнять обратное преобразование и объединение. Сегодня полностью оптические усилители восстанавливают мощность сигнала всего лишь раз в 1000 км, не требуя многочисленных оптико-электрических преобразований.

В примере на илл. 2.23 длины волн системы фиксированы. Биты с входного оптоволокна 1 попадали в выходное оптоволокно 3, биты с входного оптоволокна 2 попадали в выходное оптоволокно 1, и т. д. Однако можно создать и WDM-системы с оптической коммутацией. В подобных устройствах выходные фильтры настраиваются с помощью интерферометров Фабри — Перо или Маха — Цендера. Эти устройства позволяют управляющему компьютеру динамически менять выбранные частоты, что делает систему гибкой. Благодаря этому она способна обеспечить множество путей по фиксированному набору оптоволоконных кабелей через телефонную систему на разных длинах волн. Больше информации об оптических сетях и WDM вы можете найти в работе Гроуба и Эйзелта (Grobe and Eiselt, 2013).

## 2.5. КОММУТИРУЕМАЯ ТЕЛЕФОННАЯ СЕТЬ ОБЩЕГО ПОЛЬЗОВАНИЯ

Для соединения двух расположенных рядом компьютеров проще всего использовать кабель. Именно так организованы локальные сети (Local Area Networks, LAN). Однако при значительных расстояниях, большом количестве подключаемых устройств или если кабель должен пересечь шоссе либо другой общественный участок затраты на прокладку собственных сетей совершенно неподъемны. Более того, в абсолютном большинстве стран мира закон запрещает прокладывать частные линии передачи по территории государственной собственности (или под ней). Следовательно, разработчикам сетей приходится использовать уже существующее оборудование связи, например телефонные или сотовые сети, а также сети кабельного телевидения.

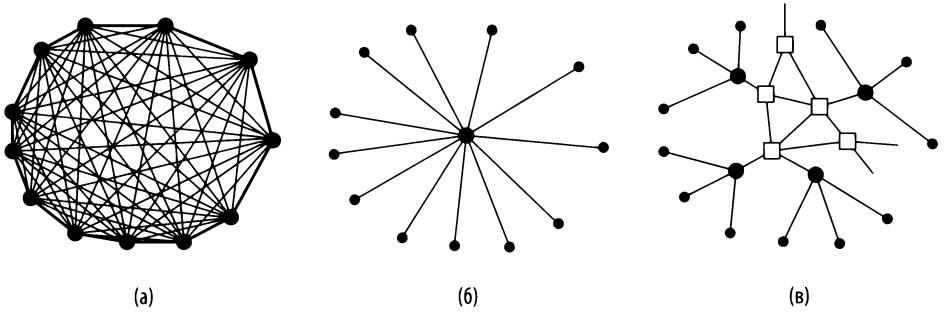
Долгое время основным фактором, ограничивающим сети обмена данными, был последний участок («последняя миля») перед потребителем. В его основе может лежать любая вышеупомянутая физическая технология, в противовес архитектуре так называемой опорной сети («backbone») в остальной сети доступа. За последнее десятилетие ситуация изменилась коренным образом, и скорость домашнего интернета 1 Гбит/с перестала быть чем-то необычным. Значительный вклад в это изменение внесли оптоволоконные кабели, все чаще развертываемые на границах сети. Но, вероятно, в некоторых странах еще более важную роль сыграли современные инженерные методы, применяемые в уже *существующих* телефонных и кабельных сетях для получения максимально широкой полосы пропускания в существующей инфраструктуре. Оказалось, что это намного дешевле, чем прокладка новых (оптоволоконных) кабелей к домам пользователей. Мы изучим различные архитектуры и характеристики всех этих физических инфраструктур связи.

Большинство существующих систем связи, особенно **коммутируемых телефонных сетей общего пользования (Public Switched Telephone Network, PSTN)**, было спроектировано много лет назад для совершенно иной цели: передачи человеческого голоса в более-менее узнаваемом виде. По кабелю, соединяющему два компьютера, можно передавать данные со скоростью в 10 Гбит/с

и более; таким образом, перед телефонной сетью ставится задача передачи битов на высоких скоростях. Первые технологии цифровых абонентских линий (DSL) позволяли передавать данные со скоростью не более нескольких мегабит в секунду; сегодня более современные DSL достигают 1 Гбит/с. В следующих разделах описывается устройство и функционирование телефонной системы. Дополнительную информацию можно найти в работе Лайно (Laino, 2017).

## 2.5.1. Структура телефонной системы

После получения Александром Грэхемом Беллом патента на телефон в 1876 году (всего на несколько часов раньше его конкурента Илайши Грея (Elisha Gray)) возник колоссальный спрос на его изобретение. Изначально на рынке продавались только телефоны, причем попарно. Протягивать провод между ними должен был сам абонент. А если владелец телефона хотел поговорить с $n$ владельцами других телефонов, приходилось тянуть отдельные провода во все $n$ домов. Всего через год города были покрыты проводами, беспорядочно опутывающими дома и деревья. Стало очевидно, что модель соединения всех телефонов попарно, приведенная на илл. 2.24 (а), не подходит.




**Илл. 2.24.** (а) Полносвязная сеть. (б) Централизованная коммутация. (в) Двухуровневая иерархия

Надо отдать Беллу должное: он быстро осознал проблему и создал компанию Bell Telephone. Первая коммутационная станция открылась в Нью-Хэйвене, штат Коннектикут, в 1878 году. От коммутатора тянулись провода в дома и офисы всех абонентов. Чтобы позвонить кому-то, абонент вращал рукоятку телефона, на коммутаторе раздавался звонок, и оператор вручную соединял звонившего с вызываемым абонентом с помощью короткого гибкого кабеля. Модель отдельного коммутатора показана на илл. 2.24 (б).

Вскоре коммутаторы Bell Telephone стали появляться повсюду как грибы после дождя. Люди захотели звонить из одного города в другой, так что Bell Telephone начали соединять коммутаторы между собой. И скоро столкнулись с той же проблемой: при соединении проводами всех коммутаторов попарно сеть быстро становилась очень запутанной. Поэтому были изобретены коммутаторы второго уровня. Через какое-то время потребовалось уже несколько

коммутаторов второго уровня, как показано на илл. 2.24 (в). В конце концов иерархия разрослась до пяти уровней.

К 1890 году телефонная система состояла из трех основных составляющих: коммутаторов; проводов, соединяющих абонентов с коммутаторами (теперь уже симметричных изолированных витых пар, а не голых проводов с землей в качестве обратного провода); и наконец, междугородних соединений коммутаторов. Техническая сторона истории телефонной системы кратко описана в работе Хоули (Hawley, 1991).

И хотя с тех пор все три составляющие претерпели значительные изменения, основная модель Bell Telephone через 100 лет осталась по существу такой же. Следующее описание, возможно, несколько упрощено, но позволяет понять, что к чему. Из каждого телефона выходит два провода, непосредственно ведущих в **оконечную телефонную станцию (end office)**, называемую также **местной центральной АТС (local central office)**. Расстояние между станциями обычно составляет от 1 до 10 км, причем в городах меньше, чем в сельской местности. Только в США насчитывается около 22 000 оконечных станций. Линия из двух проводов между телефоном абонента и оконечной станцией называется **абонентским шлейфом (local loop)**[1]. Если вытянуть в одну линию все локальные шлейфы в мире, они покрыли бы расстояние до Луны и обратно 1000 раз.

В какой-то момент 80 % капитала AT&T составляла медь в абонентских шлейфах. На тот момент AT&T была фактически крупнейшим добывающим медь предприятием в мире. К счастью, это было не слишком широко известно, иначе какой-нибудь агрессивный инвестор мог бы купить AT&T, закрыть весь телефонный бизнес в США, выкопать все провода и продать их скупщикам цветных металлов ради быстрой наживы.

Когда абонент, подключенный к местной АТС, звонит другому абоненту, подключенному к той же станции, механизм коммутации создает прямое электрическое соединение двух абонентских шлейфов, поддерживаемое на протяжении всего звонка.

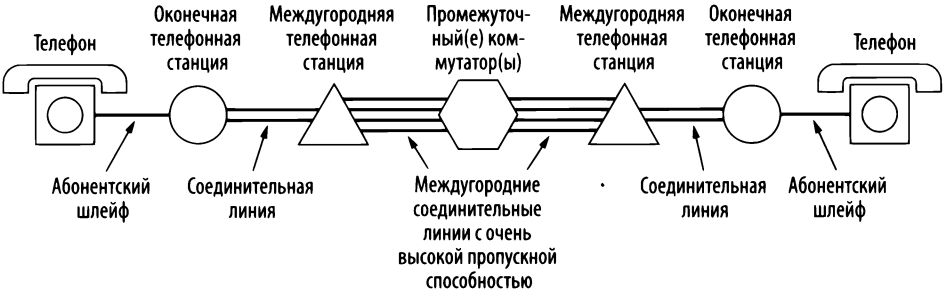
Если же вызываемый телефон подключен к другой станции, необходима иная процедура. У каждой АТС есть несколько исходящих каналов связи, ведущих к одному или нескольким соседним коммутаторам, называемым **междугородними телефонными станциями (toll office)** либо **транзитными (узловыми) телефонными станциями (tandem office)**, если они расположены в одном районе. Эти каналы называются **соединительными линиями (toll connecting trunk)**. Число различных видов коммутаторов и их топология в разных странах различаются и зависят от плотности телефонной сети.

Если линии АТС как вызывающего, так и вызываемого абонента ведут к одной междугородней станции (что весьма вероятно, если они расположены по соседству), то соединение можно произвести внутри этой станции. Телефонная сеть, состоящая лишь из телефонов (маленькие точки), оконечных телефонных

[1] Также встречаются названия «локальный шлейф», «локальная кольцевая линия» и др. — *Примеч. пер.*

станций (большие точки) и междугородних телефонных станций (квадраты), показана на илл. 2.24 (в).

Если соединительные линии АТС абонентов не ведут к одной междугородней телефонной станции, необходимо построить путь между двумя междугородними станциями. Они взаимодействуют друг с другом посредством высокоскоростных **междугородних соединительных линий (intertoll trunks, interoffice trunks)**. До распада AT&T в 1984 году в телефонной системе США для поиска такого пути применялась иерархическая маршрутизация с переходом на все более высокие уровни до тех пор, пока не будет найден общий коммутатор. Затем этот механизм сменила более гибкая неиерархическая маршрутизация. На илл. 2.25 показан механизм маршрутизации междугородних соединений.




**Илл. 2.25.** Типовой маршрут связи для междугородних звонков

Для связи используется множество различных сред передачи. В отличие от современных офисных зданий, куда обычно прокладываются кабели категории 5 или 6, абонентские шлейфы к жилым домам состоят в основном из витых пар категории 3 (хотя встречается и оптоволокно). А для соединения коммутаторов широко используются коаксиальные кабели, микроволны, а главным образом оптоволоконные кабели.

Раньше передача информации через телефонные системы была аналоговой, а сам голосовой сигнал передавался в виде электрического напряжения от источника в пункт назначения. С приходом оптоволоконных технологий, цифровой электроники и компьютеров все соединительные линии и коммутаторы стали цифровыми и единственной аналоговой частью системы остались абонентские шлейфы. Цифровая передача удобнее, поскольку не требует точного воспроизведения аналоговой формы волны, прошедшей через множество усилителей при междугороднем вызове. Достаточно безошибочно отличать 0 от 1. Благодаря этому свойству цифровая передача данных надежнее аналоговой. Кроме того, она дешевле и проще в обслуживании.

Итак, телефонная система состоит из трех основных компонентов:

1. Абонентские шлейфы (аналог витых пар между оконечными телефонными станциями и жилыми домами/офисами).
2. Соединительные линии (оптоволоконные цифровые каналы связи с очень высокой пропускной способностью, связывающие коммутаторы между собой).

3. Коммутаторы (в которых вызовы перенаправляются из одной соединительной линии в другую, электрически или оптически).

Абонентские шлейфы обеспечивают пользователям доступ к системе, а потому играют критически важную роль. К сожалению, в то же время это самое слабое звено в сети. Основная задача междугородних соединительных линий — сбор нескольких звонков и отправка их по одному и тому же оптоволоконному кабелю. Для этого применяется мультиплексирование по длинам волн (WDM). Кроме того, существует два принципиально разных способа коммутации: коммутация каналов и коммутация пакетов, которые мы рассмотрим далее.

## 2.5.2. Абонентские шлейфы: телефонные модемы, ADSL и оптоволокно

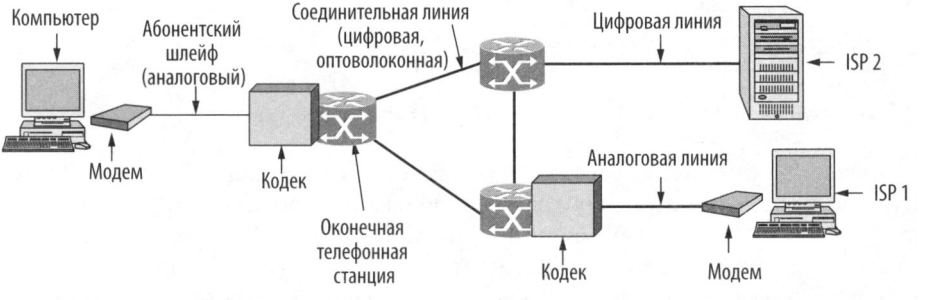
В этом разделе мы обсудим абонентские шлейфы как старого, так и нового образца. Мы расскажем о телефонных модемах, ADSL и технологии «оптоволокно в дом». В некоторых регионах абонентские шлейфы были модернизированы с применением технологии «оптоволокно в дом» (ну или «почти в дом»). Они обеспечивают работу компьютерных сетей с очень приличной пропускной способностью для сервисов передачи данных. К сожалению, прокладывать оптоволоконный кабель в жилые дома дорого. Иногда удается проложить кабель во время других коммунальных работ, требующих раскапывания улиц; абонентские шлейфы в некоторых регионах, особенно в густонаселенной городской местности, — оптоволоконные. Оптоволоконные абонентские шлейфы — редкость, хотя будущее, безусловно, за ними.

### *Телефонные модемы*

Большинству знакомы двухпроводные абонентские шлейфы, ведущие из оконечной телефонной станции в дома пользователей. Эти шлейфы часто называют последней милей, хотя фактическая их длина может составлять не одну, а несколько миль. Были приложены значительные усилия, чтобы выжать все возможное из уже существующих медных абонентских шлейфов. Телефонные модемы служат для обмена цифровой информацией между компьютерами по узкому каналу, предназначенному телефонной компанией для голосовых звонков. Когда-то модемы были распространены, но сегодня их почти везде заменили широкополосные технологии. Одна из таких технологий, ADSL, многократно использует абонентские шлейфы для отправки цифровых данных в АТС, откуда они попадают в интернет. При использовании модемов и ADSL приходится мириться с ограничениями старых абонентских шлейфов. Это относительно узкая полоса пропускания, неизбежное затухание и искажение сигналов, а также чувствительность к электрическим помехам, в частности перекрестным.

Для отправки битов через абонентский шлейф или любой другой физический канал необходимо преобразовать их в аналоговый сигнал. Это преобразование производится при помощи методов цифровой модуляции (см. раздел 2.4). А на другом конце линии аналоговый сигнал снова становится цифровым.

За преобразование потока цифровых битов в соответствующий им аналоговый сигнал и обратно отвечает **модем (modem)** — сокращение от «*мо*дулятор/*дем*одулятор». Модемы бывают самыми разными, включая телефонные, кабельные, беспроводные, а также DSL-модемы. Кабельные и DSL-модемы представляют собой отдельное аппаратное устройство, подключаемое между входящим в дом физическим каналом связи и остальной частью домашней сети. Беспроводные устройства обычно содержат встроенные модемы. Модем, что логично, размещается между (цифровым) компьютером и (аналоговой) телефонной системой, как показано на илл. 2.26.




**Илл. 2.26.** Применение аналоговой и цифровой передачи данных для связи между компьютерами. За преобразование отвечают модемы и кодеки

Телефонные модемы применяются для обмена битами между двумя компьютерами по каналам передачи голоса (вместо обычных разговоров). Основная проблема состоит в том, что эти каналы ограничены полосой пропускания 3100 Гц, вполне достаточной для передачи разговора. Эта полоса пропускания на четыре порядка меньше используемой для Ethernet или 802.11 (Wi-Fi). Неудивительно, что скорость телефонных модемов также на четыре порядка меньше, чем у Ethernet или 802.11.

Произведем вычисления и выясним, почему так происходит. Согласно теореме Найквиста, даже по идеальному каналу в 3000 Гц (которым телефонная линия явно не является) нет смысла отправлять символы со скоростью выше 6000 бод. Представим старый модем, который работает со скоростью 2400 символов/с (2400 бод) и стремится увеличить число битов на символ; при этом он передает данные в обоих направлениях (за счет использования разных частот для каждого направления).

В нашем скромном 2400-бит/с модеме логическому «0» соответствует 0 вольт, а логической «1» — 1 вольт, 1 бит на символ. Слегка усовершенствуем его: используем четыре разных символа, как в четырех фазах QPSK, что даст 2 бита на символ и позволит достичь скорости 4800 бит/с.

По мере развития технологий скорости все повышались и повышались. А большая скорость передачи данных требовала расширения набора символов (илл. 2.27). При большом числе символов даже незначительный шум в амплитуде или фазе полученного сигнала может привести к ошибке. Чтобы снизить

вероятность ошибок, в стандартах для более высокоскоростных модемов часть символов отводится на их коррекцию. Эти схемы известны под названием **треллис-модуляции (Trellis Coded Modulation, TCM)**[1]. На илл. 2.27 приведены несколько распространенных стандартов модемов.

| Стандарт модема | Бод | Бит/символ | Бит/с |
|---|---|---|---|
| V.32 | 2400 | 4 | 9600 |
| V.32 bis | 2400 | 6 | 14 400 |
| V.34 | 2400 | 12 | 28 800 |
| V.34 bis | 2400 | 14 | 33 600 |

**Илл. 2.27.** Некоторые стандарты модемов и их скорости передачи данных

Почему эта таблица заканчивается на 33 600 бит/с? Дело в том, что предел Шеннона, в соответствии со средней длиной и качеством абонентских шлейфов для телефонной системы, равен примерно 35 Кбит/с. Более высокая скорость либо пойдет вразрез с законами физики (а точнее, термодинамики), либо потребует замены абонентских шлейфов на новые (что постепенно и происходит).

Впрочем, ситуацию можно изменить. В АТС данные преобразуются в цифровую форму для отправки внутри телефонной сети (ядро сети уже давно перешло с аналоговой на цифровую передачу). Ограничение в 35 Кбит/с рассчитано для двух абонентских шлейфов, по одному с каждой стороны. Каждый шлейф добавляет в сигнал помехи. Если же как-нибудь избавиться от одного из них, можно повысить SNR и удвоить максимальную скорость.

Именно на основе такого подхода и работают 56-килобитные модемы. На одну сторону (обычно это ISP) подается цифровой поток высокого качества от ближайшей оконечной станции. Таким образом, благодаря высококачественному, как у большинства современных ISP, сигналу на одной стороне соединения общая скорость передачи данных составляет до 70 Кбит/с. Максимальная скорость передачи между двумя домашними пользователями с использованием модемов и аналоговых линий по-прежнему составляет 33,6 Кбит/с.

Почему же на практике используются 56-Кбит/с модемы, а не 70-Кбит/с? Причина — в теореме Найквиста. Телефонный канал осуществляет передачу данных внутри системы в виде цифровых измерений. Ширина диапазона частот каждого телефонного канала составляет 4000 Гц с учетом защитных полос частот. Таким образом, необходимо восстановить 8000 измерений в секунду. Число битов на измерение в Северной Америке — 8, из которых один используется в целях контроля; это позволяет передавать 56 000 бит/с пользовательских данных. В Европе пользователям доступны все 8 бит, поэтому теоретически можно

[1] Она же решетчатое кодирование, или решетчатая кодированная модуляция. — *Примеч. пер.*

было бы использовать 64 000-бит/с модемы, но ради единого международного стандарта была выбрана скорость в 56 000 бит/с.

В результате возникли стандарты модемов **V.90 и V.92**. Они предусматривают 56-килобитный входящий (от ISP к пользователю) канал передачи данных и 33,6- и 48-килобитные исходящие (от пользователя к ISP) каналы. Причина такой асимметрии в том, что от ISP пользователю поступает обычно больше информации, чем в обратном направлении. Вдобавок это позволяет выделить большую часть ограниченной полосы пропускания на нисходящий канал и повысить шансы на то, что скорость передачи данных действительно будет близка к 56 Кбит/с.

### *Цифровые абонентские линии (DSL)*

Когда телефонные компании наконец-то добрались до скорости в 56 Кбит/с, это был повод для гордости. В это же время провайдеры кабельного телевидения уже предлагали скорости до 10 Мбит/с. Предоставление интернет-доступа становилось все более важной частью бизнеса региональных телефонных компаний. Это заставило их задуматься о более конкурентоспособном продукте и предложить новые цифровые услуги по абонентскому шлейфу.

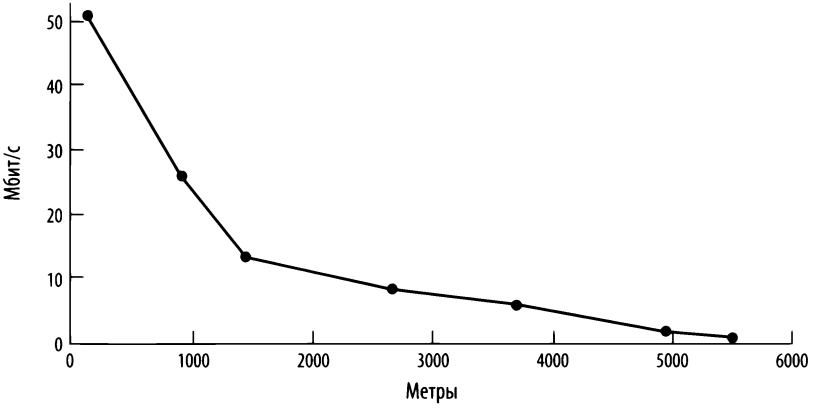
Изначально рынок был наводнен множеством пересекающихся вариантов высокоскоростного доступа в интернет под общим названием **цифровых абонентских линий (Digital Subscriber Line)**, или **xDSL**, где *x* меняется в зависимости от конкретной технологии. Сервисы с большей пропускной способностью, чем у обычных телефонных линий, иногда назывались **широкополосными (broadband)**, хотя это понятие скорее из сферы маркетинга, чем технологий. Мы обсудим наиболее популярный вариант, **асимметричный DSL**, или **ADSL**. В качестве краткого названия для всех этих вариантов мы будем использовать DSL или xDSL.

Причина медленной работы модемов в том, что телефоны изначально были предназначены для передачи человеческого голоса и вся система максимально оптимизирована под эту цель. Передача данных всегда была на втором месте. В точке подключения абонентского шлейфа к оконечной телефонной станции провод проходит через фильтр затухания, ослабляющий все частоты ниже 300 Гц и выше 3400 Гц. Срез происходит достаточно плавно (300 Гц и 3400 Гц — 3-дБ точки), и обычно указывается, что полоса пропускания равна 4000 Гц, хотя расстояние между 3-дБ точками составляет 3100 Гц. Данные в проводах также ограничиваются этой узкой полосой.

Хитрость, благодаря которой работает xDSL, заключается в том, что при подключении абонента входящий канал связи соединяется с особым сетевым коммутатором без вышеупомянутого фильтра. Это позволяет использовать всю пропускную способность абонентского шлейфа. Теперь ограничивающим фактором становятся физические характеристики линии, а не фильтр, и мы получаем около 1 МГц против искусственно созданного лимита в 3100 Гц.

К сожалению, пропускная способность абонентского шлейфа довольно быстро падает по мере удаления от оконечной станции и ослабления сигнала. Кроме того, она зависит от толщины и общего качества витой пары. График

потенциальной пропускной способности как функции расстояния приведен на илл. 2.28. В нем предполагается, что все остальные факторы — оптимальны (новые провода, качественные световоды и т. д.).




**Илл. 2.28.** Пропускная способность относительно расстояния при использовании для DSL кабелей UTP категории 3

Из графика следует проблема телефонных компаний. Выбирая скорость для своего коммерческого предложения, компания автоматически ограничивает радиус действия услуги. Это значит, что некоторым потенциальным клиентам придется отвечать: «Благодарим за интерес к нашему предложению, но вы живете на сто метров дальше от АТС, чем нужно, и не можете стать нашим абонентом. Не хотите ли переехать поближе?» Чем ниже выбранная скорость, тем длиннее радиус и больше абонентов. Но низкая скорость делает услугу менее привлекательной и меньше людей соглашается за нее платить. Вот так коммерческие соображения сталкиваются с технологическими.

Все услуги xDSL проектировались в соответствии с определенными критериями. Во-первых, они должны работать на уже имеющихся абонентских шлейфах с витыми парами категории 3. Во-вторых, не мешать существующим телефонам и факсам абонентов. В-третьих, их скорость должна значительно превышать 56 Кбит/с. В-четвертых, услуга должна быть всегда доступной и оплачиваться ежемесячно, а не поминутно.

Для удовлетворения технических требований имеющийся диапазон в 1,1 МГц абонентского шлейфа разбит на 256 независимых каналов, по 4312,5 Гц каждый. Эта архитектура показана на илл. 2.29. Для пересылки данных по этим каналам используется схема OFDM, которую мы обсуждали в предыдущем разделе. В связи с DSL ее часто называют **дискретной многотональной модуляцией (Discrete MultiTone, DMT)**. Канал 0 используется для **обычной телефонной связи (Plain Old Telephone Service, POTS)**. Каналы 1–5 не используются, чтобы голосовые и информационные сигналы не мешали друг другу. Из оставшихся 250 каналов один служит для управления входящим трафиком, еще один — для управления исходящим. Остальные каналы доступны для пользовательских данных.

**Илл. 2.29.** Функционирование ADSL на основе дискретной многотональной модуляции

В принципе, для полнодуплексной передачи данных может использоваться любой из оставшихся каналов, но гармоники, перекрестные помехи и другие эффекты не позволяют реальным системам достичь теоретически возможного предела. Количество каналов, доступных для входящего и исходящего потоков данных, выбирает поставщик услуги. Технически можно распределить их в соотношении 50/50, но большинство ISP отдает около 80–90 % пропускной способности входящему каналу, поскольку большинство пользователей скачивает больше данных, чем отправляет. Отсюда и «A» в ADSL. Распространенный вариант: 32 канала на исходящий поток данных, а остальные — на входящий. Также можно сделать несколько верхних исходящих каналов двунаправленными для повышения пропускной способности, хотя такая оптимизация потребует специального контура подавления эха.

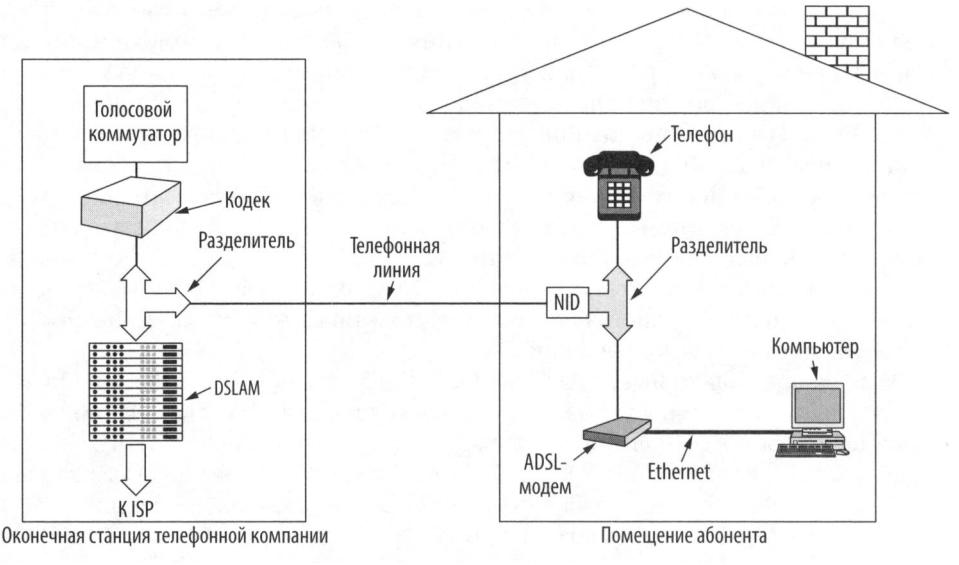
Международный стандарт ADSL — **G.dmt** — был одобрен в 1999 году. Он допускает входящую скорость до 8 Мбит/с и исходящую до 1 Мбит/с. В 2002 году его заменили более совершенным стандартом второго поколения, ADSL2, с входящей скоростью до 12 Мбит/с и исходящей до 1 Мбит/с. Стандарт ADSL2+ еще в два раза повысил входящую скорость, до 24 Мбит/с, за счет удвоения полосы пропускания (2,2 МГц через витую пару).

Следующим шагом, в 2006 году, стал **VDSL**. Входящая скорость передачи данных по коротким абонентским шлейфам достигала 52 Мбит/с, а исходящая — 3 Мбит/с. В период с 2007 по 2011 год появился ряд новых стандартов под общим названием **VDSL2**. При полосе пропускания 12 МГц на высококачественных абонентских шлейфах входящая скорость достигала 200 Мбит/с, исходящая — 100 Мбит/с. В 2015 году для шлейфов короче 250 м был предложен стандарт **Vplus**. Теоретически он позволяет достичь входящей скорости до 300 Мбит/с и исходящей до 100 Мбит/с, но реализовать это непросто. Вероятно, из уже существующих кабелей категории 3 больше выжать нельзя, разве что на более коротких расстояниях.

Внутри каналов используется модуляция QAM на скорости примерно в 4000 символов/с. При этом непрерывно отслеживается качество связи в каждом канале, с подстройкой скорости за счет использования бóльших или меньших схем модуляции, как на илл. 2.17. Скорость передачи данных в различных каналах отличается: до 15 бит/символ для канала с высоким SNR и 2, а то и 1 бит/символ для канала с низким SNR в зависимости от стандарта.

Типовая схема работы ADSL показана на илл. 2.30. Согласно этой схеме специалист телефонной компании устанавливает в помещении абонента

**устройство сопряжения с сетью (Network Interface Device, NID)**. Эта маленькая пластмассовая коробочка отмечает точку, где заканчивается оборудование, принадлежащее телефонной компании, и начинается собственность абонента. Неподалеку от NID (а иногда они даже совмещаются) располагается **разделитель (splitter)** — аналоговый фильтр, отделяющий полосу POTS (0–4000 Гц) от каналов данных. Сигнал POTS направляется к телефону или факсу, информационный сигнал — в ADSL-модем, реализующий OFDM при помощи цифрового обработчика сигналов. А поскольку большинство ADSL-модемов — внешние, компьютер подключается к модему по высокоскоростному соединению. Обычно для этого используется Ethernet, USB-кабель или 802.11.




**Илл. 2.30.** Типовая конфигурация оборудования ADSL

На другом конце провода, на стороне телефонной станции, устанавливается аналогичный разделитель. Сигнал с частотой выше 26 кГц направляется к специальному устройству — **мультиплексору доступа к цифровой абонентской линии (Digital Subscriber Line Access Multiplexer, DSLAM)**, включающему такой же цифровой сигнальный процессор, как и ADSL-модем. DSLAM преобразует сигнал в биты и отправляет пакеты в сеть интернет-провайдера.

Благодаря полному разделению системы передачи голоса и ADSL, развертывание ADSL для телефонной компании упрощается. Достаточно закупить DSLAM и разделитель и подключить абонентов ADSL к разделителю. Прочие сервисы с высокой пропускной способностью, предоставляемые по телефонной сети (например, ISDN), требуют от компаний куда больших изменений в коммутационном оборудовании.

Следующая вершина, которую предстоит покорить DSL, — скорости передачи данных в 1 Гбит/с и более. Для этого применяется множество

вспомогательных методов, включая **связывание (bonding)** — создание единого виртуального DSL-соединения за счет объединения двух или более физических DSL-соединений. Разумеется, при объединении двух витых пар пропускная способность также удваивается. В некоторых регионах в здания проводят телефонные кабели, состоящие из двойных витых пар. Изначально идея была в использовании двух отдельных телефонных линий с разными номерами в одном помещении. Но с помощью парной сцепки можно сделать на их основе одно высокоскоростное подключение к интернету. Все больше ISP в Европе, Австралии, Канаде и США развертывает технологию **G.fast**, в которой используется парное связывание. Как и в случае с остальными DSL, быстродействие G.fast зависит от расстояния, на которое передается сигнал. Недавние тесты показали, что на расстоянии 100 м скорость передачи по симметричному каналу приближается к 1 Гбит/с. В сочетании с оптоволокном это дает технологию **FTTdp (Fiber to the Distribution Point — «оптоволокно до точки распределения»)**. Оптоволокно прокладывается до точки распределения между несколькими сотнями абонентов, а на оставшемся участке (в случае VDSL2 — до 1 км, хотя и с меньшей скоростью передачи) используются медные провода. FTTdp — лишь один из вариантов использования оптоволокна не только в ядре сети, но и ближе к ее периферии. Ниже представлены другие системы такого типа.

### *Оптоволокно до точки X (FTTX)*

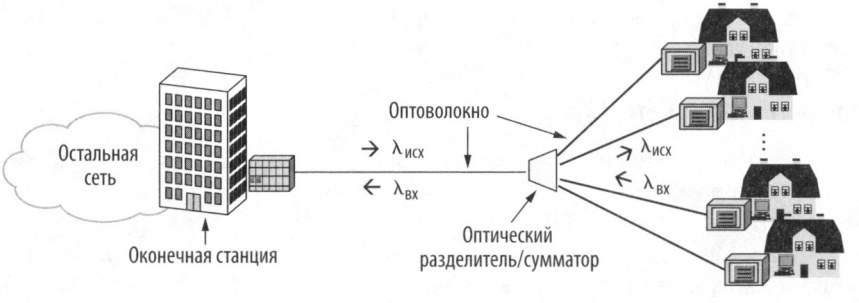
Скорость на последнем участке сети часто ограничена медными проводами, которые используются в обычных телефонных сетях. Они не способны на высокоскоростную передачу данных на столь большие расстояния, как оптоволоконный кабель. Следовательно, необходимо проложить оптоволокно как можно ближе к домам абонентов, то есть реализовать **FTTH (Fiber to the Home — «оптоволокно до дома»)**. Телефонные компании стремятся повысить быстродействие абонентского шлейфа, для чего зачастую прокладывают оптоволокно максимально близко к домам. И если даже не прямо в дом, то хотя бы поблизости. При использовании технологии **FTTN (Fiber to the Node/Neighborhood — «оптоволокно до узловой точки/микрорайона»)** кабель заканчивается в коммутационном шкафу на улице, иногда в нескольких километрах от дома абонента. В случае FTTdp оптоволокно оказывается еще ближе к домам, иногда буквально в нескольких метрах. Промежуточное положение между этими вариантами занимает **FTTC (Fiber to the Curb — «оптоволокно до бордюра»)**. Все эти виды FTTX иногда называют «оптоволокном в абонентском шлейфе», поскольку часть абонентского шлейфа составляет оптоволокно.

Существует несколько вариантов FTTX: *X* может означать подвал, бордюр или микрорайон. Все эти названия используются, чтобы указать на возможность прокладки оптоволокна ближе к дому абонента. В этом случае медные провода (витая пара или коаксиальный кабель) обеспечивают достаточно высокую скорость на последнем коротком участке. Насколько далеко прокладывать оптоволокно — вопрос экономический, выбор зависит от соотношения затрат и ожидаемой прибыли. В любом случае смысл в том, чтобы оптоволоконный

кабель перешел границу «последней мили». В нашем обсуждении мы сосредоточимся на технологии FTTH.

Как и медные провода, оптоволоконный абонентский шлейф пассивен, то есть не требует никакого оборудования для усиления или другой обработки сигналов. Оптоволокно просто переносит сигналы между жилищем абонента и оконечной станцией, снижая таким образом затраты и повышая надежность. Как правило, ведущие из домов кабели объединяются, так что от группы из 100 зданий к оконечной станции доходит только один оптоволоконный кабель. В исходящем направлении передаваемый из коммутатора сигнал разбивается оптическими разделителями, чтобы попасть во все дома. Если сигнал предназначается только для одного абонента, в целях безопасности используется шифрование. Во входящем направлении оптические сумматоры соединяют сигналы от всех домов в один, который и поступает в оконечную станцию.

Подобная архитектура, представленная на илл. 2.31, называется **пассивной оптической сетью (Passive Optical Network, PON)**. Обычно для входящей передачи данных все дома совместно используют одну длину волны, а для исходящей — другую.




**Илл. 2.31.** Пассивная оптическая сеть для технологии FTTH

Даже при разделении колоссальная пропускная способность и незначительное затухание оптоволоконного кабеля позволяют PON работать на высоких скоростях при расстоянии до 20 км. Фактическая скорость передачи данных и другие нюансы зависят от типа PON. Наиболее распространены два вида: **гигабитные PON (GPON)** и **Ethernet PON (EPON)**. GPON пришли из мира электросвязи, а потому описаны в стандарте МСЭ. EPON больше связаны с компьютерными сетями и описываются стандартом IEEE. Обе разновидности работают на скорости около гигабита и могут передавать трафик для различных нужд, включая интернет, видео и голосовые сервисы. Например, сети GPON обеспечивают входящую скорость 2,4 Гбит/с и исходящую — 1,2 или 2,4 Гбит/с.

Чтобы несколько зданий могли совместно использовать возможности одного оптоволоконного кабеля, идущего из оконечной станции, необходимы дополнительные протоколы. Во входящем направлении проблем нет. Оконечная станция может отправлять сообщения в разные дома в любом порядке. А вот одновременная передача данных из нескольких домов в исходящем направлении приведет к конфликту сигналов. Вдобавок различные дома не могут принимать

передаваемые другими домами сигналы, а значит, и не могут прослушивать, прежде чем передавать, не передает ли кто-то еще. Для решения этой проблемы устройства оконечной станции выделяют домовому оборудованию, по запросу последнего, интервалы времени для работы. Для успешного функционирования такой схемы необходимо выстроить хронометраж передачи данных от различных домов, чтобы синхронизировать все получаемые на оконечной станции сигналы. Такая архитектура аналогична архитектуре кабельных модемов, которую мы рассмотрим далее в этой главе. Больше информации о PON можно найти в работах Гроуба и Элберса (Grobe and Elbers, 2008), а также Де Андраде и др. (De Andrade et al., 2014).

## 2.5.3. Соединительные линии и мультиплексирование

Соединительные линии в телефонных сетях не только работают намного быстрее абонентских шлейфов, но и отличаются от последних еще двумя нюансами. В основной телефонной сети передается цифровая, а не аналоговая информация, то есть биты, а не голос. Из-за этого в оконечной станции нужна конвертация в цифровую форму для передачи по междугородним соединительным линиям. По соединительным линиям передаются тысячи, иногда миллионы звонков одновременно. Такое совместное использование позволяет значительно сэкономить, ведь проведение и обслуживание высокоскоростной и низкоскоростной линий стоят примерно одинаково. Совместное использование обеспечивается при помощи различных вариантов TDM и FDM.

Ниже мы кратко поговорим о преобразовании голосовых сигналов в цифровую форму для их передачи по телефонной сети. После этого мы рассмотрим применение TDM для отправки битов по соединительным линиям, включая систему TDM, используемую для оптоволокна (SONET). Затем мы обсудим применение FDM для оптоволокна: мультиплексирование по длинам волн.

### *Преобразование голосовых сигналов в цифровую форму*

На начальном этапе существования телефонных сетей голосовые звонки передавались в виде аналоговой информации. В течение долгих лет для мультиплексирования голосовых каналов по 4000 Гц каждый (3100 Гц плюс защитные полосы) в большие блоки использовались методики FDM. Например, 12 звонков в полосе от 60 до 108 кГц называются **группой**, пять групп (всего 60 звонков) — **супергруппой** и т. д. Эти методы FDM до сих пор иногда применяются для медных проводов и микроволновых каналов. Впрочем, FDM требует аналоговых электрических схем и не подходит для компьютерной обработки. TDM, напротив, можно полностью отдать на откуп цифровой электронике, поэтому в последние годы эта система получила широкое распространение. TDM работает только с цифровыми данными, а абонентские шлейфы генерируют аналоговые сигналы. Поэтому на оконечной станции, где все отдельные шлейфы сходятся и формируют исходящие соединительные линии, аналоговые сигналы преобразуются в цифры.

Преобразование происходит с помощью специального устройства — так называемого кодека (сокращение от «*ко*дировщик/*дек*одировщик»), который

применяет методику **импульсно-кодовой модуляции (Pulse Code Modulation, PCM)**. Эта методика — основа современной телефонной системы. Кодек создает 8000 сэмплов в секунду (по 125 мкс на сэмпл). Согласно теореме Найквиста этого достаточно для захвата всей информации от телефонного канала с полосой пропускания в 4 кГц. При меньшей частоте сэмплирования часть данных будет утрачена, при более высокой — дополнительных данных все равно получить нельзя. Практически все интервалы времени в любой телефонной системе кратны 125 мкс. Таким образом, стандартная скорость передачи несжатых данных для голосового телефонного звонка равна 8 битам каждые 125 мкс, то есть 64 Кбит/с.

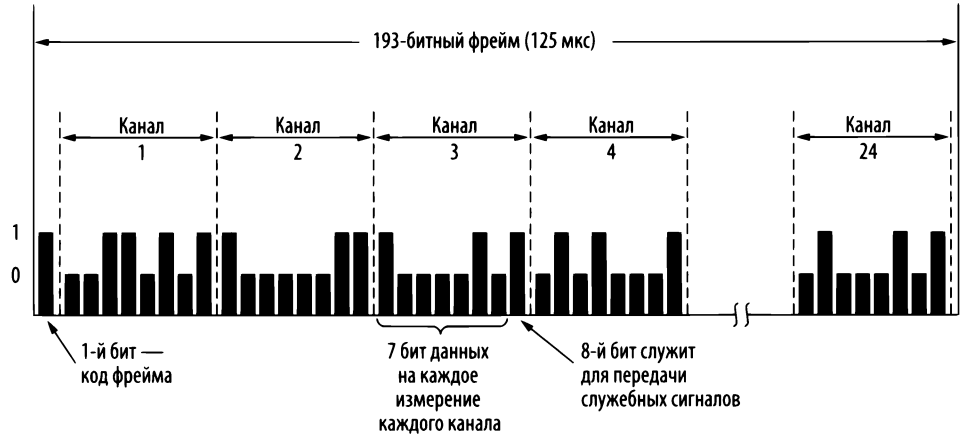
Каждый сэмпл амплитуды сигнала квантуется до 8-битного целого. Чтобы снизить погрешность, шаги квантования выбираются неравномерно. При этом используется логарифмическая шкала, вследствие чего на малые амплитуды сигналов приходится относительно больше битов, а на большие амплитуды — относительно меньше. Таким образом, погрешность пропорциональна амплитуде сигнала. Широко применяются два варианта квантования: **$\mu$-закон** в Северной Америке и Японии и **A-закон** в Европе и остальном мире. Оба варианта описаны в стандарте МСЭ G.711. Этот процесс можно представить так: динамический диапазон сигнала (отношение между минимальными и максимальными значениями) сжимается перед его квантованием (равномерным), а после восстановления аналогового сигнала — расширяется. По этой причине данный метод называют **компандированием (companding)**. Можно также сжимать сэмплы после их оцифровки, так что для них потребуется куда меньшая скорость канала данных, чем 64 Кбит/с. Впрочем, мы отложим этот вопрос до обсуждения аудиоприложений, например передачи голоса по IP.

На другой стороне звонка аналоговый сигнал восстанавливается из цифровых сэмплов путем их воспроизведения (и сглаживания). В точности соответствовать исходному аналоговому сигналу он, конечно, не будет, хотя мы и производили сэмплы со скоростью, указанной теоремой Найквиста, поскольку они были преобразованы в цифровую форму.

### *Система связи T: мультиплексирование цифровых сигналов в телефонных сетях*

**Система связи T (T-Carrier)** — спецификация передачи данных через несколько каналов TDM в одном физическом канале. TDM с PCM применяется для трансляции нескольких голосовых звонков по соединительным линиям посредством отправки сэмплов сигналов всех звонков каждые 125 мс. Когда цифровая передача данных стала реальностью, ССЭ МСЭ (тогда он носил название МККТТ) не смог согласовать международный стандарт PCM. В результате сейчас в различных странах используется множество разнообразных несовместимых между собой схем.

В Северной Америке и Японии используется система связи **T1**, показанная на илл. 2.32. (Строго говоря, формат называется DS1, а сама система связи — T1, но мы, следуя общепринятой в данной отрасли традиции, не станем вдаваться в подобные нюансы.) Система связи T1 состоит из 24 голосовых каналов, мультиплексированных в один. Каждый из этих 24 каналов, в свою очередь, выдает в исходящий поток сигналов 8 бит. Эта система связи возникла в 1962 году.




**Илл. 2.32.** Система связи T1 (1,544 Мбит/с)

Фрейм состоит из 24×8 = 192 бит плюс один дополнительный контрольный бит, то есть 193 бита каждые 125 мкс. Получается довольно приличная скорость передачи данных в 1,544 Мбит/с, из которых 8 Кбит/с отводится на вспомогательные цели. Этот 193-й бит используется для синхронизации фреймов и в сигнальных целях. В одном из вариантов он входит в состав группы из 24 фреймов, которая называется **расширенным суперфреймом (extended superframe)**. Шесть битов, на 4-й, 8-й, 12-й, 16-й, 20-й и 24 позициях, соответствуют повторяющемуся паттерну 001011... Как правило, чтобы убедиться в должной синхронизации, приемник непрерывно проверяет наличие этого паттерна. Еще шесть битов используются для отправки кода проверки ошибок, чтобы приемник мог подтвердить синхронизацию. В случае рассинхронизации приемник ищет паттерн и подтверждает код проверки ошибок для восстановления синхронизации. Оставшиеся 12 бит отводятся на контрольную информацию, необходимую для функционирования и сопровождения сети, например данные о быстродействии с удаленной стороны.

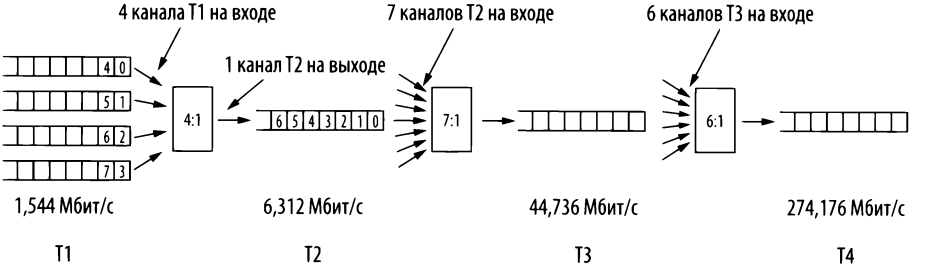
Существует несколько вариантов формата T1. В более ранних версиях сигнальная информация посылалась **внутриполосным образом (in-band)**, то есть по одному каналу с данными, используя некоторые биты. Эта архитектура представляет собой одну из форм **передачи служебных сигналов по отдельному каналу (channel-associated signaling)**, поскольку у каждого канала есть свой собственный сигнальный подканал. В одной из схем в каждом шестом фрейме используется наименее значимый бит из 8 бит сэмпла каждого канала. Этот вариант получил красочное название **передачи служебных сигналов с украденным битом (robbed-bit signaling)**. Основная его идея — несколько «украденных» битов не играют никакой роли для голосовых разговоров. Никто не услышит разницы.

Однако с данными все иначе. Отправка неправильных битов по меньшей мере бесполезна. При передаче данных с помощью более старых версий T1 в каждом из 24 каналов можно было задействовать лишь семь из восьми бит,

то есть 56 Кбит/с. Более новые варианты T1 обеспечивают свободные каналы с использованием всех битов. Свободные каналы — это именно то, что нужно компаниям, арендующим линии T1 для отправки по телефонной сети данных вместо голоса. При этом передача служебных сигналов для всех голосовых звонков производится **внеполосным образом (out-of-band)**, то есть по каналу, отделенному от данных. Зачастую передача служебных сигналов происходит по **общему**, то есть совместно используемому, **каналу (common-channel signaling)**. Для этой цели можно задействовать один из 24 каналов.

За пределами Северной Америки и Японии вместо T1 распространена система связи **E1** со скоростью 2048 Мбит/с. В ней используется 32 8-битных сэмпла, упакованных в стандартный 125-мкс фрейм. Тридцать каналов используются для информации и один-два — для передачи служебных сигналов. Каждая группа из четырех фреймов включает 64 бита для служебных сигналов, половина из которых используется для их передачи (по выделенному или общему каналу), а другая половина — для синхронизации фреймов (или же она резервируется — в каждой стране под разные нужды).

Мультиплексирование с разделением по времени позволяет объединять несколько T1 в системы более высокого порядка. На илл. 2.33 показано, как это происходит. Слева представлено четыре канала T1, мультиплексируемых в один канал T2. На уровне T2 и выше 24 голосовых канала, составляющие фрейм T1, мультиплексируются побитно, а не побайтно. Четыре потока T1 со скоростью 1,544 Мбит/с должны давать 6,176 Мбит/с, но на деле скорость T2 составляет 6,312 Мбит/с. Дополнительные биты используются для синхронизации фреймов и восстановления в случае сбоев системы связи.




**Илл. 2.33.** Мультиплексирование каналов T1 в системы связи более высокого порядка

На следующем уровне семь потоков T2 объединяются побитно в T3. Далее шесть потоков T3 соединяются в T4. На каждом шаге присутствуют небольшие накладные расходы на синхронизацию фреймов и восстановление (на случай рассинхронизации между отправителем и получателем). T1 и T3 широко используются абонентами, в то время как T2 и T4 применяются только внутри самой телефонной системы, поэтому они менее известны.

В США и остальном мире нет единого стандарта для базовой системы связи, так же как нет и согласия относительно ее мультиплексирования в систему с большей пропускной способностью. Принятую в США схему с шагами 4, 7 и 6

в остальном мире не сочли лучшим из возможных вариантов, поэтому стандарт МСЭ призывает мультиплексировать по четыре потока в один на каждом уровне. Кроме того, данные для синхронизации фреймов и восстановления также отличаются в стандартах США и МСЭ. В иерархии МСЭ используется 32, 128, 512, 2048 и 8192 канала, работающие на скоростях 2048, 8848, 34 304, 139 264 и 565 148 Мбит/с.

### *Мультиплексирование оптических сетей: SONET/SDH*

На самых первых этапах развития оптоволоконной связи каждая телефонная компания имела свою патентованную оптическую TDM-систему. После того как в 1984 году правительство США разделило AT&T, местным телефонным компаниям пришлось подключаться к многочисленным междугородним линиям с оптическими TDM-системами от различных производителей и поставщиков. Стало очевидно, что без стандартизации не обойтись. В 1985 году Bellcore, исследовательское подразделение Regional Bell Operating Companies (RBOCs), начало работу над этим стандартом, получившим название **синхронные оптические сети (Synchronous Optical NETwork, SONET)**.

Позднее к этой работе подключился МСЭ, в результате чего в 1989 году появился стандарт SONET и набор сопутствующих рекомендаций МСЭ (G.707, G.708 и G.709). Эти рекомендации МСЭ называются **синхронной цифровой иерархией (Synchronous Digital Hierarchy, SDH)**, но отличаются от SONET лишь мелкими нюансами. Практически все междугородние линии в США, да и во многих других странах в настоящее время используют SONET на физическом уровне. Дополнительную информацию вы найдете в работе Перроса (Perros, 2005).

Основные цели создания SONET:

1. Совместимость различных систем связи: SONET был призван обеспечить взаимодействие разных систем связи. Для этого понадобился общий стандарт обмена служебными сигналами с учетом длин волн, распределения интервалов времени, структуры фреймов и прочих нюансов.
2. Унификация стандарта для различных стран: пришлось приложить некоторые усилия, чтобы привести к одному виду цифровые системы США, Европы и Японии. Все они основаны на 64-Кбит/с каналах PCM, но группируют их различными (причем несовместимыми) способами.
3. Мультиплексирование цифровых каналов: SONET должен был обеспечить возможность мультиплексирования нескольких цифровых каналов. На момент создания SONET самой быстрой из распространенных в США систем цифровой связи была T3 со скоростью 44,736 Мбит/с. T4 уже существовала на бумаге, но использовалась не слишком широко, а стандарт со скоростью, превышающей T4, даже не был описан. Часть миссии SONET заключалась в том, чтобы расширить эту иерархию до скоростей порядка гигабит в секунду и выше. Кроме того, был необходим стандартный способ мультиплексирования медленных каналов в один канал SONET.
4. Поддержка управления системой: задачей SONET было обеспечить поддержку эксплуатации, администрирования и обслуживания (operations,

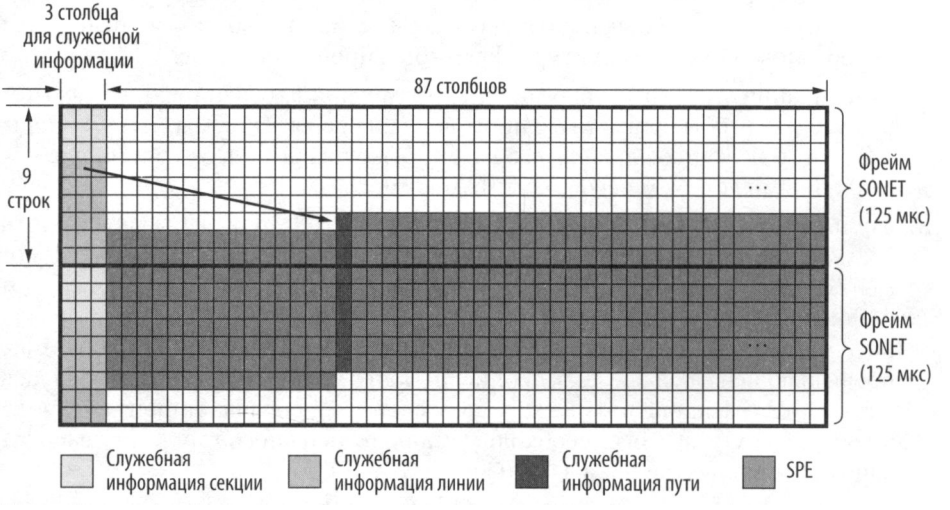
administration and maintenance, OAM), необходимых для управления. Предыдущие системы плохо с этим справлялись.

Изначально было решено сделать SONET обычной системой TDM и всю полосу пропускания оптоволокна предоставить одному каналу, выделяя слоты времени для различных подканалов. Поэтому SONET является синхронной системой. Все отправители и получатели привязаны к единому синхросигналу. Точность главного генератора синхроимпульсов, управляющего всей системой, составляет примерно 1 на $10^9$. Биты по линии SONET отправляются в исключительно точные промежутки времени, контролируемые главным генератором синхроимпульсов.

Простейший фрейм SONET представляет собой блок из 810 байт, передаваемый каждые 125 мкс. А поскольку SONET — синхронная система, фреймы генерируются вне зависимости от наличия полезных данных для отправки. Скорость 8000 фреймов/с в точности соответствует скорости получения измерений каналов PCM во всех телефонных системах.

Можно представить 810-байтные фреймы SONET в виде прямоугольника байтов, 90 столбцов в ширину и 9 строк в высоту. Таким образом, 8000 раз в секунду передается по 8 × 810 = 6480 бит, и общая скорость равна 51,84 Мбит/с. Эта схема отражает простейший канал SONET — **синхронный транспортный сигнал-1 (Synchronous Transport Signal-1, STS-1)**. Все соединительные линии SONET кратны STS-1.

Первые три столбца фрейма резервируются для управляющей информации системы, как показано на илл. 2.34. В этом блоке первые три строки содержат служебные данные секции (они генерируются и проверяются в начале каждой секции); следующие шесть строк составляют служебные данные линии (генерируются и проверяются в начале и конце каждой линии).




**Илл. 2.34.** Два идущих подряд фрейма SONET

Передатчик SONET отправляет один за другим 810-байтные фреймы без промежутков, даже если данных для отправки нет (в таком случае отправляются фиктивные данные). С точки зрения приемника они выглядят как непрерывный поток битов. Как же он различает границы фреймов? Дело в том, что первые два байта каждого фрейма содержат фиксированный паттерн. Если приемник находит этот паттерн в одном и том же месте в большом числе последовательно идущих фреймов, то он делает вывод, что синхронизирован с отправителем. Теоретически пользователь может вставлять данный паттерн в отправляемые полезные данные через равные промежутки, но на практике это невозможно по разным причинам, например из-за мультиплексирования данных от нескольких пользователей в одном фрейме.

Оставшиеся 87 столбцов каждого фрейма содержат $87 \times 9 \times 8 \times 8000 =$ $= 50{,}112$ Мбит/с пользовательских данных. Они могут быть голосовыми сэмплами в случае T1 и других систем связи или пакетами. SONET — это просто контейнер для передачи битов. **Огибающая синхронной полезной нагрузки (Synchronous Payload Envelope, SPE)** не всегда начинается в столбце 4 ряда 1. SPE может начинаться в любом месте фрейма. Первая строка служебных данных линии включает указатель на первый байт SPE. А первая строка SPE представляет собой служебные данные пути (то есть заголовок сквозного протокола подуровня пути).

Благодаря тому что SPE может начинаться в любом месте фрейма SONET и даже охватывать два фрейма, как показано на илл. 2.34, система становится более гибкой. Например, если во время формирования фиктивного фрейма SONET в источник поступает пользовательская информация, ее можно вставить в текущий фрейм, а не ждать начала следующего.

Иерархия мультиплексирования SONET/SDH приведена на илл. 2.35. В стандарте описаны скорости от STS-1 до STS-768, то есть примерно от линии T3 до 40 Гбит/с. Несомненно, со временем будут описаны и более высокие скорости. Следующей будет система OC-3072 со скоростью 160 Гбит/с, когда это станет технически выполнимым. Оптическая система, соответствующая STS-$n$ (синхронному транспортному сигналу $n$-уровня), называется OC-$n$ и совпадает с ним с точностью до бита, с той разницей, что для синхронизации требуется некоторая перестановка битов. Названия SDH отличаются — они начинаются с OC-3, поскольку в системах на основе стандартов МСЭ нет скорости, близкой к 51,84 Мбит/с. На илл. 2.35 приведены распространенные варианты скоростей, начиная с OC-3 и далее, кратные 4. Общая скорость учитывает все служебные данные. Скорость передачи SPE не учитывает служебные данные линии и секции. Скорость передачи пользовательских данных учитывает все три вида служебных данных и охватывает только 86 столбцов пользовательских данных.

Когда система связи (например, OC-3) не мультиплексируется, а переносит данные от единственного источника, в ее обозначение добавляется буква $c$ (от concatenated — «конкатенированный»). Таким образом, OC-3 — это система связи со скоростью 155,52 Мбит/с, состоящая из трех отдельных систем OC-1, а OC-3c — поток данных из одного источника на скорости в 155,52 Мбит/с. Три потока данных OC-1 в OC-3c чередуются по столбцам: столбец 1 из потока 1,

столбец 1 из потока 2, столбец 1 из потока 3, затем столбец 2 из потока 1 и т. д., в результате чего получается фрейм шириной в 270 столбцов и глубиной в 9 строк.

| SONET | | SDH | Скорость передачи данных (Мбит/с) | | |
|---|---|---|---|---|---|
| Электрическая | Оптическая | Оптическая | Общая | SPE | Пользовательских данных |
| STS-1 | OC-1 | | 51,84 | 50,112 | 49,536 |
| STS-3 | OC-3 | STM-1 | 155,52 | 150,336 | 148,608 |
| STS-12 | OC-12 | STM-4 | 622,08 | 601,344 | 594,432 |
| STS-48 | OC-48 | STM-16 | 2488,32 | 2405,376 | 2377,728 |
| STS-192 | OC-192 | STM-64 | 9953,28 | 9621,504 | 9510,912 |
| STS-768 | OC-768 | STM-256 | 39813,12 | 38486,016 | 38043,648 |

**Илл. 2.35.** Скорости мультиплексирования SONET и SDH

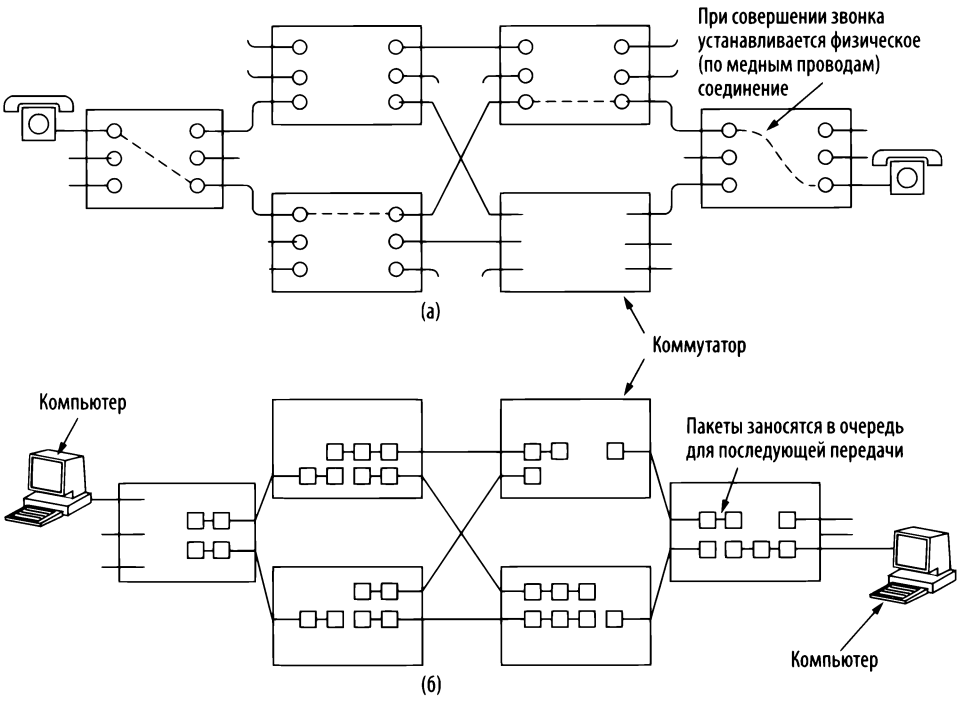
## 2.5.4. Коммутация

С точки зрения обычного телефонного инженера, телефонная система состоит из двух основных частей: наружное оборудование (абонентские шлейфы и соединительные линии), физически находящееся вне АТС, и внутреннее оборудование (коммутаторы), расположенное на АТС. До сих пор мы рассматривали только наружное оборудование. Пришло время обсудить внутреннее.

Сегодня в сетях применяются два различных метода: коммутация каналов и коммутация пакетов. Первый метод использовался в традиционных телефонных системах, а в основе технологии передачи голоса по IP лежит второй метод. Мы обсудим коммутацию каналов несколько подробнее, а затем сравним ее с коммутацией пакетов. Оба метода важны, поэтому мы вернемся к ним еще раз, когда будем говорить о сетевом уровне.

### *Коммутация каналов*

Изначально при совершении человеком или компьютером телефонного звонка коммутационное оборудование строило физический маршрут между двумя абонентами и поддерживало его во время разговора. Эта методика называется **коммутацией каналов (circuit switching)**. Схематически она представлена на илл. 2.36 (а). Каждый из шести прямоугольников соответствует коммутатору системы связи (оконечной телефонной станции, центральной телефонной станции и т. д.). В этом примере у каждой станции три входящие и три исходящие линии. При прохождении звонка через коммутатор устанавливается физическое соединение между линией связи, по которой поступил звонок, и одной из выходных линий, показанных пунктиром.



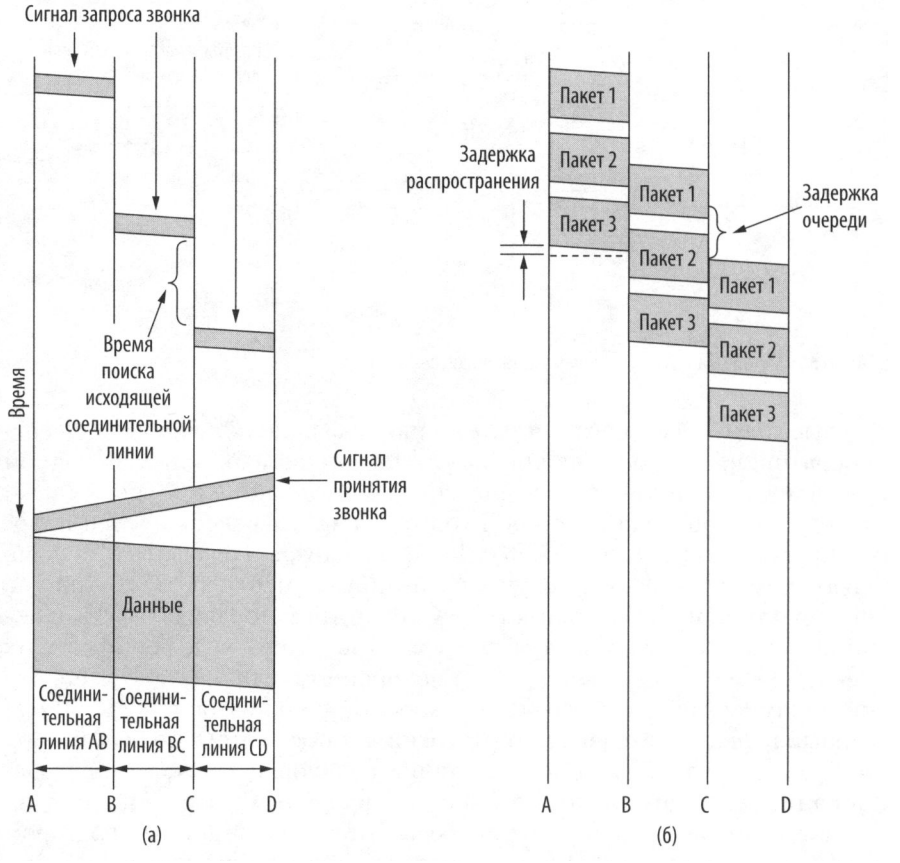

**Илл. 2.36.** (а) Коммутация каналов. (б) Коммутация пакетов

В первые годы существования телефонной связи подключение осуществлял оператор, вручную вставляя гибкий кабель во входной и выходной разъемы. С изобретением автоматического оборудования для коммутации каналов связана забавная история. Его создал в XIX веке владелец похоронного бюро Элмон Б. Строуджер (Almon B. Strowger) из штата Миссури. После изобретения телефона в случае чьей-нибудь смерти люди звонили на коммутатор и говорили телефонистке: «Соедините меня, пожалуйста, с похоронным бюро». К несчастью для мистера Строуджера, телефонистка в их городке была женой владельца другого похоронного бюро. Стало ясно, что либо он придумает автоматический коммутатор, либо разорится. И он выбрал первый вариант. В течение почти сотни лет после этого по всему миру применялось оборудование для коммутации каналов, известное под названием **декадно-шагового искателя Строуджера (Strowger gear)**. История умалчивает о том, не стала ли потерявшая работу телефонистка оператором справочного бюро, отвечая на вопросы вроде: «Каков номер телефона похоронного бюро?»

Приведенная на илл. 2.36 (а) модель, конечно, сильно упрощена, поскольку физический путь между двумя телефонами может включать микроволновые или оптоволоконные каналы связи, сочетающие путем мультиплексирования тысячи звонков. Тем не менее основная идея все та же: во время звонка устанавливается

соединение и возникает выделенный путь между абонентами, который поддерживается до завершения звонка.

Важная особенность коммутации каналов: необходимо сформировать сквозной путь между абонентами *перед* отправкой данных. Между окончанием набора номера и тем, когда зазвонит телефон, может пройти 10 с (или больше — при междугородних или международных разговорах). В это время телефонная система ищет путь, как показано на илл. 2.37 (а). Обратите внимание, что еще до начала передачи данных сигнал запроса звонка должен пройти весь путь до точки назначения, а его получение должно быть подтверждено. Во многих компьютерных приложениях (например, при проверке наличия средств на карте в POS-системах) длительное ожидание нежелательно.




**Илл. 2.37.** Хронометраж событий при: (а) коммутации каналов; (б) коммутации пакетов

Как только путь между участниками разговора установлен, задержка данных зависит только от скорости распространения электромагнитного сигнала: примерно 1000 км за 5 мс. Кроме того, благодаря выделенному маршруту можно

не бояться перегруженности — после соединения вы уже не услышите сигнала «занято».

### Коммутация пакетов

Альтернатива коммутации каналов — **коммутация пакетов (packet switching)**, показанная на илл. 2.36 (б) и описанная в главе 1. При использовании этой технологии пакеты отправляются сразу же — заранее формировать выделенный путь не требуется (в отличие от коммутации каналов). Коммутация пакетов напоминает отправку нескольких писем по почте: все они передаются независимо друг от друга. Перемещение каждого отдельного пакета до пункта назначения производят маршрутизаторы на основе передачи с промежуточным хранением данных. Данная процедура отличается от коммутации каналов, при которой после установления соединения резервируется полоса пропускания на всем протяжении пути от отправителя к получателю. Все данные в канале следуют по этому пути строго в порядке отправления. При коммутации пакетов фиксированного пути не существует. А значит, пакеты могут передаваться по разным маршрутам в зависимости от сложившихся в сети условий на момент их отправки и могут доставляться в произвольном порядке.

Сети с коммутацией пакетов ограничивают максимальный размер пакета, гарантируя тем самым, что ни один пользователь не сможет надолго (например, на большое число миллисекунд) полностью занять линию передачи. Таким образом, сети с коммутацией пакетов могут работать с интерактивным трафиком. Кроме того, это снижает задержку: первый пакет длинного сообщения передается дальше до того, как полностью прибудет второй. Но задержка пакета в памяти маршрутизатора перед дальнейшей отправкой (связанная с буферизацией данных) превышает задержку при коммутации каналов, где биты непрерывно движутся по проводам, и ничего не сохраняется для последующей отправки.

Коммутация пакетов и каналов различается не только этим. Поскольку при коммутации пакетов не резервируется полоса пропускания, пакетам иногда приходится ждать дальнейшей передачи. В результате при одновременной отправке большого числа пакетов возникает **задержка в очереди (queueing delay)** и перегруженность сети. С другой стороны, нет риска услышать сигнал «занято» и потерять возможность использовать сеть. Таким образом, при коммутации каналов сеть перегружается во время установки соединения, а при коммутации пакетов — во время отправки данных.

Но если канал зарезервирован для конкретного пользователя, а трафика нет, то полоса пропускания простаивает, хотя могла бы использоваться для другого трафика. При коммутации пакетов такого не бывает, а значит, этот метод эффективнее с точки зрения системы. Чтобы увидеть принципиальную разницу между коммутацией пакетов и коммутацией каналов, необходимо осознать следующий компромисс. Либо мы получаем гарантированный сервис с напрасной тратой ресурсов, либо — негарантированный, но без таковой. Коммутация пакетов устойчивее к сбоям, чем коммутация каналов. На самом деле именно поэтому она и была создана. Когда отказывает сетевой коммутатор, все подключенные к нему каналы обрываются и их нельзя использовать для передачи.

При коммутации пакетов можно перенаправить пакеты в обход неработающих сетевых коммутаторов.

Еще одно различие между коммутацией пакетов и каналов — тарификация трафика. При коммутации каналов (например, для голосовых звонков по телефону через PSTN) трафик тарифицируется в зависимости от расстояния и времени. В мобильной связи расстояние обычно не имеет значения (разве что при международных звонках), а время играет лишь второстепенную роль. Например, тарифный план на 2000 бесплатных минут обходится дороже плана с 1000 минут и пониженным тарифом по ночам или выходным. При коммутации пакетов, как в стационарных, так и в мобильных сетях, длительность соединения роли не играет и основным фактором является объем трафика. С домашних пользователей в США и Европе ISP обычно взимают ежемесячную абонентскую плату (так проще для ISP и понятнее для клиентов). В некоторых развивающихся странах тарификация до сих пор происходит на основе объема трафика: пользователи покупают «пакет данных» определенного размера, который используется на протяжении цикла тарификации. При этом в определенное время дня или по некоторым направлениям трафик может быть бесплатным либо не входить в ежедневную квоту. Такие сервисы иногда называют **сервисами с нулевой ставкой (zero-rated services)**. В целом ISP в опорной сети интернета тарифицируют услуги исходя из объемов трафика. В основе типовой модели тарификации лежит 95-й процентиль пятиминутных измерений. ISP измеряет трафик, прошедший через конкретное соединение за последние 5 минут; 30-дневный расчетный период включает 8640 подобных 5-минутных интервалов; ISP выставляет счет, исходя из 95-го процентиля этих измерений. Эту методику часто называют **тарификацией 95-го процентиля (95th percentile billing)**.

На илл. 2.38 приведена общая сводка различий между двумя видами коммутаций. Традиционно коммутация каналов применялась в телефонных сетях ради

| Пункт | С коммутацией каналов | С коммутацией пакетов |
|---|---|---|
| Соединение | Необходимо | Не требуется |
| Выделенный физический путь | Да | Нет |
| Все пакеты следуют по одному пути | Да | Нет |
| Пакеты прибывают в порядке отправления | Да | Нет |
| Отказ коммутатора играет критическую роль | Да | Нет |
| Доступная полоса пропускания | Фиксированная | Динамическая |
| Время возможной перегруженности сети | Во время установления соединения | На любом пакете |
| Вероятность траты полосы пропускания впустую | Да | Нет |
| Передача данных с их промежуточным хранением | Нет | Да |
| Тарификация | Поминутно | Побайтно |

**Илл. 2.38.** Сравнение сетей с коммутацией каналов и коммутацией пакетов

повышения качества звонков, а коммутация пакетов использовалась в компьютерных сетях из-за ее простоты и эффективности. Впрочем, существуют заслуживающие упоминания исключения. В некоторых более старых компьютерных сетях «под капотом» используется коммутация каналов (например, в сетях, основанных на стандарте X.25), а в более новых телефонных сетях при передаче голоса по IP используется коммутация пакетов. Для пользователей это выглядит как обычный телефонный звонок, но внутри сети происходит коммутация сетевых пакетов голосовых данных. Это способствовало развитию рынка дешевых международных звонков с помощью переговорных карточек (хотя, вероятно, с более низким качеством звонка, чем у официальных телефонных компаний).

## 2.6. СОТОВЫЕ СЕТИ

Даже если традиционная телефонная система когда-нибудь полностью перейдет на мультигигабитное оптоволокно, этого будет недостаточно. Современные пользователи хотят звонить, проверять электронную почту и просматривать веб-страницы где угодно: в самолетах, автомобилях, бассейнах и даже во время пробежек в парке. Это порождает невероятный интерес к беспроводной телефонии (а также инвестициям в нее).

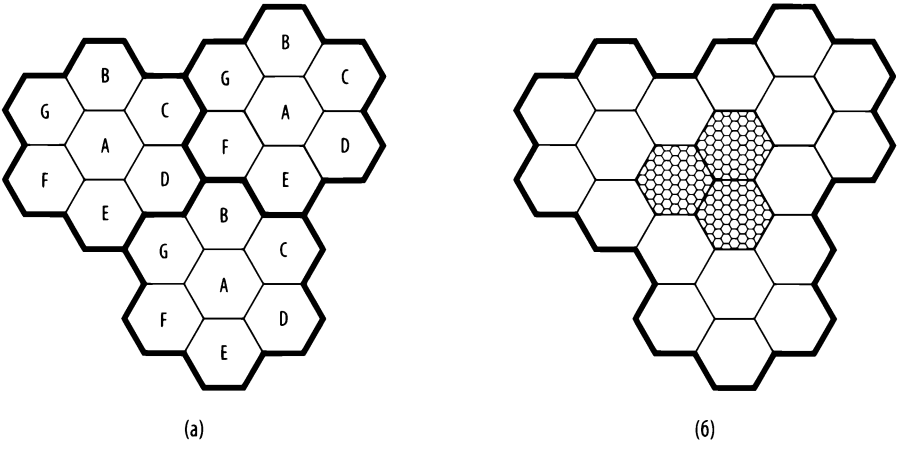
Мобильные телефонные системы используются для глобальной голосовой связи и обмена данными. Уже насчитывается пять поколений **мобильных телефонов** (иногда называемых **сотовыми**): 1G, 2G, 3G, 4G и 5G. Первые два поколения предоставляли услуги аналоговой (1G) и цифровой (2G) передачи голоса; поколение 3G — цифровой передачи голоса и данных (интернет, электронная почта и т. д.). В технологии 4G добавились новые возможности, включая дополнительные методики передачи данных физического уровня (например, восходящую передачу OFDM), а также фемтосоты на основе IP (домашние сотовые узлы, подключенные к стационарной интернет-инфраструктуре). Поколение 4G не поддерживает телефонию с коммутацией каналов, в отличие от его предшественников; в его основе — исключительно коммутация пакетов. В настоящее время постепенно развертываются сети 5G, но пройдут годы, прежде чем они полностью заменят сети предыдущих поколений. Технология 5G позволяет передавать данные на скорости до 20 Гбит/с и отличается большей плотностью размещения сотовых вышек. Особое внимание направлено на снижение сетевой задержки, чтобы обеспечить работу более широкого круга приложений, например современных интерактивных игр.

### 2.6.1. Основные понятия: соты, передача обслуживания, пейджинг

Во всех мобильных телефонных системах географические области делятся на **соты (cells)**, именно поэтому переносные телефонные аппараты иногда называют сотовыми телефонами. Смежные соты используют разные наборы частот. Главная идея, благодаря которой пропускная способность сотовых систем намного выше,

чем у их предшественников, — использование относительно маленьких сот и повторное использование частот в близко расположенных (но не смежных) сотах. Чем меньше соты, тем выше пропускная способность системы и тем экономнее потребление электроэнергии. В итоге передатчики и переносные телефонные аппараты становятся компактнее и дешевле.

Соты позволяют многократно использовать частоты, как показано на илл. 2.39 (а). Соты имеют форму, близкую к окружности, но удобнее представить их в виде шестиугольников. На илл. 2.39 (а) изображены соты одного размера, сгруппированные по семь штук. Наборы частот обозначены буквами. Обратите внимание, что каждый набор окружен буфером в две соты толщиной, в котором используются другие частоты. Это обеспечивает эффективное разделение частот и низкий уровень помех.




**Илл. 2.39.** (а) Одинаковые частоты не используются в смежных сотах. (б) Чтобы повысить число пользователей, можно использовать меньшие соты

Слишком большое количество пользователей может вызвать перегрузку системы. Чтобы этого избежать, можно снизить мощность и разбить перегруженные соты на **микросоты**. Это повышает интенсивность повторного использования частот, как показано на илл. 2.39 (б). Иногда телефонные компании создают временные микросоты с помощью переносных вышек со спутниковыми каналами связи — на спортивных мероприятиях, рок-концертах и в других местах, где множество пользователей собирается на несколько часов.

В центре сот расположены базовые станции, на которые передаются данные со всех телефонов в соте. Базовая станция состоит из компьютера и передатчика/приемника, подключенных к антенне. В маленьких системах все базовые станции подключаются к одному устройству, которое называется **центром мобильной коммутации (Mobile Switching Center, MSC)**, или **коммутатором мобильной связи (Mobile Telephone Switching Office, MTSO)**. В больших системах могут понадобиться несколько MSC, подключенных к одному MSC второго уровня, и т. д. MSC, по сути, являются аналогами оконечных телефонных станций

и фактически подключены по меньшей мере к одной такой станции. MSC обмениваются информацией с базовыми станциями, друг с другом и с **коммутируемой телефонной сетью общего пользования (Public Switched Telephone Network, PSTN)**, используя коммутацию пакетов.

В любой момент времени мобильный телефон логически относится к конкретной соте и контролируется ее базовой станцией. Когда телефон покидает эту соту, базовая станция замечает ослабление сигнала и «спрашивает» ближайшие станции, насколько хорошо они его «слышат». Получив ответы, станция передает обслуживание этого телефона той соте, в пределах которой он теперь находится. После этого телефон оповещается о смене соты. Если это происходит во время разговора, он получает предложение о переключении на новый канал (поскольку старый не может использоваться в смежных сотах). Этот процесс, называемый **передачей обслуживания (handoff)**, занимает около 300 мс. Распределением каналов занимается MSC — мозговой центр системы. Базовые станции представляют собой простые радиоретрансляторы.

Основная проблема — найти высокую точку для размещения базовых станций. Она заставила некоторых операторов мобильной связи заключить договор с Римско-католической церковью, которая владеет множеством подходящих для установки антенн возвышенностей по всему миру. Удобно и то, что они находятся под управлением одной организации.

В сотовых сетях обычно встречается четыре типа **каналов**. **Каналы управления (control channels)** — от базовой станции к мобильному устройству — используются для управления системой. **Пейджинговые каналы (paging channels)** — от базовой станции к мобильному устройству — оповещают пользователей мобильных устройств о входных звонках. **Каналы доступа (access channels)** — двунаправленные — используются для настроек звонков и распределения каналов. Наконец, **каналы данных (data channels)** — также двунаправленные — служат для передачи голоса, факсов или данных.

## 2.6.2. Технология 1G: аналоговая передача голоса

Теперь рассмотрим технологии сотовых сетей с самых первых систем. Мобильные радиотелефоны периодически использовались в морской и военной связи еще в начале XX века. В 1946 году в Сент-Луисе появилась первая система для автомобильных телефонов — с одним большим передатчиком на высотном зданий и одним каналом для передачи и приема. Для разговора пользователям приходилось нажимать кнопку, включающую передатчик и отключающую приемник. Подобные **системы «нажал — говори» (push-to-talk systems)** устанавливались повсюду в начале 1950-х. Эта технология часто используется в такси и полицейских автомобилях.

В 1960-х начала внедряться **усовершенствованная система мобильной связи (Improved Mobile Telephone System, IMTS)**. В ней также применялся мощный (200 Вт) передатчик, расположенный на возвышенности, но с двумя частотами — одна для передачи, вторая для приема, так что необходимость в переговорной кнопке отпала. А поскольку входящие и исходящие сигналы передавались по

разным каналам, пользователи мобильной связи не могли слышать чужие разговоры (в отличие от систем «нажал — говори» в старых такси).

IMTS поддерживала 23 канала в диапазоне от 150 до 450 МГц. Из-за такого маленького числа каналов пользователям нередко приходилось долго ждать освобождения линии. Кроме того, из-за большой мощности расположенных на возвышенностях передатчиков смежные системы приходилось разносить на несколько сотен километров во избежание помех. Все эти ограничения делали систему непрактичной.

Пропускную способность сотовых сетей значительно увеличила аналоговая **продвинутая система телефонной мобильной связи (Advanced Mobile Phone System, AMPS)**, созданная Bell Labs и впервые развернутая в США в 1983 году. Она также использовалась в Великобритании, где называлась TACS, затем в Японии — под названием MCS-L1. От AMPS официально отказались в 2008 году, но мы рассмотрим ее, чтобы лучше понять следующие за ней системы 2G и 3G. В AMPS размер сот варьируется от 10 до 20 км в поперечнике; в цифровых системах соты обычно меньше. В то время как в IMTS диаметром 100 км на каждой частоте мог выполняться лишь один звонок, в AMPS на ту же площадь приходится сто 10-километровых сот, благодаря чему она может обслуживать от 10 до 15 звонков на каждой частоте в удаленных друг от друга сотах.

Для разделения каналов в AMPS используется FDM. Система включает 832 полнодуплексных канала, каждый из которых состоит из пары симплексных каналов. Такая схема называется **дуплексной связью с частотным разделением каналов (Frequency Division Duplex, FDD)**. Для передачи данных с мобильного устройства на базовую станцию используется 832 симплексных канала в диапазоне 824–849 МГц; обратно данные передаются по 832 симплексным каналам в диапазоне 869–894 МГц. Ширина каждого симплексного канала — 30 кГц.

832 канала AMPS делятся на четыре категории. Поскольку повторно использовать одинаковые частоты в соседних сотах нельзя, а 21 канал в каждой соте резервируется для управления, фактически число доступных голосовых каналов в каждой соте намного меньше, чем 832 (обычно около 45).

### *Управление вызовами*

В программируемой постоянной памяти каждого телефона в AMPS содержится уникальный 32-битный серийный номер и телефонный номер из 10 цифр. Телефонный номер во многих странах содержит 10-битный код области из 3 цифр и 24-битный номер пользователя из 7 цифр. При подключении телефона он сканирует заранее запрограммированный список из 21 канала управления в поисках наиболее мощного сигнала. Затем телефон транслирует свой 32-битный серийный номер и 34-битный телефонный номер. Как и вся управляющая информация в AMPS, этот пакет отправляется в цифровой форме, многократно, с использованием кода коррекции ошибок, несмотря на то что сами голосовые каналы — аналоговые.

Когда базовая станция получает оповещение, она передает информацию об этом MSC, который фиксирует появление нового абонента, а также сообщает

домашнему MSC абонента о его текущем местоположении. В штатном режиме работы мобильный телефон регистрируется заново примерно каждые 15 минут.

Чтобы позвонить, пользователь включает телефон, набирает номер и нажимает кнопку вызова. Телефон передает вызываемый номер и его собственный идентификатор по каналу доступа. В случае конфликта попытка повторяется. Получив запрос, базовая станция сообщает о нем MSC. Если звонящий является абонентом компании данного MSC (или одного из ее партнеров), то MSC ищет свободный канал для звонка. Обнаружив канал, он отправляет его номер обратно по каналу управления. После этого мобильный телефон автоматически переключается на выбранный голосовой канал и ждет, пока вызываемый абонент не поднимет трубку.

Входящие звонки происходят иначе. Все телефоны в режиме ожидания непрерывно прослушивают пейджинговый канал на предмет предназначенных для них сообщений. При звонке на мобильный телефон (со стационарного или с другого мобильного телефона) домашний MSC вызываемого абонента получает пакет с запросом о его местонахождении. Затем пакет отправляется на базовую станцию текущей соты, которая транслирует по пейджинговому каналу запрос вида: «Устройство 14, вы здесь?». Телефон вызываемого абонента отвечает по каналу доступа «Да». После этого базовая станция сообщает ему нечто вроде: «Устройство 14, вам поступил вызов по каналу 3». Далее вызываемый телефон переключается на канал 3 и начинает звонить (или проигрывать мелодию, полученную владельцем телефона в качестве подарка на день рождения).

## 2.6.3. Технология 2G: цифровая передача голоса

Первое поколение мобильных телефонов было аналоговым, второе поколение — цифровое. У перехода на цифровые технологии есть несколько преимуществ. За счет оцифровки и сжатия голосовых сигналов повышается пропускная способность. Благодаря возможности шифрования голосовых и управляющих сигналов повышается безопасность. Это защищает от мошенничества и перехвата разговоров, как в случае преднамеренного сканирования диапазона, так и при случайном перехвате отголосков чужих звонков вследствие распространения радиоволн. Наконец, появляются новые сервисы, такие как обмен текстовыми сообщениями.

Международного стандарта для второго поколения мобильной связи не появилось, так же как и для первого. На основе 2G было разработано несколько систем, три из которых применялись весьма широко. **Продвинутая цифровая система телефонной мобильной связи (Digital Advanced Mobile Phone System, D-AMPS)** представляет собой цифровой вариант AMPS, способный сосуществовать с ней. Она использует TDM для нескольких одновременных звонков на одном частотном канале. D-AMPS описана в международном стандарте IS-54 и его преемнике IS-136. С тех пор преобладающей системой стала **Глобальная система мобильной связи (Global System for Mobile communications, GSM)**. Несмотря на медленное распространение в США, сейчас она используется практически во всем мире. В основе GSM, как и D-AMPS, лежит сочетание FDM и TDM. Совершенно иной системой, не основанной ни на FDM, ни на TDM, является **множественный доступ с кодовым разделением (Code Division**

**Multiple Access, CDMA)**, описанный в **международном стандарте IS-95**. И хотя эта технология не стала главной системой 2G, она легла в основу 3G.

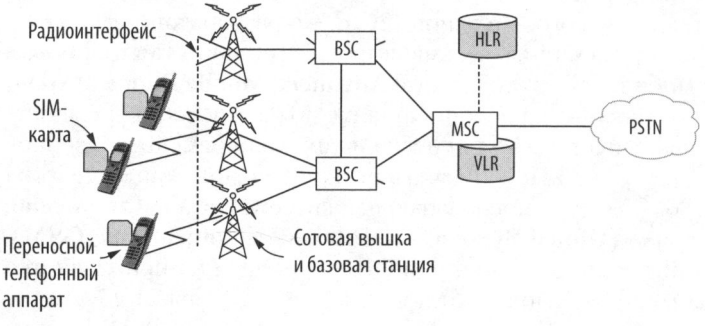
В литературе по маркетингу системы 2G (то есть цифровые) иногда называются **сервисами персональной связи (Personal Communications Services, PCS)**. Изначально под этим понимались мобильные телефоны, использующие полосу 1900 МГц, но сегодня это разграничение роли не играет. Доминирующей системой 2G во всем мире сегодня является GSM, подробно описанная далее.

## 2.6.4. GSM: Глобальная система мобильной связи

GSM появилась в 1980-х как попытка создания единого европейского стандарта 2G. Эту задачу поручили комиссии по электросвязи, носившей французское название Groupe Specialé Mobile. Первые GSM-системы были развернуты в 1991 году и быстро обрели популярность. Скоро стало очевидно, что GSM ожидает успех не только на европейском рынке, но и в отдаленных уголках мира, даже в Австралии. Поэтому GSM переименовали — для большей привлекательности в мировых масштабах.

GSM и другие системы телефонной связи, которые будут представлены далее, унаследовали от систем первого поколения сотовую архитектуру, повторное использование частот и мобильность с передачей обслуживания при перемещении абонента. Различаются лишь нюансы. Мы кратко рассмотрим основные свойства GSM. Но учтите, что в напечатанном виде стандарт GSM занимает более 5000 (!) страниц. Немалая доля этих сведений относится к технической стороне системы, особенно к архитектуре приемников для многолучевого распространения сигнала и синхронизации передатчиков и приемников. Этих вопросов мы касаться не будем.

Архитектуры GSM и AMPS совпадают, как показано на илл. 2.40, хотя названия компонентов отличаются. Сам мобильный телефон теперь состоит из переносного телефонного аппарата и съемного чипа — так называемой **SIM-карты (SIM card; Subscriber Identity Module — модуль идентификации абонента)**, содержащей информацию о пользователе и состоянии его счета. Именно SIM-карта активирует переносной телефонный аппарат и содержит всю секретную




**Илл. 2.40.** Архитектура мобильной сети GSM

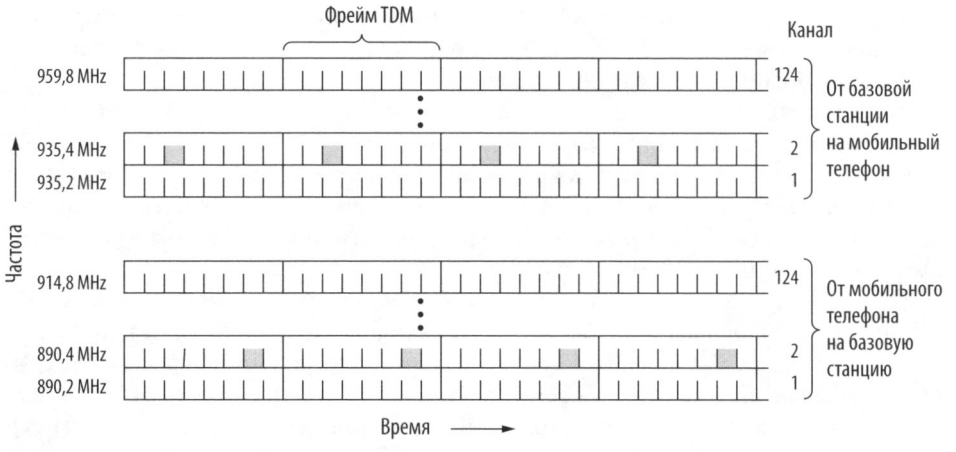
информацию, с помощью которой телефон и сеть идентифицируют друг друга и шифруют разговоры. SIM-карту можно вытащить из одного аппарата и вставить в другой: он и станет вашим телефоном с точки зрения сети.

Мобильный телефон взаимодействует с сотовыми базовыми станциями через **радиоинтерфейс (air interface)**, который мы опишем чуть позднее. Все сотовые базовые станции подключены к **контроллерам базовых станций (Base Station Controller, BSC)**. Они управляют радиоресурсами сот и отвечают за передачу обслуживания. BSC подключаются к MSC (аналогично AMPS), который маршрутизирует звонки и соединяется с PSTN.

Для маршрутизации звонков MSC требуется информация о местоположении абонентов. Он поддерживает базу данных находящихся поблизости телефонов, подключенных к управляемой им соте, — **регистр роуминговых абонентов (Visitor Location Register, VLR)**. В мобильной сети также есть база данных о последнем известном местонахождении всех телефонов — **домашний регистр местоположения (Home Location Register, HLR)**. Эта база используется для маршрутизации входящих звонков в нужную точку. Важно поддерживать актуальность обеих баз данных, поскольку мобильные телефоны постоянно перемещаются из соты в соту.

Теперь подробнее обсудим радиоинтерфейс. Во всем мире GSM работает на нескольких радиочастотах, включая 900, 1800 и 1900 МГц. Диапазон GSM шире, чем у AMPS, что позволяет обслуживать куда большее число пользователей. GSM — полнодуплексная система сотовой связи с частотным разделением каналов, как и AMPS. То есть все телефоны передают данные на одной частоте, а принимают — на другой, более высокой (на 55 МГц для GSM и на 80 МГц для AMPS). Однако, в отличие от AMPS, отдельные пары частот в GSM делятся с помощью TDM на временные слоты. Таким образом, ее могут использовать несколько мобильных телефонов.

Для обеспечения звонков нескольких мобильных телефонов каналы GSM намного шире, чем каналы AMPS (200 кГц вместо 30 кГц). На илл. 2.41 показан отдельный 200-килогерцный канал. Работающая в диапазоне 900 МГц GSM насчитывает 124 пары симплексных каналов (каждый шириной 200 кГц) и поддерживает восемь отдельных соединений, используя TDM. Каждому активному в текущий момент устройству выделяется свой временной слот на одной паре каналов. Теоретически каждая сота может поддерживать 992 канала, но многие из них недоступны во избежание конфликтов частот с соседними сотами. На илл. 2.41 все восемь заштрихованных временных слотов выделены для одного соединения, по четыре в каждом направлении. Передача и прием сигналов разнесены по разным временным слотам, поскольку GSM-радиостанции не способны передавать и принимать одновременно, а переключение с одного режима на другой занимает определенное время. Если мобильное устройство привязано к диапазону 890,4/935,4 МГц и требует временного слота 2 для передачи сигнала на базовую станцию, то оно воспользуется четырьмя нижними заштрихованными слотами (и следующими за ними по времени), передавая в каждый слот какие-либо данные, пока не будет отправлена вся информация.



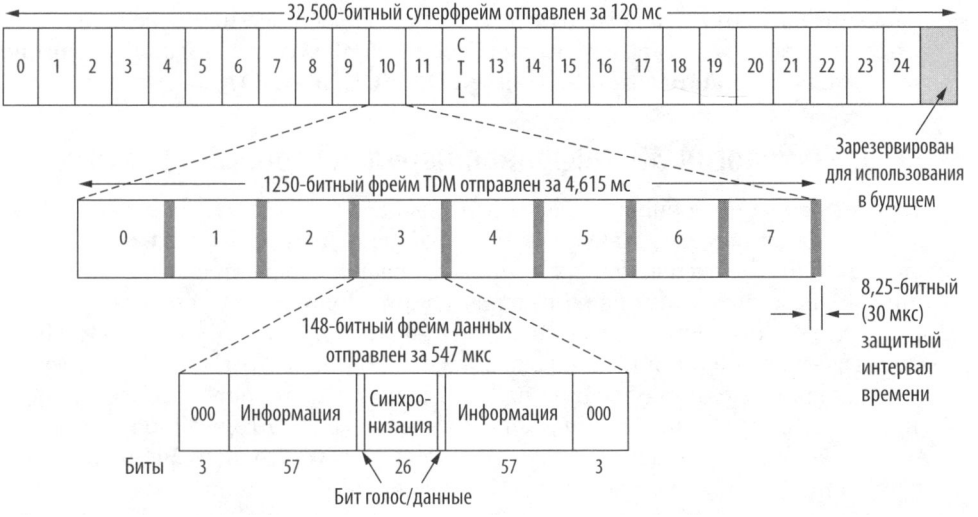

**Илл. 2.41.** 124 частотных канала GSM, каждый из которых использует восьмислотовую TDM-систему

Слоты TDM на илл. 2.41 — часть сложной иерархии фреймов. Каждый слот TDM имеет особую структуру, так же как и группы слотов, образующие суперфреймы. Упрощенная версия этой иерархии приведена на илл. 2.42. Каждый слот TDM состоит из 148-битного фрейма данных, занимающего канал на 577 мкс (включая защитный интервал времени в 30 мкс после каждого слота). Каждый фрейм начинается и завершается тремя битами 0 в целях разграничения фреймов. Он также содержит два 57-битных поля *Информация*; в каждом — контрольный бит, указывающий, для чего предназначено это поле (для голоса или данных). Между полями *Информация* располагается 26-битное поле *Синхронизация*, с помощью которого приемник производит синхронизацию по границам фреймов отправителя.

Фрейм данных передается за 547 мкс, но передатчик может отправлять лишь по одному фрейму каждые 4,615 мс, поскольку делит канал еще с семью другими устройствами. Общая скорость каждого канала, составляющая 270 833 бит/с, делится между восемью пользователями. Впрочем, как и в AMPS, служебные данные «съедают» значительную часть полосы пропускания, оставляя в конечном итоге 24,7 Кбит/с для полезных данных в расчете на каждого пользователя (до коррекции ошибок). После коррекции на голосовые данные остается 13 Кбит/с. И хотя это существенно меньше, чем 64 Кбит/с при PCM для несжатых голосовых сигналов в стационарных телефонных сетях, благодаря сжатию на стороне мобильного устройства можно достичь этого уровня без особой потери качества.

Как видно из илл. 2.42, один фрейм TDM состоит из восьми фреймов данных, а один 120-мс суперфрейм состоит из 26 фреймов TDM. Из них слот 12 используется для управления, а слот 25 зарезервирован для использования в будущем, так что для пользовательского трафика доступно только 24 фрейма.

Впрочем, помимо показанного на илл. 2.42 суперфрейма на 26 слота, используется также (не представленный) суперфрейм на 51 слот: часть из них используется для каналов управления системой. **Широковещательный канал управления (broadcast control channel)** представляет собой непрерывный поток




**Илл. 2.42.** Фрагмент структуры фреймов GSM

выходных сигналов базовой станции, содержащих ее идентификатор и данные о состоянии канала. Все мобильные устройства непрерывно отслеживают мощность сигнала, чтобы узнать о переходе в новую соту.

**Выделенный канал управления (dedicated control channel)** используется для обновления данных о местоположении, регистрации и подготовке звонков. В частности, у каждого BSC есть база данных мобильных устройств, относящихся к нему в данный момент (VLR). Необходимая для обновления VLR информация передается по выделенному каналу управления.

В системе также есть **общий канал управления (common control channel)**, разбитый на три логических подканала. Первый из них — **пейджинговый канал (paging channel)**, используемый базовыми станциями для оповещения о входящих звонках. Все мобильные устройства непосредственно следят за ним в ожидании звонков, на которые необходимо ответить. Второй — **канал произвольного доступа (random access channel)**, через который пользователь может запросить слот в выделенном канале управления. В случае конфликта двух таких запросов они искажаются и их приходится повторить позднее. С помощью выделенного канала управления станция может установить соединение. Оповещение относительно выделенного слота происходит по третьему подканалу — **каналу предоставления доступа (access grant channel)**.

Наконец, GSM отличается от AMPS способом передачи обслуживания. В AMPS MSC производит его без какой-либо помощи со стороны мобильных устройств. При использовании временных слотов GSM телефон большую часть времени ничего не посылает и не принимает. Неиспользуемые слоты дают мобильному устройству возможность измерять качество сигнала от расположенных поблизости базовых станций и отправлять полученную информацию BSC. На основе этой информации BSC определяет, когда мобильный телефон

покидает одну соту и переходит в другую, для своевременной передачи обслуживания. Эта архитектура называется **передачей обслуживания при содействии мобильных устройств (Mobile Assisted HandOff, MAHO)**.

## 2.6.5. Технология 3G: цифровая передача голоса и данных

Первое поколение мобильных телефонов предназначалось для аналоговой передачи голоса, а второе — для цифровой. Третье поколение, **3G**, служит для цифровой передачи голоса *и* данных. К широкому распространению этой технологии привело несколько факторов. Во-первых, когда появился 3G, объем передаваемых данных в стационарных сетях начал превышать объем голосового трафика; аналогичная тенденция наблюдалась и для мобильных устройств. Во-вторых, наметилась тенденция объединения телефонных, интернет- и видеосервисов. Появление смартфонов, начиная с выпущенного в 2007 году iPhone компании Apple, ускорило переход к мобильному интернету. С ростом популярности iPhone неуклонно росли и объемы данных. Изначально iPhone использовал сеть **2.5G** (по сути, немного усовершенствованную сеть 2G), пропускная способность которой была явно недостаточной, чтобы удовлетворить растущие потребности пользователей. Это обусловило переход на технологию 3G, поддерживающую более высокие скорости передачи. Годом позднее компания Apple выпустила обновленную версию iPhone с поддержкой сетей 3G.

Операторы с самого начала пытались продвигаться в направлении 3G путем перехода на технологии, иногда называемые **2.5G**. Одна из таких систем — **EDGE (Enhanced Data rates for GSM Evolution — «усовершенствованный GSM с улучшенной скоростью передачи данных»)**. По сути, это GSM с большим количеством битов на символ, что автоматически ведет к увеличению числа ошибок на символ. Поэтому в EDGE насчитывается девять схем модуляции и коррекции ошибок, возникающих из-за более высокой скорости. Эти схемы различаются задействованной долей полосы пропускания. EDGE — лишь один шаг на пути эволюции от GSM к технологиям 3G, представленным далее.

Начиная с 1992 года МСЭ пытался конкретизировать концепцию 3G, для чего выпустил рабочий проект **IMT-2000** (где IMT означает **International Mobile Telecommunications — Международный стандарт мобильной связи**). Сеть IMT-2000 должна была предоставлять своим пользователям следующие базовые сервисы:

1. Передача голосовых данных в высоком качестве.
2. Обмен сообщениями (вместо электронной почты, факсов, SMS, чатов и т. д.).
3. Мультимедийные сервисы (проигрывание музыки, просмотр видео, фильмов, телепрограмм и т. д.).
4. Доступ в интернет (веб-серфинг, включая страницы с аудио и видео).

В числе дополнительных сервисов: видеоконференции, телеприсутствие, многопользовательские игры и мобильная коммерция (оплата товаров в магазине взмахом мобильного телефона на кассовом терминале). Более того, все эти сервисы должны были предоставляться повсеместно (с автоматическим

соединением через спутник в отсутствие приземной сети), мгновенно (с постоянным подключением) и с гарантированным QoS. Другими словами, журавль в небе.

МСЭ рассчитывал на единую технологию IMT-2000 в масштабах всего земного шара, чтобы производители могли разработать для нее универсальное устройство, продаваемое и используемое повсюду. Единая технология сильно упростила бы положение дел для сетевых операторов и привлекла бы больше пользователей.

Но оказалось, что эти планы были излишне оптимистическими. Число 2000 в названии проекта означало три вещи: (1) год предполагаемого внедрения; (2) частоту в мегагерцах, на которой технология должна была работать; (3) предполагаемую пропускную способность сервиса (в килобитах в секунду). К 2000 году ничего из этого не было реализовано. МСЭ рекомендовал правительствам всех государств зарезервировать диапазон частот 2 ГГц для беспроблемного перемещения устройств из одной страны в другую. Выполнил это требование только Китай. Наконец, стало понятно, что скорость 2 Мбит/с выглядит не слишком реалистичной для *очень* мобильных пользователей (из-за невозможности достаточно быстрой передачи обслуживания). Эта скорость больше подходила для стационарных пользователей в помещении. Для пешеходов достижимой была скорость 384 Кбит/с, для подключений в автомобилях — 144 Кбит/с.

Несмотря на первоначальные неудачи, с тех пор удалось добиться многого. Было предложено несколько вариантов IMT-2000, из которых после отбора осталось два основных. Первый — **WCDMA (Wideband CDMA — широкополосный CDMA)** от компании Ericsson. Его продвигал Европейский союз, где он называется **UMTS**. Второй — **CDMA2000**, предложенный компанией Qualcomm в США.

У этих систем больше сходств, чем различий: обе основаны на широкополосном варианте CDMA. WCDMA использует каналы в 5 МГц, а CDMA2000 — в 1,25 МГц. Если посадить инженеров Ericsson и Qualcomm в одну комнату и попросить разработать единую архитектуру, вероятно, они управятся за час. Но основная проблема, не на инженерном уровне, а на политическом (как всегда). Европе нужна была система, совместимая с GSM, а США — с распространенной там системой IS-95. Каждая из сторон, естественно, поддерживала местную компанию (штаб-квартира Ericsson располагается в Швеции, Qualcomm — в Калифорнии). В результате Ericsson и Qualcomm постоянно сражались в судах по поводу патентов на технологии CDMA. Ситуация осложнилась тем, что UMTS стала единым стандартом 3G со множеством несовместимых между собой вариантов, включая CDMA2000. Эта попытка примирить враждующие лагеря и закрыть глаза на технические противоречия лишь отвлекла внимание от истинной цели всех усилий.

Преимущество WCDMA по сравнению с описанной выше упрощенной схемой CDMA — возможность отправлять данные с различной скоростью независимо друг от друга. В CDMA это достигается естественным образом, путем фиксации скоростей передачи элементарных сигналов и назначения для разных пользователей последовательностей элементарных сигналов разной длины. Например, в WCDMA количество элементов сигнала в секунду равно 3,84, а размер

кодовых последовательностей варьируется от 4 до 256 элементов сигнала. Если код состоит из 256 элементов сигнала, после коррекции ошибок остается около 12 Кбит/с полосы пропускания, и этой пропускной способности вполне достаточно для голосового звонка. Если же код включает 4 элемена сигнала, скорость передачи пользовательских данных достигает 1 Мбит/с. Коды промежуточной длины дают промежуточные скорости. Для достижения скорости в несколько мегабит в секунду мобильное устройство должно использовать более одного канала в 5 МГц одновременно.

Мы сосредоточимся на применении CDMA в сотовых сетях, поскольку это отличительная черта обеих систем. CDMA не использует ни FDM, ни TDM в чистом виде, скорее их смесь, при которой все пользователи осуществляют передачу одновременно в одном диапазоне. Когда концепция CDMA впервые была озвучена, она вызвала в коммерческих кругах примерно ту же реакцию, что у королевы Изабеллы — предложение Колумба достичь Индии, направившись в противоположную сторону. Впрочем, благодаря настойчивости компании Qualcomm CDMA достиг успеха в качестве системы 2G (IS-95) и был проработан настолько, что стал техническим фундаментом 3G.

Для мобильной телефонии недостаточно базового метода CDMA, представленного в разделе 2.4. Мы описали так называемый **синхронный CDMA (synchronous CDMA)**, при котором последовательности элементов сигналов строго ортогональны. Такая архитектура работает, только если все пользователи синхронизированы по начальному времени передачи последовательностей элементов сигналов, как в случае отправки данных от базовой станции мобильному устройству. Базовая станция может передавать последовательности сигналов, начинающиеся строго в одно время, так что сигналы окажутся ортогональными, а значит, их легко будет разделить. Но синхронизировать передачи независимых мобильных телефонов намного сложнее. Если не приложить особые усилия, данные от них поступят на базовую станцию в разное время без каких-либо гарантий ортогональности. Чтобы телефоны отправляли данные на базовую станцию без синхронизации, нужны кодовые последовательности, ортогональные друг другу при всех возможных смещениях, а не только когда они выровнены по времени начала передачи.

И хотя для данного общего случая найти строго ортогональные последовательности невозможно, длинные псевдослучайные последовательности вполне могут подойти. С высокой степенью вероятности им свойственна слабая **перекрестная корреляция (cross-correlation)** друг с другом при любых смещениях. Это значит, что если перемножить последовательности и найти скалярное произведение, результат будет мал (если бы они были ортогональны, он вообще был бы равен нулю). Интуитивно ясно, что случайные последовательности всегда должны различаться между собой. Их произведение дает случайный сигнал с низким значением. Благодаря этому приемник может отфильтровать нежелательные передачи из полученного сигнала. **Автокорреляция (auto-correlation)** псевдослучайных последовательностей, вероятнее всего, также будет низкой (за исключением таковой при нулевом смещении). Это значит, что результат умножения последовательности на сдвинутую по времени собственную копию и суммирования будет мал (за исключением случая, когда сдвиг равен нулю).

Случайная последовательность с задержкой выглядит как совершенно другая последовательность, так что мы возвращаемся к сказанному относительно перекрестной корреляции. В итоге приемник синхронизируется с началом нужной передачи в полученном сигнале.

Благодаря использованию псевдослучайных последовательностей базовая станция может принимать сообщения CDMA от несинхронизированных мобильных устройств. При обсуждении CDMA мы подразумевали, что уровень мощности сигналов от всех мобильных телефонов на стороне приемника одинаков. Если это не так, низкая перекрестная корреляция с мощным сигналом может подавить высокую автокорреляцию со слабым сигналом. Поэтому необходимо контролировать мощность передатчиков мобильных телефонов для минимизации помех между конкурирующими сигналами. Именно эти взаимные помехи и ограничивают пропускную способность систем CDMA.

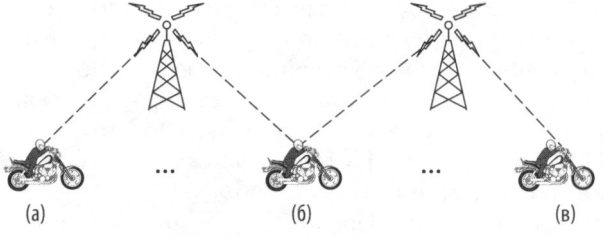
Уровень принимаемого базовой станцией сигнала зависит от того, как далеко находится передатчик и какова мощность его передачи. На разном расстоянии от базовой станции может находиться большое количество мобильных устройств. Для выравнивания мощности получаемых сигналов используется удобный эвристический алгоритм: каждое мобильное устройство отправляет на базовую станцию сигнал с мощностью, обратной мощности сигнала, полученного им от базовой станции. Другими словами, устройство, принимающее слабый сигнал от станции, использует большую мощность, чем устройство, получающее сильный сигнал. Для повышения точности базовая станция дает обратную связь с указанием повысить, снизить или не менять мощность передачи. Это происходит достаточно часто (1500 раз в секунду), поскольку должное управление мощностью сигнала важно для минимизации взаимных помех.

Теперь опишем преимущества CDMA. Во-первых, CDMA может увеличивать пропускную способность за счет использования маленьких промежутков времени, в течение которых часть передатчиков ничего не отправляет. Как при вежливом разговоре: один из собеседников говорит, а второй молчит. В среднем линия занята только 40 % времени. Однако паузы могут быть небольшими и их трудно предсказать. При работе с системами TDM или FDM невозможно переназначать временные слоты или частотные каналы настолько быстро, чтобы воспользоваться этими короткими промежутками тишины. А вот в CDMA для снижения взаимных помех пользователю достаточно ничего не передавать. При этом вероятно, что какая-то часть пользователей не будет постоянно осуществлять передачу в загруженной соте. Таким образом, CDMA использует предполагаемые промежутки тишины для увеличения возможного числа одновременных звонков.

Во-вторых, в случае CDMA все соты используют один набор частот. Чтобы разделять передачи различных пользователей, в CDMA не требуется FDM (в отличие от GSM и AMPS). Это устраняет сложные задачи частотного планирования, повышает пропускную способность, а также упрощает использование базовой станцией нескольких направленных антенн — так называемых **секторных антенн (sectored antenna)** — вместо всенаправленных. Секторные антенны сосредоточивают сигнал в нужном направлении и снижают его уровень (а значит, и помехи) во всех остальных направлениях. Это, в свою очередь, повышает

пропускную способность. Наиболее распространенной является трехсекторная архитектура. Базовая станция должна отслеживать перемещение телефонов из сектора в сектор. В случае CDMA это несложно, поскольку все частоты используются во всех секторах.

В-третьих, CDMA упрощает так называемую **мягкую передачу обслуживания (soft handoff)**, при которой телефон переходит в распоряжение новой базовой станции до того, как отключается от старой. Благодаря этому соединение не прерывается. Мягкая передача обслуживания показана на илл. 2.43. При использовании CDMA она не представляет сложностей, поскольку все частоты используются во всех секторах. Альтернативный вариант — **жесткая передача обслуживания (hard handoff)**, при которой старая базовая станция прекращает поддержку звонка до его перехода на новую. А если новая станция не способна принять управление (например, из-за отсутствия доступной частоты), то звонок внезапно обрывается. Разумеется, пользователи недовольны, но в данной архитектуре это неизбежно. Жесткая передача обслуживания традиционно используется при архитектуре FDM, чтобы избежать затрат на передачу/прием мобильным устройством на двух частотах одновременно.




**Илл. 2.43.** Мягкая передача обслуживания: (а) до; (б) во время; (в) после

## 2.6.6. Технология 4G: коммутация пакетов

В 2008 году МСЭ описал набор стандартов для систем 4G. Поколение **4G** (или **IMT Advanced**) полностью основано на технологиях сетей с коммутацией пакетов, как и его предшественники, например технология **LTE (Long Term Evolution — стандарт «долгосрочного развития»)**. Еще одного предшественника и родственную 4G технологию, 3GPP LTE, иногда называют «4G LTE». Это название может сбить с толку, поскольку «4G» фактически относится к поколению мобильной связи, а каждое поколение может насчитывать несколько стандартов. Например, МСЭ считает стандартом 4G и IMT Advanced, и LTE. К 4G относятся и другие технологии, такие как уже устаревшая WiMAX (IEEE 802.16). Формально LTE и «настоящее» 4G — различные версии стандарта 3GPP (версии 8 и 10 соответственно).

Главное преимущество 4G по сравнению с предыдущими системами 3G — использование коммутации пакетов вместо коммутации каналов. Это возможно благодаря нововведению — **развитому ядру пакетной коммутации (Evolved Packet Core, EPC)**. Фактически EPC является упрощенной IP-сетью, отделяющей голосовой трафик от сети данных. Она производит передачу как голоса, так и данных в IP-пакетах. Следовательно, EPC является сетью **передачи голоса по**

**IP (VoIP, Voice over IP)**; ее ресурсы выделяются при помощи описанных выше вариантов мультиплексирования со статистическим разделением. EPC должна распределять ресурсы между множеством пользователей так, чтобы качество передачи голоса оставалось высоким. Требования к быстродействию LTE включают, помимо прочего, пиковую пропускную способность в 100 Мбит/с входящего и 50 Мбит/с исходящего направления. Чтобы достичь таких высоких скоростей, сети 4G используют набор дополнительных частот: 700, 850, 800 МГц и др. Еще один важный момент в стандарте 4G — «спектральная эффективность», то есть количество битов, которое можно передать за секунду на заданной частоте. Для технологий 4G пиковая спектральная эффективность должна составлять 15 бит/с/Гц для нисходящего соединения и 6,75 бит/с/Гц для восходящего.

Архитектура LTE включает следующие составляющие развитого ядра пакетной коммутации, как показано в главе 1 на илл. 1.19:

1. **Обслуживающий шлюз (Serving Gateway, S-GW)**. SGW перенаправляет пакеты данных, чтобы в случае переключения с одного узла eNodeB на другой пакеты продолжали передаваться на пользовательское устройство.
2. **Узел управления мобильностью (Mobility Management Entity, MME)**. MME отслеживает пользовательское устройство, отправляет на него пейджинговые уведомления и выбирает для него SGW при первом его подключении к сети, а также во время передач обслуживания. Кроме того, он отвечает за аутентификацию устройства.
3. **Сетевой шлюз пакетного обмена данными (Packet Data Network Gateway, P-GW)**. P-GW служит интерфейсом между пользовательским устройством и сетью пакетного обмена данными (то есть сетью с коммутацией пакетов). Он выполняет функции выделения адресов в сети (например, с помощью DHCP), ограничения скорости, фильтрации, углубленного анализа пакетов и правомерного перехвата сообщений. Пользовательские устройства формируют службу, ориентированную на установление соединения со шлюзом пакетного обмена данными. Для этого используется так называемое виртуальное **EPS-соединение (EPS bearer)**, которое устанавливается при подключении устройства к сети.
4. **Сервер абонентов (Home Subscriber Server, HSS)**. MME запрашивает у HSS информацию о том, соответствует ли пользовательское устройство действующему абоненту.

Сеть 4G также включает развитую **сеть радиодоступа (Radio Access Network, RAN)**. В RAN для LTE появились специальные узлы доступа, **eNodeB**, осуществляющие операции на физическом уровне (которому и посвящена эта глава). Также в ней существуют подуровни **управления доступа к среде (Medium Access Control, MAC)**, **управления каналами радиосвязи (Radio Link Control, RLC)** и **протокола управления пакетными данными (Packet Data Control Protocol, PDCP)**, специфичные для архитектуры сотовой сети. Узлы eNodeB осуществляют управление ресурсами и допуском, планирование и другие функции контроля.

В сетях 4G голосовой трафик может передаваться через EPC с помощью **передачи голоса по LTE (Voice over LTE, VoLTE)**. Это позволяет системам связи

отправлять голосовой трафик по сетям с коммутацией пакетов и устраняет любую зависимость от устаревших сетей передачи голоса с коммутацией каналов.

## 2.6.7. Технология 5G

Около 2014 года системы LTE достигли своего пика, и люди начали задумываться: что дальше? Разумеется, за четвертым поколением следует пятое. Вопрос о том, каким именно будет **5G**, подробно обсуждался в работе Эндрюса и др. (Andrews et al., 2014). Через несколько лет под термином «5G» подразумевалось множество разных вещей — в зависимости от аудитории и того, кто говорит. По сути, очередное поколение технологий мобильных телефонных сетей свелось к двум основным факторам: более высокая скорость передачи данных и меньшая задержка, чем у сетей 4G. Конечно, это стало возможным благодаря конкретным технологиям, которые мы обсудим ниже.

Быстродействие сотовых сетей обычно оценивается по **совокупной скорости передачи данных (aggregate data rate)**, она же **пропускная способность на единицу площади (area capacity)**. Это общий объем данных в битах, который данная сеть способна передавать на единицу площади. Одна из целей, поставленных перед 5G, — увеличение пропускной способности на единицу площади на три порядка (то есть в 1000 раз больше, чем у 4G) с помощью сочетания следующих технологий:

1. Сверхуплотнение и разгрузка. Один из простейших способов повышения пропускной способности сети — увеличить количество сот на единицу площади. В то время как в сетях 1G соты были размером в сотни квадратных километров, сети 5G ориентированы на меньшие соты, включая **пикосоты** (диаметром менее 100 м) и даже **фемтосоты** (радиусом действия как у Wi-Fi, в несколько десятков метров). Одно из важнейших преимуществ уменьшения размера сот — возможность повторного использования спектра частот в заданной географической области. Это снижает число абонентов, конкурирующих за ресурсы конкретной базовой станции. Конечно, уменьшение размеров сот имеет и недостатки, в том числе усложнение управления мобильностью пользователей и передачи обслуживания.
2. Повышение полосы пропускания за счет использования волн миллиметрового диапазона. Основная часть спектра в предыдущих технологиях относилась к диапазону от нескольких сотен мегагерц до нескольких гигагерц (что соответствует волнам длиной от нескольких сантиметров до метра). Этот спектр все больше переполняется, особенно в местах скопления людей в час пик. В миллиметровом же диапазоне (20–300 ГГц, с длинами волн менее 10 мм) существуют значительные полосы неиспользуемого спектра. До недавних пор этот спектр считался неподходящим для беспроводной связи, поскольку более короткие волны хуже распространяются. Один из способов решения этой проблемы — использование больших массивов направленных антенн. Это существенный сдвиг в архитектуре по сравнению с предыдущими поколениями сотовых сетей: меняется все, начиная от свойств помех до процесса привязки пользователей к базовым станциям.

3. Повышение спектральной эффективности посредством усовершенствований технологии **MIMO («multiple input, multiple output» — «несколько входов, несколько выходов»)**. MIMO увеличивает пропускную способность радиоканала за счет использования нескольких передающих и принимающих антенн. Это позволяет использовать многолучевое распространение, при котором радиосигнал может достичь приемника двумя или более путями. MIMO впервые стала применяться для Wi-Fi и сотовых технологий 3G примерно в 2006 году. Существует довольно много вариантов MIMO; в первых сотовых стандартах применялась **MU-MIMO (Multi-User MIMO)**. Обычно эти технологии используют разнесенность пользователей в пространстве для нейтрализации взаимных помех, возможных на любом конце беспроводной передачи. **Massive MIMO** — одна из разновидностей MU-MIMO, при которой число антенн базовых станций увеличивается настолько, что их становится намного больше, чем конечных точек. Можно даже использовать трехмерный массив антенн — так называемую технологию **FD-MIMO (Full-Dimension MIMO)**.

Еще одна возможность, которую дает 5G, — **сегментация сети (network slicing)**. Операторы сотовой связи могут создавать многочисленные виртуальные сети поверх одной и той же физической инфраструктуры, выделяя части сети под конкретных потребителей. Части сети (и их ресурсы) распределяются между поставщиками приложений с разными запросами. Например, для приложения, требующего высокой пропускной способности, и для приложения с низкими требованиями можно выделить разные сегменты сети. Растет также популярность таких вспомогательных технологий сегментации сетей, как **программно-определяемые сети (Software-Defined Networking, SDN)** и **виртуализация сетевых функций (Network Functions Virtualization, NFV)**. Мы обсудим эти технологии в следующих главах.

## 2.7. КАБЕЛЬНЫЕ СЕТИ

Стационарные и беспроводные телефонные системы, безусловно, сыграют важную роль в сетевых технологиях будущего, но на сети широкополосного доступа немалое влияние окажут и кабельные системы. Сегодня многие пользователи получают по кабелю услуги телевидения, телефона и интернета. В следующих разделах мы подробно рассмотрим сеть кабельного телевидения и сравним ее с уже изученными телефонными системами. Больше информации вы можете найти в работе Харте (Harte, 2017), а также в стандарте DOCSIS 2018 (в частности, относительно архитектур современных кабельных сетей).

### 2.7.1. История кабельных сетей: ТВ-системы коллективного приема

Кабельное телевидение возникло в конце 1940-х как способ улучшения телевизионного сигнала в сельской или горной местности. Изначально система

состояла из большой антенны, установленной на возвышенности и принимающей телевизионный сигнал из эфира, усилителя — так называемой **головной станции (headend)** — и коаксиального кабеля, ведущего к домам абонентов (илл. 2.44).

**Илл. 2.44.** Первые системы кабельного телевидения

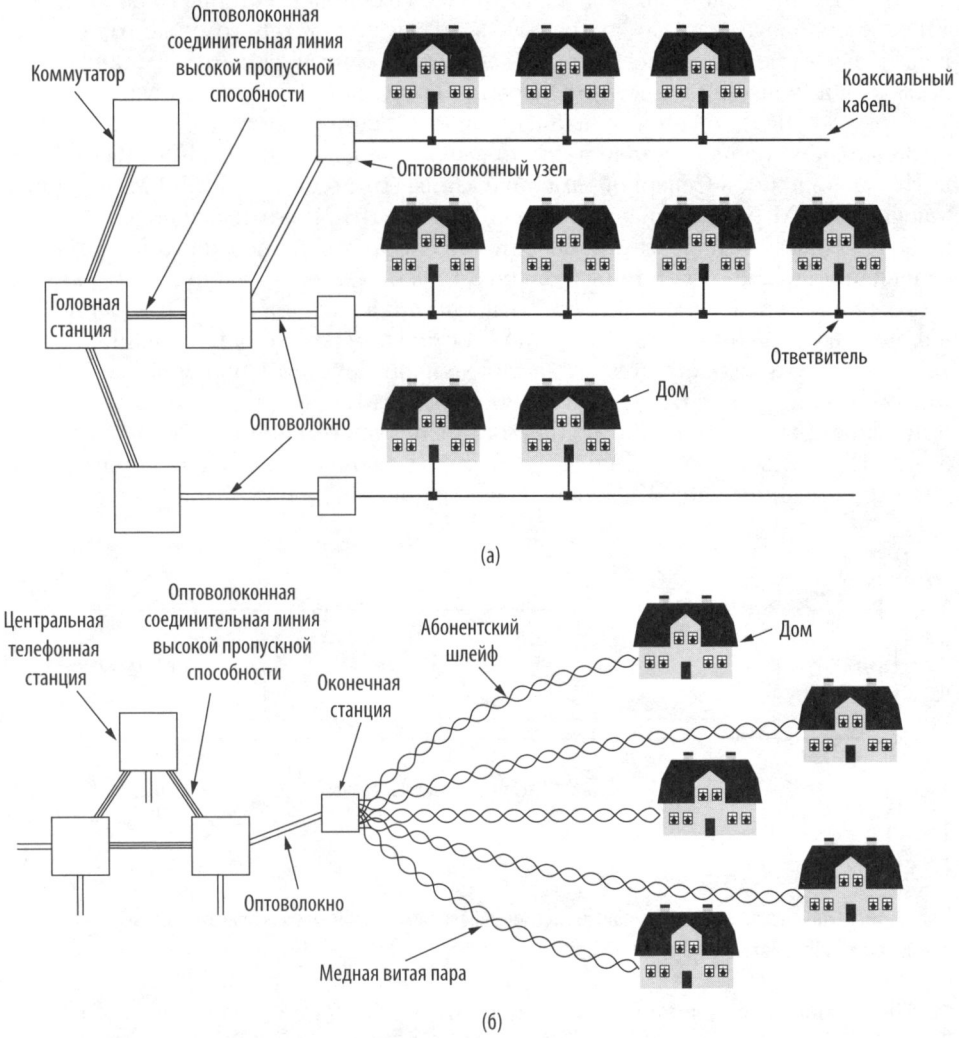
Сначала кабельное телевидение называлось **ТВ-системами коллективного приема (Community Antenna Television, CATV)** и было, по сути, семейным бизнесом. Любой разбирающийся в электронике человек мог настроить телевидение для жителей своего города, готовых оплатить расходы. По мере роста числа абонентов к первоначальной линии подсоединялись дополнительные кабели, а в случае необходимости добавлялись усилители. Передача была односторонней, от головной станции — пользователям. К 1970 году уже существовали тысячи независимых систем.

В 1974 году компания Time Inc. запустила новый канал телевидения Home Box Office, работающий исключительно через кабельную сеть. Затем появились и другие кабельные каналы: спортивные, новостные, кулинарные, исторические, научно-популярные, детские, каналы с фильмами и многие другие. Это привело к двум серьезным изменениям в отрасли. Во-первых, крупные корпорации начали скупать уже существующие кабельные системы и прокладывать новые кабели для привлечения новых пользователей. Во-вторых, возникла потребность в соединении множества систем, зачастую расположенных в разных городах, для распространения новых кабельных каналов. Операторы кабельного телевидения начали прокладывать магистрали между городами, чтобы соединить их в единую систему. Все это напоминало события в телефонной отрасли, происходившие за 80 лет до этого, когда изолированные друг от друга оконечные телефонные станции соединялись, чтобы можно было звонить по межгороду и в другие страны.

## 2.7.2. Широкополосный доступ в интернет по кабелю: сети HFC

Шли годы, кабельные системы росли, а кабели между городами сменились оптоволокном с широкой полосой пропускания, аналогично тому, как это происходило в телефонных системах. Системы, в которых на больших расстояниях

прокладывается оптоволокно, а к домам ведут коаксиальные кабели, называются **комбинированными оптокоаксиальными сетями (Hybrid Fiber Coax, HFC)**. Именно такая архитектура сегодня преобладает в современных кабельных сетях. Оптоволокно проводится все ближе и ближе к домам абонентов, как было описано в разделе, посвященном FTTX. Электронно-оптические преобразователи, служащие интерфейсом между оптоволоконной и электрической частями сети, называются **оптоволоконными узлами (fiber node)**. А поскольку пропускная способность оптоволоконного кабеля гораздо выше, чем коаксиального, один оптоволоконный узел может раздавать поток данных на несколько коаксиальных кабелей. Часть современной системы HFC показана на илл. 2.45 (а).

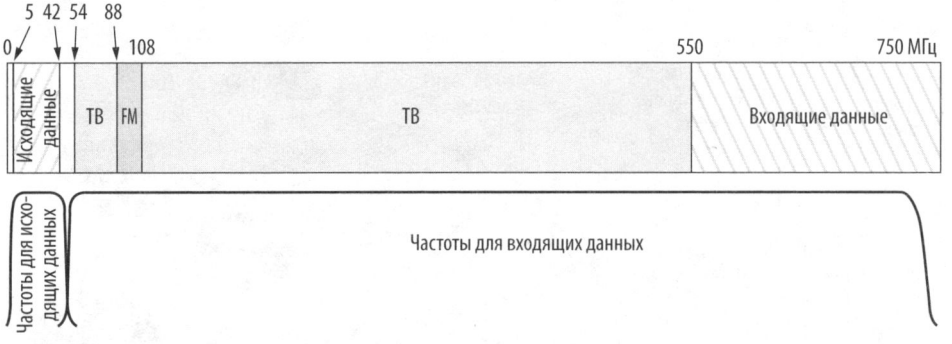



**Илл. 2.45.** (а) Комбинированная оптокоаксиальная кабельная сеть. (б) Стационарная телефонная система

В конце 1990-х многие кабельные операторы начали предоставлять услуги телефонии и доступа в интернет. Для этого им пришлось учесть различия между кабельной и телефонной проводкой. Прежде всего нужно было заменить все односторонние усилители двусторонними для поддержки как исходящей, так и входящей передачи данных. В ранних системах интернет-доступа для входящей передачи данных использовалась кабельная телевизионная сеть, а для исходящей — модемное соединение через телефонную сеть. Это было временное и топорное, но рабочее решение.

Полностью прекратить телевещание и использовать кабельную инфраструктуру исключительно для доступа в интернет операторы не решались. Это вызвало бы гнев немалого числа пользователей (в основном старшего поколения, так как молодежь уже сама отказалась от телевизора). К тому же городские власти зачастую жестко регулируют кабельный контент, так что компании при всем желании не смогли бы этого сделать. В результате пришлось искать способ мирного сосуществования телевидения и интернета в одном кабеле.

Решением проблемы стало мультиплексирование по частоте. Каналы кабельного телевидения в Северной Америке занимают полосу в 54–550 МГц (за исключением FM-радио в диапазоне 88–108 МГц). Ширина этих каналов — 6 МГц, включая защитные полосы; они могут нести сигнал одного обычного аналогового телевизионного канала или нескольких цифровых. В Европе нижняя граница этого диапазона обычно около 65 МГц, а ширина каналов 6–8 МГц (для повышенного разрешения PAL и SECAM), в остальном же схема выделения частот такая же. Нижняя часть полосы частот не используется. Современные кабели отлично себя проявляют на частоте более 550 МГц, иногда даже до 750 МГц или выше. Было принято решение организовать исходящие каналы в полосе 5–42 МГц (в Европе чуть больше), а частоты в конце диапазона использовать для входящих сигналов. Кабельный спектр показан на илл. 2.46.




**Илл. 2.46.** Распределение частот в типовой системе кабельного телевидения, используемой для доступа в интернет

Поскольку все телевизионные сигналы — входящие, можно использовать исходящие усилители, работающие только в диапазоне 5–42 МГц, и входящие, работающие только в диапазоне 54 МГц и выше, как показано на рисунке.

Возникает асимметрия входящей и исходящей полос пропускания, ведь в более высокой полосе (по сравнению с телевизионной) частот доступно больше, чем в более низкой. С другой стороны, большинству пользователей требуется больше входящего трафика, чем исходящего, так что кабельных операторов этот факт вполне устраивает. Как мы видели выше, телефонные компании, как правило, представляют асимметричный DSL-сервис, хотя технических причин для этого у них нет. Операторам приходится обновлять не только усилители, но и головные станции — на смену примитивным усилителям приходят интеллектуальные цифровые компьютерные системы с широкополосным оптоволоконным интерфейсом подключения к провайдеру. Подобные усовершенствованные головные станции иногда называют **оконечной системой кабельных модемов (Cable Modem Termination System, CMTS)**.

## 2.7.3. DOCSIS

Для подключения на «последней миле» кабельные компании используют технологию физического уровня HFC, а также оптоволокно и беспроводные соединения. Технология HFC широко распространена в США, Канаде, Европе и в других странах. В ней используются **DOCSIS (Data Over Cable Service Interface Specification)** — стандарты передачи данных по коаксиальному кабелю, разработанные CableLabs.

Версия 1.0 стандарта DOCSIS была выпущена в 1997 году. Ограничения по входящей и исходящей скорости для DOCSIS 1.0 были 38 Мбит/с, для DOCSIS 1.1 — 9 Мбит/с. С появлением DOCSIS 2.0 (2001) исходящая пропускная способность увеличилась втрое. Поддержка IPv6 и «склейки» каналов исходящего и входящего трафика в DOCSIS 3.0 (2006) резко повысила потенциальную пропускную способность для каждого абонента до сотен мегабит в секунду. DOCSIS 3.1 (2013), в котором появилось мультиплексирование с ортогональным частотным разделением каналов (OFDM), расширилась полоса пропускания и повысилась эффективность, позволил достичь входящей скорости более чем 1 Гбит/с для каждого абонента. Позднее в стандарт DOCSIS 3.1 были внесены обновления. Среди них — полнодуплексные операции (2017), благодаря которым стала возможна мультигигабитная симметричная входная/выходная пропускная способность, а также DOCSIS Low Latency (2018) и прочий функционал для снижения времени задержки.

Комбинированная оптокоаксиальная часть сети (HFC) очень динамична, поскольку операторы кабельных сетей регулярно производят разделение оптоволоконных узлов. Благодаря этому оптоволокно оказывается все ближе к домам абонентов, а число домов в одном узле сокращается. В результате пропускная способность для каждого дома повышается. В некоторых случаях HFC на последнем участке заменяется на «оптоволокно в дом», а многие новые сети сразу строятся по этому принципу.

Абонентам кабельного интернета необходим кабельный модем DOCSIS, играющий роль узла сопряжения домашней сети и сети ISP. Каждый кабельный модем передает данные по одному входящему и одному исходящему каналу.

Выделение каналов происходит с помощью FDM. В DOCSIS 3.0 используется несколько каналов. Обычная схема работы выглядит так: входящий канал шириной 6 или 8 МГц модулируется при помощи QAM-64 (в случае кабеля высокого качества — QAM-256). Использование канала 6 МГц и QAM-64 дает скорость около 36 Мбит/с. С учетом передачи служебных сигналов пропускная способность сети составляет около 27 Мбит/с. При использовании QAM-256 скорость передачи полезных данных равна приблизительно 39 Мбит/с. В Европе значения больше примерно на треть из-за большей полосы пропускания.

Интерфейс между модемом и домашней сетью довольно прост: обычно это Ethernet-соединение. Сегодня многие пользователи интернета подключают кабельный модем к точке доступа Wi-Fi для создания домашней беспроводной сети. В некоторых случаях провайдер предоставляет клиенту отдельное устройство, сочетающее в себе кабельный модем и беспроводную точку доступа. Интерфейс между модемом и остальной сетью ISP сложнее, поскольку требует согласования совместного использования ресурсов множеством абонентов кабельной сети, подключенных к одной головной станции. Совместное использование происходит на канальном, а не физическом уровне, но мы обсудим его в этой главе, чтобы соблюсти последовательность.

## 2.7.4. Совместное использование ресурсов в сетях DOCSIS: узлы и мини-слоты

Существует важное принципиальное различие между HFC-системой с илл. 2.45 (а) и телефонной системой с илл. 2.45 (б). В отдельном жилом микрорайоне один кабель совместно используют многие дома, в то время как в телефонной системе у каждого здания — свой абонентский шлейф. Совместное использование кабелей для телевещания выглядит вполне естественным. Все программы транслируются по кабелю, и неважно, сколько зрителей их смотрит, 10 или 10 000. Однако при совместном использовании того же кабеля для выхода в интернет число пользователей имеет большое значение. Если один из них решит скачать очень большой файл или просмотреть в потоковом режиме фильм в 8K, для остальных эта полоса пропускания будет недоступна. Чем больше пользователей совместно использует один кабель, тем выше конкуренция за полосу пропускания. В телефонных системах этой особенности нет: если вы скачиваете большой файл по ADSL-каналу, это не приносит вашим соседям никаких неудобств. С другой стороны, пропускная способность коаксиального кабеля намного выше, чем у витой пары. На практике в каждый конкретный момент доступная пользователю полоса пропускания во многом зависит от трафика других абонентов, подключенных к тому же кабелю. Далее мы поговорим об этом подробнее.

Кабельные ISP решили эту проблему за счет разделения длинных кабелей и подключения каждого из них напрямую к оптоволоконному узлу. Полоса пропускания между головной станцией и оптоволоконными узлами достаточно велика, так что при небольшом числе абонентов в каждом сегменте кабеля он способен справиться с нужным объемом трафика. Типичный узел 10–15 лет

назад охватывал 500–2000 домов, хотя число домов на узел продолжает снижаться в целях увеличения скорости доступа. Рост числа пользователей и объема трафика за последнее десятилетие привел к необходимости все больше разделять кабели и добавлять все новые оптоволоконные узлы. К 2019 году типичный узел охватывал около 300–500 домов, хотя в некоторых местах провайдеры реализовали архитектуры N + 0 HFC (так называемые «Fiber Deep»), позволяющие снизить это число чуть ли не до 70. Благодаря этому можно отказаться от каскадов усилителей и прокладывать оптоволокно от головных станций сети непосредственно к узлам на последнем сегменте коаксиального кабеля.

После подключения кабельный модем начинает просматривать входящие каналы в поисках специального пакета, периодически отправляемого головной станцией. Пакет содержит системные параметры для модемов, только что начавших работу в сети. При получении этого пакета новый модем оповещает о своем появлении по одному из исходящих каналов. В ответ на это головная станция назначает для него входящий и исходящий каналы. Позже она может их переназначить, если это понадобится для балансировки нагрузки.

В исходящем направлении радиочастотного шума больше, чем во входящем, поскольку система изначально не была рассчитана на передачу данных. Помехи от множества абонентов направляются к головной станции, поэтому в модемной связи используются более консервативные подходы от QPSK до QAM-128, в которых часть символов используется для защиты от ошибок с помощью треллис-модуляции. Благодаря меньшему числу битов на символ в исходящем направлении асимметрия входящей и исходящей скоростей оказывается намного сильнее, чем на илл. 2.46.

Современные DOCSIS-модемы запрашивают временные слоты для передачи, а CMTS выделяет один или несколько слотов в зависимости от загруженности. Одновременные пользователи конкурируют за входящий и исходящий доступ. Для совместного использования исходящей полосы пропускания сеть применяет TDM. Время делится на **мини-слоты**; каждый абонент производит обмен данными в свой мини-слот. Периодически головная станция объявляет о начале нового цикла мини-слотов. Однако модемы получают это оповещение в разное время по мере распространения сигнала по кабелю. Каждый модем сам вычисляет начало первого мини-слота, исходя из своей удаленности от головной станции.

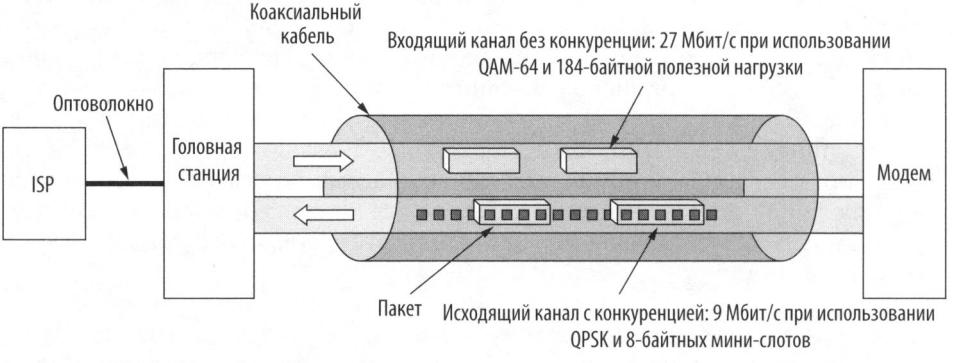
Для правильного расчета времени модему важно определить точное расстояние до головной станции. Для этого он отправляет специальный пакет и засекает время получения ответа. Этот процесс называется **пристрелкой (ranging)**. Любой исходящий пакет при достижении головной станции должен попасть в один или несколько последовательных мини-слотов. Длительность мини-слотов в разных сетях различается. Полезная нагрузка обычно составляет 8 байт.

Во время инициализации головная станция выделяет каждому модему мини-слот для запроса исходящей полосы пропускания. Чтобы отправить пакет, компьютер передает его модему и тот запрашивает необходимое число мини-слотов. Если головная станция одобряет запрос, она посылает по входящему каналу оповещение, сообщающее модему о зарезервированных для его пакета

мини-слотах. Далее начинается отправка пакета в выделенном для него мини-слоте. Посредством специального поля в заголовке можно запросить передачу дополнительных пакетов.

Как правило, нескольким модемам назначается один и тот же мини-слот, что приводит к конфликту (несколько модемов пытается отправить данные одновременно). CDMA разрешает нескольким абонентам совместно использовать один и тот же мини-слот, хотя в результате доступная каждому из них скорость снижается. Можно не использовать CDMA, но тогда подтверждение запроса, вероятно, не будет получено из-за конфликта. В этом случае модем просто ждет некоторое время и повторяет попытку. После каждой последующей неудачи время ожидания удваивается. Для читателя, знакомого с теорией компьютерных сетей: этот алгоритм представляет собой слотированную версию ALOHA с двоичной экспоненциальной задержкой. Использовать Ethernet в кабельной системе не получится, поскольку станции не могут прослушивать такую среду передачи. Мы вернемся к этому вопросу в главе 4.

Входящие каналы работают иначе, чем исходящие. Данные отправляет только головная станция, так что никакой конкуренции и необходимости в мини-слотах нет. Объем входящего трафика обычно намного больше, чем объем исходящего, поэтому размер пакетов фиксирован — 204 байта. Часть этих 204 байт составляет код коррекции ошибок Рида — Соломона и еще некоторые служебные данные. Для пользовательских данных остается 184 байта. Эти значения были выбраны из соображений совместимости с цифровым телевидением, использующим MPEG-2, так что формат телевизионных и входящих каналов данных одинаков. Общая логика соединений показана на илл. 2.47.




**Илл. 2.47.** Типовой вид исходящих и входящих каналов в Северной Америке

## 2.8. СПУТНИКИ СВЯЗИ

В 1950-х и начале 1960-х годов предпринимались попытки построения систем связи путем отражения сигналов от покрытых металлом метеозондов.

К сожалению, принимаемый сигнал был слишком слаб. Затем ВМФ США обратили внимание на своеобразный метеозонд, который постоянно находится в небе, — Луну. В итоге была создана действующая система связи «корабль — берег», основанная на отражении сигналов от естественного спутника Земли.

Продвинуться в этой сфере дальше стало возможно только после запуска первого спутника связи. Ключевое отличие между искусственным и естественным спутником в том, что искусственный способен усиливать сигналы перед отправкой назад на землю, в результате чего эта диковинная технология превращается в мощную систему связи.

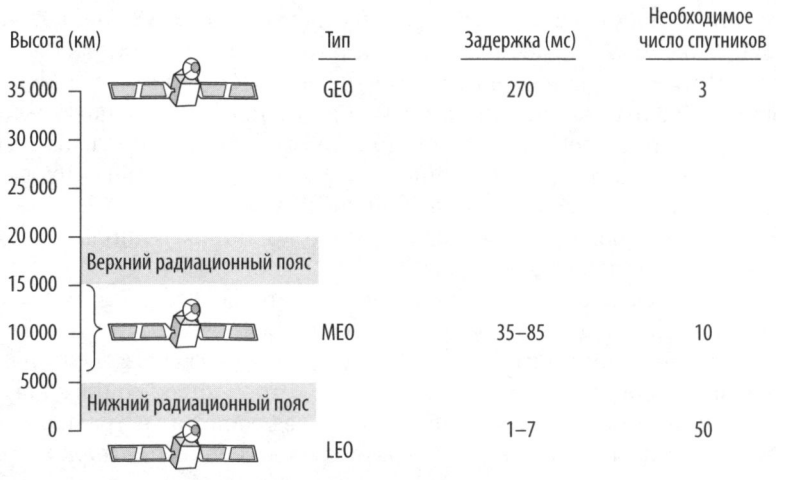
Спутники связи обладают интересными свойствами, делающими их заманчивыми для многих прикладных задач. Проще всего представить спутник связи как своего рода большой повторитель микроволн, висящий в небе. Он содержит несколько **транспондеров (transponder)**, каждый из которых прослушивает свою долю спектра и усиливает полученные сигналы с последующей их ретрансляцией на другой частоте во избежание взаимных помех со входящим сигналом. Подобный режим работы называется **прямой ретрансляцией (bent pipe)**[1]. Чтобы управлять отдельными потоками данных в общем диапазоне и перенаправлять их, можно добавить цифровую обработку. Кроме того, спутник может получать цифровую информацию и ретранслировать ее. Такой способ восстановления сигналов повышает эффективность работы по сравнению с прямой ретрансляцией, ведь при этом спутник не усиливает содержащийся в сигнале шум. Пучки сигналов от спутника могут быть довольно широкими и покрывать значительную долю земной поверхности или узкими, охватывая область лишь в несколько сотен километров в диаметре.

Согласно законам Кеплера, период обращения спутника пропорционален радиусу его орбиты в степени 3/2. Чем выше находится спутник, тем больше период обращения. Близ поверхности Земли он составляет около 90 минут. Следовательно, находящиеся на низкой орбите спутники довольно быстро пропадают из виду (поскольку движутся). Для непрерывного покрытия необходимо множество спутников и наземных антенн для их отслеживания. На высоте примерно 35 800 км период составляет 24 часа, на высоте в 384 000 км — около 1 месяца, в чем может убедиться любой желающий, понаблюдав за Луной.

Период обращения спутника важен, но это не единственный нюанс, который необходимо учитывать, выбирая место его размещения. Еще одна проблема: радиационные пояса Земли (пояса Ван Аллена). Это слои заряженных частиц, удерживаемых магнитным полем Земли и способных довольно быстро разрушить любой спутник, попавший внутрь пояса. С учетом всех факторов остается три области для безопасного размещения спутников. Эти области и некоторые их свойства приведены на илл. 2.48. Ниже мы вкратце опишем спутники, расположенные в каждой из них.

---

[1] Дословно «изогнутая труба». — *Примеч. пер.*




**Илл. 2.48.** Спутники связи и некоторые их характеристики: высота над земной поверхностью, длительность прохождения сигнала туда и обратно, а также число спутников, необходимое для полного покрытия поверхности Земли

## 2.8.1. Геостационарные спутники

В 1945 году писатель-фантаст Артур Кларк вычислил, что спутник, находящийся на высоте 35 800 км на круговой экваториальной орбите, будет казаться неподвижным в небе, так что отслеживать его не нужно (Clarke, 1945). Он описал полноценную систему связи, использующую подобные (пилотируемые) **геостационарные спутники**, включая их орбиты, солнечные батареи, радиочастоты и процедуры запуска. К сожалению, он пришел к выводу, что его идея нереализуема из-за невозможности размещения на орбите хрупких усилителей на вакуумных лампах, требующих большого количества энергии. Поэтому Кларк больше не развивал эту идею, хотя и посвятил ей несколько научно-фантастических рассказов.

Изобретение транзисторов в корне изменило ситуацию, и в июле 1962 года был запущен первый искусственный спутник связи «Телстар». С тех пор спутники связи стали многомиллиардным и единственным высокорентабельным бизнесом, связанным с космическим пространством. Эти спутники, расположенные на большой высоте, часто называют **спутниками GEO (Geostationary Earth Orbit — геостационарная околоземная орбита)**.

При современном уровне технологий не имеет смысла располагать геостационарные спутники чаще чем с интервалом в 2 градуса в 360-градусной экваториальной плоскости. В противном случае возникнут взаимные помехи. Следовательно, в небе могут одновременно находиться только 360/2 = 180 таких спутников. Впрочем, каждый транспондер может использовать несколько частот и схем поляризации для повышения доступной полосы пропускания.

Во избежание хаоса в небе выделением мест для спутников занимается МСЭ. Этот процесс чрезвычайно политизирован. Одни государства, едва вышедшие из

каменного века, требуют «своих» мест на орбите (для дальнейшей перепродажи тому, кто заплатит больше). Другие считают, что государственная собственность не распространяется до Луны и ни одна страна не имеет права на участок орбиты над ее территорией. Ситуация осложняется тем, что коммерческая связь — отнюдь не единственный способ применения таких спутников. Телевизионные компании, правительства, военные — все хотят свой «кусок космического пирога».

Современные спутники бывают довольно большими, весят более 5000 кг и потребляют несколько киловатт электроэнергии, производимой солнечными батареями. Притяжение Солнца, Луны и планет стремится сместить их с назначенных мест и ориентации на орбите. Противодействие этому эффекту называется **поддержанием стационарной орбиты (station keeping)** и осуществляется с помощью установленных на спутнике ракетных двигателей. Когда заканчивается топливо (обычно лет через десять), спутник начинает беспомощно дрейфовать и «кувыркаться», поэтому его приходится отключать. В конце концов он сходит с орбиты, входит в атмосферу и сгорает либо (крайне редко) падает на Землю.

Участки орбиты — далеко не единственное яблоко раздора. Частоты тоже нужно распределять, поскольку передачи по входящим каналам создают помехи для микроволновых устройств. В связи с этим МСЭ выделил определенные полосы частот для спутниковых пользователей (основные приведены на илл. 2.49). Первой из них стала полоса C, доступная для коммерческого спутникового трафика. В ней было выделено два диапазона частот: нижний — для входящего трафика (от спутника) и верхний — для исходящего (на спутник). Для одновременного движения трафика в обоих направлениях необходимы два канала. Эти каналы и так переполнены, поскольку используются в распространенных системах связи для приземных микроволновых соединений. Полосы L и S были добавлены в результате международного соглашения в 2000 году. Впрочем, они довольно узкие и тоже перегружены.

| Полоса | Входящий трафик | Исходящий трафик | Ширина полосы пропускания | Проблемы |
|---|---|---|---|---|
| L | 1,5 ГГц | 1,6 ГГц | 15 МГц | Малая ширина полосы пропускания; переполнена |
| S | 1,9 ГГц | 2,2 ГГц | 70 МГц | Малая ширина полосы пропускания; переполнена |
| C | 4,0 ГГц | 6,0 ГГц | 500 МГц | Помехи от приземной связи |
| Ku | 11 ГГц | 14 ГГц | 500 МГц | Дожди |
| Ka | 20 ГГц | 30 ГГц | 3500 МГц | Дожди; стоимость оборудования |

**Илл. 2.49.** Основные спутниковые полосы частот

Вторая по высоте частот полоса, доступная для коммерческих операторов связи, — полоса Ku (K under; «K нижняя»). Этот диапазон пока еще не слишком перегружен, и на его верхних частотах спутники можно размещать через

1 градус; скорость передачи может достигать более 500 Мбит/с. Впрочем, есть другая проблема: дожди. Вода хорошо поглощает столь короткие микроволны. К счастью, сильные ливни обычно ограничиваются небольшой территорией, так что обойти эту проблему можно за счет нескольких наземных станций, расположенных далеко друг от друга, вместо одной. Но это решение имеет свою цену в виде стоимости дополнительных антенн, кабелей и электроники для быстрого переключения между станциями. Также для коммерческой спутниковой связи была выделена полоса в диапазоне Ka (K above; «K верхняя»), но для ее использования требуется весьма дорогостоящее оборудование. Помимо коммерческих диапазонов, существует также множество военных и правительственных.

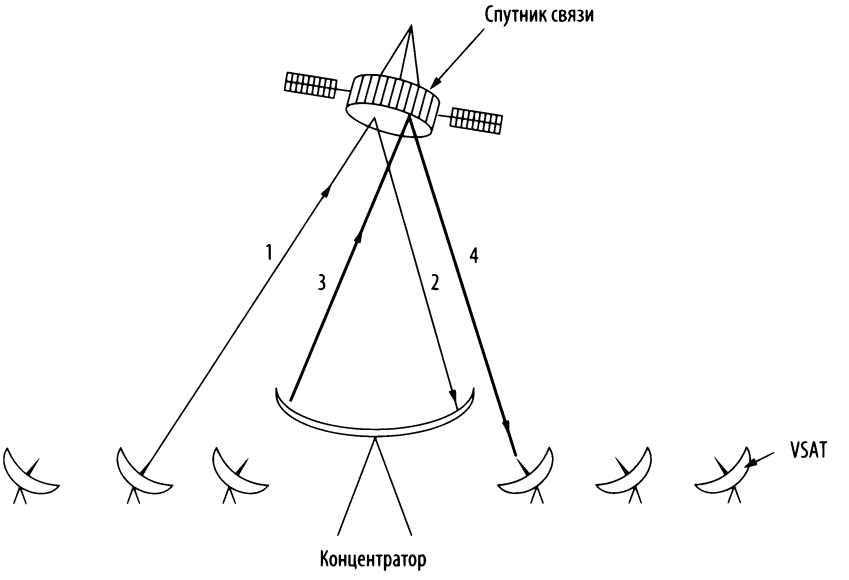
Современный спутник обычно содержит около 40 транспондеров с полосой пропускания, как правило, 36 МГц. Обычно они осуществляют прямую ретрансляцию, но в некоторых последних моделях спутников есть мощности для обработки «на борту», что дает возможность производить более сложные операции. В первых спутниках деление транспондеров по каналам было статическим: полоса пропускания просто разбивалась на фиксированные полосы частот. Теперь пучки сигналов транспондеров делятся по временным слотам и пользователи используют их по очереди. В который раз убеждаемся, что схемы TDM и FDM могут применяться в разнообразных ситуациях.

Пространственный пучок сигнала — его **зона покрытия (footprint)** — первых геостационарных спутников охватывал примерно 1/3 земной поверхности. В результате колоссального снижения стоимости, размеров и требований к мощности микроэлектроники стала возможной гораздо более совершенная стратегия широкополосной трансляции. Каждый спутник снабжен несколькими антеннами и транспондерами. Любой нисходящий пучок можно сфокусировать на небольшой географической области и осуществлять несколько одновременных входящих и исходящих передач. Обычно у подобных **остронаправленных пучков (spot beam)** эллиптическая форма, а размер может быть всего несколько сотен километров в диаметре. Американские спутники связи обычно используют один широкий пучок сигнала для непрерывного участка из 48 штатов и еще два остронаправленных пучка для Аляски и Гавайев.

Новым витком развития в мире спутников связи стало создание недорогих микростанций **VSAT (Very Small Aperture Terminals)**; см. работу Абрамсона (Abramson, 2000). Диаметр антенн этих крошечных терминалов составляет всего 1 м или даже меньше (в отличие от 10 м у стандартных антенн GEO), а их выходная мощность — около 1 Вт. Скорость исходящего соединения обычно не более 1 Мбит/с, а входящего — до нескольких мегабит в секунду. Эта технология применяется в спутниковых системах прямого телевещания для односторонней передачи.

Во многих системах VSAT мощность микростанций недостаточна для прямой связи друг с другом (через спутник, разумеется). Для ретрансляции трафика между различными VSAT необходима особая наземная станция, **концентратор (hub)**, на которой установлена большая антенна с высоким коэффициентом усиления (илл. 2.50). В данной конфигурации либо передатчик, либо приемник имеет огромную антенну и мощный усилитель. Недостатком системы является

более длительная задержка, но есть и существенный плюс — более дешевые терминалы для конечных пользователей.




**Илл. 2.50.** VSAT с использованием концентратора

Системы VSAT имеют колоссальные перспективы применения в сельской местности, особенно в развивающихся странах. Во многих уголках мира нет ни проводной связи, ни сотовых вышек. Бюджет большинства развивающихся стран не позволяет прокладывать телефонные линии в тысячи крошечных деревушек. Возводить сотовые вышки проще, но их нужно соединять проводами с общенациональной телефонной сетью. Установка однометровой тарелки VSAT с питанием от солнечных батарей зачастую становится оптимальным решением. VSAT — это технология, которая может положить конец опутыванию всего мира проводами. Кроме того, она способна обеспечить интернет-доступ пользователям смартфонов в регионах без приземной инфраструктуры (то есть в большей части развивающегося мира).

У спутников связи есть несколько особенностей, резко отличающих их от приземных двухточечных соединений. Прежде всего долгий путь сигнала до геостационарного спутника и обратно приводит к существенной задержке (несмотря на то что сигнал движется со скоростью света, почти 300 000 км/с). В зависимости от расстояния между пользователем и наземной станцией, а также от высоты спутника над горизонтом сквозная задержка составляет 250–300 мс. Обычно время прохождения сигнала туда и обратно составляет 270 мс (540 мс для системы VSAT с концентратором).

Для сравнения: задержка распространения сигнала в приземных микроволновых каналах связи составляет примерно 3 мкс/км, а в коаксиальном или

оптоволоконном кабеле — примерно 5 мкс/км. Разница объясняется тем, что электромагнитные сигналы быстрее распространяются в воздухе, чем в плотной среде.

Важное свойство спутников состоит в том, что они по своей сути широковещательные. Стоимость отправки сообщения на тысячу устройств в зоне покрытия транспондера такая же, как и на одно устройство. В некоторых случаях это очень удобно. Например, спутник может транслировать популярные веб-страницы в кэш множества компьютеров на огромной территории. И хотя широковещательную трансляцию можно имитировать при помощи двухточечных линий связи, спутниковое вещание обойдется намного дешевле. При этом с точки зрения защиты информации спутники крайне небезопасны: все могут слышать всё. Для обеспечения конфиденциальности необходимо шифрование.

Еще одна особенность спутников — стоимость передачи сообщения не зависит от расстояния, проходимого сигналом. Звонок через океан ничуть не дороже, чем звонок в соседний дом. Спутники также отличаются превосходными показателями частоты ошибок, а необходимая инфраструктура развертывается практически мгновенно, что очень важно в случае чрезвычайных ситуаций и для военных.

## 2.8.2. Среднеорбитальные спутники

На гораздо более низких высотах, между двумя радиационными поясами, находятся **среднеорбитальные спутники MEO (Medium-Earth Orbit — средняя околоземная орбита)**. С Земли можно наблюдать, как они медленно перемещаются по долготе. Спутники MEO делают оборот вокруг планеты примерно за 6 часов. Соответственно, их движение по небу необходимо отслеживать. А поскольку они располагаются ниже, чем GEO, зона покрытия земной поверхности у них меньше. Зато для связи с ними достаточно куда более слабого передатчика. В настоящее время MEO применяются в навигационных системах чаще, чем в телекоммуникациях, так что мы не будем останавливаться на них подробно. Примером спутников MEO служит группа из 30 спутников **системы глобального позиционирования (Global Positioning System, GPS)**.

## 2.8.3. Низкоорбитальные спутники

Еще ближе к поверхности земли располагаются **низкоорбитальные спутники LEO (Low-Earth Orbit — низкая околоземная орбита)**. Для создания полноценной системы необходимо большое количество таких спутников, поскольку они быстро перемещаются по орбите. С другой стороны, благодаря низкому расположению LEO наземным станциям не требуется много энергии, а задержка прохождения сигнала туда и обратно составляет всего 40–150 мс. Стоимость запуска также существенно меньше. В этом разделе мы рассмотрим два примера спутниковых группировок, используемых для сервисов голосовой связи: Iridium и Globalstar.

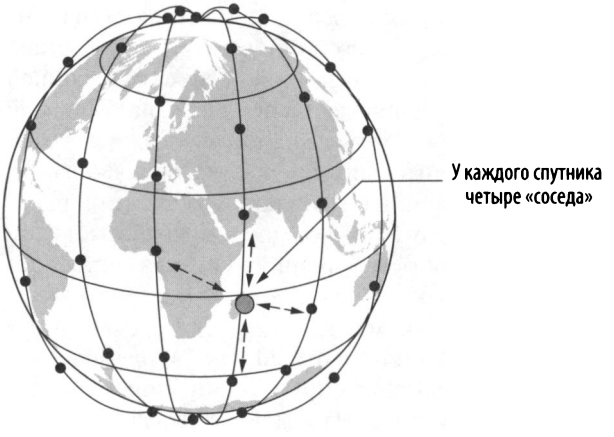
В первые 30 лет спутниковой эры спутники с низкой стационарной орбитой использовались редко, поскольку они слишком быстро появляются и выходят из зоны действия передатчика. В 1990 году Motorola положила начало новой

эпохе, запросив у FCC разрешение на запуск 77 низкоорбитальных спутников для проекта **Iridium** (иридий — 77-й элемент таблицы Менделеева). Позднее план несколько пересмотрели и решили использовать только 66 спутников, так что проект следовало бы переименовать в Dysprosium (диспрозий — 66-й элемент), но, пожалуй, это напоминает название какой-то болезни. Идея заключалась в следующем: как только один спутник исчезает из поля зрения, на смену ему приходит другой. Это предложение вызвало ажиотаж среди остальных телекоммуникационных компаний. Внезапно каждая из них захотела запустить свою цепочку низкоорбитальных спутников.

Спустя семь лет поиска партнеров и финансирования, в ноябре 1998 года, проект был наконец запущен. К сожалению, спрос на крупногабаритные и тяжелые спутниковые телефоны оказался ничтожным, поскольку к этому времени невероятно разрослись мобильные сети. В результате Iridium оказался нерентабельным и обанкротился в августе 1999 года, став одним из самых впечатляющих корпоративных фиаско в истории. Спутники и прочие активы стоимостью $5 млрд позднее были приобретены инвестором за $25 млн на своего рода космической гаражной распродаже. Остальные коммерческие спутниковые проекты ждала та же участь.

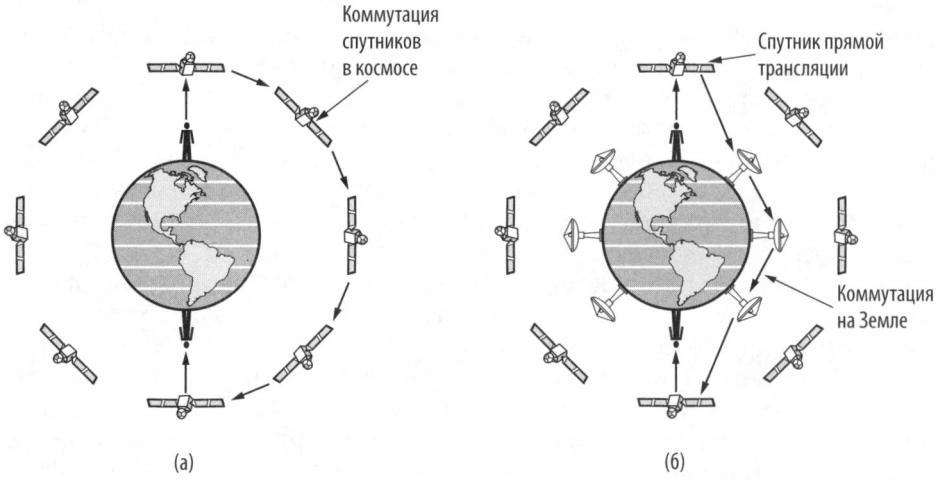
Сервис Iridium был вновь запущен в марте 2001 года и с тех пор демонстрирует стабильный рост. Он предоставляет услуги передачи голоса, данных, пейджинговых сообщений, факсов, а также навигационные сервисы повсюду — на земле, в воздухе и на море. Портативные устройства поддерживают прямую связь со спутниками Iridium. Среди клиентов сервиса — судоходные и авиационные компании, предприятия, занимающиеся поиском нефтяных месторождений, а также путешественники по регионам, где нет телекоммуникационной инфраструктуры (пустыни, горы, Южный полюс и некоторые развивающиеся страны).

Спутники Iridium располагаются на круговых полярных орбитах на высоте 670 км. Они вытянуты в цепочки с севера на юг, по одному спутнику на каждые 32 градуса долготы (илл. 2.51). Каждый спутник насчитывает до 48 ячеек (остронаправленных пучков сигналов) и до 3840 потенциальных каналов, часть которых используется для пейджинга и навигации, а остальные — для данных и голоса.




**Илл. 2.51.** Спутники Iridium располагаются в виде шести цепочек вокруг Земли

Как видно на илл. 2.51, шесть цепочек спутников охватывают всю Землю. Интересная особенность Iridium — связь между удаленными пользователями происходит в космосе. Это показано на илл. 2.52 (а): вызывающий абонент находится на Северном полюсе, спутник расположен непосредственно над его головой. У каждого спутника есть четыре «соседа», с которыми он может обмениваться информацией — два в той же цепочке (показаны на рисунке) и еще два в смежных цепочках (не показаны). Спутники ретранслируют звонок по этой сетке, пока он не попадает к вызываемому абоненту на Южном полюсе.




**Илл. 2.52.** (а) Ретрансляция в космосе. (б) Ретрансляция на Земле

Существует альтернатива Iridium — проект **Globalstar**. Он построен на 48 спутниках LEO, но использует другую схему коммутации. Если Iridium ретранслирует звонки между спутниками (для этого они оснащаются сложным коммутационным оборудованием), Globalstar использует принцип прямой трансляции. Как видно на илл. 2.52 (б), звонок с Северного полюса поступает на спутник, затем отправляется на большую наземную станцию где-то во владениях Санта-Клауса. Далее звонок направляется по приземной сети до ближайшей к вызываемому абоненту наземной станции и попадает к нему по каналу прямой трансляции. Преимущество этой схемы в том, что все наиболее сложные ее составные части находятся на Земле. Это значительно упрощает их обслуживание. Кроме того, большие антенны наземных станций позволяют передавать сильные сигналы и принимать слабые. Благодаря этому можно использовать даже маломощные телефоны. В конце концов, мощность телефонного сигнала составляет всего несколько милливатт, поэтому попадающий на наземную станцию сигнал довольно слаб (даже после усиления спутником).

Постепенно запускаются все новые спутники (около 20 штук в год), включая и более крупные, весом более 5000 кг. Для организаций с ограниченным бюджетом были изобретены очень маленькие спутники. Чтобы повысить доступность

космических исследований, ученые из Калифорнийского политехнического и Стэнфордского университетов в 1999 году совместно описали стандарт для миниатюрных спутников и пусковой установки. Стандарт был призван значительно снизить стоимость запуска; подробнее см. в работе Ньюджента и др. (Nugent et al., 2008). Миниатюрные спутники — **кубсаты (cubesats)** — представляют собой кубики со стороной 10 × 10 × 10 см[1], каждый из которых весит не более килограмма. Стоимость их запуска не превышает $40 000. Пусковая установка обычно отправляется в качестве дополнительной полезной нагрузки при коммерческих полетах в космос. Она представляет собой трубку с кубсатами (до трех штук), которые выстреливаются на орбиту при помощи пружин. Уже запущено несколько десятков кубсатов, и регулярно запускаются новые. Большинство из них связываются с наземными станциями на полосах частот УВЧ и ОВЧ.

Помимо прочего, спутники LEO применяются в создании опорной сети спутникового интернета. Проект OneWeb изначально предполагает группировку из нескольких сотен спутников. В случае успеха будет обеспечен высокоскоростной интернет-доступ в тех местах, где его ранее не было. OneWeb будет работать в диапазоне Ku с использованием технологии Progressive Pitch, при которой спутники слегка поворачиваются во избежание взаимных помех с геостационарными спутниками, передающими в той же полосе частот.

## 2.9. СРАВНЕНИЕ РАЗЛИЧНЫХ СЕТЕЙ ДОСТУПА

Теперь сравним свойства различных типов сетей доступа, о которых мы говорили выше.

### 2.9.1. Наземные сети доступа: кабельные, оптоволоконные и ADSL

У кабельных сетей, FTTH и ADSL больше сходств, чем различий. Они предлагают пользователям похожие сервисы и, вследствие ожесточенной конкуренции, все более сопоставимые цены. На сегодняшний день в опорной сети всегда используется оптоволокно независимо от выбранной технологии; отличия проявляются лишь на последнем участке, на физическом и канальном уровнях. Оптоволоконные и ADSL-провайдеры предоставляют абонентам более стабильный доступ, поскольку пропускная способность выделяется под конкретного пользователя. Согласно последней статистике в США, например, ежегодным отчетам проекта MBA (Measuring Broadband America) Федеральной комиссии по связи (FCC), реальные скорости провайдеров, как правило, соответствуют рекламным обещаниям.

В сетях ADSL и FTTH подключение новых абонентов практически не влияет на качество услуг для остальных пользователей, поскольку каждый получает выделенную линию до самого дома. В то же время абоненты кабельных

[1] Стандарт допускает объединение нескольких кубиков в один спутник. — *Примеч. пер.*

систем совместно используют пропускную способность одного узла, и если кто-то начинает потреблять больше трафика, остальные пользователи ощущают перегруженность сети. Поэтому сегодня поставщики услуг кабельного интернета часто резервируют для пользователей большую пропускную способность, чем нужно. Большинство современных стандартов DOCSIS (например, DOCSIS 3.0) требуют от кабельных модемов деления минимум на четыре канала для достижения входящей скорости примерно в 170 Мбит/с, а исходящей — в 120 Мбит/с (где примерно 10 % пропускной способности занимают служебные данные).

В конечном счете максимальная скорость кабельного интернета ограничена возможностями коаксиального кабеля; в оптоволоконном кабеле доступный спектр намного больше. В кабельной сети скорость передачи данных в узле снижается по мере подключения к нему новых пользователей. В связи с этим кабельные провайдеры разделяют особо загруженные кабели, подключая каждый непосредственно к оптоволоконному узлу; эту практику иногда называют **разделением узлов (node split)**. Как уже упоминалось, число домов из расчета на один узел неуклонно падает по мере прокладывания кабельными провайдерами оптоволокна все ближе к границе сети.

Кабельные, оптоволоконные и ADSL-сети доступны в разных регионах, а быстродействие сети зависит не только от самой технологии, но и от того, как именно она применяется. Большинство домашних пользователей в развитых странах имеют доступ к телефонной линии, но не все живут достаточно близко к оконечной телефонной станции, чтобы пользоваться ADSL. Некоторым приходится обходиться 56-килобитными модемными соединениями, особенно в сельской местности. На самом деле даже в США существуют обширные территории, где линия T1 со скоростью 1,544 Мбит/с является недоступной роскошью. В крупных городах Европы, благодаря большей плотности населения, оптоволоконный интернет на скорости 500 Мбит/с — обычное дело. Иногда скорость даже достигает 1 Гбит/с.

Кабельный интернет есть далеко не у всех. Если в вашем регионе проложен кабель и работает провайдер, то подключиться легко, при этом расстояние до оптоволоконного узла или головной станции значения не имеет. Однако в некоторых регионах, особенно малонаселенных, кабельное и оптоволоконное подключение остается проблемой. По большому счету, высокоскоростной доступ в интернет сегодня все еще зависит от прокладки кабеля или оптоволокна до домов. Все большее разделение узлов кабельных сетей требует проведения оптоволокна вглубь микрорайонов на замену существующей кабельной инфраструктуре. Даже в случае ADSL уже в нескольких километрах от центральной станции скорость существенно падает. Чтобы предоставить пользователям в малонаселенных районах высокую скорость, приходится прокладывать оптоволокно все ближе к краю сети, например до сетевого узла (FTTN). Все это стоит больших денег.

Исторически сложилось, что телефонная инфраструктура (и сети DSL) обычно более надежны, чем коаксиальный кабель. Однако по данным MBA Федеральной комиссии по связи, разрыв между ними постепенно сокращается, а большинство кабельных и DSL-сервисов достигают надежности в «две

девятки» (они доступны 99 % времени, что означает несколько десятков часов простоя в год). Спутники и общегородские беспроводные сети менее надежны. Для сравнения: показатели обычной телефонной сети — «пять девяток», что соответствует всего нескольким минутам недоступности в год (см. работу Бишофа и др.; Bischof et al., 2018).

ADSL, будучи двухточечной средой передачи, по сути, более безопасна, чем кабель. Любой пользователь может считывать проходящие по кабелю пакеты вне зависимости от их истинного адресата. Поэтому любой уважающий себя поставщик услуг кабельного интернета шифрует весь трафик в обоих направлениях. Тем не менее ситуация, в которой сосед видит ваши зашифрованные сообщения, не так безопасна, как ситуация, в которой он не видит их вообще.

## 2.9.2. Спутники и наземные сети

Полезно сравнить спутниковые и наземные сети. Не так давно казалось, что будущее телекоммуникаций — за спутниками. В конце концов, телефонные системы мало поменялись за предыдущие сто лет и вряд ли серьезно поменяются за следующие сто. Такой медленный прогресс во многом был обусловлен регуляторной политикой, предполагающей предоставление достойного сервиса голосовой связи по разумной цене (это и было достигнуто) в обмен на гарантированную доходность вложений. Для передачи данных пользователи могли воспользоваться модемами со скоростью 1200 бит/с. Других вариантов не было.

Появление конкуренции на рынке связи в 1984 году в США и чуть позднее в Европе резко изменило положение дел. Телефонные компании начали заменять свои междугородние сети оптоволоконным кабелем и предоставлять сервисы с высокой пропускной способностью, например ADSL. Кроме того, они перестали завышать цены на междугородние звонки для дотирования местных звонков. Неожиданно наземное оптоволокно стало казаться победителем этого соревнования.

Между тем у спутников связи есть свои ниши на рынке, где оптоволокно не может с ними конкурировать. Во-первых, они с легкостью выигрывают у оптоволокна, если развернуть систему связи необходимо как можно быстрее. Оперативность важна для систем военной связи во время войны, а также при реагировании на чрезвычайные ситуации в мирное время. После мощного землетрясения на Суматре в 2004 году и последующего цунами благодаря спутникам удалось за 24 часа наладить систему связи для сотрудников чрезвычайных служб. Это стало возможным благодаря развитому рынку спутниковой связи. Крупные игроки, например Intelsat, имеющий в распоряжении более 50 спутников, могут предоставлять в аренду мощности практически в любом уголке земного шара. А пользователи уже существующих спутниковых сетей могут легко и быстро установить VSAT на солнечных батареях и получить мегабитный канал связи.

Еще один сегмент рынка — регионы с неразвитой наземной инфраструктурой. Сегодня пользователи хотят быть на связи, куда бы они ни отправились. Мобильные сети охватывают районы с высокой плотностью населения, но плохо

работают в других местах (например, на море или в пустыне). Iridium же предоставляет сервис голосовой связи по всему миру, даже на Южном полюсе. Кроме того, наземная инфраструктура порой обходится недешево в зависимости от рельефа и прав на землю. У Индонезии, например, есть свой спутник для местного телефонного трафика. Запустить его оказалось дешевле, чем протягивать тысячи подводных кабелей между 13 677 островами архипелага.

Третья ниша — широковещательная трансляция. Сообщение, отправленное спутником, могут получить тысячи наземных станций одновременно. Поэтому спутники применяются для распределения значительной доли сетевых телепрограмм по локальным станциям. Сегодня существует масштабный рынок спутникового цифрового теле- и радиовещания непосредственно конечным пользователям, установившим дома или в машине спутниковые приемники. Транслировать можно и множество других видов контента. Например, организация, предоставляющая тысячам дилеров поток рыночных данных (цен на акции, облигации или товары), может значительно сэкономить, используя спутниковую систему вместо других средств связи.

В США есть несколько конкурирующих между собой спутниковых провайдеров, в том числе Hughes (также известный как DISH; в прошлом — EchoStar) и Viasat, работающих в основном со спутниками GEO и MEO, хотя некоторые постепенно переходят на LEO. По данным проекта MBA, в 2016 году они оказались в числе немногих интернет-провайдеров, быстродействие которых постепенно снижалось, скорее всего, из-за роста числа абонентов и ограниченной пропускной способности. Согласно отчету, они предлагали скорости не выше 10 Мбит/с.

Тем не менее в последние годы спутниковый интернет вызывает все больший интерес, особенно в таких сегментах рынка, как онлайн-доступ на борту самолета. Иногда для этого применяется прямой обмен сообщениями с мобильными широкополосными вышками, но при трансокеанских полетах этот вариант не подходит. Еще один метод решения проблемы ограниченной пропускной способности в самолетах состоит в передаче данных группе спутников на геостационарной орбите. Некоторые другие компании — упомянутая выше OneWeb и Boeing — работают над созданием опорной интернет-сети на основе спутников LEO. Это все еще несколько нишевый рынок, поскольку пропускная способность ожидается в районе 50 Мбит/с — намного ниже, чем у наземного интернета.

Похоже, что основной системой связи в будущем станет сочетание оптоволокна и сотовых сетей, а спутники будут использоваться в особых случаях. Впрочем, нужно учитывать экономическую составляющую. Несмотря на то что пропускная способность оптоволокна выше, вполне возможно, что на некоторых рынках спутники смогут успешно конкурировать с ним по цене. Стоимость запуска спутников может резко упасть вследствие развития технологий (например, если какой-нибудь космический аппарат будущего сможет выводить на орбиту десятки спутников за раз), а низкоорбитальные спутники могут внезапно стать популярными. При таком развитии событий неизвестно, победит ли оптоволокно в этом соревновании.

## 2.10. НОРМАТИВНОЕ РЕГУЛИРОВАНИЕ ФИЗИЧЕСКОГО УРОВНЯ

Различные аспекты физического уровня требуют нормативных и управленческих решений, принципиально влияющих на создание и использование технологий. Мы вкратце обсудим текущую деятельность по разработке стратегий как в наземных (то есть телефонных и кабельных), так и в беспроводных сетях.

### 2.10.1. Распределение частот

Основная проблема, связанная со спектром электромагнитных волн, заключается в эффективном и справедливом **распределении частот (spectrum allocation)**. Если разрешить множеству пользователей в пределах региона передавать данные в одном диапазоне, это, скорее всего, приведет к возникновению взаимных помех. Чтобы предотвратить полный хаос, существуют общенациональные и международные соглашения по использованию частот. Всем нужна высокая скорость передачи данных, а значит, и более широкий диапазон частот. Правительства выделяют части спектра для AM- и FM-радио, телевидения и мобильных телефонов, а также для телефонных компаний, полиции, судоходства, навигации, вооруженных сил, государственных служб и многих других конкурирующих пользователей. Одно из агентств МСЭ-R (WRC) пытается координировать выделение частот таким образом, чтобы можно было производить устройства, работающие во многих странах. Впрочем, рекомендации МСЭ-R необязательны для государств и иногда отвергаются Федеральной комиссией по связи, распределяющей частоты в США (обычно потому, что какая-нибудь могущественная политическая структура не хочет отдавать требуемую часть спектра).

Даже когда часть спектра выделяется под конкретные цели, например, для мобильной связи, остается вопрос распределения частот между компаниями. В прошлом широко применялись три алгоритма. Наиболее старый из них, **«конкурс красоты» (beauty contest)**, требует от каждого оператора связи пояснить, почему его предложение лучше всего отвечает общественным интересам. Затем государственные служащие решают, какое из этих предложений нравится им больше всего. Распределение чиновниками объектов стоимостью в миллиарды долларов приводит к взяточничеству, коррупции и непотизму. Более того, даже кристально честному госслужащему, который сочтет, что иностранная компания лучше справится с задачей, чем любая из местных, придется давать много неприятных пояснений.

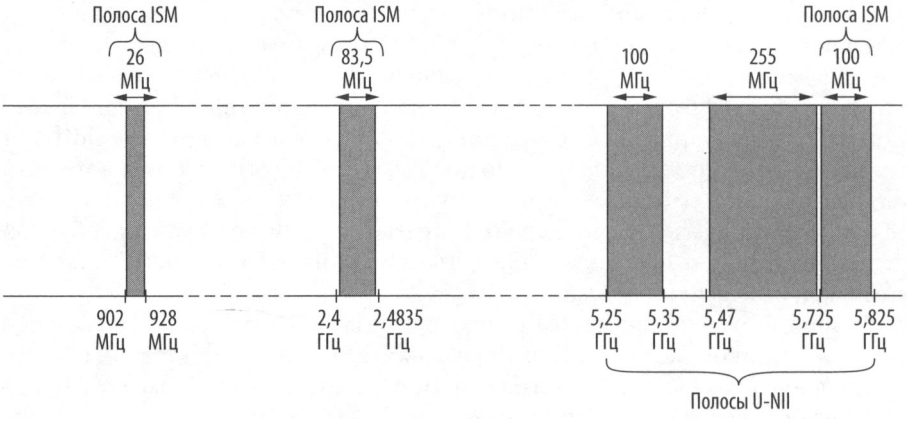
Это наблюдение привело к появлению второго алгоритма: **лотереи (lottery)** среди заинтересованных компаний. Но проблема с лотереями состоит в том, что участвовать в них могут даже компании, которые не собираются использовать выделенный им диапазон. Если конкурс выиграет, скажем, ресторан или обувной магазин, то он может просто перепродать спектр оператору связи с большой прибылью и безо всякого риска.

Ситуация, в которой случайные, но проворные компании получали колоссальные прибыли, никого не устраивала. В результате был придуман третий подход: распределение спектра путем **торгов (auction)**, в которых выигрывает тот, кто предложит большую цену. Британское правительство, распродавая частоты для мобильных 3G-систем, ожидало выручить примерно $4 млрд, а получило около

$40 млрд — все из-за ажиотажа среди операторов связи, до смерти боявшихся упустить лакомый кусочек. Это пробудило алчность в представителях других правительств, и они запустили свои собственные аукционы. Схема сработала, но некоторые операторы связи залезли в долги настолько, что оказались на грани банкротства. Даже в лучшем случае им понадобятся многие годы, чтобы окупить расходы на лицензию.

Существует и совершенно иной подход к выделению частот: вообще их не распределять, а вместо этого всем разрешить передачу на любой частоте, при этом регулируя мощность станций малой дальности, чтобы они не мешали друг другу. Именно поэтому правительства некоторых стран зарезервировали определенные полосы частот — так называемые **ISM («Industrial, Scientific, Medical» — «промышленные, научные, медицинские»)** — для свободного использования. Дистанционное управление гаражными дверями, радиотелефоны, радиоуправляемые игрушки, беспроводные компьютерные мыши и многие другие беспроводные домашние устройства используют полосы ISM. Для минимизации взаимных помех между этими не согласованными между собой устройствами FCC требует ограничения мощности их передатчиков (например, до 1 Вт) и применения методик распределения сигналов по диапазону частот. Кроме того, эти устройства не должны мешать работе радиолокационных станций.

В разных странах расположение полос ISM в спектре отличается. Например, полосы частот, на которых сетевые устройства могут работать без лицензии в США, показаны на илл. 2.53. Полоса частот 900 МГц использовалась в первых версиях 802.11, но уже переполнена. Полоса частот 2,4 ГГц в большинстве стран доступна и широко используется для 802.11b/g и Bluetooth, хотя и подвержена помехам от микроволновых печей и радиолокационных станций. Часть спектра на частоте 5 ГГц включает диапазон **U-NII (Unlicensed National Information Infrastructure — «нелицензируемая национальная информационная инфраструктура»)**. Полосы 5 ГГц относительно малоразвиты, но благодаря наибольшей полосе пропускания и использованию их в таких спецификациях Wi-Fi, как 802.11ac, обрели немалую популярность и тоже перегружены.




**Илл. 2.53.** Полосы ISM и U-NII, используемые в США беспроводными устройствами

Не требующие лицензии полосы частот в последние десятилетия имели оглушительный успех. Возможность бесплатного использования части спектра привела к внедрению массы новшеств в беспроводных LAN и PAN. Об этом свидетельствует повсеместное внедрение таких технологий, как 802.11 и Bluetooth. Сегодня некоторые провайдеры даже предлагают технологию LTE-U, которая состоит в развертывании сотовой сети LTE в нелицензируемом диапазоне. Эта технология позволит мобильным устройствам работать на свободных частотах наряду со спектром, специально выделенным для сотовых сетей. Благодаря LTE-U проводные операторы, размещающие точки доступа Wi-Fi в миллионах домов, смогут превратить сеть точек доступа в сеть сотовых базовых станций. Конечно, при использовании сотовыми телефонами нелицензируемой части спектра могут возникнуть сложности. Например, устройства на этих частотах не должны мешать друг другу и по возможности уже существующим устройствам (incumbent devices) крупных операторов. Также возможны проблемы, связанные с надежностью и производительностью, ведь при использовании LTE-U устройствам приходится идти на компромиссы с другими устройствами в нелицензируемом спектре, от устройств Wi-Fi до радионянь.

Различные изменения в сфере регулирования за последние 10 лет открывают дорогу новым инновациям в сфере беспроводных технологий. Одно из таких изменений в США — тенденция к выделению дополнительных нелицензируемых частей спектра. В 2009 году FCC разрешила свободное использование **«окон» (white spaces)** в районе 700 МГц — выделенных, но не используемых на местном уровне полос частот. К освобождению этих «окон» привел полный переход с аналогового на цифровое телевидение в США в 2010 году. Сложность была в том, что нелицензируемые устройства должны были «видеть» все расположенные поблизости лицензируемые передатчики (включая беспроводные микрофоны), обладающие приоритетом на использование данных частот. Помимо этого, в 2001 году FCC открыла для работы без лицензии диапазон 57–64 ГГц. Это колоссальный кусок спектра, больше, чем все остальные ISM-полосы, вместе взятые. Он вполне способен поддерживать высокоскоростные сети, пригодные для беспроводного потокового телевещания в высоком качестве в пределах гостиной. В районе 60 ГГц радиоволны поглощаются кислородом. Это значит, что сигналы не способны распространяться на большое расстояние, но для сетей малой дальности этот диапазон вполне подходит. Высокие частоты (частота 60 ГГц относится к КВЧ — «миллиметровой» полосе частот, чуть ниже инфракрасного излучения) заставили изготовителей оборудования немало потрудиться, но сегодня соответствующие устройства уже есть на рынке.

В США были перепрофилированы и проданы на торгах и другие полосы спектра, включая 2,5 ГГц и 2,9 ГГц, 3,7–4,2 ГГц (C-диапазон, ранее используемый для спутниковой связи), а также 3,5, 6, 24, 28, 37 и 49 ГГц. FCC рассматривает возможность использования некоторых очень высоких частот для ближней связи, например диапазон 95 ГГц. В конце 2018 года FCC запустила первые торги по 5G и запланировала их продолжение на ближайшие годы. Это откроет для мобильной широкополосной связи значительную часть спектра и позволит получить более высокую пропускную способность, необходимую для приложений потоковой видеопередачи и IoT. На частотах 24 и 28 ГГц выставляется

на продажу примерно по 3000 лицензий. При этом FCC предоставляет скидки малому бизнесу и операторам, работающим в сельской местности. Также запланированы торги на частоты 37, 39 и 49 ГГц. В других странах некоторые из этих диапазонов не подлежат лицензированию. Например, автомобильная промышленность в Германии успешно пролоббировала выделение полосы 3,5 ГГц для частных компаний; остальные европейские страны, вероятно, вскоре последуют ее примеру.

## 2.10.2. Сотовые сети

Любопытно, что политические и незначительные маркетинговые решения оказывают колоссальное влияние на развертывание сотовых сетей в США и Европе. Первую мобильную систему, разработанную в США компанией AT&T, FCC позднее сделало обязательной для всей страны. В результате на всей территории США существовала единая аналоговая система, и приобретенный в Калифорнии телефон успешно работал в Нью-Йорке. И напротив, когда мобильные телефоны появились в Европе, каждая страна разработала свою собственную систему, что привело к полному провалу.

Европа учла допущенную ошибку, и с появлением цифровых систем государственные управления почтово-телеграфной и телефонной связи согласовали и стандартизировали единую систему (GSM), так что любой европейский телефон работает на всей территории Европы. К тому времени в США пришли к выводу, что правительство не должно участвовать в стандартизации, и цифровые системы были отданы на откуп рынку. В результате этого решения изготовители оборудования стали производить разные виды мобильных телефонов, и в США появились две основные — и совершенно несовместимые — мобильные телефонные системы, а также несколько небольших систем.

Несмотря на изначальное преимущество США, процент владения и использования мобильных телефонов в Европе сейчас намного выше. В частности, благодаря единой системе, работающей по всей Европе вне зависимости от оператора, но есть и другие причины.

Вторая область, в которой США и Европа различаются, — малозаметный вопрос емкости пула телефонных номеров. В США номера мобильных и стационарных телефонов не различаются. Следовательно, звонящий никак не может узнать, (212) 234-5678 — это обычный телефон (звонок дешевый или даже бесплатный) или мобильный (звонок стоит достаточно дорого). Чтобы абоненты не боялись звонить, телефонные компании решили, что владельцы мобильных телефонов должны платить за входящие вызовы. В результате многие не хотели покупать мобильные телефоны, опасаясь получить внушительный счет просто за прием звонков. В Европе у мобильных телефонов отдельный код (подобно номерам 800- и 900-), благодаря чему их можно сразу определить. Поэтому для мобильной связи в Европе действует стандартное правило — «платит звонящий» (за исключением международных звонков, где оплата делится между звонящим и вызываемым абонентами).

Третий нюанс, серьезно повлиявший на ситуацию, — широкое распространение в Европе телефонов с предоплаченными разговорами (до 75 % в некоторых

регионах). Их можно купить во многих магазинах и даже через интернет. На баланс карты заранее внесена сумма, скажем, в 20 или 50 евро, и его можно пополнять (с помощью секретного PIN-кода) по мере исчерпания. В результате в Европе практически у любого подростка и даже у маленьких детей есть мобильный телефон (как правило, предоплаченный), так что родители знают, где находится их ребенок, но при этом не боятся, что он наговорит на большую сумму. Редко используемые телефоны обходятся практически бесплатно, поскольку отсутствует ежемесячная абонентская плата и платеж за входящие звонки.

Распродажа на торгах заветных частот для 5G в сочетании с большим количеством технологических новшеств, обсуждавшихся в этой главе, способна полностью перевернуть рынок сотовых сетей в ближайшие несколько лет. Уже сейчас можно наблюдать рост числа **виртуальных операторов мобильных сетей (Mobile Virtual Network Operators, MVNOs)**. Это беспроводные операторы связи, у которых нет своей сетевой инфраструктуры для предоставления услуг пользователям. Постепенно соты уменьшаются, частоты повышаются и наращивается серийный выпуск оборудования для маленьких сот. Пользуясь этим, MVNO платит другим операторам за подключение к их системе. MVNO могут задействовать либо свои собственные компоненты архитектуры LTE, либо инфраструктуру базового оператора связи. Виртуальные операторы, использующие свою собственную опорную сеть, называются полноценными MVNO. Qualcomm, Intel и другие компании разрабатывают типовую архитектуру аппаратного обеспечения маленьких сот. Это может привести к полному разделению сетевой периферии, особенно в сочетании с использованием нелицензируемого спектра. Также наблюдается тенденция к переходу на инфраструктуру «прозрачных» eNodeB: они подключаются к центральной станции, которая предоставляет виртуальные сервисы EPC. Подобная архитектура реализована в проекте M-CORD организации Open Networking Foundation.

## 2.10.3. Телефонная сеть

До 1984 года как локальную, так и междугороднюю связь в большей части США десятилетиями обеспечивала компания Bell System. В 1970-х годах правительство США решило, что такая монополия незаконна, и попыталось в судебном порядке раздробить компанию. Попытка удалась, и 1 января 1984 года компания AT&T разделилась на AT&T Long Lines, 23 местные компании **Bell (Bell Operating Companies, BOC)** и еще несколько фирм. Чтобы обеспечить рентабельность, 23 BOC были сгруппированы в 7 региональных BOC (RBOC). Вся структура связи в США поменялась за одну ночь по решению суда (а *вовсе не* из-за принятого Конгрессом США закона).

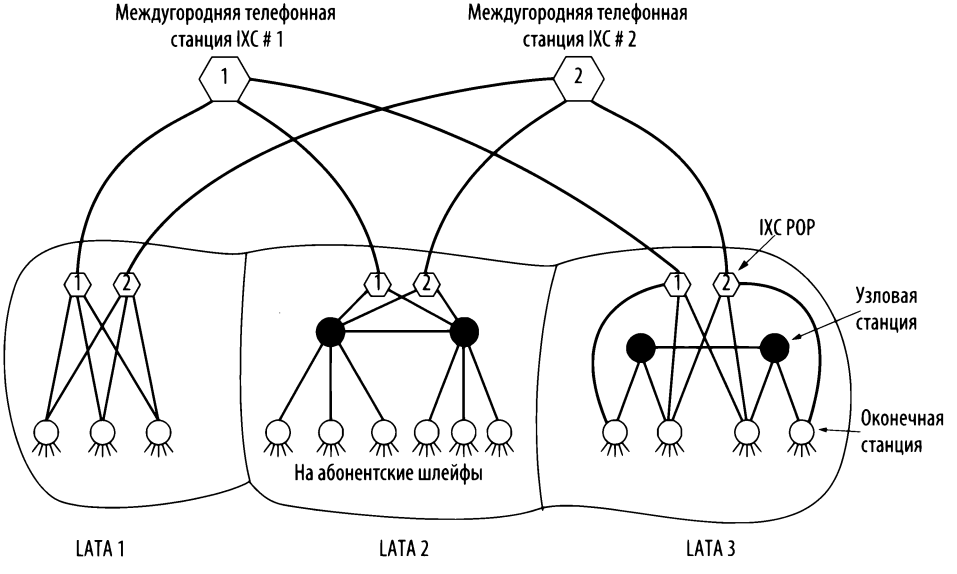
Подробные условия этой дивестиции[1] описаны в **Поправках к окончательному судебному решению (Modification of Final Judgment, MFJ)**; название — образцовый оксюморон. Это событие привело к усилению конкурентной борьбы,

[1] Изъятие капиталовложений, продажа части активов или всей компании. — *Примеч. ред.*

улучшению обслуживания потребителей и снижению расценок на междугородние и международные звонки для частных и бизнес-пользователей. Впрочем, по мере отказа от перекрестного субсидирования за счет междугородних звонков тарифы на местную связь росли, и ей пришлось перейти на самоокупаемость. Конкуренция по аналогичному сценарию внедряется сейчас и во многих других странах.

Совершенно новая конкурентная среда привела к появлению важнейшего элемента в архитектуре телефонной сети. Для четкого распределения обязанностей территория США была разделена на 164 **зоны местного доступа и передачи (Local Access and Transport Area, LATA)**. Размер зоны LATA приблизительно соответствует одному коду телефонной зоны. В каждой LATA был один **оператор местной связи (Local Exchange Carrier, LEC)**, обладающий монополией на традиционные телефонные сервисы в данной зоне. Важнейшими операторами являлись BOC, хотя в некоторых зонах их функции выполняла одна или несколько независимых телефонных компаний (всего их насчитывается 1500).

Новым элементом архитектуры сети стала компания нового типа, обрабатывающая весь трафик между LATA, — **оператор линий межстанционного обмена (IntereXchange Carrier, IXC)**. Первоначально единственным крупным IXC была компания AT&T Long Lines, но сегодня в этой сфере конкурирует несколько серьезных компаний, например Verizon и Sprint. При разделении AT&T особое внимание уделялось равенству всех IXC по качеству линий связи, тарифам и количеству цифр телефонных кодов. На илл. 2.54 показано, как это было реализовано. Мы видим три примера LATA, каждая — с несколькими оконечными телефонными станциями. В LATA 2 и 3 также есть небольшая иерархия узловых телефонных станций (междугородних телефонных станций внутри LATA).




**Илл. 2.54.** Взаимосвязи LATA, LEC и IXC. Все круги представляют собой коммутаторы LEC, а шестиугольники относятся к IXC с соответствующим номером

Любой IXC, желающий обрабатывать инициированные в LATA звонки, может создать в ней коммутатор — **точку присутствия (Point of Presence, POP)**. Местный оператор должен соединить все IXC со всеми оконечными станциями — напрямую, как в LATA 1 и 3, или косвенно, как в LATA 2. Технические и финансовые условия соединения должны быть одинаковы для всех IXC. Соблюдение этого требования позволяет абоненту в LATA 1 свободно выбирать, через какой IXC звонить абонентам в LATA 3.

Согласно поправкам к окончательному судебному решению, IXC было запрещено предлагать сервис местных звонков, а LEC — предлагать услуги связи между LATA (хотя и тем и другим разрешалось заниматься любым другим бизнесом, например ресторанным). В 1984 году это предписание выглядело довольно логичным. К сожалению, по мере развития технологий законы устаревают. Поправки не учитывали ни кабельное телевидение, ни мобильные телефоны. А когда к услугам кабельного телевидения добавился доступ в интернет, а популярность мобильных телефонов резко выросла, LEC и IXC начали скупать кабельных и мобильных операторов или сливаться с ними.

К 1995 году Конгресс США пришел к выводу, что сохранять разделение между разнообразными видами компаний нет смысла. Был подготовлен законопроект, сохранявший конкуренцию, но позволивший операторам кабельного телевидения, местной, дальней и мобильной телефонной связи заниматься ранее недоступным бизнесом. Идея была в том, что любая компания могла предлагать абонентам единый комплексный пакет услуг, включающий кабельное телевидение, телефонию и информационные сервисы, а конкуренция базировалась на качестве и стоимости услуг. Этот законопроект был утвержден в феврале 1996 года и перевернул всю практику регулирования телекоммуникаций. В результате некоторые BOC превратились в IXC, а другие компании, например операторы кабельного телевидения, стали предлагать услуги местной телефонии, конкурируя с LEC.

Интересная особенность закона 1996 года — то, что он требует от LEC **переносимости местных номеров (local number portability)**. Это значит, что пользователь может сохранить телефонный номер при смене компании местной связи. В 2003 году появилась переносимость мобильных номеров (а также переносимость между стационарными и мобильными номерами). Такая возможность устранила важную для многих людей проблему, и они стали чаще менять LEC. В результате конкуренция на рынке телекоммуникаций в США существенно выросла; другие страны последовали их примеру. Некоторые государства нередко наблюдают за подобными экспериментами в США: если все благополучно, они поступают так же, если нет — пробуют что-то иное.

В последние годы в сфере регулирования работы телефонных компаний было относительно спокойно. Основная деятельность развернулась вокруг интернет-провайдеров. Недавно было внесено два регуляторных постановления относительно пробелов в безопасности протокола передачи сигналов **SS7 (Signaling System 7)**. С его помощью сотовые сети обмениваются друг с другом информацией. Этот протокол оказался плохо защищенным, и Конгресс США потребовал от FCC принять соответствующие меры. Еще одно любопытное постановление, связанное с Актом о телекоммуникациях 1996 года, касается классификации

текстовых сообщений. В отличие от голосового трафика в телефонных сетях, который относится к услугам связи (как звонки по телефону), сообщения SMS («текстовые сообщения») классифицируются как услуга обмена информацией (подобно мгновенным сообщениям и прочим интернет-сервисам). В результате SMS регулируются совершенно другим набором законодательных актов, а они определяют все — от тарификации до правил защиты персональной информации.

## 2.11. РЕЗЮМЕ

Физический уровень — основа любой сети. Законы природы налагают на все каналы связи два фундаментальных ограничения, которые определяют их пропускную способность. Это предел Найквиста, относящийся к каналам без помех, и предел Шеннона, описывающий зашумленные каналы.

Среды передачи данных бывают проводными и беспроводными. Основные проводные среды: витая пара, коаксиальный и оптоволоконный кабель; беспроводные — наземные радиоволны, микроволны, инфракрасные волны, лазерные лучи и спутники.

Методы цифровой модуляции позволяют пересылать биты через проводные и беспроводные среды передачи в виде аналоговых сигналов. Линейные коды используются при передаче сигналов в базовой полосе частот, кроме того, сигналы можно передавать в полосе пропускания путем модуляции амплитуды, частоты и фазы несущего сигнала. Несколько пользователей могут совместно использовать канал при помощи мультиплексирования по времени, частоте или с кодовым разделением.

Важнейшим элементом множества глобальных сетей является телефонная система. Ее основные компоненты — абонентские шлейфы, соединительные линии и коммутаторы. ADSL обеспечивает скорость до 40 Мбит/с по абонентскому шлейфу за счет его разделения на множество параллельных вспомогательных несущих частот. Это намного быстрее, чем скорость телефонных модемов. Пассивные оптические сети обеспечивают еще более высокие скорости доступа за счет прокладки оптоволоконного кабеля ближе к домам абонентов. Цифровые данные передаются по соединительным линиям. Чтобы обеспечить множество соединений с высокой пропускной способностью, используется мультиплексирование по длинам волн, а для совместного использования высокоскоростного соединения несколькими пользователями применяется мультиплексирование по времени. Свою роль при этом играет как коммутация пакетов, так и коммутация каналов.

Еще одна система сетевого доступа — кабельная инфраструктура, которая постепенно эволюционировала от коаксиальной в комбинированную оптокоаксиальную сеть. Благодаря этому сегодня многие провайдеры предлагают абонентам скорость до 1 Гбит/с (а в ближайшие годы она может вырасти и до 10 Гбит/с). Эта архитектура отличается от коаксиальной тем, что полоса пропускания может совместно использоваться несколькими абонентами одного узла.

Стационарная телефонная система не подходит для мобильных устройств. На сегодняшний день мобильные телефоны широко применяются для передачи

как голоса, так и данных. С момента внедрения 4G все голосовые данные передаются по сети с коммутацией пакетов. Первое поколение, 1G, было аналоговым, в нем доминировала AMPS. 2G было цифровым; GSM по сей день наиболее распространенная система мобильной связи в мире. 3G — цифровое поколение, основанное на широкополосном CDMA. Главная инновация 4G — переход на базовую сеть с коммутацией пакетов. Отличительные характеристики 5G — меньший размер сот, массовый MIMO и использование гораздо более широкого спектра частот.

Многие аспекты физического уровня зависят не только от самих технологий, но и от регулирующих организаций, например комитетов по стандартизации и регулятивных органов. Большая часть регулирования приходится на беспроводной диапазон. Требования к пропускной способности постоянно растут, и регулятивные органы активно ищут способы более рационального использования спектра. К этим способам относятся перераспределение и продажа с торгов ранее выделенных частот.

## ВОПРОСЫ И ЗАДАЧИ

1. Является ли нефтепровод симплексной, полудуплексной или полнодуплексной системой (или ни одной из них)? А река или связь по рации?
2. В чем состоят преимущества и недостатки (если они есть) оптоволоконного кабеля над медными проводами в качестве среды передачи данных?
3. Какова полоса пропускания 0,1 мкм спектра на длине волны в 1 мкм?
4. Необходимо передать последовательность снимков экрана компьютера по оптоволоконному кабелю. Разрешение экрана составляет 3840 × 2160 пикселей, каждый из которых занимает 24 бита. Какая скорость передачи данных потребуется при частоте 60 снимков экрана в секунду?
5. На илл. 2.5 полоса частот слева уже, чем остальные. Почему?
6. Сегодня операции, выполняемые цифровыми компьютерами, реализуются с помощью электрических сигналов. Как бы изменилась цифровая связь, если бы их удалось эффективно реализовать с помощью лазера? Почему современные компьютеры так не работают?
7. Радиоантенны зачастую работают наилучшим образом, если их диаметр равен длине радиоволны. Диаметр обычной антенны варьируется от 1 см до 1 м. Какому диапазону частот это соответствует?
8. Многолучевое замирание максимально, если расхождение двух лучей по фазе составляет 180 градусов. При какой разности путей достигается максимальное замирание для соединения 1 ГГц длиной 100 км?
9. Лазерный луч 1 мм шириной нацелен на датчик 1 мм шириной, расположенный на расстоянии 100 м на крыше здания. При каком угловом отклонении (в градусах) луч промахнется мимо датчика?
10. Вычислите коэффициенты Фурье для функции $f(t) = t \ \ (0 \le t \le 1)$.

11. Двоичный сигнал на частоте 5 ГГц посылается по каналу с соотношением сигнал/шум в 40 дБ. Какова минимальная верхняя граница максимальной скорости передачи данных? Поясните свой ответ.
12. Измерения канала 3 кГц без помех берутся каждую миллисекунду. Какова максимально возможная скорость передачи данных? Как изменится максимальная скорость передачи данных, если канал зашумлен с соотношением сигнал/шум в 30 дБ?
13. Справедлива ли теорема Найквиста для высококачественного одномодового оптоволоконного кабеля или только для медных проводов?
14. Ширина телевизионных каналов составляет 6 МГц. Сколько бит в секунду можно отправить по такому каналу при использовании четырехуровневых цифровых сигналов? Предполагается, что канал не зашумлен.
15. Какова максимально достижимая скорость передачи данных при отправке двоичного сигнала по каналу 3 кГц с соотношением сигнал/шум в 20 дБ?
16. По каналу с кодированием 4B/5B отправляются данные со скоростью 64 Мбит/с. Какова минимальная полоса пропускания, используемая этим каналом?
17. На квадратурной диаграмме все точки лежат на горизонтальной оси. Какая модуляция при этом используется?
18. Может ли станция, использующая QAM-64, отправлять 3 бита на символ? Поясните почему.
19. Какова минимальная полоса пропускания, необходимая для достижения скорости передачи данных в $B$ бит/с, если сигнал передается с помощью NRZ, MLT-3 или манчестерского кодирования? Поясните свой ответ.
20. Докажите, что при кодировании данных 4B/5B при помощи NRZI тактовые переходы будут происходить по крайней мере через каждые четыре бита.
21. В квадратурной диаграмме модема, аналогичной представленной на илл.. 2.17, точки данных находятся на следующих координатах: (1, 1), (1, –1), (–1, 1) и (–1, –1). Какой скорости (в битах в секунду) может достичь модем с такими параметрами при 1200 символов/с?
22. Сколько частот использует полнодуплексный модем QAM-64?
23. Десять сигналов, требующих по 4000 Гц каждый, мультиплексируются в единый канал с помощью FDM. Какая минимальная полоса пропускания необходима для этого мультиплексированного канала? При этом ширина защитных полос частот составляет по 400 Гц.
24. Пусть A, B и C одновременно передают биты 0 с помощью CDMA-системы с последовательностью элементарных сигналов с илл. 2.22 (a). Какая последовательность элементарных сигналов получится в итоге?
25. При обсуждении ортогональности последовательностей элементарных сигналов CDMA мы утверждали, что если $\mathbf{S} \cdot \mathbf{T} = 0$, то и $\mathbf{S} \cdot \bar{\mathbf{T}} = 0$. Докажите это утверждение.

26. Рассмотрим свойство ортогональности последовательностей элементарных сигналов CDMA с другой стороны. Каждая пара битов из двух последовательностей может совпадать или не совпадать. Объясните свойство ортогональности с точки зрения совпадения/несовпадения битов.
27. Приемник CDMA получает следующие элементарные сигналы: (−1 +1 −3 +1 −1 −3 +1 +1). Предположим, что исходные последовательности представлены на илл. 2.22 (а). Какие станции осуществляли передачу и какие биты отправила каждая из них?
28. На илл. 2.22 представлены четыре станции, способные передавать сигналы. Добавим сюда еще четыре станции. Укажите последовательности элементарных сигналов для них.
29. Какова вероятность того, что нормализованное внутреннее произведение двух случайных последовательностей из 128 элементарных сигналов будет равно 1/4 или больше?
30. Как в телефонных (стационарных), так и в телевизионных сетях множество конечных пользователей подключается к одной оконечной станции, головной станции или оптоволоконному узлу. Могут ли эти системы обеспечить большую устойчивость к ошибкам, чем обычные телефонные сети, которые мы обсуждали в главе 1?
31. Сколько кодов оконечных станций было до 1984 года, когда все обозначались трехзначным кодом региона и первыми тремя цифрами местного номера? Коды регионов начинались с любой цифры от 2 до 9, второй цифрой был 0 или 1, заканчиваться они могли любой цифрой. Первые две цифры всех локальных номеров находились в диапазоне от 2 до 9. Третья цифра могла быть любой.
32. Простая телефонная система состоит из двух оконечных станций, подключенных к одной междугородней станции посредством полнодуплексной соединительной линии в 1 МГц. В среднем за 8-часовой рабочий день с каждого телефона производится четыре звонка. Средняя длительность звонка составляет 6 минут. Десять процентов звонков — междугородние/международные (то есть проходят через междугороднюю телефонную станцию). Какое максимальное число телефонов может поддерживать оконечная станция при ширине канала 4 кГц? Поясните, почему телефонная компания может принять решение о поддержке меньшего числа телефонов, чем максимально позволяет оконечная станция.
33. Число абонентов региональной телефонной компании — 15 млн. Их телефоны подключены к центральной станции при помощи медной витой пары. Средняя длина витых пар составляет 10 км. Поперечное сечение каждой кабельной жилы представляет собой круг диаметром 1 мм, плотность меди равна 9 г/см$^3$, а продать ее можно по $6 за килограмм. Какова суммарная стоимость меди в абонентских шлейфах?
34. Какова максимальная скорость передачи данных, доступная на модеме стандарта V.32, если скорость передачи в бодах составляет 4800 и коррекция ошибок не применяется?

35. Стоимость быстрого микропроцессора упала настолько, что его можно ставить в каждый модем. Как это повлияет на обработку ошибок в телефонных линиях? Можно ли благодаря этому отказаться от проверки/коррекции ошибок на уровне 2?
36. Использующая DMT ADSL-система выделяет 3/4 доступных каналов передачи данных на нисходящее соединение. В каждом канале при этом используется модуляция QAM-64. Какова пропускная способность нисходящего соединения?
37. Почему интервал дискретизации по времени PCM установлен в 125 мкс?
38. Какое соотношение сигнал/шум необходимо для работы системы связи T1 на канале 1 МГц?
39. Сравните максимальную скорость передачи данных канала шириной 4 кГц без помех при использовании:

    а) аналогового кодирования (например, QPSK) при 2 битах на измерение;
    б) системы T1 PCM.

40. В случае сбоя и рассинхронизации система связи T1 пытается заново синхронизироваться по первым битам фреймов. Сколько фреймов необходимо просмотреть (в среднем) для повторной синхронизации с вероятностью ошибки 0,001?
41. Каков процент накладных расходов в канале T1 (то есть какой процент от пропускной способности в 1,544 Мбит/с недоступен для конечного потребителя)? Как он соотносится с аналогичным показателем для каналов OC-1 и OC-768?
42. Синхросигнал SONET сдвигается с приблизительной скоростью 1 такт на $10^9$. Через какое время смещение станет равно 1 биту? Каково практическое значение этого расчета?
43. На илл. 2.35 скорость передачи пользовательских данных для OC-3 равна 148 608 Мбит/с. Выведите это значение из параметров SONET OC-3. Чему будут равны общая скорость, SPE и скорость передачи пользовательских данных для канала OC-3072?
44. Для работы с более низкими скоростями, чем в STS-1, SONET использует систему виртуальных трибутарных потоков (virtual tributaries, VT). VT представляет собой часть пользовательских данных, которую можно вставить во фрейм STS-1, заполняя фрейм данных в сочетании с другими частями пользовательских данных. В VT1.5 используется 3 столбца, в VT2 — 4, в VT3 — 6, а в VT6 — 12 столбцов фрейма STS-1. Какой VT подойдет для:

    а) сервиса DS-1 (1,544 Мбит/с)?
    б) европейского сервиса CEPT-1 (2,048 Мбит/с)?
    в) сервиса DS-2 (6,312 Мбит/с)?

45. Чему равна доступная для пользователя полоса пропускания соединения OC-12c?

46. В чем состоит различие (если оно вообще есть) между демодулятором (часть модема) и кодером (часть кодека)? (В конце концов, и тот и другой служат для преобразования аналоговых сигналов в цифровые.)
47. Каждая из трех сетей с коммутацией пакетов содержит $n$ узлов. Топология первой из них — типа «звезда» с коммутатором в центре, второй — двунаправленное кольцо, а третья — полносвязная (все узлы в ней соединены проводами между собой). Какова длина пути передачи на транзитных участках в худшем, среднем и наилучшем случаях?
48. Сравните время задержки при пересылке сообщения размером $x$ бит по состоящему из $k$-транзитных участков пути в сети с коммутацией каналов и (не слишком загруженной) сети с коммутацией пакетов. Время подготовки канала равно $s$ секунд, задержка распространения сигнала — $d$ секунд на участок, размер пакета — $p$ бит, скорость передачи данных $b$ бит/с. При каких условиях время задержки пакетной сети будет меньше? Объясните, при каких условиях сеть с коммутацией пакетов предпочтительнее сети с коммутацией каналов.
49. Необходимо передать $x$ бит пользовательских данных по состоящему из $k$-транзитных участков пути в сети с коммутацией каналов в виде последовательности пакетов; каждый пакет содержит $p$ бит данных и $h$ бит заголовка, причем $x >> p + h$. Скорость передачи данных канала равна $b$ бит/с; задержкой распространения сигнала можно пренебречь. При каком значении $p$ общее время задержки окажется минимальным?
50. В типичной системе мобильной связи с шестиугольными сотами запрещается повторно использовать полосу частот в смежной соте. Сколько частот можно использовать в конкретной соте, если всего для использования доступно 840 частот?
51. На самом деле соты редко располагаются столь равномерно, как показано на илл. 2.39. Даже форма отдельных сот обычно неправильная. Приведите возможные причины этого явления. Как неправильная форма соты влияет на распределение в ней частот?
52. Оцените приблизительно число микросот PCS диаметром 100 м, необходимое для покрытия всего Сан-Франциско (120 кв. км).
53. Иногда в мобильной сети при пересечении пользователем границы между двумя сотами звонок внезапно обрывается, хотя все передатчики и приемники функционируют идеально. Почему?
54. Телефонные системы на низком уровне обладают звездчатой топологией, все абонентские шлейфы в микрорайоне сходятся в одной оконечной станции. И напротив, система кабельного телевидения состоит из одного длинного кабеля, извивающегося между всеми домами микрорайона. Допустим, телевизионный кабель представляет собой 10-Гбит/с оптоволокно, а не медный провод. Можно ли с его помощью имитировать телефонную модель, при которой у каждого пользователя есть свой личный канал к оконечной станции? Если да, сколько домов с одним телефонным номером в каждом можно подключить к одному такому оптоволоконному кабелю?

55. Система кабельного телевидения насчитывает 100 коммерческих каналов, в которых телевизионные программы чередуются с рекламой. Что это больше напоминает: TDM или FDM?
56. Кабельная компания хочет предоставить интернет-доступ в микрорайон, который состоит из 5000 домов. Она использует коаксиальный кабель и распределение спектра, что обеспечивает пропускную способность 100 Мбит/с в нисходящем направлении из расчета на кабель. Для привлечения абонентов компания гарантирует каждому абоненту постоянную пропускную способность минимум 2 Мбит/с в нисходящем направлении. Опишите, что потребуется этой компании для реализации этих гарантий.
57. Опираясь на распределение спектра из илл. 2.46 и приведенную в тексте информацию, вычислите, какую пропускную способность (в мегабитах в секунду) выделяет кабельная система в исходящем и входящем направлениях.
58. С какой скоростью пользователь кабельного интернета может принимать данные, если сеть больше ничем не загружена? При этом интерфейс пользователя представляет собой:
    а) 10-Мбит/с Ethernet;
    б) 100-Мбит/с Ethernet;
    в) 54-Мбит/с беспроводной канал.
59. 66 низкоорбитальных спутников проекта Iridium разделены на шесть цепочек, огибающих Землю. Период их обращения составляет 90 минут. Каков в этом случае средний промежуток передачи обслуживания для неподвижного передатчика?
60. Представьте спутник, расположенный на высоте геостационарных спутников, орбитальная плоскость которого наклонена к экваториальной плоскости на угол $\varphi$. Будет ли этот спутник казаться неподвижно висящим в небе неподвижному наблюдателю, находящемуся на земной поверхности на широте $\varphi$ градусов (в Северном полушарии)?
61. Вычислите сквозное время прохождения пакета для геостационарных спутников (высота над земной поверхностью 35 800 км), среднеорбитальных спутников (высота 18 000 км) и низкоорбитальных спутников (высота 750 км).
62. Чему будет равно время задержки звонка с Северного полюса на Южный, если он проходит через спутники Iridium? При этом время коммутации на спутниках равно 10 мс, а радиус Земли — 6371 км.
63. Сколько времени займет передача файла размером в 1 Гбайт с одной VSAT на другую при использовании приведенного на илл. 2.50 концентратора? При этом скорость исходящего канала — 1 Мбит/с, входящего — 7 Мбит/с; используется коммутация каналов со временем подготовки канала 1,2 с.
64. Вычислите время передачи в предыдущем упражнении при использовании коммутации пакетов вместо коммутации каналов при размере пакета в 64 Кбайт, задержке коммутации на спутнике и концентраторе 10 мкс и размере заголовка пакета 32 байта.

65. Мультиплексирование нескольких потоков данных STS-1, называемых трибутарными, играет важную роль в SONET. Мультиплексор 3:1 мультиплексирует три входных трибутарных потока STS-1 в один выходной поток STS-3. Это происходит побайтно. То есть роль первых трех выходных байтов играют первые байты трибутарных потоков 1, 2 и 3 и т. д. Напишите программу, моделирующую такой мультиплексор 3:1. Программа должна включать пять процессов. Основной процесс создает четыре процесса: по одному для каждого из трех входных трибутарных потоков STS-1 и один для мультиплексора. Каждый из трибутарных процессов читает фрейм STS-1 из входного файла в виде последовательности из 810 байт и отправляет свои фреймы (побайтно) процессу-мультиплексору. Процесс-мультиплексор получает эти байты и выдает фрейм STS-3 (побайтно) для записи в стандартный поток вывода. Для обмена сообщениями между процессами используйте конвейеры.

66. Напишите программу, реализующую CDMA. Длина последовательности элементарных сигналов — 8, а число передающих станций — 4. Ваша программа должна состоять из трех наборов процессов: четырех процессов-передатчиков (t0, t1, t2 и t3), одного объединяющего процесса и четырех процессов-приемников (r0, r1, r2 и r3). Основная программа, играющая также роль объединяющего процесса, сначала считывает четыре последовательности элементарных сигналов (в биполярном формате) из стандартного потока ввода и последовательность из 4 бит (по одному на каждый процесс-передатчик) и порождает четыре пары процессов-передатчиков и процессов-приемников. Каждой паре процессов-передатчиков/приемников (t0,r0; t1,r1; t2,r2; t3,r3) назначается последовательность элементарных сигналов, а каждому процессу-передатчику соответствует 1 бит (первый бит — процессу t0, второй бит — t1 и т. д.). Далее каждый процесс-передатчик вычисляет сигнал для передачи (последовательность из 8 бит) и отправляет его объединяющему процессу. После получения сигналов от всех четырех процессов-передатчиков объединяющий процесс объединяет их и отправляет результат четырем процессам-приемникам. Каждый процесс-приемник вычисляет полученный бит и выводит его в стандартный поток вывода. Для обмена сообщениями между процессами используйте конвейеры.

# ГЛАВА 3
# Канальный уровень

В этой главе мы рассмотрим принципы построения второго уровня нашей модели — канального уровня (иногда его также называют уровнем передачи данных). Мы обсудим алгоритмы, обеспечивающие надежную и эффективную передачу целых блоков информации, фреймов (сравните с физическим уровнем, задачей которого является передача отдельных битов), между двумя смежными устройствами. Имеются в виду два компьютера, физически соединенные каналом связи, действующим по принципу провода (это может быть коаксиальный кабель, телефонная линия или беспроводной канал). Основное свойство такого канала заключается в том, что биты принимаются в том же порядке, в каком передаются.

На первый взгляд может показаться, что данная проблема настолько проста, что тут нечего и изучать, — устройство *A* просто посылает биты по каналу, а устройство *B* их оттуда извлекает. К сожалению, в каналах связи иногда случаются ошибки при передаче данных. Кроме того, скорость передачи ограниченна, а время распространения сигнала не равно нулю. Это оказывает серьезное влияние на эффективность передачи данных. Все эти факторы должны учитываться при использовании протоколов связи, которым и посвящена эта глава.

После знакомства с ключевыми вопросами устройства канального уровня мы обсудим его протоколы, изучив природу ошибок и методы их обнаружения и исправления. Затем мы рассмотрим ряд протоколов по мере нарастания их сложности. Каждый следующий протокол будет решать все больше задач этого уровня. Наконец, мы приведем несколько примеров протоколов передачи данных на канальном уровне.

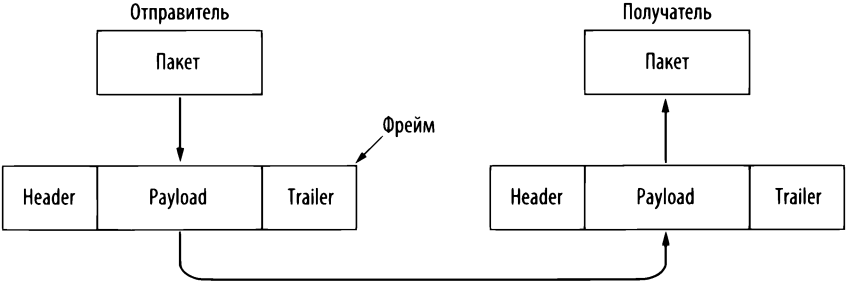
## 3.1. КЛЮЧЕВЫЕ ВОПРОСЫ ПРОЕКТИРОВАНИЯ КАНАЛЬНОГО УРОВНЯ

Канальный уровень использует определенные службы нижележащего физического уровня для отправки и получения битов по потенциально ненадежным коммуникационным каналам, которые могут терять данные. У него есть ряд специфических функций. К ним относятся:

1. Обеспечение строго очерченного служебного интерфейса для сетевого уровня (раздел 3.1.1).

2. Формирование отдельных фреймов из последовательностей байтов (раздел 3.1.2).
3. Обнаружение и исправление ошибок передачи (раздел 3.1.3).
4. Управление потоком данных, исключающее «затопление» медленных приемников быстрыми передатчиками (раздел 3.1.4).

Для этих целей канальный уровень берет пакеты, полученные с сетевого уровня, и вставляет их в специальные **фреймы (frames)**, также называемые **кадрами**, для передачи. В каждом фрейме содержатся поля `Header` (Заголовок), `Payload` (Пользовательские данные) и `Trailer` (Трейлер). Структура фрейма показана на илл. 3.1. Управление фреймами — это основная задача канального уровня. В следующих разделах мы более подробно изучим обозначенные выше цели. Кроме того, в ненадежных беспроводных сетях использование протоколов для улучшения канала связи часто увеличивает производительность в дальнейшем.



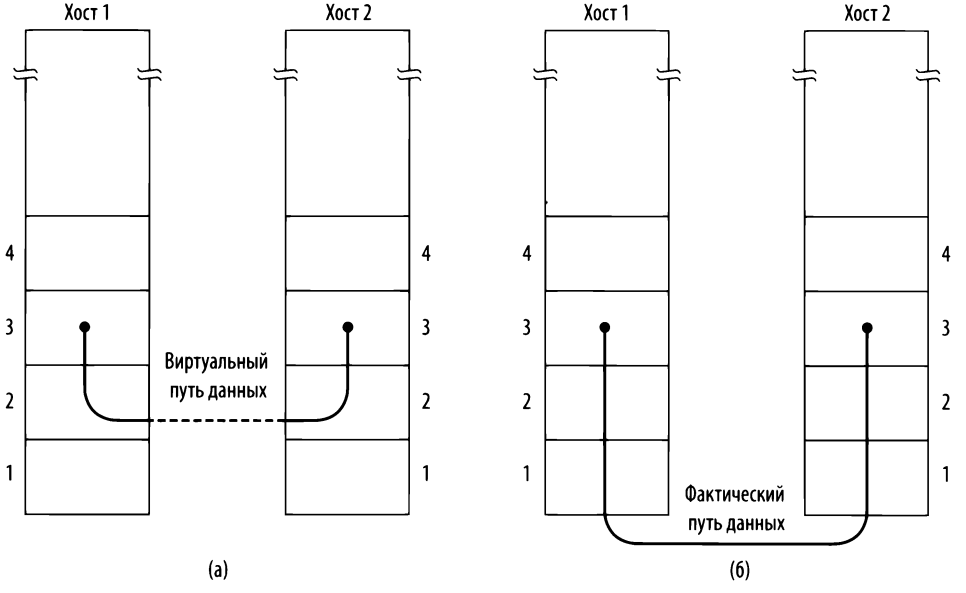

**Илл. 3.1.** Соотношение между пакетами и фреймами

Эта глава в основном посвящена детальному рассмотрению канального уровня и соответствующих протоколов. Но многие обсуждаемые здесь вопросы (например, контроль ошибок и управление потоками) в некоторых сетях относятся также к транспортным и другим протоколам. Обеспечение надежности — это общая цель, для достижения которой должны слаженно работать все уровни. На самом деле во многих сетях эти функции являются прерогативой верхних уровней и вообще не относятся к канальному уровню, который выполняет лишь простейшие задачи. С другой стороны, это не столь важно, потому что принципы все равно остаются неизменными. Аргументом в пользу их рассмотрения на примере канального уровня является то, что здесь они проявляются в наиболее простой форме и их легко показать в деталях.

## 3.1.1. Службы, предоставляемые сетевому уровню

Задача канального уровня заключается в предоставлении служб сетевому. Основная служба канального уровня заключается в обмене данными между сетевыми уровнями отправляющего и целевого устройств. На сетевом уровне отправителя находится некая сущность (назовем ее процессом), которая передает пакеты на

канальный уровень для их дальнейшей отправки по назначению. Канальный уровень должен отправить данные адресату так, чтобы они достигли его сетевого уровня, как показано на илл. 3.2 (а). В действительности данные передаются по пути, показанному на илл. 3.2 (б), однако проще представить себе два канальных уровня, которые связываются друг с другом с помощью протокола передачи данных. Поэтому на протяжении всей главы будет использоваться модель, изображенная на илл. 3.2 (а).




**Илл. 3.2.** (а) Виртуальное соединение. (б) Реальное соединение

Канальный уровень может предоставлять различные службы. Их набор может отличаться в разных протоколах. Как правило, встречаются следующие варианты, которые мы рассмотрим далее более подробно.

1. Служба без подтверждений и без установки соединения.
2. Служба с подтверждениями и без установки соединения.
3. Служба с подтверждениями, ориентированная на установление соединения.

Алгоритм работы службы без подтверждений и без установки соединения заключается в том, что отправляющее устройство посылает независимые фреймы принимающему, которое не отвечает подтверждением о получении. Хороший пример канального уровня, предоставляющего службу такого класса, — Ethernet. Соединения заранее не устанавливаются и не разрываются после передачи. Если какой-либо фрейм теряется из-за шума в линии, то на канальном уровне не предпринимается никаких попыток восстановить его. Этот класс служб приемлем при очень низком уровне ошибок. В этом случае задача восстановления

потерянных данных может быть решена на верхних уровнях. Кроме того, такие службы подходят для трафика в реальном времени, например голосовых или видеоданных, поскольку в этом случае лучше получить данные в искаженном виде, чем с большой задержкой.

Следующим шагом в сторону повышения надежности является служба с подтверждениями и без установки соединения. При ее использовании соединение также не устанавливается, но получение каждого фрейма подтверждается. В результате отправитель знает, дошел ли фрейм до пункта назначения в целости или потерялся. Если в течение установленного интервала времени подтверждения не поступает, фрейм отправляется снова. Такая служба полезна при передаче по ненадежным каналам, в частности беспроводным. Хороший пример — протокол 802.11 (Wi-Fi).

Вероятно, следует отметить, что предоставление подтверждений на канальном уровне является скорее оптимизацией, чем требованием. Сетевой уровень всегда может послать пакет и ожидать подтверждения его доставки. Если за установленный период времени подтверждение не будет получено отправителем, сообщение может быть выслано еще раз. Проблема в том, что эта стратегия зачастую оказывается неэффективной. Обычно фреймы имеют жесткое ограничение максимальной длины, связанное с аппаратными требованиями, к тому же существует определенная задержка доставки. На сетевом уровне эти параметры неизвестны. Сообщения могут делиться, скажем, на 10 фреймов. В среднем два из них потеряются по дороге. Передача данных этим методом может занять очень много времени. Если подтверждать получение отдельных фреймов, то ошибки можно исправлять напрямую и гораздо быстрее. В надежных каналах (например, оптоволоконных) накладные расходы на подтверждение на канальном уровне только снизят пропускную способность, однако для беспроводной связи (ненадежной по своей природе) такие расходы окупятся и уменьшат время передачи длинных сообщений.

Наиболее сложная служба, предоставляемая канальным уровнем, — ориентированная на установление соединения служба с подтверждениями. При использовании данного метода источник и приемник устанавливают соединение перед обменом данными. Каждый посылаемый фрейм нумеруется, а канальный уровень гарантирует, что все фреймы действительно приняты на другой стороне линии, что каждый фрейм принят всего один раз и что все они получены в правильном порядке. Таким образом, ориентированная на установление соединения служба предоставляет процессам сетевого уровня эквивалент надежного потока битов. Она подходит для длинных ненадежных связей, таких как спутниковый канал или междугороднее телефонное соединение. В службе без установления соединения возможно, что при потере подтверждения один и тот же фрейм будет послан (и получен) несколько раз. Это лишняя нагрузка на канал и неразумное расходование полосы пропускания.

При использовании службы, ориентированной на установление соединения, передача данных состоит из трех фаз. В первой фазе возникает соединение, при этом обе стороны инициализируют переменные и счетчики, необходимые для слежения за тем, какие фреймы уже приняты, а какие — еще нет. Во второй фазе передаются фреймы с данными. Наконец, в третьей соединение разрывается,

при этом освобождаются все переменные, буферы и прочие ресурсы, использовавшиеся его для поддержания.

## 3.1.2. Формирование фрейма

Для обслуживания сетевого уровня канальный уровень использует службы, предоставляемые ему физическим уровнем. Физический уровень принимает необработанный поток битов и пытается передать его по назначению. Если канал зашумлен (как обычно бывает с беспроводными и большинством проводных соединений), то на физическом уровне добавляются избыточные сигналы, чтобы снизить количество ошибок до допустимого уровня. Однако поток битов, получаемый на уровне передачи данных, не застрахован от ошибок. У некоторых битов могут быть другие значения, количество принятых битов может быть меньше, равно или больше числа переданных. Канальный уровень должен обнаружить ошибки и, если нужно, исправить их.

Обычно канальный уровень разделяет поток битов на отдельные фреймы и вычисляет для каждого из них короткий маркер, называемый контрольной суммой. Контрольная сумма добавляется во фрейм перед тем, как он пересылается дальше. (Алгоритмы подсчета контрольных сумм представлены ниже в этой главе.) Когда целевое устройство получает фрейм, контрольная сумма вычисляется заново на его основе. Если она отличается от содержащейся во фрейме, то канальный уровень понимает, что при передаче произошла ошибка, и принимает меры (например, игнорирует испорченный фрейм и посылает отправляющему устройству сообщение об ошибке).

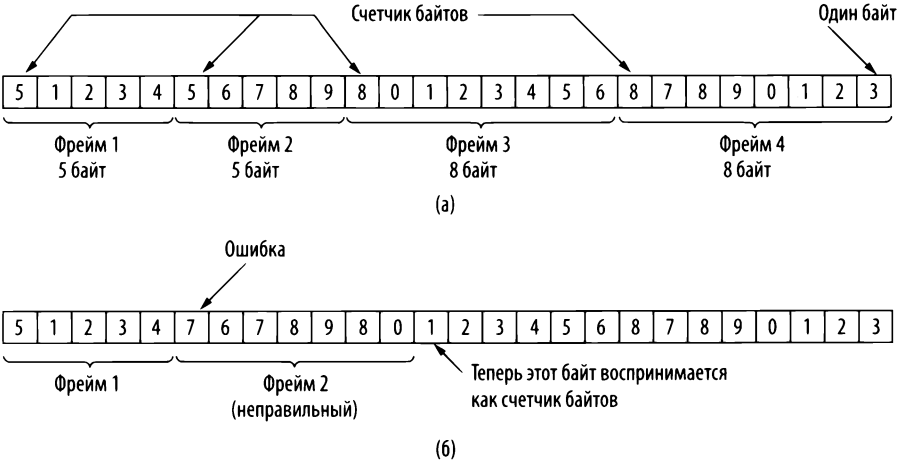
Разбиение потока битов на отдельные фреймы представляет собой более сложную задачу, чем это может показаться на первый взгляд. В хорошей системе приемник с легкостью находит отметки начала новых фреймов, минимально нагружая полосу пропускания. Мы рассмотрим четыре метода маркировки границ фреймов.

1. Подсчет байтов.
2. Флаговые байты с байт-стаффингом.
3. Использование сигнальных битов с бит-стаффингом.
4. Применение запрещенных сигналов физического уровня.

Первый метод формирования фреймов заключается в указании количества байтов фрейма в поле заголовка. Канальный уровень на принимающем устройстве видит это поле, узнает, сколько байтов последует, и таким образом определяет, где находится конец фрейма. Этот прием проиллюстрирован на илл. 3.3 (а) для четырех небольших фреймов размером 5, 5, 8 и 8 байт.

Недостаток такой системы в том, что при передаче может быть искажен сам счетчик. Например, если при подсчете байтов второго фрейма произойдет ошибка в единственном бите и число 5 превратится в 7, как показано на илл. 3.3 (б), то целевое устройство потеряет синхронизацию и не сможет правильно обнаружить начало следующего фрейма. Даже если контрольная сумма не сойдется (скорее всего) и устройство поймет, что фрейм принят неверно, то оно все равно

не сможет определить начало следующего фрейма. Запрашивать повторную передачу фрейма бесполезно, поскольку неизвестно, сколько байтов нужно пропустить до начала повторной передачи. По этой причине на сегодняшний день метод подсчета байтов отдельно практически не применяется.



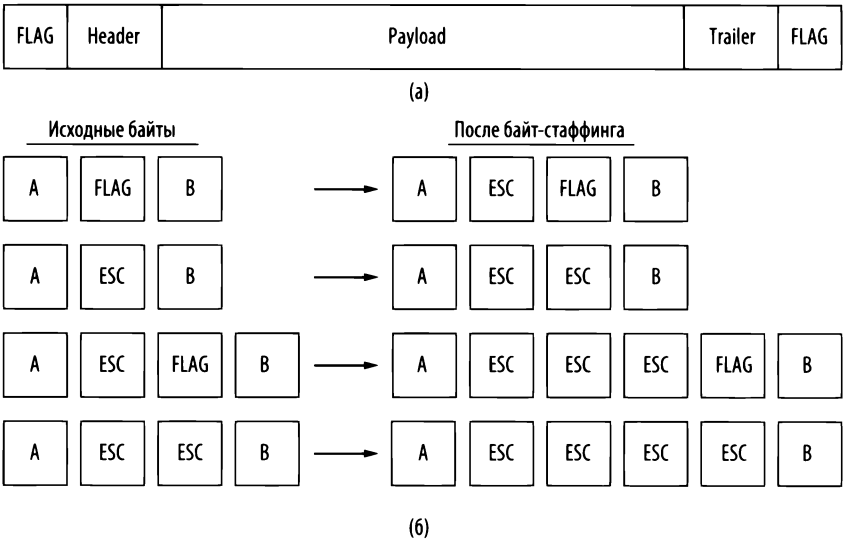

**Илл. 3.3.** Поток байтов: (а) без ошибок; (б) с одной ошибкой

Второй метод формирования фреймов решает проблему восстановления синхронизации после сбоя при помощи маркировки начала и конца каждого фрейма специальными байтами. Зачастую в качестве разделителя используется один и тот же байт, называемый **флаговым (flag byte)**. Он устанавливается в начальной и конечной точке фрейма. Этот байт помечен на илл. 3.4 (а) как FLAG. Два соседних флаговых байта говорят о том, что закончился один фрейм и начался другой. Таким образом, если приемник теряет синхронизацию, ему необходимо просто найти два флаговых байта, с помощью которых он распознает конец текущего фрейма и начало следующего.

Однако одна проблема все же остается. В передаваемых данных, особенно если это двоичные данные (например, фотографии или музыка), запросто может встретиться последовательность, используемая в качестве флагового байта. Возникновение такой ситуации, скорее всего, собьет синхронизацию. Один из способов решения проблемы — добавление специального escape-символа (знака переключения кода, ESC) непосредственно перед случайно совпавшим флаговым байтом внутри фрейма. Таким образом, настоящий флаг можно отличить по наличию или отсутствию перед ним ESC. Канальный уровень получателя вначале убирает эти escape-символы, затем передает фрейм на сетевой уровень. Такой метод называется **байт-стаффингом (byte stuffing)**.

Следующий логичный вопрос: а что, если и символ ESC случайно окажется в данных? Решение такое же: вставить перед фиктивным escape-символом настоящий. На стороне получателя первый символ ESC будет удален, а следующий байт данных останется, даже если это еще один байт ESC или флаговый байт.

На илл. 3.4 (б) показаны некоторые примеры. В любом случае байтовая последовательность после ее очищения от вставленных символов в точности совпадает с исходной. Найти границу фрейма можно все так же по двум последовательным флаговым байтам до удаления дополнительных символов ESC.




**Илл. 3.4.** (а) Фрейм, ограниченный флаговыми байтами. (б) Четыре примера байтовых последовательностей до и после байт-стаффинга

Схема байт-стаффинга, представленная на илл. 3.4, — это немного упрощенная модель протокола **PPP (Point-to-Point Protocol, протокол «точка-точка»)**, который служит для передачи пакетов по коммуникационным каналам и широко используется в сети интернет. Мы подробно обсудим протокол PPP в разделе 3.5.1.

Третий метод разделения потока битов на фреймы позволяет обойти недостатки байт-стаффинга, который обязывает использовать исключительно 8-битные байты. Делить данные на фреймы можно на уровне битов, причем фреймы могут содержать произвольное число битов и состоять из блоков любого размера. Данный метод был разработан для некогда популярного протокола **HDLC (High-level Data Link Control — высокоуровневый протокол управления каналом передачи данных)**. Каждый фрейм начинается и завершается специальной последовательностью битов, 01111110 (или 0x7E в шестнадцатеричной системе). Это все тот же флаговый байт. Если в битовом потоке встретится пять единиц подряд, уровень передачи данных автоматически вставит в выходной поток нулевой бит. **Бит-стаффинг (bit stuffing)** аналогичен байт-стаффингу, при котором во фрейм вставляется escape-символ перед случайным флагом. Он также гарантирует минимальную плотность передачи, помогающую сохранять синхронизацию на физическом уровне. По этой причине бит-стаффинг применяется в протоколе USB.

Когда принимающая сторона встречает пять единиц подряд, за которыми следует ноль, она автоматически удаляет его. Бит-стаффинг, как и байт-стаффинг, является абсолютно прозрачным для сетевого уровня обоих устройств. Если флаговая последовательность 01111110 встречается в данных пользователя, она передается в виде 011111010, но в памяти целевого устройства сохраняется в исходном виде: 01111110. При этом вышележащие уровни остаются в полном неведении о применении бит-стаффинга. На илл. 3.5 приведен пример этого метода.

**Илл. 3.5.** Бит-стаффинг. (а) Исходные данные. (б) Данные на линии. (в) Данные, сохраненные в памяти после удаления вставленных битов

Благодаря бит-стаффингу границы между двумя фреймами могут быть безошибочно распознаны с помощью флаговой последовательности. Таким образом, если принимающая сторона потеряет границы фреймов, ей нужно всего лишь отыскать в полученном потоке битов флаговый байт, поскольку он встречается только на границах фреймов и никогда не встречается в данных пользователя.

Побочный эффект как бит-стаффинга, так и байт-стаффинга состоит в том, что длина фрейма зависит от содержимого, то есть от входящих в него данных. Например, если в данных нет флаговых байтов, то 100 байт можно передать во фрейме размером приблизительно 100 байт. Если же данные состоят исключительно из флаговых байтов, то перед каждым из них вставляется ESC и длина фрейма увеличивается примерно до 200 байт. При бит-стаффинге увеличение составляет около 12,5 %, так как к каждому байту добавляется 1 бит.

Последний метод формирования фреймов напрямую связан с особенностями физического уровня. В главе 2 мы узнали, что при кодировании битов в виде сигналов для облегчения работы получающей стороны добавляются избыточные данные. Это означает, что в самих данных некоторые сигналы не появляются. Например, в линии 4B/5B четыре бита данных сопоставляются с пятью сигнальными битами, для того чтобы гарантировать удовлетворительную передачу. Таким образом, 16 из 32 возможных сигналов не используются. Некоторые из зарезервированных сигналов можно применять для обозначения начальной и конечной границы фреймов. Фактически для разграничения фреймов можно применять «нарушения правил кодирования» (неправильные символы). Преимущество этого метода в том, что использование зарезервированных сигналов делает поиск границ фрейма чрезвычайно простым. При этом заполнять данные дополнительными байтами или битами не требуется.

Во многих протоколах канального уровня для повышения безопасности используются различные сочетания описанных методов. В протоколах Ethernet и 802.11 фрейм часто начинается с четко определенного шаблона — **преамбулы (preamble)**. Этот шаблон может быть довольно длинным (для 802.11 это обычно

72 бита), что упрощает адресату задачу приема пакета. Вслед за преамбулой в заголовке передается поле длины (счетчик битов). Он нужен для обнаружения конца фрейма.

### 3.1.3. Обработка ошибок

Решив проблему маркировки начала и конца фрейма, мы сталкиваемся с новой проблемой: как гарантировать сетевому уровню принимающего устройства доставку всех фреймов и при этом расположить их в правильном порядке. Предположим, что у адресата есть возможность определить, какую информацию содержит полученный фрейм — правильную или ошибочную (подробнее о кодах, позволяющих распознать и исправить ошибки передачи, мы поговорим далее). В линиях, где подтверждение передачи не требуется, отправитель просто передает фреймы, не заботясь о том, дошли ли они до адресата. Но для ориентированной на установление соединения службы с подтверждениями этого недостаточно.

Обычно для гарантии надежной доставки отправителю посылается информация о том, что происходит на другом конце линии. Протокол требует от получателя передавать обратно специальные управляющие фреймы, содержащие позитивные или негативные сообщения о полученных фреймах. Позитивное сообщение означает, что отправленный фрейм успешно получен на том конце линии. Негативное сообщение говорит о том, что с фреймом что-то случилось и его нужно передать снова.

Кроме того, отправленный фрейм может полностью исчезнуть из-за неисправности оборудования или какой-нибудь помехи (например, шума). В этом случае принимающая сторона его просто не получит и, соответственно, никак не отреагирует. Аналогично, если фрейм подтверждения теряется, то отправитель также не узнает, как ему действовать дальше. Очевидно, что протокол, с помощью которого отправитель отсылает фрейм и ожидает подтверждения (положительного или отрицательного ответа), в случае потери фрейма зависнет навсегда (например, если произойдет сбой оборудования или коммуникационного канала).

Чтобы избежать зависаний сети в случае полной потери фреймов, используются таймеры канального уровня. После передачи фрейма включается таймер, отсчитывая интервал времени, достаточный для получения целевым устройством этого фрейма, его обработки и отправки подтверждения. В нормальной ситуации фрейм корректно принимается, а подтверждение отсылается и доходит до отправителя, прежде чем истечет установленный интервал времени; только после этого таймер отключается.

Однако если исходный фрейм или подтверждение теряются по пути, установленный интервал времени истекает и отправитель получает сообщение о возможной проблеме. Самое простое решение для отправителя — послать фрейм еще раз. Однако при этом возникает опасность получения одного и того же фрейма несколько раз на канальном уровне целевого устройства и повторной передачи его сетевому уровню. Чтобы этого не случилось, необходимо последовательно пронумеровать отсылаемые фреймы, чтобы получатель мог отличить повторы от оригиналов.

Вопрос управления таймерами и порядковыми номерами, гарантирующими доставку каждого фрейма адресату на сетевом уровне ровно один раз, не больше и не меньше, — очень важная задача, которая решается на канальном (или более высоком) уровне. Далее мы подробно рассмотрим методы такого управления, изучив ряд постепенно усложняющихся примеров.

### 3.1.4. Управление потоком

Еще один важный вопрос разработки канального уровня (а также более высоких уровней) — что делать с отправителем, постоянно передающим фреймы быстрее, чем получатель способен их получать. Такая ситуация может возникнуть, если у передающей стороны оказывается более быстрый и мощный компьютер, чем у принимающей. Представьте смартфон, запрашивающий веб-страницу с высокотехнологичного сервера. Мощнейшее устройство развертывает пожарный шланг, наводит его на бедный телефон и поливает его данными до тех пор, пока тот не захлебнется. Даже при полном отсутствии ошибок передачи данных в определенный момент получатель просто не сможет продолжать обработку все прибывающих фреймов и начнет их терять.

Очевидно, что для предотвращения подобной ситуации следует что-то предпринять. В настоящее время применяются два подхода. Первый из них — **управление потоком с обратной связью (feedback-based flow control)**: получатель отсылает отправителю сообщение, в котором разрешает продолжить передачу или просто сообщает о своем состоянии. При втором подходе, **управлении потоком с ограничением (rate-based flow control)**, в протокол встраивается механизм, ограничивающий скорость, с которой отправитель может передавать данные. Обратная связь с получателем отсутствует.

В этой главе мы рассмотрим только подход с обратной связью, поскольку подход с ограничением используется исключительно на транспортном уровне (подробнее об этом — в главе 5). Управление потоком с обратной связью осуществляется на канальном уровне, но чаще — на более высоких. При этом оборудование канального уровня работает достаточно быстро, чтобы информация не терялась. Например, об аппаратной реализации этого уровня в виде карт **NIC (Network Interface Card — сетевая интерфейсная карта)** говорят, что она работает со скоростью «передачи по кабелю» (то есть фреймы обрабатываются так же быстро, как прибывают). Канальный уровень не отвечает за переполнение, эта проблема решается на более высоких уровнях.

Существуют различные схемы управления потоком с обратной связью, но большинство из них использует один и тот же принцип. Протокол содержит четко заданные правила, определяющие, когда отправитель может отослать следующий фрейм. Эти правила часто запрещают отправку фрейма до тех пор, пока получатель не даст разрешения, явно или неявно. Например, при установке соединения получатель может сказать: «Сейчас вы можете отправить мне $n$ фреймов, но не посылайте следующие фреймы, пока я не попрошу вас продолжить». В данной главе мы рассмотрим разные механизмы, основанные на этом принципе.

## 3.2. ОБНАРУЖЕНИЕ И КОРРЕКЦИЯ ОШИБОК

Из главы 2 мы узнали, что у каналов передачи данных большой разброс по характеристикам. В некоторых, например в оптоволоконных каналах телекоммуникационных сетей, вероятность ошибки крайне низкая, поэтому потеря данных происходит редко. Но количество ошибок в беспроводных или старых локальных сетях в десятки раз больше, и они даже считаются нормой. Для того чтобы полностью исключить их, потребуются слишком большие расходы с точки зрения производительности. Отсюда следует вывод: ошибки при передаче данных останутся важным фактором еще на долгие годы. Поэтому нам необходимо изучить методы их обнаружения и устранения.

Разработчики сетей создали две основные стратегии для борьбы с ошибками, основанные на добавлении к передаваемым данным некоторой избыточной информации. В одном случае с ее помощью принимающая сторона может определить, какие данные должны были прийти, в другом — это всего лишь оповещение об ошибке (без указания ее типа), после которого получатель запрашивает повторную передачу. Первая стратегия использует **корректирующие коды (error-correcting codes)**, вторая — **коды для обнаружения ошибок (error-detecting codes)**. Использование корректирующего кода часто называют **упреждающей коррекцией ошибок (Forward Error Correction, FEC)**.

Каждая стратегия занимает свою нишу. В высоконадежных (например, оптоволоконных) каналах дешевле использовать код для обнаружения ошибок и просто заново передавать поврежденные блоки. А беспроводные соединения, в которых возникает множество ошибок, чаще используют избыточность информации, позволяющей определить, какие данные должны были прийти. FEC применяется в зашумленных каналах, поскольку вероятность ошибки при повторной передаче так же велика, как и при первой.

Чтобы определить, какой метод лучше подойдет в конкретной ситуации, нужно понять, какой тип ошибок более вероятен. Но ни одна из стратегий не позволит справиться со всеми возможными ошибками, поскольку дополнительные биты, передаваемые для повышения надежности, также могут быть повреждены в пути. Было бы неплохо, если бы каналы передачи могли отличать такие биты от битов полезных данных, но это невозможно. Для канала все биты одинаковы. Поэтому, чтобы избежать необнаруженных ошибок, необходимо использовать достаточно надежные коды, чтобы успешно справляться со всеми прогнозируемыми ошибками.

В одной модели считается, что причина ошибок — экстремально высокие значения теплового шума, который изредка на короткие промежутки времени перекрывает сигнал, порождая изолированные однобитовые ошибки. Вторая модель предполагает, что ошибки чаще возникают целыми последовательностями, а не поодиночке. Объясняется это физическими процессами, вызывающими неполадки. Это может быть глубокое замирание беспроводного канала или временная электрическая помеха в кабельном канале.

Обе модели имеют практическую значимость, но у каждой есть свои преимущества и недостатки. Последовательность или пакет ошибок могут быть предпочтительнее одиночных ошибок. Данные всегда отправляются блоками.

Предположим, что размер блока равен 1000 бит, а вероятность ошибки равна 0,001 на один бит. Если бы ошибки были независимыми, то они бы обнаруживались почти в каждом блоке. Однако если возникнет целая последовательность ошибок, то в среднем из ста блоков только один будет поврежден. С другой стороны, последовательность исправить намного сложнее, чем изолированные ошибки.

Существуют и другие типы ошибок. Иногда местоположение ошибки известно. Например, физический уровень получает аналоговый сигнал, значение которого отличается от ожидаемого нуля или единицы, и сообщает, что бит потерян. В этой ситуации речь идет о **канале со стиранием (erasure channel)**. В каналах со стиранием ошибки исправлять проще, чем в каналах, где значения битов меняются на противоположные: даже если значение бита утеряно, по крайней мере нам известно, где кроется ошибка. Однако воспользоваться преимуществами каналов со стиранием удается нечасто.

Далее мы рассмотрим корректирующие коды и коды для обнаружения ошибок. Главное, не забывать о двух вещах. Во-первых, мы изучаем этот вопрос применительно к канальному уровню, так как именно здесь перед нами впервые встает проблема надежной пересылки группы битов. Однако эти коды используются гораздо шире, ведь вопрос надежности является общей проблемой. Коды исправления ошибок можно часто встретить на физическом уровне (особенно в зашумленных каналах), а также на более высоких уровнях, главным образом при рассылке мультимедийной информации в режиме реального времени. Коды обнаружения ошибок применяются на канальном, сетевом и транспортном уровнях.

Во-вторых, следует помнить, что коды ошибок относятся к прикладной математике. Если только вы не крупный специалист по полям Галуа или свойствам разреженных матриц, используйте надежные коды, полученные из проверенных источников, и не пытайтесь конструировать собственные. Так делается во многих стандартных протоколах; одни и те же коды будут встречаться вам снова и снова. Далее мы подробно изучим простой код, а затем коснемся нескольких более сложных. Так вы сможете лучше понять их преимущества и недостатки и познакомиться с кодами, применяемыми на практике.

## 3.2.1. Корректирующие коды

Мы рассмотрим четыре разных корректирующих кода:

1. Коды Хэмминга.
2. Двоичные сверточные коды.
3. Коды Рида — Соломона.
4. Коды с малой плотностью проверок на четность.

Все эти коды добавляют к отправляемой информации избыточные данные. Фрейм состоит из битов данных, то есть информационных битов ($m$), и избыточных, или контрольных, битов ($r$). В **блочном коде (block code)** $r$ вычисляется как простая функция от $m$: $r$ и $m$ связаны друг с другом, как если бы они находились

в огромной таблице соответствий. В **систематическом коде (systematic code)** $m$ пересылается напрямую вместе с $r$ и не кодируется перед отправкой. В **линейном коде (linear code)** $r$ вычисляется как линейная функция от $m$. Очень часто используется функция исключающего ИЛИ (XOR) или суммирование по модулю 2. Это означает, что для кодирования используются такие операции, как умножение матриц или простые логические схемы. Далее в этом разделе речь пойдет о линейных систематических блочных кодах (если не будет указано иное).

Допустим, полная длина фрейма равна $n$ (то есть $n = m + r$). Обозначим это как $(n, m)$-код. Набор из $n$ бит, содержащий информационные и контрольные биты, часто называют $n$-битным **кодовым словом**, или **кодовой комбинацией (codeword)**. **Кодовая норма (code rate)**, или просто норма, — это часть кодового слова, несущая информацию, которая не является избыточной, то есть $m/n$. На практике значения нормы могут сильно отличаться. Например, для зашумленного канала обычной нормой считается 1/2, то есть половина полученной информации будет избыточной. В каналах высокого качества норма близка к единице и к большим сообщениям добавляется лишь несколько контрольных битов.

Чтобы понять, как исправляются ошибки, сначала необходимо познакомиться с самим понятием ошибки. Если рассмотреть два кодовых слова, например 10001001 и 10110001, можно определить, сколько соответствующих разрядов в них отличаются. В данном примере различаются 3 бита. Чтобы это узнать, нужно сложить два кодовых слова по модулю 2 (операция XOR) и сосчитать количество единиц в результате:

$$\begin{array}{r} 10001001 \\ 10110001 \\ \hline 00111000 \end{array}$$

Количество битов, различающихся в двух кодовых словах, называется **расстоянием Хэмминга (Hamming distance)** в честь американского математика Ричарда Хэмминга (Hamming, 1950), или **минимальным кодовым расстоянием**. Смысл в том, что если два кодовых слова находятся на кодовом расстоянии $d$, то для преобразования одного в другое понадобится $d$ ошибок в одиночных битах.

С помощью алгоритма вычисления контрольных разрядов можно построить полный список всех допустимых кодовых слов и найти в нем пару комбинаций с минимальным кодовым расстоянием. Это расстояние Хэмминга для всего кода.

В большинстве приложений передачи данных все $2^m$ возможных сообщений являются допустимыми, однако в результате вычисления контрольных битов не все $2^n$ потенциальных кодовых слов используются. Более того, при $r$ контрольных битов допустимыми считаются лишь $2^m/2^n$ или $1/2^r$ кодовых слов, а не все возможные $2^m$. Именно разреженность данных в кодовой комбинации позволяет получателю распознавать и исправлять ошибки.

Способность блочного кода находить и исправлять ошибки зависит от расстояния Хэмминга. Чтобы гарантированно обнаружить $d$ ошибок, необходим код с минимальным кодовым расстоянием, равным $d + 1$, поскольку $d$ однобитовых ошибок не смогут превратить одну допустимую комбинацию в другую. Когда приемник встречает запрещенную кодовую комбинацию, он «понимает», что при

передаче произошла ошибка. Аналогично для исправления $d$ ошибок требуется код с минимальным кодовым расстоянием $2d + 1$, так как в этом случае даже при $d$ однобитных ошибок результат окажется ближе к исходному кодовому слову, чем к любому другому. Это означает, что исходную кодовую комбинацию можно однозначно определить, основываясь на предположении, что возникновение большего числа ошибок менее вероятно.

В качестве простейшего примера корректирующего кода рассмотрим код, в котором всего четыре допустимые комбинации:

0000000000, 0000011111, 1111100000 и 1111111111

В этом коде минимальное кодовое расстояние равно 5, а значит, он может исправлять двойные и обнаруживать четверные ошибки. Если принимающая сторона получит кодовое слово 0000000111, ожидая только однобитовые или двухбитовые ошибки, она «поймет», что оригинал должен быть равен 0000011111. Однако если тройная ошибка изменит 0000000000 на 0000000111, она будет исправлена неверно. Если же ожидаются все перечисленные ошибки, то их можно распознавать. Ни одна из полученных кодовых комбинаций не входит в список допустимых. Следовательно, произошла ошибка. Очевидно, что в данном примере невозможно одновременно исправлять двойные ошибки и распознавать четверные, потому что для этого полученное кодовое слово нужно будет интерпретировать двумя разными способами.

В нашем примере задачу декодирования, то есть поиск допустимого кодового слова, наиболее близкого к полученному, можно выполнить простым просмотром. К сожалению, в более общем случае, когда приходится оценивать все кодовые слова, это заняло бы очень много времени. Вместо этого разрабатываются практические коды, которые по определенным подсказкам ищут наиболее подходящее исходное кодовое слово.

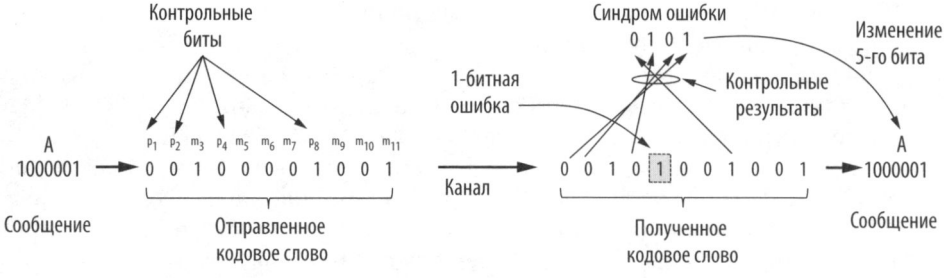
Попробуем создать код, состоящий из $m$ информационных и $r$ контрольных битов, способный исправлять одиночные ошибки. Каждому из $2^m$ допустимых сообщений будет соответствовать $n$ недопустимых кодовых слов, с кодовым расстоянием 1. Их можно получить инвертированием каждого из $n$ битов $n$-битного кодового слова. Таким образом, любому из $2^m$ допустимых сообщений должны соответствовать $n + 1$ кодовых комбинаций. Поскольку общее количество возможных кодовых комбинаций равно $2^n$, то $(n + 1)2^m \leq 2^n$. Так как $n = m + r$, получаем:

$$(m + r + 1) \leq 2r \tag{3.1}$$

При заданном $m$ формула описывает нижний предел контрольных битов, необходимых для исправления одиночных ошибок.

Кроме того, этот теоретический нижний предел может быть достигнут с помощью метода Хэмминга (Hamming, 1950). В **кодах Хэмминга** биты кодового слова нумеруются последовательно слева направо, начиная с 1. Биты с номерами, равными степеням 2 (1, 2, 4, 8, 16 и т. д.), являются контрольными. Остальные биты (3, 5, 6, 7, 9, 10 и т. д.) заполняются $m$ битами данных. Такой шаблон для кода Хэмминга (11, 7) с 7 битами данных и 4 контрольными битами показан на

илл. 3.6. Каждый контрольный бит обеспечивает сумму по модулю 2 или четность некоторой группы битов, включая себя самого. Один бит может входить в несколько вычислений контрольных битов. Чтобы определить, в какие группы контрольных сумм будет входить бит данных в $k$-й позиции, следует разложить $k$ на степень двойки. Например, 11 = 8 + 2 + 1, а 29 = 16 + 8 + 4 + 1. Каждый бит проверяется только теми контрольными битами, номера которых входят в этот ряд разложения (например, 11-й бит проверяется битами 1, 2 и 8). В нашем примере контрольные биты вычисляются для проверки на четность суммы в сообщении, представляющем букву «A» в кодах ASCII.




**Илл. 3.6.** Пример кода Хэмминга (11, 7), исправляющего однобитную ошибку

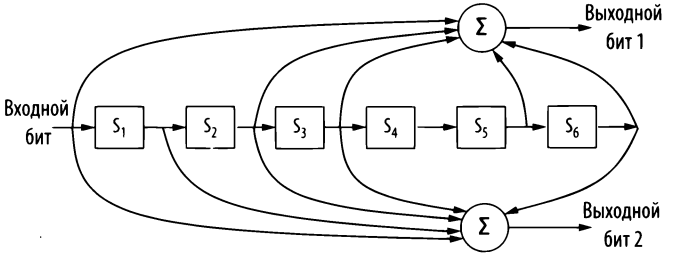
Расстояние Хэмминга этого кода равно 3, то есть он позволяет исправлять одиночные ошибки (или распознавать двойные). Причина такой сложной нумерации битов данных и контрольных битов становится очевидной, если рассмотреть процесс декодирования. Когда приходит кодовое слово, приемник учитывает значения полученных контрольных битов и заново вычисляет их. Их называют результатами проверки. Если контрольные биты верны, то для четных сумм все результаты проверки должны быть равны нулю. В этом случае кодовое слово принимается как правильное.

Если не все результаты проверки равны нулю, это означает, что обнаружена ошибка. Набор результатов проверки формирует **синдром ошибки (error syndrome)**, с помощью которого ошибка выявляется и исправляется. На илл. 3.6 в канале произошла ошибка в одном бите. Результаты проверки равны 0, 1, 0 и 1 для $k$ = 8, 4, 2 и 1 соответственно. Таким образом, синдром равен 0101 или 4 + 1 = 5. Согласно схеме, пятый бит содержит ошибку. Поменяв его значение (а это может быть контрольный бит или бит данных) и отбросив контрольные биты, мы получим правильное сообщение: букву «A» в кодах ASCII.

Кодовые расстояния полезны для понимания блочных кодов, а коды Хэмминга применяются в физических модулях оперативной памяти RAM. Однако в большинстве сетей используются более надежные коды. Второй тип кода, с которым мы познакомимся, — **сверточный код (convolutional code)**. Из всех кодов, представленных в данной книге, только он не относится к блочному типу. Кодировщик обрабатывает последовательность входных битов и генерирует последовательность выходных битов. В отличие от блочного кода, никакие ограничения на размер сообщения не накладываются, также не существует границы

кода. Значение выходного бита зависит от значения текущего и предыдущих входных битов — если у кодировщика есть возможность использовать память. Число предыдущих битов, от которого зависит выход, называется **длиной кодового ограничения (constraint length)** для данного кода. Сверточные коды описываются с учетом кодовой нормы и длины кодового ограничения.

Сверточные коды широко применяются в существующих сетях, например, они входят в систему GSM, спутниковые сети и сети 802.11. В качестве примера можно рассмотреть популярный сверточный код, представленный на илл. 3.7. Он называется сверточным кодом NASA с $r = 1/2$ и $k = 7$. Впервые этот код был использован при подготовке полета космического корабля *Voyager*, запущенного в 1977 году. С тех пор он применяется повсюду, к примеру, в сетях 802.11.




**Илл. 3.7.** Двоичный сверточный код NASA, применяемый в сетях 802.11

На илл. 3.7 для каждого входного бита слева создается два выходных бита справа. Эти выходные биты получаются путем применения операции XOR к входному биту и внутреннему состоянию. Так как кодирование работает на уровне битов и использует линейные операции, это двоичный линейный сверточный код. Поскольку один входной бит создает два выходных бита, кодовая норма $r$ равна 1/2. Этот код не систематический, так как входные биты никогда не передаются на выход напрямую без изменений.

Внутреннее состояние хранится в шести регистрах памяти. При поступлении на вход очередного бита значения в регистрах сдвигаются вправо. Например, если на вход подается последовательность 111, а в первоначальном состоянии в памяти только нули, то после подачи первого, второго и третьего бита оно будет меняться на 100000, 110000 и 111000 соответственно. На выходе получатся значения 11, затем 10 и затем 01. Чтобы полностью подавить первоначальное состояние регистров (тогда оно не будет влиять на результат), требуется 7 сдвигов. Таким образом, длина кодового ограничения для данного кода равна 7: $k = 7$.

Чтобы декодировать сверточный код, нужно найти последовательность входных битов, которая с наибольшей вероятностью породила наблюдаемую последовательность выходных битов (включая любые ошибки). Для небольших значений $k$ это делается с помощью широко распространенного алгоритма, разработанного Витерби (Viterby), см. работу Форни (Forney, 1973). Алгоритм проходит по наблюдаемой последовательности, сохраняя для каждого шага и для

каждого возможного внутреннего состояния входную последовательность, которая породила бы наблюдаемую последовательность с минимальным числом ошибок. Входная последовательность, которой соответствует минимальное число ошибок, и есть наиболее вероятное исходное сообщение.

Сверточные коды очень популярны на практике, так как в декодировании легко учесть неопределенность значения бита (0 или 1). Предположим, что –1 В соответствует логическому уровню 0, а +1 В — логическому уровню 1. Мы получили для двух битов значения 0,9 В и –0,1 В. Вместо того чтобы сразу определять соответствие: 1 для первого бита и 0 для второго, — можно принять 0,9 В за «очень вероятную единицу», а –0,1 В за «возможный ноль» и скорректировать всю последовательность. Для более надежного исправления ошибок к неопределенностям можно применять различные расширения алгоритма Витерби. Такой подход к обработке неопределенности значения битов называется **декодированием с мягким принятием решений (soft-decision decoding)**. И наоборот, если мы решаем, какой бит равен нулю, а какой единице, до последующего исправления ошибок, такой метод называется **декодированием с жестким принятием решений (hard-decision decoding)**.

Третий тип корректирующего кода, с которым мы познакомимся, называется **кодом Рида — Соломона (Reed–Solomon code)**. Как и коды Хэмминга, коды Рида — Соломона относятся к линейным блочным кодам и часто являются систематическими. Но в отличие от кодов Хэмминга, которые применяются к отдельным битам, коды Рида — Соломона работают на группах из $m$-битовых символов. Разумеется, математика здесь намного сложнее, поэтому мы используем аналогию.

Коды Рида — Соломона основаны на том факте, что каждый многочлен $n$-й степени однозначно определяется $n + 1$ точками. Например, если линия задается формулой $ax + b$, то восстановить ее можно по двум точкам. Дополнительные точки, лежащие на той же линии, излишни, но они полезны для исправления ошибок. Предположим, что две точки данных представляют линию. Мы отправляем их по сети. Также мы передаем еще две контрольные точки, лежащие на той же линии. Если в значение одной из точек по пути закрадывается ошибка, то точки данных все равно можно восстановить, подогнав линию к полученным правильным точкам. Три точки окажутся на линии, а одна, ошибочная, будет лежать вне ее. Обнаружив правильное положение линии, мы исправим ошибку.

В действительности коды Рида — Соломона определяются как многочлены на конечных полях. Для $m$-битовых символов длина кодового слова составляет $2m - 1$ символов. Очень часто выбирают значение $m = 8$, то есть одному символу соответствует один байт. Тогда длина кодового слова — 255 байт. Широко используется код (255, 233), он добавляет 22 дополнительных символа к 233 символам данных. Декодирование с исправлением ошибок выполняется при помощи алгоритма, разработанного Берлекэмпом и Мэсси (Berlekamp, Massey). Он эффективно осуществляет подгонку для кодов средней длины (Massey, 1969).

Коды Рида — Соломона широко применяются на практике благодаря эффективности в исправлении ошибок, особенно последовательных. Они используются в сетях DSL, в кабельных и спутниковых сетях, а также для исправления ошибок

на CD, DVD и Blu-ray. Поскольку работа идет на базе $m$-битных символов, однобитные ошибки и $m$-битные пакеты ошибок обрабатываются одинаково — как одна символьная ошибка. При добавлении $2t$ избыточных символов код Рида — Соломона способен исправить до $t$ ошибок в любом из переданных символов. Это означает, например, что код (255, 233) с 32 избыточными символами исправляет до 16 символьных ошибок. Так как символы могут быть последовательными, а размер их обычно составляет 8 бит, то возможно исправление пакетов ошибок в 128 битах. Еще лучше, если модель ошибок связана с удалением данных (например, царапина на компакт-диске, уничтожившая несколько символов). В таком случае возможно исправление до $2t$ ошибок.

Коды Рида — Соломона часто используются в сочетании с другими кодами, например сверточными, и вот почему. Сверточные коды эффективно обрабатывают изолированные однобитные ошибки, но с последовательностью ошибок они, скорее всего, не справятся, особенно если ошибок в полученном потоке битов слишком много. Добавив внутрь сверточного кода код Рида — Соломона, можно очистить поток битов от последовательностей ошибок. Таким образом, получившийся составной код обеспечит надежную защиту как от одиночных, так и от массовых ошибок.

Наконец, рассмотрим код **LDPC (Low-Density Parity Check — код с малой плотностью проверок на четность)**. Коды LDPC — это линейные блочные коды, изобретенные Робертом Галлагером и описанные в его докторской диссертации (Gallager, 1962). О ней быстро забыли, как и о большинстве других диссертаций. Однако в 1995 году, когда вычислительные мощности сделали огромный скачок вперед, представленный в ней код был изобретен заново.

В коде LDPC каждый выходной бит формируется из некоторого подмножества входных битов. Это приводит нас к матричному представлению кода с низкой плотностью единиц — отсюда и название. Полученные кодовые слова декодируются алгоритмом аппроксимации, который итеративно ищет наилучшее приближение, составленное из полученных данных, пока не получает допустимое кодовое слово. Так происходит устранение ошибок.

Коды LDPC удобно применять для блоков большого размера. Они превосходно справляются с ошибками — лучше, чем многие другие коды (включая те, с которыми мы уже познакомились). По этой причине коды LDPC активно добавляются в новые протоколы. Они являются частью стандарта цифрового телевидения, сетей Ethernet 10 Гбит/с, ЛЭП, а также последней версии 802.11. Очевидно, что эти коды обязательно будут использоваться и в новых сетях, которые еще находятся на стадии разработки.

## 3.2.2. Коды для обнаружения ошибок

Корректирующие коды широко применяются в беспроводных системах связи, известных зашумленностью и ненадежностью по сравнению с оптоволокном. Передать что-либо, не используя эти коды, практически невозможно. Однако при отправке данных по оптоволокну или высококачественному медному проводу уровень ошибок гораздо ниже, поэтому их обнаружение и повторная передача данных — более подходящий метод.

Мы рассмотрим три кода для обнаружения ошибок. Все они относятся к линейным систематическим блочным кодам:

1. Код с проверкой на четность.
2. Код с контрольными суммами.
3. Циклический избыточный код.

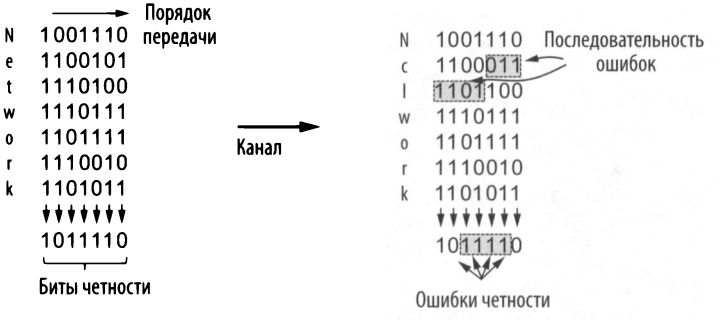
Чтобы понять, в каких ситуациях обнаружение ошибок эффективнее их исправления, рассмотрим первый из перечисленных кодов. К отправляемым данным присоединяется единственный **бит четности (parity bit)**, который выбирается так, чтобы число единичных битов в кодовом слове было четным (или нечетным). Это аналогично вычислению бита четности в виде суммы по модулю 2 для битов данных (или применению операции XOR). Например, если отправляется комбинация 1011010 и число единиц должно быть четным, то в конце добавляется ноль и последовательность превращается в 10110100. Если же число единиц должно быть нечетным, то комбинация превращается в 10110101. Расстояние Хэмминга у кода с единственным битом четности равно двум, так как любая однобитовая ошибка меняет четность кодового слова на неправильную. Это означает, что данный код позволяет распознавать однобитовые ошибки.

Рассмотрим канал с изолированными ошибками, возникающими с вероятностью $10^{-6}$ на бит. Такое значение может показаться очень небольшим, но для длинного кабельного канала, в котором распознавать ошибки довольно сложно, оно в лучшем случае считается допустимым. Типичные LAN характеризуются вероятностью ошибки $10^{-10}$. Пусть блок данных состоит из 1000 бит. Как видно из представленного выше уравнения (3.1), чтобы создать код, исправляющий однократные ошибки в 1000-битном блоке, потребуется 10 контрольных битов. Для 1 Мбит данных это составит 10 000 проверочных бит. Чтобы просто обнаружить одиночную однобитную ошибку, достаточно одного бита четности на блок. На каждые 1000 блоков будет выявляться одна ошибка, и придется переслать повторно еще один блок (1001 бит), чтобы исправить ее. Таким образом, суммарные накладные расходы на обнаружение ошибки и повторную передачу составят всего 2001 бит на 1 Мбит данных против 10 000 бит, необходимых для кода Хэмминга.

Проблема данной схемы в том, что если к блоку добавлять всего один бит четности, то гарантированно распознаваться будет только одна однобитовая ошибка в блоке. В случае возникновения длинной последовательности ошибок вероятность обнаружения ошибки будет всего лишь 0,5, что абсолютно неприемлемо. Этот недостаток может быть исправлен, если рассматривать каждый посылаемый блок как прямоугольную матрицу $n$ бит шириной и $k$ бит высотой (принцип ее построения был описан выше). Если вычислить и отправить один бит четности для каждой строки, то можно гарантированно обнаружить до $k$ однобитных ошибок (если в каждой строке будет не больше одной ошибки).

Однако можно сделать кое-что еще, чтобы повысить уровень защиты от последовательностей ошибок, — биты четности можно вычислять в порядке, отличном от того, в каком данные отправляются по каналу связи. Этот способ

называется **чередованием (interleaving)**. В нашем примере мы будем вычислять бит четности для каждого из $n$ столбцов, но биты данных будут отправляться в виде $k$ строк в обычном порядке: сверху вниз и слева направо. В последней строке отправим $n$ бит четности. На илл. 3.8 порядок пересылки показан для $n = 7$ и $k = 7$.




**Илл. 3.8.** Чередование битов четности для обнаружения последовательностей ошибок

С помощью чередования код, обнаруживающий (или исправляющий) изолированные ошибки, преобразуется в код, обнаруживающий (или исправляющий) пакеты ошибок. На илл. 3.8 мы видим, что при возникновении таких пакетов длиной $n = 7$ ошибочные биты находятся в разных столбцах. (Последовательность ошибок не предполагает, что все биты в ней неправильные; подразумевается, что ошибки есть как минимум в первом и последнем битах. На илл. 3.8 из семи сбойных битов на самом деле изменено значение только четырех.) В каждом из $n$ столбцов повреждено будет не больше одного бита, поэтому биты четности в них помогут выявить ошибку. В данном методе $n$ бит четности в блоках из $kn$ бит данных применяются для обнаружения одной последовательности ошибок длиной $n$ бит или меньше.

Последовательность ошибок длиной $n + 1$ не будет обнаружена, если будут инвертированы первый и последний биты, а все остальные останутся неизменными. Если в блоке при передаче возникнет длинная последовательность или несколько коротких, вероятность того, что четность любого из $n$ столбцов случайным образом окажется верной, равна 0,5. Следовательно, возможность необнаружения ошибки будет равна $2^{-n}$.

Второй тип кода для обнаружения ошибок — с использованием **контрольной суммы (checksum)** — весьма напоминает группу кодов, применяющих биты четности. Под «контрольной суммой» часто подразумевают любую группу контрольных битов, связанных с сообщением, независимо от способа их вычисления. Группа битов четности — также пример контрольной суммы. Однако существуют и другие, более надежные варианты, основанные на текущей сумме битов данных в сообщении. Контрольная сумма обычно помещается в конец сообщения, в качестве дополнения функции суммирования. Таким образом, ошибки можно обнаружить путем суммирования всего полученного кодового

слова: битов данных и контрольной суммы. Если результат равен нулю, значит, ошибок нет.

Один из примеров такого кода — 16-битная контрольная сумма, используемая во всех пакетах IP при передаче данных в интернете (см. работу Брейдена и др.; Braden et al., 1988). Она представляет собой сумму битов сообщения, поделенного на 16-битные слова. Так как данный метод работает со словами, а не с битами (как при использовании битов четности), то ошибки, оставляющие четность неизменной, все же изменяют значение суммы, а значит, могут быть обнаружены. Например, если бит младшего разряда в двух разных словах меняется с 0 на 1, то проверка четности этих битов не выявит ошибку. Однако добавление к 16-битной контрольной сумме двух единиц даст другой результат, и ошибка станет очевидной.

Контрольная сумма, применяемая в интернете, вычисляется с помощью обратного кода или арифметики с дополнением до единицы, а не как сумма по модулю $2^{16}$. В арифметике обратного кода отрицательное число представляет собой поразрядное дополнение своего положительного эквивалента. Большинство современных компьютеров использует арифметику с дополнением до двух, в которой отрицательное число является дополнением до единицы плюс один. В арифметике с дополнением до двух сумма с дополнением до единицы эквивалентна сумме по модулю $2^{16}$, причем любое переполнение старших битов добавляется обратно к младшим битам. Такой алгоритм обеспечивает единообразный охват данных битами контрольной суммы. Иначе могут быть добавлены два старших бита, вызвать переполнение и потеряться, не изменив контрольную сумму. Но есть и еще одно преимущество. Дополнение до единицы имеет два представления нуля: все нули и все единицы. Таким образом, одно значение (например, все нули) указывает, что контрольной суммы нет и дополнительное поле для этого не требуется.

Десятилетиями существовало мнение, что фреймы, для которых вычисляется контрольная сумма, содержат случайные значения битов. Анализ алгоритмов вычисления контрольных сумм всегда проводился с учетом именно такого предположения. Изучение фактических данных, выполненное Партриджем и др. (Partridge et al., 1995), показало, что данное предположение неверно. Следовательно, нераспознанные ошибки проскальзывают в некоторых случаях намного чаще, чем считалось ранее.

В частности, контрольная сумма для интернета, несмотря на эффективность и простоту, в определенных ситуациях слабо защищает от ошибок именно потому, что это простая сумма. Она не позволяет распознать удаление или добавление нулевых данных, а также случаи, когда части сообщения меняются местами или расщепляются таким образом, что склеенными оказываются части двух разных пакетов. Может казаться, что подобные ошибки вряд ли произойдут в случайных процессах, но они вполне вероятны в сетях с неправильно работающим оборудованием.

Более эффективный алгоритм — **контрольная сумма Флетчера** (Fletcher, 1982). Он включает компонент, отвечающий за позицию: произведение данных и соответствующей позиции добавляется к текущей сумме. Это обеспечивает более точное обнаружение изменений в положении данных.

В некоторых случаях две приведенные выше схемы приемлемы на более высоких уровнях, но обычно на канальном уровне широко используется другой, более надежный метод обнаружения ошибок — **циклический избыточный код (Cyclic Redundancy Check, CRC)**, также известный как **полиномиальный код**. В основе полиномиальных кодов лежит представление битовых строк в виде многочленов с коэффициентами, равными только 0 или 1. Фрейм из $k$ бит рассматривается как список коэффициентов многочлена степени $k - 1$, состоящего из $k$ членов от $x^{k-1}$ до $x^0$. Старший (самый левый) бит фрейма соответствует коэффициенту при $x^{k-1}$, следующий — коэффициенту при $x^{k-2}$ и т. д. Например, число 110001 состоит из 6 бит и, следовательно, представляется в виде многочлена пятой степени с коэффициентами 1, 1, 0, 0, 0 и 1: $x^5 + x^4 + x^0$.

С данными многочленами осуществляются арифметические действия по модулю 2 в соответствии с алгебраической теорией поля. При этом перенос при сложении и заем при вычитании не производится. И сложение, и вычитание эквивалентны XOR. Например:

| | | | |
|---|---|---|---|
| 10011011 | 00110011 | 11110000 | 01010101 |
| +11001010 | +11001101 | −10100110 | −10101111 |
| 01010001 | 11111110 | 01010110 | 11111010 |

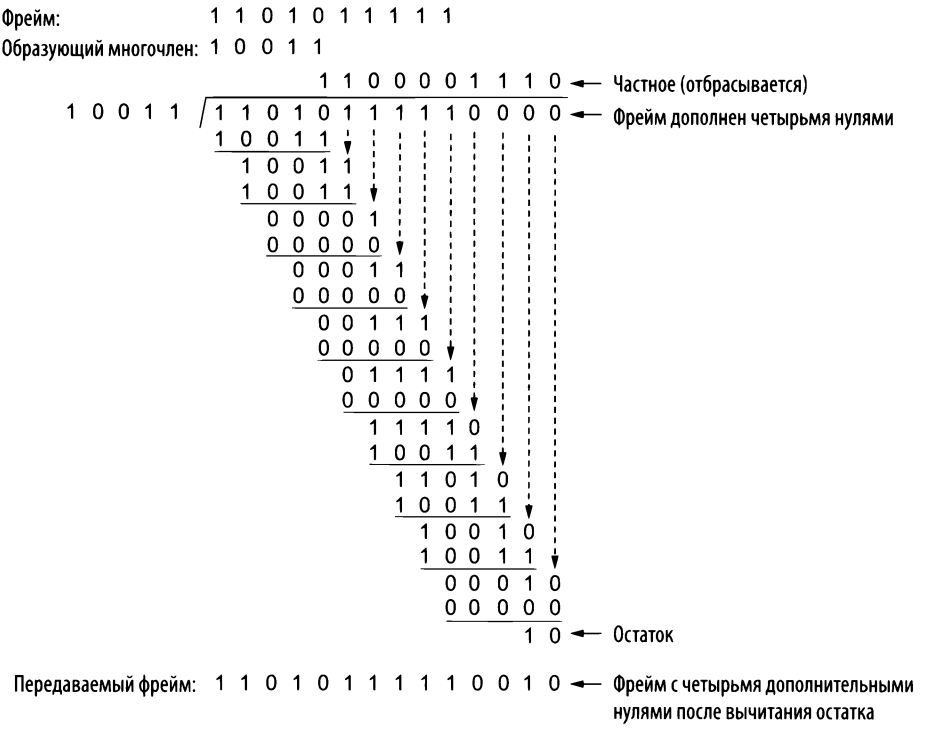
Деление чисел осуществляется точно так же, как и деление обычных двоичных чисел, с той разницей, что вычитание снова производится по модулю 2. Считается, что делитель «уходит» в делимое, если в делимом столько же битов, сколько в делителе.

При использовании циклического кода отправитель и получатель должны прежде всего договориться насчет **образующего многочлена**, $G(x)$. Старший и младший биты в нем должны быть равны 1. Вычисление CRC для фрейма из $m$ бит, соответствующего многочлену $M(x)$, возможно, если этот фрейм длиннее образующего многочлена. Идея состоит в добавлении CRC в конец фрейма так, чтобы получившийся многочлен делился на $G(x)$ без остатка. Получатель, приняв фрейм, содержащий контрольную сумму, пытается разделить его на образующий многочлен $G(x)$. Ненулевой остаток от деления означает ошибку.

Алгоритм вычисления CRC выглядит следующим образом:

1. Пусть $r$ — степень многочлена $G(x)$. Добавим $r$ нулевых битов в конец фрейма так, чтобы он содержал $m + r$ бит и соответствовал многочлену $x^rM(x)$.
2. Разделим по модулю 2 битовую строку, соответствующую многочлену $x^rM(x)$, на битовую строку, соответствующую образующему многочлену $G(x)$.
3. Вычтем по модулю 2 остаток от деления (он должен быть не более $r$ бит) из битовой строки, соответствующей многочлену $x^rM(x)$. Результат и будет передаваемым фреймом, который мы назовем многочленом $T(x)$.

На илл. 3.9 показаны вычисления для фрейма 1101011111 и образующего многочлена $G(x) = x^4 + x + 1$.




**Илл. 3.9.** Пример вычисления CRC

Важно отметить, что многочлен $T(x)$ делится (по модулю 2) на $G(x)$ без остатка. В любой задаче деления, если вычесть остаток из делимого, результат будет кратным делителю. Например, в десятичной системе счисления при делении 210 278 на 10 941 остаток равен 2399. Если вычесть 2399 из 210 278, то результат (207 879) будет делиться на 10 941 без остатка.

Теперь проанализируем возможности этого метода. Какие ошибки он способен обнаружить? Представим, что произошла ошибка при передаче фрейма и вместо многочлена $T(x)$ получатель принял $T(x) + E(x)$. Каждый единичный бит многочлена $E(x)$ соответствует инвертированному биту в пакете. Если в многочлене $E(x)$ $k$ бит равны 1, это значит, что произошло $k$ однобитных ошибок. Отдельный пакет ошибок характеризуется единицей в начале, комбинацией нулей и единиц и конечной единицей; остальные биты равны 0.

Получатель делит принятый фрейм с контрольной суммой на образующий многочлен $G(x)$, то есть вычисляет $[T(x) + E(x)]/G(x)$. $T(x)/G(x)$ равно 0, поэтому результат вычислений равен $E(x)/G(x)$. Ошибки, которые случайно окажутся кратными образующему многочлену $G(x)$, останутся незамеченными, остальные будут обнаружены.

Если происходит однобитная ошибка, то $E(x) = x^i$, где $i$ — номер ошибочного бита. Если образующий многочлен $G(x)$ содержит два или более члена, то $E(x)$

никогда не будет делиться на него без остатка, поэтому будут обнаружены все однобитные ошибки.

В случае двух изолированных однобитных ошибок $E(x) = x^i + x^j$, где $i > j$. Это также можно записать в виде $E(x) = x^j(x^{i-j} + 1)$. Предположим, что $G(x)$ не делится на $x$. Тогда достаточное условие обнаружения всех двойных ошибок — неделимость многочлена $x^k + 1$ на $G(x)$ для любого $k$ от 1 до максимального значения $i - j$ (то есть до максимальной длины фрейма). Известны простые многочлены с небольшими степенями, обеспечивающие защиту для длинных фреймов. Например, многочлен $x^{15} + x^{14} + 1$ не является делителем для $x^k + 1$ для любого $k$ от 1 до 32 768.

Если ошибка затронет нечетное количество битов во фрейме, многочлен $E(x)$ будет содержать нечетное число членов (например, $x^5 + x^2 + 1$, но не $x^2 + 1$). Интересно, что в системе арифметических операций по модулю 2 многочлены с нечетным числом членов не делятся на $x + 1$. Если в качестве образующего выбрать многочлен, который делится на $x + 1$, то с его помощью можно обнаружить все ошибки, состоящие из нечетного количества инвертированных битов. Как показывает статистика, уже за счет этого можно обнаружить половину имеющихся ошибок.

И наконец, что наиболее важно, полиномиальный код с $r$ контрольными битами будет обнаруживать все пакеты ошибок длиной $\leq r$. Пакет ошибок длиной $k$ может быть представлен в виде многочлена $x^i(x^{k-1} + \ldots + 1)$, где $i$ определяет, насколько далеко от правой границы фрейма располагается пакет ошибок. Если образующий многочлен $G(x)$ содержит член $x^0$, то $x^i$ не будет его множителем. Поэтому если степень выражения в скобках меньше степени $G(x)$, то остаток от деления никогда не будет нулевым.

Если длина последовательности ошибок равна $r + 1$, то остаток от деления будет нулевым, только если последовательность идентична $G(x)$. По определению, первый и последний биты последовательности должны быть равны 1, поэтому ее совпадение с образующим многочленом зависит от $r - 1$ промежуточных битов. Если все комбинации считать равновероятными, то возможность такой нераспознаваемой ошибки равна $(1/2)^{r-1}$.

Также следует отметить, что при возникновении пакета ошибок длиннее $r + 1$ битов или нескольких коротких пакетов вероятность пропуска ошибки составляет $(1/2)^r$ при условии, что все комбинации битов равновероятны.

Некоторые образующие многочлены стали международными стандартами. Один из них применяется в IEEE 802 (он основан на многочлене, используемом в Ethernet):

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1.$$

Помимо других полезных свойств, этот многочлен обладает способностью обнаруживать любые пакеты ошибок длиной не более 32 бит и все пакеты, дающие нечетное число битов. Начиная с 1980-х годов он применяется очень широко. И все же его нельзя назвать наилучшим. Выполнив обстоятельные компьютерные вычисления, Кастаньоли и др. (Castagnoli et al., 1993) и Купман (Koopman, 2002) обнаружили самые эффективные коды CRC. Расстояние Хэмминга, соответствующее сообщениям обычной длины, для них равно 6, в то время как для CRC-32 стандарта IEEE оно равняется всего 4.

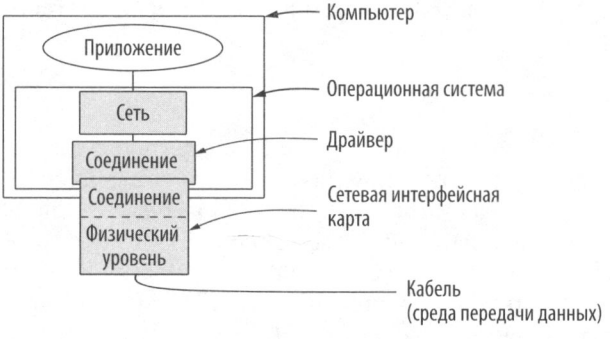
Хотя алгоритм вычисления CRC может показаться сложным, Петерсон и Браун (Peterson and Brown, 1961) продемонстрировали, что можно создать простую схему для аппаратной проверки и подсчета CRC на основе схемы с регистром сдвига. В дальнейшем с регулярной периодичностью появлялись более новые и более быстрые реализации (см. работу Митры и Найека; Mitra and Nyack, 2017). CRC до сих пор применяется на практике повсеместно в аппаратном обеспечении. Десятки сетевых стандартов работают на основе кодов CRC, включая почти все LAN (например, Ethernet, 802.11) и линии связи «точка-точка» (пакеты, пересылаемые по SONET).

## 3.3. ЭЛЕМЕНТАРНЫЕ ПРОТОКОЛЫ ПЕРЕДАЧИ ДАННЫХ НА КАНАЛЬНОМ УРОВНЕ

Знакомство с протоколами мы начнем с рассмотрения трех примеров, в порядке возрастания сложности. Прежде чем приступить к их изучению, полезно обозначить некоторые допущения, лежащие в основе данной модели связи.

### 3.3.1. Исходные упрощающие допущения

**Независимые процессы.** Для начала предположим, что физический, канальный и сетевой уровни являются независимыми процессами, которые взаимодействуют между собой путем передачи сообщений. Типичная реализация показана на илл. 3.10. Процесс физического уровня и часть процесса канального уровня выполняются на специальном оборудовании: **сетевой интерфейсной карте (Network Interface Card, NIC)**. Остальная часть процесса канального уровня и процесс сетевого уровня осуществляются на центральном процессоре (ЦП), являясь частью операционной системы. При этом программное обеспечение процесса канального уровня зачастую принимает вид **драйвера устройства**. Также возможны другие варианты реализации, например, три процесса, выполняющиеся на **сетевом ускорителе** или на ЦП с программно определяемой частотой. На самом деле оптимальная реализация меняется с течением времени




**Илл. 3.10.** Реализация физического, канального и сетевого уровней

по мере развития технологий. В любом случае представление трех уровней в виде отдельных процессов делает их обсуждение концептуально более четким, а также подчеркивает их независимость друг от друга.

**Однонаправленная передача данных.** Следующее ключевое допущение состоит в том, что устройство *A* хочет отправить на устройство *B* большой поток данных, используя надежную, ориентированную на установление соединений службу. Позднее мы рассмотрим случай, когда одновременно с этим *B* также хочет передать данные на *A*. Предполагается, что устройство *A* имеет бесконечный источник данных, готовых к отправке, и что ему никогда не нужно ждать их генерации. Когда канальный уровень *A* запрашивает данные, сетевой уровень их сразу же предоставляет. (Это ограничение позже будет отброшено.)

**Надежные устройства и процессы.** Также предполагается, что компьютеры не выходят из строя. При передаче могут возникать ошибки, но не проблемы, связанные с поломкой оборудования или случайной перезагрузкой.

С точки зрения канального уровня пакет, полученный по интерфейсу от сетевого уровня, рассматривается как чистые данные, и каждый бит должен быть доставлен сетевому уровню принимающего устройства. Тот факт, что он может интерпретировать часть пакета как заголовок, не касается канального уровня.

## 3.3.2. Базовая схема передачи и приема данных

Получив пакет от сетевого уровня отправителя, канальный уровень формирует из него фрейм, добавляя заголовок и трейлер (см. илл. 3.1). Таким образом, фрейм состоит из встроенного пакета, некоторой служебной информации (в заголовке) и контрольной суммы (в трейлере). Затем фрейм передается канальному уровню целевого устройства. Мы будем исходить из наличия подходящих библиотечных процедур, например `to_physical_layer` для отправки фрейма и `from_physical_layer` для его получения. Они вычисляют и добавляют или проверяют контрольную сумму (обычно это делается аппаратно), так что при разработке протоколов, представленных в этом разделе, можно об этом не беспокоиться. К примеру, они могут использовать рассмотренный ранее алгоритм CRC (циклический избыточный код).

Изначально получатель ничего не должен делать, он просто ожидает. В протоколах, рассматриваемых в этой главе, ожидание событий канальным уровнем происходит путем вызова процедуры `wait_for_event(&event)`. Эта процедура возвращает управление, только когда что-то происходит (например, получение фрейма). При этом переменная `event` сообщает, что именно случилось. Наборы возможных событий отличаются в разных протоколах, поэтому будут описываться для каждого протокола отдельно. Следует отметить, что на практике канальный уровень не находится в холостом цикле ожидания событий (согласно нашему допущению), а получает прерывание, когда это событие происходит. При этом он приостанавливает текущие процессы и обрабатывает полученный фрейм. Но для простоты мы проигнорируем эти детали и будем исходить из того, что канальный уровень все свое время посвящает работе с одним каналом.

Когда принимающая сторона получает фрейм, контрольная сумма вычисляется заново. Если она неверна (то есть при передаче возникли ошибки),

канальный уровень получает соответствующую информацию (`event=cksum_err`). Если фрейм приходит без ошибок, канальному уровню об этом также сообщается (`event=frame_arrival`), после чего он может получить этот фрейм у физического уровня с помощью процедуры `from_physical_layer`. Получив неповрежденный фрейм, канальный уровень проверяет управляющую информацию, находящуюся в заголовке, и если все в порядке, часть фрейма передается сетевому уровню; заголовок фрейма не передается ни при каких обстоятельствах.

Для запрета передачи сетевому уровню любой части заголовка фрейма есть веская причина: необходимо обеспечить полное разделение сетевого и канального уровней. До тех пор пока сетевой уровень ничего не знает о формате фрейма и протоколе канального уровня, их изменения не потребуют смены программного обеспечения самого сетевого уровня. Это происходит только при установке на компьютер новой сетевой карты. Поддержание жестко заданного интерфейса между сетевым и канальным уровнями значительно упрощает разработку программ, так как протоколы различных уровней могут развиваться независимо.

На илл. 3.11 показаны некоторые объявления (на языке C), общие для многих протоколов, обсуждаемых ниже. Выделяются пять типов данных: `boolean`, `seq_nr`, `packet`, `frame_kind` и `frame`. Тип `boolean` представляет собой перечисляемый тип, переменные которого могут принимать значение `true` или `false`. Тип `seq_nr` является целым без знака, используемым для нумерации фреймов, благодаря которой их можно различать. Нумерация идет от 0 до числа *MAX_SEQ* включительно, определяемого для конкретного протокола. Тип `packet` — это единица информации, используемая при обмене данными между сетевым и канальным уровнями одного компьютера или двумя сетевыми уровнями. В нашей модели пакет всегда состоит из *MAX_PKT* байт, хотя на практике он обычно имеет переменную длину.

Тип `frame` содержит четыре поля: `kind`, `seq`, `ack` и `info`; первые три содержат управляющую информацию, а последнее может включать данные, которые необходимо передать. Три управляющих поля вместе называются **заголовком фрейма (frame header)**.

Поле `kind` сообщает о наличии данных внутри фрейма, так как некоторые протоколы отличают фреймы, содержащие только управляющую информацию, от фреймов, в которых также есть данные. Поле `seq` используется для хранения последовательного номера фрейма, `ack` — для подтверждения. Подробнее их применение будет описано ниже.

Поле данных фрейма `info` содержит один пакет. В управляющем фрейме это поле не задействуется. На практике используется поле `info` переменной длины, в управляющих фреймах оно полностью отсутствует.

Важно понимать взаимоотношения между пакетом и фреймом (см. илл. 3.1). Сетевой уровень создает пакет, принимая сообщение от транспортного уровня и добавляя к нему свой заголовок. Пакет передается канальному уровню, который включает его в поле `info` исходящего фрейма. Когда целевое устройство получает фрейм, канальный уровень извлекает пакет из фрейма и передает его сетевому. Таким образом, сетевой уровень может действовать так, будто устройства обмениваются пакетами напрямую.

```
#define MAX_PKT 1024                                    /* определяет размер пакета в байтах */
typedef enum {false, true} boolean;                     /* тип boolean */
typedef unsigned int seq_nr;                            /* порядковые номера фреймов
                                                           или подтверждений */
typedef struct {unsigned char data[MAX_PKT];} packet;   /* определение пакета */
typedef enum {data, ack, nak} frame_kind;               /* определение типа фрейма */

typedef struct {                                        /* фреймы, транспортируемые
                                                           на данном уровне */
 frame_kind kind;                                       /* тип фрейма */
 seq_nr seq;                                            /* порядковый номер */
 seq_nr ack;                                            /* номер подтверждения */
 packet info;                                           /* пакет сетевого уровня */
} frame;
/* Ожидать события и вернуть тип события в переменной event. */
void wait_for_event(event_type *event);

/* Получить пакет у сетевого уровня для передачи по каналу. */
void from_network_layer(packet *p);

/* Передать информацию из полученного пакета сетевому уровню. */
 void to_network_layer(packet *p);

/* Получить пришедший пакет у физического уровня и скопировать его в r. */
void from_physical_layer(frame *r);

/* Передать фрейм физическому уровню для отправки. */
void to_physical_layer(frame *s);

/* Запустить таймер и разрешить событие timeout. */
void start_timer(seq_nr k);

/* Остановить таймер и запретить событие timeout. */
void stop_timer(seq_nr k);

/* Запустить вспомогательный таймер и разрешить событие ack_timeout. */
void start_ack_timer(void);

/* Остановить вспомогательный таймер и запретить событие ack_timeout. */
void stop_ack_timer(void);

/* Разрешить сетевому уровню инициировать событие network_layer_ready. */
void enable_network_layer(void);

/* Запретить сетевому уровню инициировать событие network_layer_ready. */
void disable_network_layer(void);

/* Макрос inc развертывается прямо в строке: циклически увеличить переменную k. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

**Илл. 3.11.** Общие определения для последующих протоколов. Определения располагаются в файле protocol.h

На илл. 3.11 также перечислен ряд процедур. Это библиотечные процедуры, детали которых зависят от конкретной реализации; их внутреннее устройство мы

рассматривать не будем. Как уже упоминалось ранее, процедура `wait_for_event` представляет собой холостой цикл ожидания какого-либо события. Процедура `to_network_layer` используется канальным уровнем для отправки пакетов сетевому, `from_network_layer` — для получения пакетов от него. Обратите внимание: процедуры `from_physical_layer` и `to_physical_layer` служат для обмена фреймами между канальным и физическим уровнями, тогда как процедуры `to_network_layer` и `from_network_layer` применяются для передачи пакетов между канальным и сетевым уровнями. Другими словами, процедуры `to_network_layer` и `from_network_layer` относятся к интерфейсу между уровнями 2 и 3, а процедуры `from_physical_layer` и `to_physical_layer` — к интерфейсу между уровнями 1 и 2.

В большинстве протоколов предполагается использование ненадежного канала, который может случайно потерять целый фрейм. Чтобы избежать неприятных последствий, при отправке фрейма передающий канальный уровень запускает таймер. Если за установленный интервал времени ответ не получен, срок ожидания истекает и канальный уровень получает сигнал прерывания.

В приведенных здесь протоколах этот сигнал реализован в виде значения `event=timeout`, возвращаемого процедурой `wait_for_event`. Для запуска и остановки таймера используются процедуры `start_timer` и `stop_timer` соответственно. Событие `timeout` может произойти, только если был запущен таймер, но еще не была вызвана процедура `stop_timer`. Процедуру `start_timer` разрешается запускать во время работающего таймера. Такой вызов просто перезапускает таймер, и отсчет начинается заново (до нового перезапуска или выключения).

Процедуры `start_ack_timer` и `stop_ack_timer` запускают и останавливают вспомогательные таймеры, используемые для создания подтверждений в некоторых ситуациях.

Процедуры `enable_network_layer` и `disable_network_layer` применяются в более сложных протоколах, где уже не предполагается, что у сетевого уровня всегда есть пакеты для отправки. Когда канальный уровень разрешает работу сетевого, последний может посылать сигнал прерывания, когда ему нужно передать пакет. Такое событие обозначается как `event = network_layer_ready`. Когда сетевой уровень отключен, он не может инициировать такие события. Канальный уровень тщательно следит за включением и выключением сетевого и не допускает ситуации, когда тот заваливает его пакетами, для которых нет места в буфере.

Последовательные номера фреймов всегда находятся в пределах от 0 до *MAX_SEQ* включительно. Число *MAX_SEQ* отличается в разных протоколах. Для увеличения последовательного номера фреймов на 1 циклически (то есть с обнулением при достижении числа *MAX_SEQ*) используется макрос `inc`. Он определен в виде макроса, поскольку используется прямо в строке в тех местах программы, где быстродействие является критичным. Как мы увидим далее, производительность сети часто ограничена скоростью выполнения протоколов. Определение простых операций в виде макросов (а не процедур) не снижает удобочитаемости программы, увеличивая при этом ее быстродействие.

Объявления на илл. 3.11 входят во все последующие протоколы. В целях экономии места и для наглядности они были извлечены и собраны вместе, но, по сути, они должны быть объединены с протоколами. В языке C такое объединение производится путем размещения определений в специальном файле заголовка (в данном случае protocol.h) и включения их в файлы протокола с помощью `#include` – директивы препроцессора C.

## 3.3.3. Симплексные протоколы канального уровня

В данном разделе мы рассмотрим три простых протокола, из них каждый следующий способен справиться с более реалистичной ситуацией.

### *Протокол «Утопия»: без управления потоком и без исправления ошибок*

В качестве первого примера мы рассмотрим самый простой протокол. Данные передаются только в одном направлении, а опасений, что где-то может произойти ошибка, даже не возникает. Сетевые уровни передающего и целевого устройств находятся в состоянии постоянной готовности. Время обработки минимально, размер буфера неограничен. А главное, линия связи между канальными уровнями никогда не теряет и не искажает фреймы. Этот совершенно нереальный протокол под названием «Утопия» показан на илл. 3.12. Он всего лишь демонстрирует базовую структуру, необходимую для построения настоящего протокола.

Протокол состоит из двух процедур, `sender1` (отправитель) и `receiver1` (получатель). Процедура `sender1` работает на канальном уровне отправляющего устройства, а процедура `receiver1` – на канальном уровне целевого. Ни последовательные номера, ни подтверждения не используются, поэтому *MAX_SEQ* не требуется. Единственным возможным событием является `frame_arrival` (то есть получение неповрежденного фрейма).

Процедура `sender1` представляет собой бесконечный цикл `while`, отправляющий данные на линию с максимально возможной скоростью. Тело цикла состоит из трех действий: получение пакета от сетевого уровня (всегда исправно работающего), формирование исходящего пакета с помощью переменной s и передача пакета адресату. «Утопия» использует только поле `info`, поскольку другие поля фрейма относятся к обработке ошибок и управлению потоком, а они в данном протоколе не применяются.

Процедура `receiver1` так же проста. Вначале она ожидает, пока что-нибудь произойдет (как уже упоминалось, единственным событием в данном протоколе может быть получение неповрежденного фрейма). Когда фрейм приходит, процедура `wait_for_event` возвращает управление, при этом переменной `event` присваивается значение `frame_arrival` (которое все равно игнорируется). Вызов процедуры `from_physical_layer` удаляет вновь прибывший фрейм из аппаратного буфера и помещает его в переменную `r`. Наконец, порция данных передается сетевому уровню, а канальный уровень переходит в режим ожидания следующего фрейма.

```
/* Протокол 1 («Утопия») обеспечивает только одностороннюю передачу данных — от отправителя к получателю.
   Предполагается, что в канале связи нет ошибок и получатель способен мгновенно обрабатывать входящие данные.
   Соответственно, отправитель в цикле передает данные на линию с максимально доступной для него скоростью.
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(void)
{
 frame s;                              /* буфер для исходящего фрейма */
 packet buffer;                        /* буфер для исходящего пакета */
 while (true) {
     from_network_layer(&buffer);      /* получить у сетевого уровня пакет для передачи */
     s.info = buffer;                  /* скопировать его во фрейм s для передачи */
     to_physical_layer(&s);            /* послать фрейм по каналу */
 }                                     /* Мы дни за днями шепчем: «Завтра, завтра».
                                          Так тихими шагами жизнь ползет
                                          К последней недописанной странице.
                                                — Макбет, V, v */
}
void receiver1(void)
{
 frame r;
 event_type event;                     /* заполняется процедурой ожидания событий, но не используется
                                          здесь */
 while (true) {
     wait_for_event(&event);           /* единственная возможность — доставка фрейма (событие frame_
                                          arrival) */
     from_physical_layer(&r);          /* получить прибывший фрейм */
     to_network_layer(&r.info);        /* передать данные сетевому уровню */
 }
}
```

**Илл. 3.12.** Неограниченный симплексный протокол «Утопия»

Протокол «Утопия» абсолютно нереалистичен, так как он не способен ни управлять потоком данных, ни исправлять ошибки. Он напоминает службу без подтверждения и без установки соединения, которая надеется, что все эти проблемы решаются на более высоких уровнях. Однако даже такая служба обладает некоторыми способностями распознавать ошибки.

## *Добавляем управление потоком: протокол с остановкой и ожиданием*

Усложним задачу: предположим, отправитель посылает данные слишком быстро и получатель не успевает их обработать. В реальности такая ситуация может возникнуть в любой момент, поэтому крайне важно научиться ее предотвращать. Допущение об отсутствии ошибок в канале связи сохраняется. Линия остается симплексной.

Одно из решений — сконструировать целевое устройство так, чтобы его мощности хватало на обработку непрерывного потока последовательных фреймов (или же установить на канальном уровне достаточно низкую скорость передачи

во избежание перегрузки получателя). У принимающей стороны должен быть буфер большого объема, а скорость обработки — не ниже скорости передачи данных. Кроме того, он должен быстро передавать фреймы сетевому уровню. Это наихудшее из возможных решений. Оно требует специального оборудования, а если линия загружена слабо, то ресурсы расходуются зря. Кроме того, такое решение всего лишь перекладывает проблему слишком быстрой передачи на чужие плечи: в данном случае ее приходится решать сетевому уровню.

Лучшее решение проблемы — обратная связь со стороны получателя. Передав пакет сетевому уровню, он посылает источнику небольшое служебное сообщение, разрешающее отправку следующего фрейма. Отправитель, отослав фрейм, должен ждать этого разрешения. Подобная задержка — простейший пример протокола с управлением потоком.

Протоколы, в которых отправитель посылает один фрейм, после чего ожидает подтверждения, называются **протоколами с остановкой и ожиданием (stop-and-wait)**. На илл. 3.13 приведен пример такого симплексного протокола.

Хотя пересылка данных в этом примере осуществляется по симплексному принципу, по направлению от отправителя получателю, на практике фреймы идут и в обратную сторону. Следовательно, линия связи между двумя канальными уровнями должна поддерживать двунаправленную передачу. Однако данный протокол диктует жесткое чередование направлений пересылки: источник и получатель отправляют фреймы строго по очереди. Для такой реализации хватило бы полудуплексного физического канала.

Как и в протоколе 1, в начале цикла отправитель извлекает пакет с сетевого уровня, формирует из него фрейм и отправляет фрейм по линии связи. Отличие в том, что теперь он должен ждать получения фрейма с подтверждением, прежде чем запустить новую итерацию цикла и обратиться к сетевому уровню за следующим пакетом. В данной модели канальный уровень отправителя даже не просматривает входящий фрейм, поскольку он всегда означает только одно: подтверждение.

Единственное отличие между процедурами `receiver2` и `receiver1` состоит в том, что после передачи пакета сетевому уровню `receiver2` посылает подтверждение обратно отправителю, после чего идет на следующую итерацию цикла. Поскольку для отправителя важно само прибытие ответного фрейма, а не его содержание, то получателю не нужно заполнять его специальной информацией.

### *Добавляем исправление ошибок: порядковые номера и протокол ARQ*

Теперь рассмотрим реальную ситуацию: канал связи, в котором могут быть ошибки. Фреймы могут либо повреждаться, либо теряться. Однако мы будем считать, что если фрейм был изменен при передаче, то аппаратное обеспечение целевого устройства определит это, подсчитав контрольную сумму. Если фрейм поврежден таким образом, что контрольная сумма сходится (что очень маловероятно), то этот и любой другой протокол могут дать сбой, то есть отправить на сетевой уровень пакет с ошибками.

```
/* Протокол 2 (с ожиданием) также обеспечивает только одностороннюю передачу данных,
   от отправителя к получателю. Снова предполагается, что в канале связи нет ошибок. Однако
   на этот раз емкость буфера получателя ограниченна и, кроме того, ограниченна скорость
   обработки данных получателем. Поэтому протокол должен не допускать отправления данных
   быстрее, чем получатель способен их обработать. */
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender2(void)
{
frame s;                              /* буфер для исходящего фрейма */
packet buffer;                        /* буфер для исходящего пакета */
event_type event;                     /* frame_arrival является единственным возможным событием */
while (true) {
    from_network_layer(&buffer);      /* получить у сетевого уровня пакет для передачи */
    s.info = buffer;                  /* скопировать его во фрейм s для передачи */
    to_physical_layer(&s);            /* отправка фрейма */
    wait_for_event(&event);           /* не продолжать, пока на это не будет получено разрешения */
 }
}
void receiver2(void)
{
 frame r, s;                          /* буферы для фреймов */
 event_type event;                    /* frame_arrival является единственным возможным событием */
 while (true) {
     wait_for_event(&event);          /* единственная возможность — прибытие фрейма (событие frame_
                                         arrival) */
     from_physical_layer(&r);         /* получить входящий фрейм */
     to_network_layer(&r.info);       /* передать данные сетевому уровню */
     to_physical_layer(&s);           /* передать пустой фрейм, чтобы «разбудить» отправителя */
 }
}
```

**Илл. 3.13.** Симплексный протокол с ожиданием и остановкой

На первый взгляд может показаться, что можно улучшить протокол 2, добавив таймер. Получатель будет возвращать подтверждение только в случае получения правильных данных. Неверные пакеты будут просто игнорироваться. Через некоторое время у отправителя истечет интервал времени и он отправит фрейм еще раз. Этот процесс будет повторяться до тех пор, пока фрейм наконец не прибудет в целости.

В приведенной выше схеме имеется один критический недостаток. Прежде чем читать дальше, попытайтесь понять, что же неверно в данном варианте протокола.

Чтобы разобраться, что может пойти не так, вспомните, что цель канального уровня заключается в предоставлении безошибочной прозрачной связи между двумя процессами сетевого уровня. Сетевой уровень устройства *A* передает серию пакетов своему канальному уровню, который должен обеспечить доставку идентичной серии пакетов сетевому уровню устройства *B* его канальным уровнем. В частности, сетевой уровень *B* не может распознать потерю или дублирование

пакета, поэтому канальный уровень должен гарантировать, что повтора не произойдет ни при каких обстоятельствах.

Рассмотрим следующий сценарий.

1. Сетевой уровень устройства *A* передает пакет 1 своему канальному уровню. Пакет доставляется в целости на устройство *B* и передается его сетевому уровню. *B* посылает фрейм подтверждения обратно на *A*.
2. Фрейм подтверждения полностью теряется в канале связи. Он просто не попадает на устройство *A*. Все было бы намного проще, если бы терялись только информационные, но не управляющие фреймы, но, к сожалению, канал связи не способен их различать.
3. У канального уровня устройства *A* внезапно истекает отведенный интервал времени. Не получив подтверждения, оно предполагает, что отправленный им фрейм с данными был поврежден или потерян, и посылает его еще раз.
4. Дубликат фрейма в целости прибывает на канальный уровень *B* и передается на сетевой уровень. В итоге часть файла, переданного с *A* на *B*, дублируется. Копия файла на устройстве *B* будет неверной, и ошибка не будет обнаружена, другими словами, протокол даст сбой.

Разумеется, необходим некий механизм, с помощью которого получатель смог бы различать новые фреймы и переданные повторно. Наиболее очевидное решение — нумерация фреймов. Отправитель указывает порядковый номер фрейма в его заголовке. Благодаря этому принимающее устройство отличает новый фрейм от дубликата, который необходимо проигнорировать.

Необходимо, чтобы протокол выполнялся без ошибок, а нумерация не занимала много места в заголовке фрейма, поскольку соединение должно использоваться эффективно. Возникает вопрос: каково минимальное количество битов, достаточное для порядкового номера фрейма? В зависимости от протокола можно выделить один или несколько битов (или байтов). Важно, чтобы номера были достаточно большими для правильной работы протокола, иначе он будет бесполезен.

Единственная неопределенность в данном протоколе может возникнуть между фреймом $m$ и следующим за ним фреймом $m + 1$. Если $m$ потерян или поврежден, получатель не подтвердит его и отправитель повторит передачу. Когда он будет успешно принят, получатель отправит подтверждение. Именно здесь находится источник потенциальной проблемы. В зависимости от наличия подтверждения отправитель дублирует фрейм $m$ или передает новый фрейм $m + 1$.

На стороне отправителя событием, запускающим передачу фрейма $m + 1$, является получение подтверждения доставки фрейма $m$. Но это означает, что фрейм $m - 1$ уже передан и подтверждение его доставки отправлено и получено. В противном случае протокол не стал бы посылать новый фрейм. Следовательно, неопределенность может возникнуть только между двумя соседними фреймами.

Таким образом, для нумерации фрейма достаточно всего одного бита информации (со значением 0 или 1). В каждый момент времени получатель ожидает прибытия фрейма с определенным порядковым номером. Фрейм с верным номером принимается, передается сетевому уровню, затем отправляется подтверждение его получения. Номер следующего ожидаемого фрейма увеличивается по

модулю 2 (то есть 0 становится 1, а 1 – 0). Фрейм с неверным номером отбрасывается как дубликат. Однако последнее подтверждение повторяется, чтобы сообщить отправителю, что фрейм получен полностью.

Пример такого протокола представлен на илл. 3.14. Протокол, в котором отправитель ожидает положительного подтверждения, прежде чем перейти к пере-

```
/* Протокол 3 (PAR) обеспечивает симплексную передачу данных по ненадежному каналу.
#define MAX_SEQ 1                                   /* в протоколе 3 должно быть равно 1 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
 seq_nr next_frame_to_send;                         /* порядковый номер следующего исходящего фрейма */
 frame s;                                           /* временная переменная */
 packet buffer;                                     /* буфер для исходящего пакета */
 event_type event;

 next_frame_to_send = 0;                            /* инициализация исходящих последовательных номеров */
 from_network_layer(&buffer);                       /* получить первый пакет у сетевого уровня */
 while (true) {
     s.info = buffer;                               /* сформировать фрейм для передачи */
     s.seq = next_frame_to_send;                    /* вставить порядковый номер во фрейм */
     to_physical_layer(&s);                         /* послать фрейм по каналу */
     start_timer(s.seq);                            /* запустить таймер ожидания подтверждения */
     wait_for_event(&event);                        /* ждать событие frame_arrival, cksum_err или timeout */
     if (event == frame_arrival) {
           from_physical_layer(&s);                 /* получить подтверждение */
          if (s.ack == next_frame_to_send) {
               stop_timer(s.ack);                   /* остановить таймер
                 from_network_layer(&buffer);       /* получить следующий исходящий пакет */
                 inc(next_frame_to_send);           /* инвертировать значение переменной next_frame_to_send */
          }
     }
 }
}
void receiver3(void)
{
 seq_nr frame_expected;
 frame r, s;
 event_type event;

 frame_expected = 0;
 while (true) {
     wait_for_event(&event);                        /* ожидание возможных событий: frame_arrival, cksum_err */
     if (event == frame_arrival) {                  /* пришел неповрежденный фрейм */
           from_physical_layer(&r);                 /* получить пришедший фрейм */
           if (r.seq == frame_expected) {           /* именно этот фрейм и ожидался */
                to_network_layer(&r.info);          /* передать данные сетевому уровню */
                inc(frame_expected);                /* в следующий раз ожидать фрейм с другим порядковым номером */
          }
          s.ack = 1 - frame_expected;               /* номер фрейма, для которого посылается подтверждение */
          to_physical_layer(&s);                    /* отправить подтверждение */
     }
 }
}
```

**Илл. 3.14.** Протокол с положительным подтверждением и повторной передачей

сылке следующего фрейма, часто называется **PAR (Positive Acknowledgement with Retransmission — положительное подтверждение с повторной передачей)** или **ARQ (Automatic Repeat reQuest — автоматический запрос повторной передачи)**. Подобно протоколу 2, он передает данные только в одном направлении.

Протокол 3 отличается от своих предшественников тем, что и отправитель, и получатель содержат переменную, значение которой хранится, пока канальный уровень находится в режиме ожидания. Отправитель запоминает номер следующего фрейма в переменной `next_frame_to_send`, а получатель записывает номер следующего ожидаемого фрейма в переменной `frame_expected`. Каждый протокол имеет короткую фазу инициализации перед началом бесконечного цикла.

После передачи фрейма отправитель запускает таймер. Если он уже работал, он перезапускается для отсчета нового полного интервала времени. Этот интервал должен быть достаточно большим, чтобы даже при наихудшем сценарии фрейм успел дойти до получателя, тот успел его обработать и подтверждение вернулось к отправителю. Только по истечении отведенного времени можно предположить потерю фрейма или его подтверждения и отправить дубликат. Если интервал слишком короткий, то передающее устройство будет повторно посылать слишком много фреймов, в которых нет необходимости. Хотя лишние фреймы в данном случае не повлияют на правильность приема данных, это снизит производительность системы.

После передачи фрейма отправитель запускает таймер и ждет какого-либо события. Возможны три ситуации: либо придет неповрежденный фрейм подтверждения, либо будет получен поврежденный фрейм подтверждения, либо просто истечет интервал времени. В первом случае отправитель возьмет у сетевого уровня следующий пакет и разместит его в буфере поверх предыдущего, а также увеличит порядковый номер фрейма. Если же прибудет поврежденный фрейм подтверждения или время истечет, то ни буфер, ни номер не будут изменены и будет отправлен дубликат фрейма. В любом случае затем посылается содержимое буфера (следующий пакет либо дубликат предыдущего).

Когда неповрежденный фрейм прибывает к получателю, проверяется его номер. Если это не дубликат, то фрейм принимается и передается сетевому уровню, после чего формируется подтверждение. Дубликаты и поврежденные фреймы на сетевой уровень не передаются, но при их получении подтверждается прибытие последнего правильного фрейма, благодаря чему отправитель понимает, что нужно перейти к следующему фрейму или повторить передачу поврежденного.

## 3.4. ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ

В предыдущих протоколах фреймы данных передавались только в одну сторону. На практике чаще всего требуется передача в обоих направлениях. Кроме того, канальный уровень работает более эффективно, если по нему можно отправлять сразу несколько фреймов, не дожидаясь, пока придет подтверждение. Далее мы изучим обе эти концепции и рассмотрим несколько примеров протоколов, реализующих поставленные цели.

## 3.4.1. Цель: двунаправленная передача, отправка сразу нескольких фреймов

Ниже представлен метод так называемого вложенного подтверждения, позволяющий протоколу канального уровня обеспечить двунаправленную (дуплексную) передачу, а также метод раздвижного окна, позволяющий повысить эффективность передачи за счет отправки сразу нескольких фреймов.

### *Двунаправленная передача данных: вложенное подтверждение*

Один из способов осуществления дуплексной передачи — использование двух экземпляров описанных выше протоколов, каждый из которых передает данные по отдельной симплексной связи (в противоположных направлениях). При этом каждое соединение включает прямой канал для данных и обратный — для подтверждений. В обоих случаях пропускная способность обратного канала почти не используется.

Гораздо эффективнее использовать для дуплексной передачи один канал. К тому же в протоколах 2 и 3 фреймы уже передавались по каналу в двух направлениях, при этом обратный канал обладает той же пропускной способностью, что и прямой. В такой модели фреймы данных и подтверждения, которые устройство *A* отправляет устройству *B*, перемешиваются. Получатель может отличить фрейм данных от фрейма с подтверждением по специальному полю заголовка фрейма — `kind`.

Помимо чередования подтверждений и информационных фреймов возможно и другое улучшение протокола. Приняв фрейм данных, получатель может не посылать фрейм с подтверждением сразу, а подождать, пока сетевой уровень даст ему следующий пакет. Подтверждение добавляется к исходящему фрейму данных с помощью поля `ack` в заголовке. В результате на передачу подтверждения почти не расходуются ресурсы. Подход, при котором подтверждения временно откладываются и прикрепляются к следующему исходящему фрейму данных, называется **вложенным подтверждением (piggybacking)**.

Основное преимущество вложенного подтверждения — более эффективное использование пропускной способности канала. Поле `ack` в заголовке фрейма занимает всего несколько битов, тогда как отдельный фрейм потребует заголовка и контрольной суммы. Кроме того, чем меньше входящих фреймов, тем меньше нагрузка на получателя. В следующем протоколе, который мы рассмотрим, расходы на поле вложенного подтверждения в заголовке фрейма составляют всего 1 бит (они редко превышают несколько битов).

Однако использование вложенного подтверждения ведет к появлению новых проблем. Как долго канальный уровень должен ждать пакета, с которым следует переслать подтверждение? Если он будет ждать дольше, чем отправитель, то последний пошлет фрейм повторно и сама идея подтверждений потеряет смысл. Если бы канальный уровень мог предсказывать будущее, он бы знал, ждать ему пакет или отправить подтверждение отдельно. К сожалению, это невозможно, поэтому следует установить еще один интервал ожидания (меньший, чем у отправителя), по истечении которого подтверждение отправляется отдельным

фреймом. Если же сетевой уровень успеет передать пакет канальному уровню, то подтверждение будет отослано вместе с ним в одном фрейме.

### Раздвижное окно

Следующие три протокола являются двунаправленными и принадлежат к классу протоколов **раздвижного окна (sliding window)**. Как будет показано ниже, они отличаются друг от друга эффективностью, сложностью и требованиями к размерам буфера. Во всех протоколах раздвижного окна каждый исходящий фрейм содержит порядковый номер (в диапазоне от 0 до некоторого максимума). Этот номер должен поместиться в поле размером $n$ бит, поэтому его максимальное значение обычно составляет $2^n - 1$. В протоколах раздвижного окна с остановкой и ожиданием $n = 1$; это ограничивает порядковый номер значениями 0 и 1, однако в более сложных версиях может использоваться произвольное значение $n$.

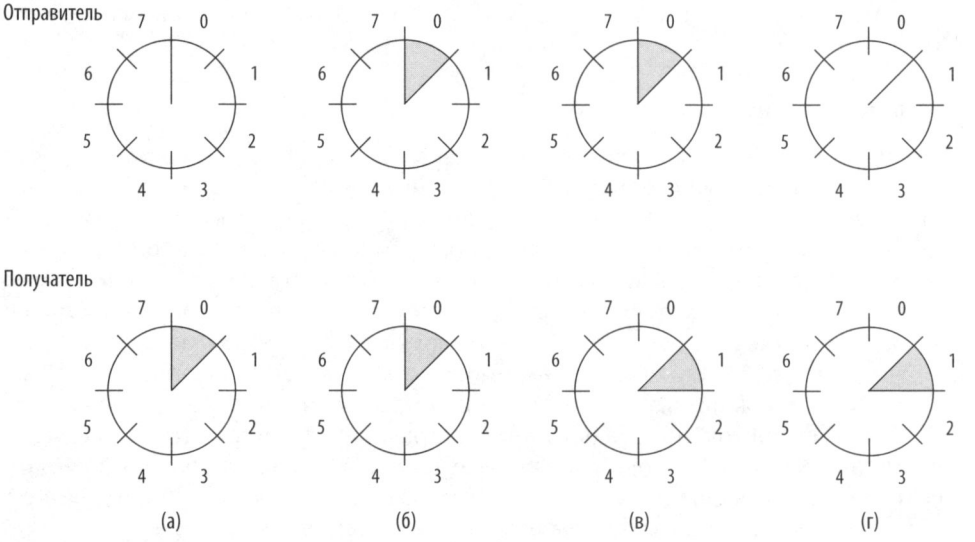
Суть всех протоколов раздвижного окна заключается в том, что отправитель постоянно работает с набором порядковых номеров, соответствующих фреймам, которые ему разрешено передавать. Эти фреймы находятся в **передающем окне**. Аналогично получатель работает с **приемным окном**, содержащим набор фреймов, которые можно принять. Окна получателя и отправителя не должны иметь одинаковые нижний и верхний пределы или даже быть одного размера. В одних протоколах размеры фиксируются, а в других они могут увеличиваться или уменьшаться по мере передачи или приема фреймов.

Данные протоколы предоставляют канальному уровню большую свободу в отношении последовательности передачи и приема фреймов. Но требование доставки пакетов целевому сетевому уровню в том же порядке, в котором они были получены от передающего сетевого уровня, по-прежнему действует. Физический канал связи также должен доставлять фреймы в порядке отправления (подобно проводу).

Порядковые номера в передающем окне соответствуют уже отправленным фреймам, для которых еще не пришли подтверждения. Пришедший от сетевого уровня пакет получает наибольший порядковый номер, и верхняя граница окна увеличивается на единицу. Когда поступает подтверждение, на единицу возрастает нижняя граница окна. Таким образом, окно постоянно содержит список неподтвержденных фреймов. Пример такого протокола показан на илл. 3.15.

Фреймы, находящиеся в окне отправителя, могут быть потеряны или повреждены во время передачи, поэтому их нужно хранить в памяти на случай возможной повторной отправки. Таким образом, если максимальный размер окна равен $n$, то отправителю потребуется $n$ буферов для хранения неподтвержденных фреймов. Если окно достигает максимального размера, канальный уровень должен отключить сетевой уровень до тех пор, пока не освободится буфер.

Окно принимающего канального уровня соответствует фреймам, которые он может принять. Любой фрейм, попадающий в окно, помещается в буфер получателя. Когда приходит фрейм с порядковым номером, соответствующим нижнему краю окна, он передается на сетевой уровень и окно сдвигается на одну позицию. Любой фрейм, не попадающий в окно, удаляется. В любом случае формируется подтверждение, чтобы отправитель мог понять, как ему действовать




**Илл. 3.15.** Раздвижное окно размера 1 с 3-битным порядковым номером. (а) Начальная ситуация. (б) После отправки первого фрейма. (в) После приема первого фрейма. (г) После приема первого подтверждения

дальше. Обратите внимание, что если размер окна равен единице, то канальный уровень может принимать фреймы только в установленном порядке, однако при больших размерах окна последовательность может нарушаться. Сетевому уровню, напротив, данные всегда предоставляются в строгом порядке, независимо от размера окна канального уровня.

На илл. 3.15 показан пример для окна с максимальным размером 1. Вначале фреймов в окне нет, поэтому оно пустое и его верхний и нижний края совпадают, однако с течением времени ситуация меняется. В отличие от окна отправителя, окно получателя всегда сохраняет первоначальный размер, сдвигаясь по мере приема и передачи на сетевой уровень очередного фрейма.

## 3.4.2. Примеры дуплексных протоколов раздвижного окна

Теперь рассмотрим пример простого однобитного протокола раздвижного окна, а также протоколы, способные обеспечить повторную передачу ошибочных фреймов в случае передачи сразу нескольких фреймов.

### *Однобитное раздвижное окно*

Прежде чем рассматривать общий случай, изучим протокол раздвижного окна, равного 1. Такой протокол использует метод ожидания: отослав фрейм, отправитель должен дождаться подтверждения, прежде чем послать следующий фрейм.

Данный протокол показан на илл. 3.16. Как и другие протоколы, он начинается с определения некоторых переменных. Переменная `next_frame_to_send`

содержит номер фрейма, который отправитель пытается послать. Аналогично переменная `frame_expected` хранит номер фрейма, ожидаемого получателем. В обоих случаях возможными значениями могут быть только 0 и 1.

```
/* Протокол 4 (раздвижное окно) является дуплексным
#define MAX_SEQ 1                                   /* в протоколе 4 должно быть равно 1
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
 seq_nr next_frame_to_send;                         /* только 0 или 1 */
 seq_nr frame_expected;                             /* только 0 или 1 */
 frame r, s;                                        /* временные переменные */
 packet buffer;                                     /* текущий отправленный пакет */
 event_type event;

 next_frame_to_send = 0;                            /* номер следующего фрейма в исходящем потоке */
 frame_expected = 0;                                /* номер ожидаемого фрейма */
 from_network_layer(&buffer);                       /* получить первый пакет у сетевого уровня */
 s.info = buffer;                                   /* подготовить первый фрейм для передачи */
 s.seq = next_frame_to_send;                        /* вставить порядковый номер во фрейм */
 s.ack = 1 - frame_expected;                        /* подтверждение, вложенное во фрейм данных */
 to_physical_layer(&s);                             /* передать фрейм */
 start_timer(s.seq);                                /* запустить таймер ожидания подтверждения */
 while (true) {
     wait_for_event(&event);                        /* ждать события frame_arrival, cksum_err
                                                       или timeout */
     if (event == frame_arrival) {                  /* фрейм пришел в целости */
          from_physical_layer(&r);                  /* получить пришедший фрейм */
          if (r.seq == frame_expected) {            /* обработать входящий поток фреймов */
                to_network_layer(&r.info);          /* передать пакет сетевому уровню */
                inc(frame_expected);                /* инвертировать порядковый номер фрейма,
                                                       ожидаемого в следующий раз */
          }
          if (r.ack == next_frame_to_send) {        /* обработать исходящий поток фреймов */
                stop_timer(r.ack);                  /* остановить таймер */
                from_network_layer(&buffer);        /* получить следующий пакет у сетевого уровня */
                inc(next_frame_to_send);            /* инвертировать порядковый номер посылаемого
                                                       фрейма */
          }
     }
     s.info = buffer;                               /* подготовить фрейм для передачи */
     s.seq = next_frame_to_send;                    /* вставить порядковый номер во фрейм */
     s.ack = 1 - frame_expected;                    /* порядковый номер последнего полученного
                                                       фрейма */
     to_physical_layer(&s);                         /* передать фрейм */
     start_timer(s.seq);                            /* запустить таймер ожидания подтверждения */
 }
}
```

**Илл. 3.16.** Однобитный протокол раздвижного окна

В обычной ситуации только один канальный уровень может начинать передачу. Другими словами, лишь одна из программ должна содержать обращения к процедурам `to_physical_layer` и `start_timer` вне основного цикла. Отправитель получает первый пакет от своего сетевого уровня, создает из него фрейм и посылает его. Когда этот (или другой) фрейм приходит, получающий канальный уровень проверяет, не является ли он дубликатом, аналогично протоколу 3. Если это тот фрейм, который ожидался, он передается сетевому уровню и окно получателя сдвигается вверх.

Поле подтверждения содержит номер последнего полученного без ошибок фрейма. Если он совпадает с номером фрейма, который пытается передать отправитель, тот понимает, что этот фрейм успешно принят получателем и что можно пересылать следующий. В противном случае он продолжает попытки передачи того же фрейма. При каждом получении фрейма производится отправка фрейма в обратном направлении.

Теперь изучим протокол 4 и посмотрим, насколько он устойчив к нестандартным ситуациям. Представим, что устройство *A* пытается послать фрейм 0 устройству *B*, при этом *B* пытается отправить *A* фрейм 0. Предположим, что на *A* установлен слишком короткий период ожидания подтверждения. Соответственно, *A* посылает серию одинаковых фреймов со значениями полей `seq` = 0 и `ack` = 1.

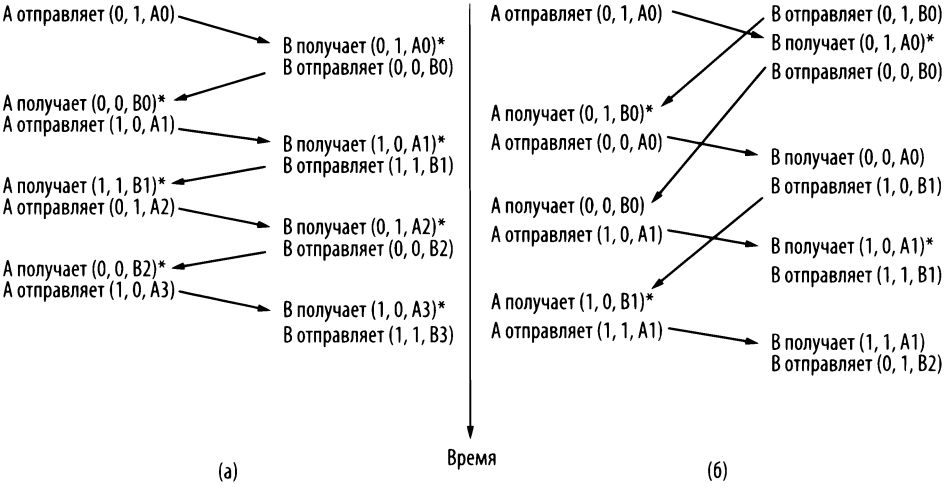
Когда первый неповрежденный фрейм придет на устройство *B*, он будет принят и значение переменной `frame_expected` будет равно 1. Все последующие входящие фреймы будут проигнорированы, поскольку *B* теперь ожидает фрейм с порядковым номером 1, а не 0. Более того, поскольку у всех дубликатов значение поля `ack` = 1, а устройство *B* продолжает ожидать подтверждения для фрейма 0, оно не станет запрашивать новый пакет у своего сетевого уровня.

В ответ на каждый отвергнутый дубликат, присылаемый устройством *A*, *B* посылает фрейм, содержащий поля `seq` = 0 и `ack` = 0. Наконец, один из этих фреймов принимается *A*, в результате чего *A* переходит к передаче следующего пакета. Никакая комбинация потерянных фреймов или преждевременно истекших интервалов ожидания не может заставить этот протокол ни выдать сетевому уровню дубликат пакета, ни пропустить пакет, ни зависнуть. Протокол работает корректно.

Однако отношения между протоколами могут быть довольно непростыми. Если обе стороны одновременно вышлют друг другу начальный пакет, возникает запутанная ситуация, представленная на илл. 3.17. В левой части рисунка показан случай нормального функционирования протокола. Правая часть демонстрирует отклонение. Если *B* ожидает первый фрейм от *A*, прежде чем послать свой первый фрейм, то последовательность будет как в левой части рисунка (а). При этом принимается каждый фрейм.

Но если *A* и *B* одновременно начнут передачу, их первые фреймы пересекутся и канальные уровни попадут в ситуацию (б). В ситуации (а) с каждым фреймом приходит новый пакет и дубликатов нет. В случае (б) половина фреймов содержит дубликаты, несмотря на то что ошибок в канале связи не было. Причиной может стать преждевременный тайм-аут, даже если одна сторона явно

начинает диалог первой. Получается, если время ожидания истечет слишком быстро, фрейм может быть послан три и более раза, приводя к ненужной трате ценных ресурсов.




**Илл. 3.17.** Два сценария для протокола 4. (а) Нормальная ситуация. (б) Нештатная ситуация. Обозначения в скобках: (seq, ack, номер пакета). Звездочка показывает, что сетевой уровень принял пакет

## *Протокол с возвратом к n*

До сих пор мы по умолчанию подразумевали, что время, необходимое на передачу фрейма от отправителя к получателю и на обратную передачу подтверждения, пренебрежимо мало. Иногда это совершенно не так. Длительное время прохождения фреймов по сети может значительно снижать эффективность использования пропускной способности канала. В качестве примера рассмотрим 50-килобитный спутниковый канал. Время прохождения сигнала в оба конца равно 500 мс. Попытаемся использовать протокол 4 для пересылки фреймов размером 1000 бит через спутник. В момент времени $t = 0$ начинается отправка первого фрейма. При $t = 20$ мс фрейм полностью отправлен. Не раньше чем при $t = 270$ мс получатель принимает фрейм целиком и высылает подтверждение. Отправитель получает его в момент времени $t = 520$ мс (в самом лучшем случае — при нулевом времени ожидания на принимающей стороне и коротком фрейме подтверждения). Это означает, что отправляющее устройство заблокировано в течение 500/520, или 96 % времени. Другими словами, использовалось только 4 % доступной полосы пропускания. Очевидно, что сочетание большого времени прохождения сигнала, высокой пропускной способности и коротких фреймов совершенно неприемлемо с точки зрения эффективности.

Описанная выше проблема — следствие правила, которое обязывает отправителя дожидаться подтверждения, прежде чем посылать следующий фрейм. Смягчив это требование, можно значительно повысить эффективность. Нужно

разрешить отправителю посылать не один фрейм, а несколько, например $w$, прежде чем остановиться и перейти в режим ожидания подтверждений. Если подобрать достаточно большое число $w$, то отправитель сможет безостановочно посылать фреймы, так как подтверждения для предыдущих фреймов будут приходить до того, как окно заполнится. Это предотвратит блокировку отправителя.

Чтобы найти подходящее значение $w$, необходимо понять, сколько фреймов «вмещается» в канал, в то время как они передаются от отправителя к получателю. Емкость определяется путем умножения полосы пропускания в битах в секунду на время пересылки в одну сторону. Это значение можно разделить на число битов во фрейме, чтобы выразить число фреймов. Назовем эту величину $BD$ **(bandwidth-delay product — полоса пропускания, умноженная на задержку)**. Следовательно, $w$ должно соответствовать $2BD + 1$. $2BD$ — это число фреймов, которое может находиться в пути (неподтвержденные фреймы), если отправитель передает фреймы непрерывно (с учетом времени на получение подтверждения). Единица прибавляется, так как фрейм подтверждения отправляется только после приема полного фрейма данных.

В качестве примера возьмем канал с полосой пропускания 50 Кбит/с, в котором на пересылку фрейма в одну сторону тратится 250 мс. Значение $BD$ равно 12,5 Кбит/с или 12,5 фрейма, каждый из которых включает 1000 бит. $2BD + 1$ равно 26 фреймам. Отправитель начинает, как и ранее, с передачи фрейма 0 и отправляет очередной фрейм каждые 20 мс. К тому моменту, когда он закончит отправку 26 фреймов (при $t = 520$ мс), как раз прибудет подтверждение фрейма 0. Затем подтверждения станут приходить каждые 20 мс. Таким образом, отправитель будет получать разрешения на передачу следующего фрейма как раз вовремя. Начиная с этого момента у отправителя будет 25 или 26 неподтвержденных фреймов и, следовательно, достаточно будет окна размером 26.

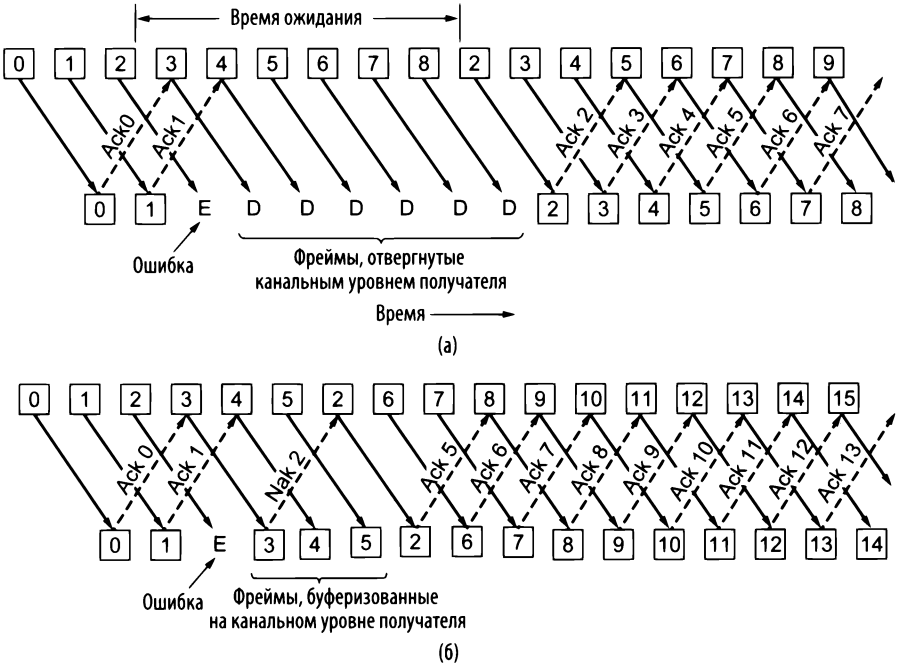
При меньшем размере окна канал будет загружен не на 100 %, так как иногда отправитель будет блокироваться. Коэффициент загруженности можно выразить как долю времени, когда отправитель не заблокирован:

$$\text{Коэффициент загруженности канала} \le \frac{w}{1+2BD}.$$

Данное значение выражает верхнюю границу, при этом не учитывается время на обработку фрейма и считается, что длина фрейма подтверждения равна нулю (обычно они действительно короткие). Из этого неравенства понятно, что для высоких значений $BD$ необходимо устанавливать большое значение размера окна $w$. Если задержка большая, то отправитель быстро опустошит свое окно даже при средней полосе пропускания, как в примере со спутником. Если полоса пропускания широкая, то даже при средней задержке отправитель также быстро обработает окно, если только оно не отличается особо крупным размером (например, канал с пропускной способностью 1 Гбит/с и задержкой в 1 мс удерживает 1 Мбит). Если у протокола с остановкой и ожиданием значение $w = 1$, то даже при задержке распространения, равной всего одному фрейму, его эффективность падает ниже 50 %.

Метод одновременной отправки сразу нескольких фреймов называется **конвейерной обработкой (pipelining)**. При конвейерном режиме передачи фреймов по ненадежному каналу возникает ряд серьезных проблем. Что произойдет, если в середине длинного потока повредится или потеряется фрейм? Большое количество последующих фреймов придет к получателю прежде, чем отправитель обнаружит ошибку. Когда поврежденный фрейм приходит к получателю, он, конечно, должен быть отвергнут. Но что делать получателю со всеми правильными последующими фреймами? Как уже говорилось, получающий канальный уровень обязан передавать пакеты сетевому уровню, соблюдая строгий порядок.

Существует два базовых подхода к исправлению ошибок при конвейерной обработке. Они показаны на илл. 3.18.




**Илл. 3.18.** Конвейеризация и коррекция ошибок. (а) Эффект при размере окна, равном 1. (б) Эффект при размере окна больше 1

Первый способ называется **возвратом к *n* (go-back-n)** и заключается в том, что получатель просто игнорирует все фреймы, следующие за ошибочным. Для таких фреймов подтверждения не посылаются. Эта стратегия соответствует окну получателя размером 1. Другими словами, канальный уровень отказывается принимать какой-либо фрейм, кроме фрейма со следующим номером, который он должен передать сетевому уровню. Если окно отправителя заполнится раньше, чем истечет период времени ожидания, конвейер начнет простаивать. Наконец, лимит времени у отправителя истечет, и он станет передавать повторно сразу

все фреймы, не получившие подтверждения, начиная с поврежденного или потерянного фрейма. Такой подход при высоком уровне ошибок может привести к потере большой доли пропускной способности канала.

На илл. 3.18 (а) изображен возврат к *n* при большом окне получателя. Фреймы 0 и 1 корректно принимаются, и высылается подтверждение этого факта. Однако фрейм 2 потерялся или был испорчен. Ничего не подозревающий отправитель продолжает посылать фреймы, пока не выйдет время ожидания фрейма 2. Только после этого он возвращается к месту сбоя и заново передает все фреймы, начиная с фрейма 2 (отправляя 2, 3, 4 и т. д.).

### *Выборочный повтор*

Протокол с возвратом к *n* хорошо работает, если ошибки встречаются нечасто, однако при плохом соединении он впустую тратит время и ресурсы, передавая фреймы по два раза. В качестве альтернативы можно использовать протокол с **выборочным повтором (selective repeat)**, который позволяет получателю принимать и буферизировать фреймы, переданные после поврежденного или утерянного фрейма.

При этом неверный фрейм отбрасывается. Когда заканчивается время ожидания подтверждения, отправитель посылает еще раз только самый старый фрейм, для которого не пришло подтверждение. Если вторая попытка будет успешной, получатель сможет последовательно передать накопившиеся пакеты сетевому уровню. Выборочный повтор используется, когда размер окна получателя больше 1. При большом окне этот подход может потребовать значительного количества памяти для принимающего канального уровня.

Выборочный повтор часто комбинируется с отправкой получателем **отрицательного подтверждения (negative acknowledgement, NAK)** при обнаружении ошибки (например, неверной контрольной суммы или измененного порядка следования фреймов). NAK стимулируют повторную отправку еще до того, как закончится время ожидания подтверждения от отправителя. Таким образом, эффективность работы несколько повышается.

На илл. 3.18 (б) фреймы 0 и 1 снова принимаются корректно, а фрейм 2 теряется. После получения фрейма 3 канальный уровень получателя замечает, что один фрейм выпал из последовательности. Для фрейма 2 отправителю посылается NAK, однако фрейм 3 сохраняется в специальном буфере. Далее приходят фреймы 4 и 5, они также буферизируются канальным уровнем вместо передачи на сетевой уровень. NAK 2 приходит к отправителю, заставляя его переслать фрейм 2. Когда последний оказывается у получателя, у уровня передачи данных уже имеются фреймы 2, 3, 4 и 5, которые сразу же в нужном порядке отдаются сетевому уровню. Теперь можно выслать подтверждение получения всех фреймов, включая пятый, что и показано на рисунке. Если NAK вдруг потеряется, то отправитель по окончании времени ожидания подтверждения сам повторит отправку фрейма 2 (и только его), однако это может произойти значительно позже, чем при помощи NAK.

Выбор одной из двух приведенных выше стратегий является компромиссом между эффективным использованием пропускной способности и размером буфера канального уровня. В зависимости от того, что в конкретной ситуации является

более критичным, может использоваться тот или иной метод. На илл. 3.19 показан протокол с возвратом к *n*, в котором канальный уровень принимает фреймы по порядку. Все фреймы, следующие за ошибочным, игнорируются. В данном протоколе мы впервые отказались от допущения, что у сетевого уровня всегда есть неограниченное количество пакетов для отсылки. Когда появляется готовый для отправки пакет, сетевой уровень может инициировать событие `network_layer_ready`. Чтобы контролировать размер окна отправителя или число неподтвержденных фреймов в любой момент времени, канальный уровень должен иметь возможность на время отключать сетевой. Для этой цели служит пара библиотечных процедур: `enable_network_layer` и `disable_network_layer`.

В любой момент времени максимальное число неподтвержденных фреймов не совпадает с количеством порядковых номеров. Для протокола с возвратом к *n* таких фреймов может быть *MAX_SEQ*, при этом имеется *MAX_SEQ* + 1 порядковых номеров: от 0 до *MAX_SEQ*. В протоколе с выборочным повтором мы увидим еще более жесткое ограничение. Чтобы понять, почему оно необходимо, рассмотрим сценарий с *MAX_SEQ* = 7.

1. Отправитель посылает фреймы с 0-го по 7-й.
2. Вложенное подтверждение для фрейма 7 приходит к отправителю.
3. Отправитель посылает следующие восемь фреймов, снова с номерами с 0 по 7.
4. Еще одно вложенное подтверждение для фрейма 7 доставляется отправителю.

Вопрос: все восемь фреймов из второго набора благополучно дошли до адресата или все они потерялись (включая проигнорированные фреймы после ошибочного)? В обеих ситуациях получатель отправит фрейм 7 в качестве подтверждения. У отправителя нет способа отличить один случай от другого. По этой причине максимальное количество неподтвержденных фреймов должно быть ограничено числом *MAX_SEQ* (а не *MAX_SEQ* + 1).

Хотя в протоколе 5 фреймы, поступившие после ошибки, не буферизируются получателем, отправитель должен хранить отправленные фреймы в своем буфере, пока не получит для них подтверждение.

Если поступает подтверждение на фрейм *n*, фреймы *n* – 1, n – 2 (и все предыдущие фреймы) автоматически считаются подтвержденными. Такой тип подтверждения называется **кумулятивным (cumulative acknowledgement)**. Он наиболее полезен в случае потери или повреждения предыдущих подтверждений. Получив подтверждение, канальный уровень проверяет, не освободился ли у него буфер (то есть не появилось ли свободное место в окне). Если место доступно, то заблокированному ранее сетевому уровню можно снова разрешить инициировать события `network_layer_ready`.

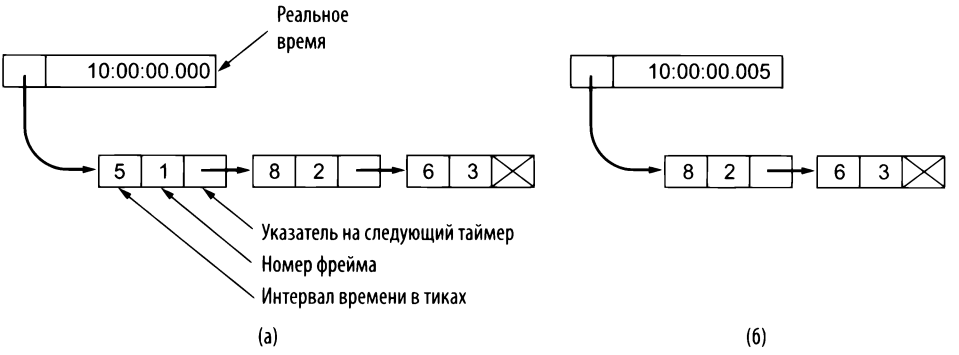
Для этого протокола предполагается, что всегда есть обратный трафик, по которому можно отправлять вложенные подтверждения. Протокол 4 не нуждается в подобном допущении, поскольку он отправляет фрейм каждый раз при получении входящего фрейма, даже если он уже был отправлен. В следующем протоколе проблема отсутствия обратного трафика будет решена гораздо более элегантным способом.

```
/* Протокол 5 (конвейерный) допускает наличие нескольких неподтвержденных фреймов. Отправитель может передать
   до MAX_SEQ фреймов, не ожидая подтверждения. Кроме того, в отличие от предыдущих протоколов, не предполагается,
   что у сетевого уровня всегда есть новые пакеты. При появлении нового пакета сетевой уровень инициирует событие
   network_layer_ready. */

#define MAX_SEQ 7

typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_
type;

#include "protocol.h"


static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Возвращает true, если a <=b < c циклично; иначе false.
 if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
       return(true);
    else
       return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Подготовить и послать информационный фрейм. */
 frame s;                                  /* временная переменная */
 s.info = buffer[frame_nr];                /* вставить пакет во фрейм */
 s.seq = frame_nr;                         /* вставить порядковый номер во фрейм */
 s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);   /* подтверждение, вкладываемое
                                                          во фрейм данных */
 to_physical_layer(&s);                    /* передать фрейм */
 start_timer(frame_nr);                    /* запустить таймер ожидания подтверждения */
}

void protocol5(void)
{
 seq_nr next_frame_to_send;                /* MAX_SEQ > 1; используется для исходящего потока */
 seq_nr ack_expected;                      /* самый старый неподтвержденный фрейм */
 seq_nr frame_expected;                    /* следующий фрейм, ожидаемый во входящем потоке */
 frame r;                                  /* временная переменная */
 packet buffer[MAX_SEQ+1];                 /* буферы для исходящего потока */
 seq_nr nbuffered;                         /* количество использующихся в данный момент выходных
                                              буферов */
 seq_nr i;                                 /* индекс массива буферов */
 event_type event;
 enable_network_layer();                   /* разрешить события network_layer_ready */
 ack_expected = 0;                         /* номер следующего ожидаемого входящего подтверждения */
 next_frame_to_send = 0;                   /* номер следующего посылаемого фрейма */
 frame_expected = 0;                       /* номер ожидаемого входящего фрейма */
 nbuffered = 0;                            /* вначале буфер пуст */
 while (true) {
    wait_for_event(&event);                /* четыре возможных события: см. event_type выше */
```

```
    switch(event) {
      case network_layer_ready:        /* у сетевого уровня есть пакет для передачи */
       /* Получить, сохранить и передать новый фрейм
       from_network_layer(&buffer[next_frame_to_send]);  /* получить новый пакет у сетевого
                                                            уровня */
       nbuffered = nbuffered + 1;      /* увеличить окно отправителя */
       send_data(next_frame_to_send, frame_expected, buffer); /* передать фрейм */
       inc(next_frame_to_send);        /* увеличить верхний край окна отправителя */
       break;
      case frame_arrival:              /* пришел фрейм с данными или с подтверждением */
       from_physical_layer(&r);        /* получить пришедший фрейм у физического уровня */
       if (r.seq == frame_expected) {
            /* Фреймы принимаются только по порядку номеров. */
            to_network_layer(&r.info); /* передать пакет сетевому уровню */
            inc(frame_expected);       /* передвинуть нижний край окна получателя */
       }
       /* Подтверждение для фрейма n подразумевает также фреймы n - 1, n - 2 и т. д. */
       while (between(ack_expected, r.ack, next_frame_to_send)) {
            /* Отправить подтверждение вместе с информационным фреймом. */
            nbuffered = nbuffered - 1; /* в буфере на один фрейм меньше */
            stop_timer(ack_expected);  /* фрейм пришел в целости; остановить таймер */
            inc(ack_expected);         /* уменьшить окно отправителя */
       }
       break;
      case cksum_err: break;           /* плохие фреймы просто игнорируются */
      case timeout:                    /* время истекло; передать повторно все неподтвержденные
                                          фреймы
       next_frame_to_send = ack_expected; /* номер первого посылаемого повторно фрейма */
       for (i = 1; i <= nbuffered; i++) {
            send_data(next_frame_to_send, frame_expected, buffer); /* переслать повторно
                                                                      1 фрейм */
            inc(next_frame_to_send);   /* приготовиться к пересылке следующего фрейма */
       }
    }
    if (nbuffered < MAX_SEQ)
       enable_network_layer();
    else
       disable_network_layer();
 }
}
```

**Илл. 3.19.** Протокол раздвижного окна с возвратом к *n*

Поскольку протокол 5 хранит несколько неподтвержденных фреймов, ему требуется несколько таймеров, по одному на фрейм. Для каждого фрейма время считается независимо от других. Однако все таймеры могут симулироваться программно, с помощью единственных аппаратных часов, периодически вызывающих прерывания. Данные таймеров могут храниться в программе в виде связанного списка. Каждый узел этого списка хранит число временных интервалов системных часов, оставшихся до истечения срока ожидания, а также номер фрейма и указатель на следующий узел списка.




**Илл. 3.20.** Программная симуляция работы нескольких таймеров. (а) Очередь из нескольких периодов ожидания. (б) Ситуация после истечения первого периода ожидания

Пример того, как можно реализовать несколько таймеров, приведен на илл. 3.20 (а). Предположим, что часы изменяют свое состояние каждую 1 мс. Пусть начальное значение реального времени будет 10:00:00.000, при этом имеются три таймера тайм-аутов, установленные на 10:00:00.005, 10:00:00.013 и 10:00:00.019. Каждый раз, когда аппаратные часы изменяют свое значение, реальное время обновляется и счетчик этих изменений в начале списка уменьшается на единицу. Когда значение счетчика становится равным нулю, инициируется тайм-аут, а узел удаляется из списка, как показано на илл. 3.20 (б). Такая организация таймеров не требует большой работы при каждом прерывании от системных часов, хотя при вызове процедур `start_timer` и `stop_timer` требуется сканирование списка. В протоколе 5 у данных процедур имеется входной параметр, означающий номер фрейма, таймер которого нужно запустить или остановить.

В этом протоколе и отправитель, и получатель работают с окнами неподтвержденных и допустимых номеров фреймов соответственно. Размер окна отправителя начинается с нуля и растет до определенного уровня. Размер окна получателя, напротив, всегда фиксированного размера. Получатель должен иметь буфер для каждого фрейма, номер которого находится в пределах окна. С каждым буфером связан бит, показывающий, занят буфер или свободен. Когда приходит фрейм, функция `between` проверяет, попал ли его порядковый номер в окно. Если да, то фрейм принимается и хранится в буфере. Это действие производится независимо от того, является ли он следующим фреймом, ожидаемым сетевым уровнем. Он должен храниться на канальном уровне до тех пор, пока все предыдущие фреймы не будут переданы сетевому уровню в правильном порядке. Пример протокола, использующего такой алгоритм, показан на илл. 3.21.

Способность протокола принимать фреймы в произвольной последовательности накладывает дополнительные ограничения на номера фреймов по сравнению с протоколами, в которых все пакеты принимались строго по порядку. Проще всего проиллюстрировать это на примере. Предположим, порядковый номер фрейма состоит из 3 бит, поэтому отправитель может посылать до семи фреймов, прежде чем перейти в режим ожидания подтверждения.

Начальное состояние окон отправителя и получателя изображено на илл. 3.22 (а). Отправитель передает фреймы с 0-го по 6-й. Окно получателя позволяет ему принимать любые фреймы с номерами от 0 по 6 включительно. Все семь фреймов успешно доставляются, поэтому получатель подтверждает их прием и передвигает окно для приема фреймов с номерами 7, 0, 1, 2, 3, 4 и 5, как показано на илл. 3.22 (б). Все семь буферов помечаются как свободные.

Именно в этот момент происходит авария: молния ударяет в телефонный столб и стирает все подтверждения. Протокол обязан отработать правильно, несмотря ни на какие чрезвычайные ситуации. Отправитель, не дождавшись подтверждений, посылает повторно фрейм 0. Когда он приходит получателю, производится проверка на предмет того, попадает ли он в его окно. На илл. 3.22 (б) фрейм 0, к сожалению, попадает в новое окно и потому принимается получателем как новый фрейм. Получатель также отправляет вложенное подтверждение для фрейма 6, поскольку были приняты все фреймы с 0-го по 6-й.

Отправитель с радостью узнает, что все переданные им фреймы успешно достигли адресата, поэтому он тут же передает фреймы 7, 0, 1, 2, 3, 4 и 5. Фрейм 7 принимается получателем, и содержащийся в нем пакет передается сетевому уровню. Сразу после этого принимающий канальный уровень проверяет наличие фрейма 0, обнаруживает его и передает старый буферизированный пакет сетевому уровню как новый. Таким образом, сетевой уровень получает неверный пакет; это означает, что протокол со своей задачей не справился.

Причина неудачи в том, что при сдвиге окна получателя новый интервал допустимых номеров фреймов перекрыл старый. Соответственно, присылаемый набор фреймов может содержать как новые фреймы (если все подтверждения были получены), так и повторно высланные старые (если подтверждения были потеряны). У принимающей стороны нет возможности отличить их.

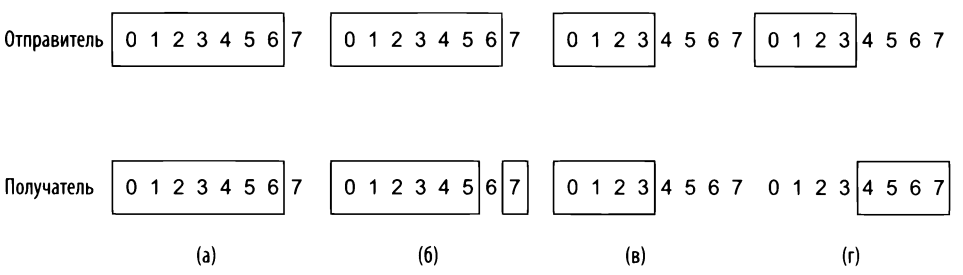
Чтобы решить эту проблему, нужно убедиться, что в сдвинутом положении окно не перекроет исходное окно. Размер окна не должен превышать половины от количества порядковых номеров, как показано на илл. 3.22 (в) и 3.22 (г). Например, если для порядковых номеров используются 3 бита, они должны изменяться в пределах от 0 до 7. В таком случае в любой момент времени только четыре фрейма могут быть неподтвержденными. Таким образом, если будут получены фреймы с 0-го по 3-й и будет передвинуто окно для приема фреймов с 4-го по 7-й, получатель сможет безошибочно отличить повторную передачу (фреймы с 0-го по 3-й) от новых фреймов (с 4-го по 7-й). Поэтому в протоколе 6 применяется окно размером $(MAX_SEQ + 1)/2$.

Возникает новый вопрос: сколько буферов должно быть у получателя? Ни при каких условиях он не должен принимать фреймы, номера которых не попадают в окно. Поэтому количество необходимых буферов равно размеру окна, а не диапазону порядковых номеров. В приведенном выше примере 3-битных номеров требуется четыре буфера с номерами от 0 до 3. Когда приходит фрейм $i$, он помещается в буфер $i \bmod 4$. Обратите внимание, что хотя $i$ и $(i + 4)$, взятые по модулю 4, «соревнуются» за один и тот же буфер, они никогда не оказываются в одном окне одновременно, потому что это привело бы к увеличению размера окна по крайней мере до 5.

```
/* Протокол 6 (выборочный повтор) принимает фреймы в любом порядке, но передает их сетевому уровню, соблюдая
порядок. С каждым неподтвержденным фреймом связан таймер. При срабатывании таймера передается повторно только
этот фрейм, а не все неподтвержденные фреймы, как в протоколе 5.
#define MAX_SEQ 7                       /* должно быть 2^n-1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready,
ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                  /* отрицательное подтверждение (nak) еще не посылалось */
seq_nr oldest_frame = MAX_SEQ+1;        /* начальное значение для симулятора */
static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* То же, что и в протоколе 5, но короче и запутаннее.
 return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}
static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected,
packet buffer[ ])
{
/* Сформировать и послать данные, а также положительное или отрицательное подтверждение */
 frame s;                               /* временная переменная */
 s.kind = fk;                           /* kind == data, ack или nak */
 if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
 s.seq = frame_nr;                      /* имеет значение только для информационных фреймов */
 s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
 if (fk == nak) no_nak = false;         /* один nak на фрейм, пожалуйста */
 to_physical_layer(&s);                 /* передать фрейм */
 if (fk == data) start_timer(frame_nr % NR_BUFS);
 stop_ack_timer();                      /* отдельный фрейм с подтверждением не нужен */
}
void protocol6(void)
{
 seq_nr ack_expected;                   /* нижний край окна отправителя */
 seq_nr next_frame_to_send;             /* верхний край окна отправителя + 1 */
 seq_nr frame_expected;                 /* нижний край окна получателя */
 seq_nr too_far;                        /* верхний край окна получателя + 1 */
 int i;                                 /* индекс массива буферов */
 frame r;                               /* временная переменная */
 packet out_buf[NR_BUFS];               /* буферы для исходящего потока */
 packet in_buf[NR_BUFS];                /* буферы для входящего потока */
 boolean arrived[NR_BUFS];              /* входящая битовая карта */
 seq_nr nbuffered;                      /* количество использующихся в данный момент выходных буферов */
 event_type event;
 enable_network_layer();                /* инициализация */
 ack_expected = 0;                      /* номер следующего ожидаемого входящего подтверждения */
 next_frame_to_send = 0;                /* номер следующего посылаемого фрейма */
 frame_expected = 0;
 too_far = NR_BUFS;
 nbuffered = 0;                         /* вначале буфер пуст */
 for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
 while (true) {
    wait_for_event(&event);             /* пять возможных событий: см. event_type выше */
    switch(event) {
      case network_layer_ready:         /* получить, сохранить и передать новый фрейм */
       nbuffered = nbuffered + 1;       /* увеличить окно отправителя */
```

```
        from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* получить новый
                                                                пакет у сетевого уровня */
        send_frame(data, next_frame_to_send, frame_expected, out_buf); /* передать
                                                                фрейм */
        inc(next_frame_to_send);    /* увеличить верхний край окна отправителя */
        break;
        case frame_arrival:         /* пришел фрейм данных или с подтверждением */
        from_physical_layer(&r);    /* получить пришедший фрейм у физического уровня */
        if (r.kind == data) {
            /* Фрейм пришел в целости. */
            if ((r.seq != frame_expected) && no_nak)
              send_frame(nak, 0, frame_expected, out_buf);
                       else start_ack_timer();
            if (between(frame_expected,r.seq,too_far) &&
                      (arrived[r.seq%NR_BUFS]==false)) {
                /* Фреймы могут приниматься в любом порядке. */
                arrived[r.seq % NR_BUFS] = true;  /* пометить буфер как занятый */
                in_buf[r.seq % NR_BUFS] = r.info; /* поместить данные в буфер */
                while (arrived[frame_expected % NR_BUFS]) {
                    /* Передать пакет сетевому уровню и сдвинуть окно
                    to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                    no_nak = true;
                    arrived[frame_expected % NR_BUFS] = false;
                    inc(frame_expected);  /* передвинуть нижний край окна получателя */
                    inc(too_far);         /* передвинуть верхний край окна получателя */
                    start_ack_timer();    /* запустить вспомогательный таймер на случай, если потре-
                                             буется пересылка подтверждения отдельным фреймом */
                }
            }
        }
        if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),
                                    next_frame_to_send))
            send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);
        while (between(ack_expected, r.ack, next_frame_to_send)) {
            nbuffered = nbuffered - 1; /* отправить подтверждение вместе с информационным фреймом */
            stop_timer(ack_expected % NR_BUFS); /* фрейм пришел в целости */
            inc(ack_expected);                  /* передвинуть нижний край окна отправителя */
        }
        break;
       case cksum_err:
        if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* поврежденный фрейм */
        break;
       case timeout:
        send_frame(data, oldest_frame, frame_expected, out_buf); /* время истекло */
        break;
       case ack_timeout:
        send_frame(ack,0,frame_expected, out_buf); /* истек период ожидания «попутки» для
                                                      подтверждения; послать подтверждение */
    }
    if (nbuffered < NR_BUFS) enable_network_layer();
                        else disable_network_layer();
}
}
```

**Илл. 3.21.** Протокол раздвижного окна с выборочным повтором




**Илл. 3.22.** Пример работы протокола. (а) Начальная ситуация при размере окна 7. (б) Семь фреймов были посланы и приняты, но не подтверждены. (в) Начальная ситуация при размере окна 4. (г) Ситуация после того, как четыре фрейма были отправлены и получены, но не подтверждены

По этой же причине количество необходимых таймеров также равно числу буферов, а не диапазону порядковых номеров; то есть с каждым буфером связывается один таймер. Когда интервал времени истекает, содержимое буфера высылается повторно.

Протокол 6 также ослабляет неявное допущение, что загрузка канала довольно высока. Мы сделали это предположение в протоколе 5, в котором подтверждения вкладывались во фреймы данных, отсылаемые в обратном направлении. Если обратный поток информации невелик, подтверждения могут задерживаться на довольно большой период времени, создавая проблемы. В исключительной ситуации, когда в одном направлении посылается много информации, а во встречном — вообще ничего, протокол останавливается, как только окно отправителя достигает максимума.

В протоколе 6 эта проблема решена. Когда приходит последовательный фрейм данных, процедура `start_ack_timer` запускает вспомогательный таймер. Если таймер сработает раньше, чем появится фрейм с данными для передачи, то будет выслано отдельное подтверждение. Прерывание от вспомогательного таймера называется событием `ack_timeout`. При такой организации возможен однонаправленный поток данных, так как отсутствие встречных фреймов данных, в которые можно было бы вкладывать подтверждения, больше не является препятствием. Требуется всего один таймер. При вызове процедуры `start_ack_timer`, если таймер уже запущен, ничего не происходит. Таймер не сбрасывается и не продлевается, так как он нужен лишь для обеспечения некоторого минимального количества подтверждений.

Важно, что период времени вспомогательного таймера должен быть существенно короче интервала ожидания подтверждения. При этом условии подтверждение доставки правильного фрейма должно приходить прежде, чем у отправителя истечет период ожидания и он повторит передачу этого фрейма.

Протокол 6 использует более эффективную стратегию обработки ошибок, чем протокол 5. При появлении у получателя подозрений в том, что произошла ошибка, он высылает отправителю фрейм отрицательного подтверждения (NAK). Он представляет собой запрос на повторную передачу фрейма. NAK используется в двух случаях: если фрейм пришел поврежденным или если

его номер отличается от ожидаемого (возможность потери фрейма). Чтобы избежать передачи нескольких запросов на повторную передачу одного и того же фрейма, получатель должен запоминать, был ли уже отправлен NAK для данного фрейма. В протоколе 6 для этой цели применяется переменная `no_nak`, принимающая значение `true`, если NAK для ожидаемого фрейма (с номером `frame_expected`) еще не был послан. Если NAK повреждается или теряется по дороге, это не имеет большого значения, так как у отправителя рано или поздно истечет период ожидания положительного подтверждения и он вышлет недостающий фрейм еще раз. Если неправильный фрейм пришел после того, как NAK был выслан и потерян, переменной `no_nak` опять присваивается `true` и запускается вспомогательный таймер. Когда время истечет, для восстановления синхронизации текущих состояний отправителя и получателя будет отправлено положительное подтверждение (ACK).

Иногда время, необходимое для прохождения фрейма по каналу, его обработки и отправки подтверждения, остается практически неизменным. В этом случае отправитель может установить время ожидания чуть больше ожидаемого интервала между отправкой фрейма и получением подтверждения. NAK при этом использовать бесполезно.

Однако в других ситуациях время прохождения сигнала в обе стороны может сильно варьироваться. Если встречный поток данных нерегулярен, то время прихода подтверждений также будет непостоянным, уменьшаясь при наличии встречного потока и увеличиваясь при его отсутствии. Перед отправителем возникает непростой выбор значения времени ожидания. Если выбрать слишком короткий интервал, то увеличится риск ненужных повторных передач. При выборе чересчур большого значения протокол будет тратить много времени на ожидания после ошибки. В обоих случаях пропускная способность тратится впустую. В целом если среднеквадратичное отклонение интервала ожидания подтверждения намного больше самого интервала, то таймер может быть установлен довольно «свободно». При этом NAK могут существенно ускорить повторную передачу потерянных или поврежденных фреймов.

С вопросом тайм-аутов и отрицательных подтверждений тесно связана проблема определения фрейма, вызвавшего тайм-аут. В протоколе 5 это всегда фрейм с номером `ack_expected`, поскольку он является старшим. В протоколе 6 нет столь простого способа определить фрейм с истекшим интервалом ожидания. Предположим, были переданы фреймы с 0-го по 4-й, то есть список неподтвержденных фреймов выглядит так: 01234 (от первого к последнему). Теперь допустим, что у фрейма 0 истекает интервал ожидания и он передается повторно, затем посылается фрейм 5 (новый), далее интервал ожидания истекает у фреймов 1 и 2 и посылается фрейм 6 (также новый). В результате список неподтвержденных фреймов принимает вид 3405126, начиная с самого старого и заканчивая самым новым. Если весь встречный поток данных потеряется, интервалы ожидания этих семи фреймов истекут именно в таком порядке.

Чтобы не усложнять и без того непростой пример протокола, мы не углубляемся в детали управления таймером. Вместо этого мы просто предполагаем, что при наступлении тайм-аута переменной `oldest_frame` присваивается номер фрейма, интервал времени которого истек.

# 3.5. ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ ПРОТОКОЛОВ КАНАЛЬНОГО УРОВНЯ

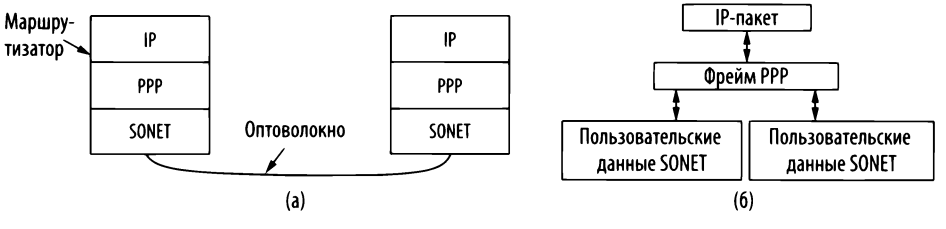
Для связи компьютеров в пределах одного здания широко применяются локальные сети, однако большинство глобальных сетей построено на линиях «точка-точка». С локальными сетями мы познакомимся в главе 4. Здесь рассмотрим три наиболее распространенных случая использования протоколов канального уровня в двухточечных каналах интернета. Первый случай — передача пакетов по оптоволоконным каналам SONET. Такие каналы широко применяются, например, для соединения маршрутизаторов, установленных в разных концах сети провайдера. Вторым примером является использование каналов ADSL в пределах локального абонентского шлейфа телефонной сети. Наконец, мы обсудим применение каналов DOCSIS в пределах локального шлейфа кабельной сети — с их помощью к интернету подключаются миллионы отдельных пользователей и компаний.

Чтобы установить такие типы соединений, требуются не только двухточечные каналы, но также телефонные и кабельные модемы, арендованные линии и т. д. Для пересылки пакетов используется стандартный протокол **двухточечного соединения PPP (Point-to-Point Protocol)**. PPP описан в стандарте RFC 1661 и доработан в более поздних документах: RFC 1662 и др. (Симпсон; Simpson, 1994a, 1994b). В каждой из трех разновидностей каналов — SONET, ADSL и DOCSIS — он применяется по-разному.

## 3.5.1. Передача пакетов по каналам SONET

SONET, с которым мы познакомились в разделе 2.5.3, — это протокол физического уровня, наиболее часто используемый в оптоволоконных каналах, которые составляют магистраль различных коммуникационных сетей, включая телефонную. SONET обеспечивает строго определенную скорость передачи данных (например, 2,4 Гбит/с в канале OC-48). Поток битов организован в виде пакетов фиксированного размера, которые посылаются каждые 125 мкс, независимо от наличия в них пользовательских данных.

Чтобы передавать пакеты по таким каналам, необходим некоторый механизм формирования фреймов, способный отличать возникающие иногда пакеты от непрерывного потока битов, в котором они передаются. Для этого на IP-маршрутизаторах работает протокол PPP, как показано на илл. 3.23.




**Илл. 3.23.** Передача пакета по каналам SONET. (а) Стек протоколов. (б) Взаимоотношения между фреймами

PPP — это улучшенный вариант более простого **интернет-протокола для последовательной линии SLIP (Serial Line Internet Protocol)**. Он обнаруживает ошибки, поддерживает несколько протоколов, разрешает аутентификацию и выполняет ряд других задач. Благодаря широкому набору параметров PPP может использоваться в качестве трех основных методов.

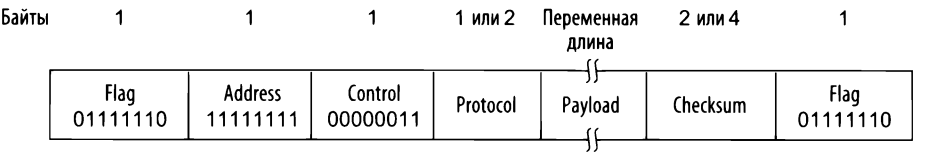
1. Метод формирования фреймов, однозначно обозначающий конец одного фрейма и начало следующего. Формат фреймов также обеспечивает обнаружение ошибок.
2. Протокол управления каналом **LCP (Link Control Protocol)**, позволяющий устанавливать соединения, тестировать их, договариваться о параметрах использования и снова отключать их, когда они не нужны.
3. Способ договориться о параметрах сетевого уровня независимо от используемого в нем протокола. Данный метод состоит в том, что для каждого поддерживаемого сетевого уровня имеется свой **сетевой протокол управления NCP (Network Control Protocol)**.

Чтобы не изобретать велосипед, для PPP был выбран формат фрейма, близкий к формату **высокоуровневого протокола управления каналом HDLC (High-level Data Link Control)**, некогда популярного представителя раннего семейства протоколов.

В отличие от бит-ориентированного протокола HDLC, PPP является байт-ориентированным. В частности, в PPP применяется байт-стаффинг, поэтому все фреймы состоят из целого числа байтов. HDLC использует бит-стаффинг; это позволяет отправить фрейм размером, к примеру, 30,25 байта.

Существует второе важное отличие. HDLC предоставляет надежную передачу за счет метода «раздвижного окна», подтверждений и тайм-аутов, как мы видели выше. PPP также может обеспечить надежность в зашумленной среде, например в беспроводных сетях; детали определены в стандарте RFC 1663. Однако на практике это применяется редко. Вместо этого для предоставления службы без установки соединения и без подтверждений в интернете применяется ненумерованный режим.

Формат фрейма PPP показан на илл. 3.24. Все PPP-фреймы начинаются со стандартного флагового байта протокола HDLC 0x7E (01111110). Если этот байт встречается в поле `Payload`, он предваряется управляющим байтом 0x7D, а следующий за ним представляет собой экранированный байт, сложенный по модулю 2 со значением 0x20 (при этом переключается пятый бит). Например, 0x7D 0x5E — это управляющая последовательность для флагового байта 0x7E. Это означает, что начало и конец фрейма можно найти, просто просканировав содержимое на наличие байта 0x7E. Больше он нигде встречаться не будет. Правило удаления заполняющих битов при получении — найти значение 0x7D, удалить его, а следующий байт сложить по модулю 2 со значением 0x20. Кроме того, между фреймами необходим только один флаговый байт. Несколько флаговых байтов могут применяться для заполнения канала при отсутствии фреймов данных для отправки получателю.




**Илл. 3.24.** Полный формат фрейма PPP для работы в ненумерованном режиме

После стартового фрейма идет поле `Address` (Адрес), которому всегда присваивается двоичное значение 11111111; это означает, что все станции должны принимать этот фрейм. Использование такого значения позволяет избежать необходимости назначать адреса передачи данных.

За полем адреса следует поле `Control` (Управляющее поле), его значение по умолчанию равно 00000011. Это число означает ненумерованный фрейм.

Так как по умолчанию поля `Address` и `Control` являются константами, протокол LCP позволяет двум сторонам договориться о возможности пропустить оба поля и сэкономить таким образом по 2 байта на фрейм.

Четвертое поле фрейма PPP — `Protocol` (Протокол). Оно определяет тип пакета, который содержится в поле `Payload`. Номера, начинающиеся с бита 0, отведены для IP версий 4 и 6, а также других протоколов сетевого уровня, таких как IPX и AppleTalk. С бита 1 начинаются коды, используемые в конфигурационных протоколах PPP, включая LCP и различные NCP для каждого поддерживаемого протокола сетевого уровня. Размер поля `Protocol` по умолчанию составляет 2 байта, однако путем переговоров с помощью LCP он может быть уменьшен до одного байта. Вероятно, разработчики перестраховались на случай, если когда-либо будет использоваться более 256 протоколов.

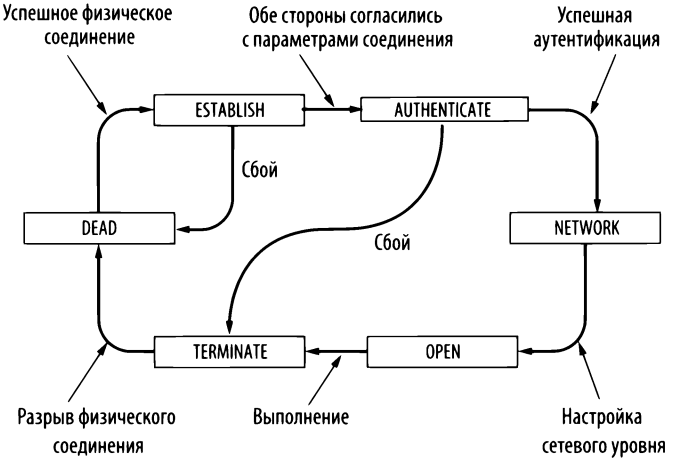
Поле `Payload` может быть переменной длины, вплоть до некоторого оговоренного максимального значения. Если размер не определен во время установки соединения при помощи LCP, то по умолчанию используется длина 1500 байт. При необходимости данные пользователя могут дополняться специальными символами.

Следом за `Payload` располагается поле `Checksum` (Контрольная сумма). В обычном состоянии оно занимает 2 байта, но в случае необходимости по договоренности может занимать четыре. Четырехбайтная контрольная сумма фактически представляет собой 32-битный код CRC, образующий многочлен которого представлен в конце раздела 3.2.2. Двухбайтная контрольная сумма также является стандартным кодом CRC.

Итак, PPP — это механизм формирования фреймов, работающий со многими протоколами в разных физических средах. Варианты использования PPP в сетях SONET описаны в стандарте RFC 2615 (см. Малис и Симпсон; Malis and Simpson, 1999). Применяется четырехбайтная контрольная сумма, так как она считается основным способом распознавания ошибок передачи на физическом, канальном и сетевом уровнях. Рекомендуется не сжимать поля `Address`, `Control` и `Protocol`, так как каналы SONET и так работают на относительно высокой скорости.

Есть и еще одна интересная особенность. Перед тем как попасть в поток данных SONET, полезная информация протокола PPP скремблируется (как описано в разделе 2.4.3). При скремблировании данные складываются по модулю 2 с длинной псевдослучайной последовательностью. Только после этого они пересылаются получателю. Проблема в том, что потоку данных SONET для успешной синхронизации требуется частая смена значений битов. В колебаниях голосового сигнала это происходит естественным образом, но при пересылке данных пользователь сам выбирает, какую информацию передать. Он может, к примеру, отправить длинную последовательность нулей и вызвать тем самым проблемы. Благодаря скремблированию вероятность такого развития событий ничтожно мала.

Перед тем как передавать фреймы PPP по каналу SONET, необходимо установить и настроить соединение PPP. Этапы, через которые проходит линия связи при ее установлении, использовании и разъединении, показаны на илл. 3.25.




**Илл. 3.25.** Диаграмма состояний установки и разрыва соединения PPP

Изначально линия в состоянии *DEAD* (отключена), то есть соединения на физическом уровне не существует. После создания физического соединения линия переходит в состояние *ESTABLISH* (установление соединения). В этот момент начинаются переговоры о параметрах с помощью протокола LCP. Узлы PPP обмениваются пакетами LCP (они содержатся в поле `Payload` фрейма PPP). Это необходимо для выбора из перечисленных выше параметров PPP. Инициирующий узел предлагает варианты, а отвечающие узлы либо соглашаются с ними, либо отвергают частично или полностью. Они также могут делать свои предложения.

При успешном результате переговоров линия переходит в фазу *AUTHENTICATE* (аутентификация). Теперь обе стороны по желанию могут проверить, кем является собеседник. После успешной аутентификации в фазе *NETWORK* (сеть) происходит обмен пакетами NCP для настройки сетевого уровня. NCP сложно

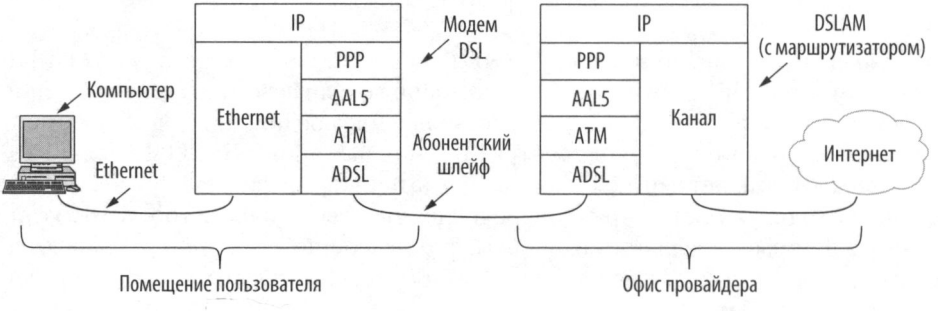
описать общими словами, так как каждый из них отличается в зависимости от конкретного протокола сетевого уровня и поддерживает конфигурационные запросы, характерные только для него. Например, для IP наиболее важной задачей является назначение IP-адресов собеседникам на обоих концах линии.

Когда линия переходит в фазу *OPEN* (открыть), можно начинать передачу данных. Именно в этой фазе IP-пакеты пересылаются в PPP-фреймах по линии SONET. Когда передача данных закончена, линия переходит к фазе *TERMINATE* (завершить), а затем снова в состояние *DEAD* (выключено), когда физическое соединение разрывается.

## 3.5.2. ADSL

**ADSL (Asymmetric Digital Subscriber Line — асимметричная цифровая абонентская линия)** соединяет миллионы домашних пользователей с интернетом на скоростях, равных нескольким мегабитам в секунду. Для этого используются те же телефонные провода, по которым предоставляются услуги традиционной телефонии. В разделе 2.5.2 описано, как на стороне пользователя устанавливается устройство под названием DSL-модем. Он отправляет биты по абонентскому шлейфу, адресуя их DSLAM (DSL Access Multiplexer — мультиплексор доступа DSL), установленному в местном офисе телефонной компании. Теперь мы более подробно рассмотрим процесс передачи пакетов по каналам ADSL.

Общая схема работы протоколов и устройств показана на илл. 3.26. В разных сетях применяются различные протоколы, поэтому для демонстрации мы выбрали наиболее популярный сценарий. Домашний компьютер пользователя посылает IP-пакеты DSL-модему, используя канальный уровень, например сеть Ethernet. Затем DSL-модем отправляет IP-пакеты по абонентскому шлейфу на устройство DSLAM с помощью протоколов, которые мы рассмотрим далее. На стороне DSLAM или подключенного к нему маршрутизатора (в зависимости от реализации) IP-пакеты извлекаются, поступают в сеть провайдера и достигают точки назначения в интернете.




**Илл. 3.26.** Стек протоколов ADSL

Показанные на илл. 3.26 протоколы, работающие в канале ADSL, начинаются с низшего, физического уровня. Они основаны на мультиплексировании

с ортогональным делением частот (также известном как цифровая многоканальная тональная модуляция), с которым мы познакомились в разделе 2.4.4. Ближе к вершине стека, под сетевым уровнем IP, находится PPP. Это тот же самый протокол PPP, который мы изучили при обсуждении передачи пакетов по сетям SONET. Он точно так же устанавливает и настраивает связь для передачи IP-пакетов.

Между ADSL и PPP находятся ATM и AAL5. Это новые протоколы, с которыми мы ранее не встречались. Протокол **асинхронной передачи данных ATM (Asynchronous Transfer Mode)** был разработан в начале 1990-х годов и широко рекламировался при первом запуске. Была обещана сетевая технология, которая решит все мировые телекоммуникационные проблемы, объединив голос, текстовые данные, кабельное телевидение, телеграф, почтовых голубей, связанные нитью консервные банки и все остальные способы передачи информации в единую интегрированную систему, способную удовлетворить любые требования пользователей. К сожалению, ожидания не оправдались. В целом ATM столкнулся с теми же проблемами, о которых мы упомянули в разговоре о протоколах OSI: неудачное время, плохая архитектура и реализация, а также политические нюансы. Тем не менее ATM все же добился большего успеха, чем OSI. И хотя он не покорил весь мир, его активно применяют в некоторых линиях широкополосного доступа, таких как DSL, и особенно в WAN-каналах телефонных сетей.

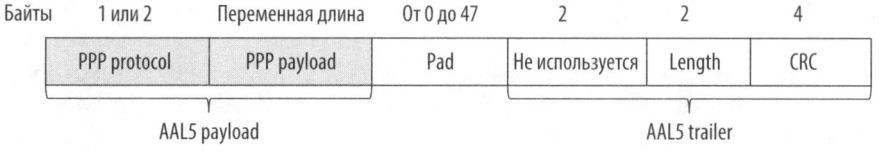
ATM представляет собой канальный уровень, основанный на пересылке **ячеек (cells)** информации фиксированной длины. «Асинхронная передача» означает, что нет необходимости отправлять ячейки постоянно, как это происходит с битами в синхронных линиях типа SONET. Ячейки пересылаются только тогда, когда имеется готовая к передаче информация. ATM — это технология, ориентированная на установление соединения. В заголовок каждой ячейки встраивается идентификатор **виртуального контура (virtual circuit)**, и устройства используют этот идентификатор для пересылки ячеек по маршрутам внутри установленных соединений.

Длина каждой ячейки составляет 53 байта: 48 байт пользовательских данных плюс 5 байт заголовка. Используя ячейки небольшого размера, ATM гибко разделяет полосу пропускания физического канала между разными пользователями. Это полезно, например, при одновременной передаче голоса и данных по одному каналу. При пересылке фрагментов голосовой информации не возникнет длинных задержек благодаря отсутствию больших пакетов данных. Нестандартный выбор длины ячейки (сравните с более распространенным значением — степенью двойки) отражает важность политической составляющей разработки протокола. 48 байт под полезную информацию — это компромисс между 32-байтными ячейками, которые хотела использовать Европа, и 64-байтными, за которые голосовали США. Краткое описание протокола представили Сиу и Джейн (Siu and Jain, 1995).

Для пересылки данных по сети ATM необходимо преобразовать их в последовательность ячеек. Преобразование выполняется на уровне адаптации протокола ATM путем сегментации и обратной сборки. Для разных служб, пересылающих различную информацию (от периодических голосовых сэмплов до

пакетных данных), были определены несколько уровней адаптации. Основной, используемый для пакетных данных — это **уровень адаптации ATM 5 (ATM Adaptation Layer 5, AAL5)**.

Фрейм AAL5 показан на илл. 3.27. Роль заголовка в нем исполняет трейлер (`AAL5 trailer`), содержащий сведения о длине (`Length`), а также 4-байтный код CRC для обнаружения ошибок. Разумеется, это тот же самый CRC, используемый протоколом PPP и сетями стандарта IEEE 802, такими как Ethernet. Ван и Кроукрофт (Wang and Crowcroft, 1992) продемонстрировали, что это достаточно сильная конфигурация для обнаружения нетрадиционных ошибок, например сбоя в порядке следования ячеек. Помимо пользовательских данных (`AAL5 payload`), во фрейме AAL5 есть биты заполнения (`Pad`). Они дополняют общую длину, чтобы она была кратной 48 байтам. Таким образом, фрейм можно поделить на целое число ячеек. Хранить адреса внутри фрейма не нужно, так как идентификатор виртуального контура, имеющийся в каждой ячейке, не даст ей заблудиться и приведет к нужному получателю.




**Илл. 3.27.** Фрейм AAL5, содержащий данные PPP

Итак, мы познакомились с протоколом ATM. Осталось только обсудить, как его задействует протокол PPP при ADSL-подключении. Это делается с помощью еще одного стандарта, который так и называется — **PPP с использованием ATM (PPP over ATM, PPPoA)**. В действительности данный стандарт нельзя назвать протоколом (поэтому на илл. 3.26 его нет). Скорее это спецификация, описывающая, как одновременно применять протокол PPP и фреймы AAL5. Она описана в стандарте RFC 2364 (см. Гросс и др.; Gross et al., 1998).

Пользовательские данные AAL5 содержат только два поля PPP: `Protocol` и `Payload`, как показано на илл. 3.27. Поле `Protocol` сообщает устройству DSLAM на другом конце линии, чем являются эти пользовательские данные: пакетом IP, LCP или другого протокола. Принимающая сторона знает, что ячейки содержат информацию PPP, так как виртуальный контур ATM настраивается соответствующим образом.

Для фрейма AAL5 механизмы формирования фрейма PPP не требуются, всю работу выполняют ATM и AAL5. Дополнительно создавать фреймы было бы попросту бессмысленно. Код CRC протокола PPP также не нужен, поскольку AAL5 включает такой же CRC. Механизм выявления ошибок дополняет кодирование физического уровня, применяемое в каналах ADSL (код Рида — Соломона для исправления ошибок и 1-байтный CRC для распознавания оставшихся отклонений, не выявленных другими способами). Это куда более сложный механизм устранения ошибок, чем тот, что применяется при пересылке данных в сетях SONET. Причина проста — линии ADSL гораздо зашумленнее.

### 3.5.3. DOCSIS

Согласно общепринятому определению, **стандарт передачи данных по сетям кабельного телевидения по коаксиальному кабелю (Data Over Cable Service Interface Specification, DOCSIS)** подразумевает наличие двух компонентов — физического уровня (PHY) в том виде, как он был описан в предыдущей главе[1], и уровня управления доступом к среде (Media Access Control, MAC), представленный в главе 4. DOCSIS находится выше физического уровня и берет на себя решение таких задач сетевого уровня, как распределение пропускной способности для восходящего и нисходящего потока (управление потоком), формирование фреймов и исправление ошибок (конечно, иногда это происходит на физическом уровне). Все эти концепции уже были рассмотрены ранее в данной главе. Далее мы узнаем, как эти задачи решаются в протоколе DOCSIS.

Фрейм DOCSIS содержит разнообразную информацию, включая показатели качества обслуживания и служебные данные о фрагментации или конкатенации фреймов. Однонаправленная последовательность фреймов называется **служебным потоком (service flow)**. Основные служебные потоки передают управляющие сообщения от головной станции кабельных модемов (Cable Modem Termination System, CMTS), расположенной в офисе оператора кабельной сети, к каждому модему. Служебный поток снабжается уникальным идентификатором и часто ассоциируется с одним из следующих классов обслуживания: «наилучший из возможных», «опрос» (при котором кабельный модем явным образом запрашивает полосу пропускания) и «предоставление сервиса» (модем передает пакеты данных с гарантированным сроком их получения). Основным является служебный поток по умолчанию, в котором передаются все фреймы, которые не были отнесены к какому-либо другому сервису. Во многих конфигурациях широкополосного доступа используются только исходящий и входящий служебные потоки по умолчанию — между кабельным модемом и головной станцией, в которых передается весь пользовательский трафик и все управляющие сообщения. Сети DOCSIS были изначально разработаны с расчетом на то, что данные будут в основном передаваться во входящем направлении. Характер трафика некоторых современных приложений, например приложений для видеоконференций, уже не укладывается в эту схему; в то же время новейшие облачные игровые сервисы (такие, как Stadia, GeForce Now, xCloud) могут использовать входящий трафик в еще больших масштабах, обеспечивая непрерывную потоковую передачу данных со скоростью до 30–35 Мбит/с.

После включения кабельного модема он устанавливает соединение с головной станцией, которая, как правило, позволяет ему подключиться к остальной части сети. Зарегистрировавшись на головной станции, он получает от нее исходящие и входящие каналы связи, а также ключи шифрования. Несущие частоты исходящей и входящей передачи предоставляют два общих канала для всех кабельных модемов. В случае входящей передачи все модемы, подключенные к станции,

[1] Его также иногда называют подуровнем физической среды (Physical Media Dependent, PMD).

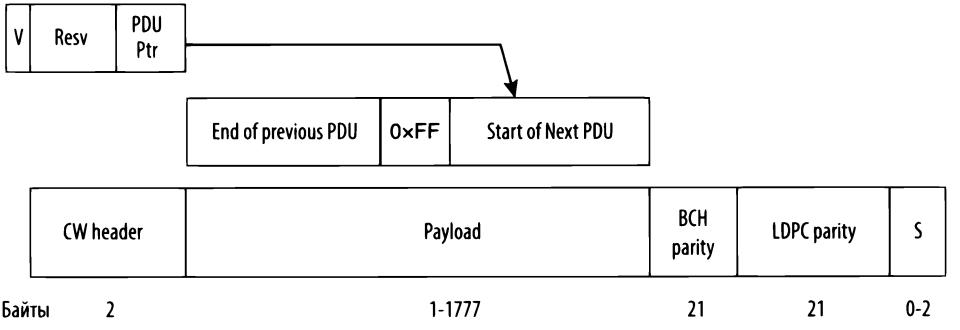
получают все передаваемые пакеты. При исходящей передаче данные передаются множеством модемов, и станция является единственным получателем данных. Между головной станцией и каждым кабельным модемом может быть несколько физических путей.

До версии DOCSIS 3.1 пакеты входящего потока разделялись на фреймы MPEG длиной 188 байт; при этом каждый фрейм содержал 4-байтный заголовок и пользовательские данные в 184 байта (так называемый уровень конвергенции MPEG). Помимо самих данных, головная станция периодически отправляет модему управляющую информацию о ранжировании, выделении каналов и других задачах, связанных с их распределением. Эти задачи выполняются уровнем управления доступом к среде (MAC). В целях совместимости версия DOCSIS 3.1 по-прежнему поддерживает данный уровень конвергенции, но он больше не используется для входящей передачи данных.

Канальный уровень DOCSIS организует передачу в соответствии с **профилями модуляции (modulation profiles)**. Профиль модуляции представляет собой список заказов модуляции (то есть битовых нагрузок), сопоставленных с поднесущими OFDM. При нисходящей передаче головная станция может использовать отдельный профиль для каждого кабельного модема, но обычно он используется для целой группы модемов с одинаковыми или близкими параметрами производительности. Принимая в расчет идентификационные данные служебного потока и параметры QoS, канальный уровень (в DOCSIS 3.1), теперь называемый **уровнем конвергенции (convergence layer)**, группирует пакеты с одинаковым профилем в один буфер пересылки. Обычно для каждого профиля отводится отдельный буфер небольшого размера во избежание значительной задержки. Затем алгоритм формирования кодовых слов преобразует фреймы DOCSIS в соответствующие кодовые слова с упреждающим исправлением ошибок (FEC). При этом он извлекает пакеты из буферов разных профилей, сменяя буфер на границе каждого кодового слова. В кодировке FEC фрейм DOCSIS выглядит как поток битов, а не как последовательность байтов. DOCSIS использует кодовые слова LDPC. В случае входящей передачи полное кодовое слово может содержать до 2027 байт, где 1799 байт (или меньше) служат для передачи данных, а 225 байт — для проверки четности. В каждом байте фрейма DOCSIS первым передается самый младший бит. Когда значение занимает несколько байтов, старшие байты передаются перед младшими — такой способ упорядочения называется **сетевым порядком (network order)**. На головной станции также применяется байт-стаффинг: при отсутствии фреймов DOCSIS в нисходящем потоке станция вставляет заполненные нулями поднесущие в символы OFDM или просто вставляет последовательности единиц в кодовые слова, как показано на илл. 3.28.

Начиная с версии 3.0, DOCSIS поддерживает технологию **связывания каналов (channel bonding)**, которая позволяет одному абоненту использовать сразу несколько исходящих и входящих каналов. Данный подход представляет собой разновидность **агрегации каналов (link aggregation)** — объединения нескольких физических соединений (или портов) в одно логическое. В версии DOCSIS 3.0 можно связать до 32 входящих и до 8 исходящих каналов шириной 6–8 МГц. Версия DOCSIS 3.1 предоставляет такие же возможности связывания, но

каналы могут быть гораздо шире — до 192 МГц при входящей и до 96 МГц при исходящей передаче; в DOCSIS 3.0 эти цифры составляют 6–8 МГц и до 6,4 МГц соответственно. Кроме того, модем с поддержкой DOCSIS 3.1 может связывать каналы разных типов (например, один канал OFDM шириной 192 МГц и четыре канала SC-QAM шириной 6 МГц).




**Илл. 3.28.** Преобразование фрейма DOCSIS в кодовые слова

V — значение; Resv — зарезервировано; PDU Ptr — указатель протокольного блока данных; End of previous PDU — конец предыдущего протокольного блока данных; Start of Next PDU — начало следующего протокольного блока данных; CW header — заголовок кодового слова; Payload — пользовательские данные; BCH parity — код BCH проверки на четность; LDPC parity — код LDPC проверки на четность

## 3.6. РЕЗЮМЕ

Задача канального уровня — преобразование необработанного потока битов, поступающего с физического уровня, в поток фреймов, которые могут использоваться сетевым уровнем. Канальный уровень может представлять этот поток с различной степенью надежности, начиная от служб без установки соединения и без подтверждения и заканчивая стабильными службами, ориентированными на установление соединения.

Для формирования фреймов используются разнообразные методы, включая подсчет байтов, байт-стаффинг и бит-стаффинг. Протоколы канального уровня могут обеспечить контроль ошибок для обнаружения и исправления поврежденных и повторной передачи потерянных фреймов. Во избежание перегрузки медленного приемника быстрым отправителем применяется управление потоком. Популярный механизм «раздвижных окон» — простой способ объединения контроля ошибок и управления потоком. Для окна размером в один пакет применяется протокол с остановкой и ожиданием.

Коды с обнаружением и исправлением ошибок добавляют к сообщениям избыточную информацию с помощью ряда математических методов. Для исправления ошибок широко применяются сверточные коды и коды Рида — Соломона. Все большую популярность завоевывают коды с малой плотностью проверок на четность. Применяемые на практике коды с обнаружением ошибок

включают циклический контроль избыточности и контрольные суммы. Все эти коды можно применять на канальном, а также на физическом и более высоких уровнях.

Мы рассмотрели ряд протоколов, обеспечивающих надежную работу канального уровня за счет подтверждений и повторной передачи или, если взять более реалистичный пример, за счет запросов ARQ (Automatic Repeat reQuest). Начав с обсуждения идеальной среды передачи, где отсутствуют ошибки, и идеального приемника, который может обработать входящий поток любого размера, мы познакомились с управлением потоком, затем с контролем ошибок с помощью порядковых номеров и, наконец, с алгоритмом с остановкой и ожиданием. Затем мы перешли к алгоритму «раздвижного окна», позволяющему обмениваться данными в двух направлениях, и узнали о концепции вложенного подтверждения. Последние два протокола используют конвейерную передачу множества фреймов, чтобы отправитель не блокировался, увеличивая задержку передачи. Получатель может либо отбрасывать все фреймы, за исключением очередного в последовательности, либо помещать неупорядоченные фреймы в буфер и отправлять отрицательные подтверждения для более эффективного использования полосы пропускания. Первая стратегия называется протоколом с возвратом к $n$, а вторая — протоколом с выборочным повтором.

В интернете в качестве основного протокола линий «точка-точка» используется PPP. Он предоставляет службу без установки соединения и без подтверждения. Для разделения фреймов применяются флаговые байты, а для распознавания ошибок — коды CRC. С помощью этого протокола пакеты передаются по множеству типов соединений, включая каналы SONET в глобальных сетях и ADSL для домашних подключений. Для предоставления доступа в интернет по имеющейся сети кабельного телевидения используется протокол DOCSIS.

## ВОПРОСЫ И ЗАДАЧИ

1. В сети Ethernet для разделения фреймов используется преамбула в сочетании со счетчиком байтов. Что произойдет, если пользователь попытается отправить данные, содержащие такую преамбулу?
2. В потоке данных, для которого применяется алгоритм байт-стаффинга, встречается следующий фрагмент данных: A B ESC C ESC FLAG FLAG D. Каким будет выходной поток после выполнения алгоритма?
3. Каковы максимальные накладные расходы для алгоритма байт-стаффинга?
4. Вы принимаете следующий фрагмент данных: 0110 0111 1100 1111 0111 1101. При этом известно, что протокол использует бит-стаффинг. Как будут выглядеть эти данные после удаления вставленных битов?
5. Может ли потеря, вставка или модификация одного бита при использовании бит-стаффинга вызвать ошибку, которую нельзя будет выявить с помощью контрольной суммы? Если нет, то почему? Если да, то каким образом? Играет ли при этом какую-либо роль длина контрольной суммы?

6. Сообщение верхнего уровня разбито на 10 фреймов, у каждого из которых шанс дойти до назначения без повреждений составляет 80 %. Если канальный уровень не обеспечивает контроль ошибок, сколько раз в среднем потребуется пересылать все сообщение?
7. При каких обстоятельствах протокол без обратной связи (например, с кодом Хэмминга) предпочтительнее протоколов с обратной связью, обсуждаемых в этой главе?
8. Для обеспечения большей надежности, чем та, которую предоставляет единственный бит четности, в некотором коде с обнаружением ошибок один бит четности суммирует все четные биты, а другой — все нечетные. Каким будет у этого кода расстояние Хэмминга?
9. Заметив, что используемая вами служба обмена мгновенными сообщениями не обеспечивает обнаружения ошибок, вы решили сами реализовать простейший механизм их выявления путем двукратной отправки каждого сообщения. Какими при этом будут расстояние Хэмминга и кодовая норма? Насколько эффективен этот способ в сравнении с использованием бита четности?
10. Допустим, вы обеспечиваете обнаружение ошибок путем двукратной отправки каждого сообщения. Если возникнут две однобитные ошибки, какова вероятность того, что они останутся незамеченными? Какой будет эта вероятность при использовании бита четности? Какой метод позволяет выявлять больше ошибок?
11. 8-битный байт с двоичным значением 10101111 необходимо закодировать, используя код Хэмминга с *четным* количеством единиц. Как будет выглядеть двоичное значение после кодирования?
12. 8-битный байт с двоичным значением 10011010 необходимо закодировать, используя код Хэмминга с *нечетным* количеством единиц. Как будет выглядеть двоичное значение после кодирования?
13. Приемник получает 12-битный код Хэмминга с контролем четности, шестнадцатеричное значение которого равно 0xB4D. Как (в шестнадцатеричном виде) выглядела исходная последовательность? Предполагается, что ошибочным может быть только 1 бит.
14. У кодов Хэмминга расстояние равно 3, что позволяет с их помощью исправлять одиночные ошибки или выявлять двойные. Можно ли применять их для одновременного выполнения обеих задач? Обоснуйте свою точку зрения. Сколько ошибок может быть исправлено, если расстояние Хэмминга равно $n$? А сколько обнаружено?
15. Имеется протокол, в котором на каждые 16 байт данных сообщения добавляется 1 байт избыточных данных. Может ли этот протокол использовать код Хэмминга для исправления одиночных ошибок?
16. Один из способов обнаружения ошибок заключается в передаче данных в виде блока из $n$ рядов по $k$ бит с добавлением битов четности к каждому ряду и каждой строке. Бит в нижнем правом углу — это бит четности, проверяющий свою строку и столбец. Будет ли такая схема выявлять все одиночные

ошибки? Двойные? Тройные? Докажите на примере, что эта схема не в состоянии обнаруживать некоторые четырехбитные ошибки.

17. Сколько ошибок можно обнаружить и исправить в предыдущей задаче?
18. Составьте формулу для вычисления минимального количества избыточных битов $r$, добавляемых в сообщение $m$ с целью исправления всех одиночных и *двойных* ошибок.
19. Принимая во внимание ответ на предыдущий вопрос, объясните популярность сложных вероятностных механизмов исправления ошибок, таких как рассмотренные в данной главе сверточные коды и коды LDPC.
20. Предположим, что данные передаются в блоках размером 1000 бит. Каков максимальный коэффициент ошибок, при котором механизм с обнаружением ошибок и повторной передачей (1 бит четности на блок) покажет себя лучше, чем код Хэмминга? Предполагается, что ошибки в битах не зависят друг от друга, а во время повторной передачи они не возникают.
21. В блоке битов из $n$ строк и $k$ столбцов используются горизонтальные и вертикальные биты четности для обнаружения ошибок. Какова вероятность того, что инверсия 4 бит не будет обнаружена?
22. Предположим, что сообщение 1001 1100 1010 0011 передается с использованием контрольной суммы для интернета (4-битное слово). Какова будет контрольная сумма?
23. Чему равен остаток от деления $x^7 + x^5 + 1$ на образующий многочлен $x^3 + 1$?
24. Поток битов 10011101 передается с использованием стандартного метода циклического избыточного кода (CRC), описанного в данной главе. Образующий многочлен равен $x^3 + 1$. Какая битовая последовательность будет реально передаваться? Предполагается, что третий бит слева при передаче инвертировался. Докажите, что эта ошибка будет обнаружена приемником. Приведите пример ошибок в битах передаваемой строки, которые приемник обнаружить не сможет.
25. Поток битов 11100110 передается с использованием стандартного метода циклического избыточного кода (CRC), описанного в этой главе. Образующий многочлен равен $x^4 + x^3 + 1$. Какая битовая последовательность будет реально передаваться? Предполагается, что третий бит слева при передаче инвертировался. Докажите, что эта ошибка будет обнаружена приемником. Приведите пример ошибок в битах передаваемой строки, которые приемник обнаружить не сможет.
26. Протоколы канального уровня всегда размещают код CRC в трейлере, а не в заголовке. Почему?
27. При обсуждении протокола ARQ приводился пример сценария, в котором получатель принимает две копии одного и того же фрейма из-за потери подтверждения. Возможно ли, что получатель примет несколько копий одного фрейма, если ни один из фреймов (данных или подтверждения) утерян не будет?

28. Скорость передачи данных в канале составляет 4 Кбит/с, а время распространения сигнала — 20 мс. При каком размере фреймов эффективность протокола с остановкой и ожиданием составит по меньшей мере 50 %?
29. Протоколы *A* и *B* различаются только размером посылающего окна. Протокол *A* использует окно передачи размером 20 фреймов. *B* представляет собой протокол с остановкой и ожиданием. Эти протоколы используются в двух одинаковых каналах. Если *A* обеспечивает почти 100%-ю эффективность использования пропускной способности, то каков этот показатель у *B*?
30. Протокол с остановкой и ожиданием обеспечивает 25%-ю эффективность использования пропускной способности, передавая 900-битные фреймы по каналу с задержкой односторонней передачи 50 мс. Чему равна пропускная способность этого канала в битах в секунду?
31. Протокол с остановкой и ожиданием обеспечивает 60%-ю эффективность использования пропускной способности, передавая 300-битные фреймы по каналу, пропускная способность которого равна 50 Кбит/с. Чему равна задержка односторонней передачи у этого канала?
32. Протокол с остановкой и ожиданием передает 800-битные фреймы по каналу с задержкой односторонней передачи 8 мс и с пропускной способностью 1200 Кбит/с. Какова при этом эффективность использования пропускной способности?
33. Протокол «раздвижного окна» использует 1000-битные фреймы и фиксированный размер посылающего окна, равный 3. Он обеспечивает почти 100%-ю эффективность использования пропускной способности канала, которая равна 250 Кбит/с. Вы решили использовать этот протокол в более мощном канале. У него такая же величина задержки, а пропускная способность в два раза больше. Какую эффективность использования пропускной способности обеспечит данный протокол в новом канале?
34. Возможно ли, что в протоколе 3 отправитель запустит таймер, когда тот уже работает? Если да, то в какой ситуации? Если нет, то почему?
35. Кабель T1 длиной 3000 км используется для передачи 64-байтных фреймов при помощи протокола 5. Если задержка распространения сигнала составляет 6 мкс/км, сколько битов следует выделить для порядковых номеров фреймов?
36. Представьте себе протокол «раздвижного окна», в котором так много битов на порядковые номера фреймов, что номера никогда не используются дважды. Какое соотношение должно связывать четыре границы окна и его размер (постоянный и одинаковый для отправителя и получателя)?
37. Когда прибывает фрейм данных, протокол 6 проверяет, отличается ли его номер от ожидаемого и равна ли переменная *no_nak* значению *true*. При выполнении обоих условий посылается NAK. В противном случае запускается вспомогательный таймер. Предположим, что в тексте программы пропущен оператор `else`. Повлияет ли это на правильность работы протокола?

38. Предположим, что из конца текста программы протокола 6 удалены три строки цикла while. Повлияет ли это на правильность работы протокола или же только на его быстродействие? Обоснуйте ваш ответ.
39. Допустим, в условиях предыдущей задачи используется другой протокол — протокол «раздвижного окна». При каком размере окна отправителя коэффициент загруженности канала будет равен 100 %? Время на обработку протокола на стороне отправителя и получателя можно не учитывать.
40. Допустим, в коде протокола 6 из оператора switch будет удален случай, соответствующий возникновению ошибок в контрольной сумме. Как это изменение скажется на работе протокола?
41. В протоколе 6 код для *frame_arrival* содержит раздел, используемый для отрицательных подтверждений (NAK). Этому участку программы передается управление, когда получаемый фрейм представляет собой NAK и выполняется другое условие. Приведите пример сценария, в котором наличие этого условия является важным.
42. Протокол 6 применяется на свободной от ошибок линии со скоростью 1 Мбит/с. Максимальный размер фрейма 1000 бит. Новые пакеты формируются примерно один раз в секунду. Интервал тайм-аута установлен на период 10 мс. Если отключить специальный таймер подтверждений, то будут происходить лишние тайм-ауты. Сколько раз в среднем будет передаваться одно сообщение?
43. В протоколе 6 значение $MAX_SEQ = 2^n - 1$. Разумеется, это желательное условие для эффективного использования битов заголовка, но его необходимость не доказана. Будет ли протокол корректно работать, например, при $MAX_SEQ = 4$?
44. Фреймы длиной 1000 бит посылаются по спутниковому каналу с пропускной способностью 1 Мбит/с и временем прохождения 270 мс. Подтверждения всегда вкладываются во фреймы данных. Заголовки фреймов очень короткие. Используются 3-битные порядковые номера. Какой будет максимальная эффективность использования канала при применении:
    1) протокола с остановкой и ожиданием;
    2) протокола 5;
    3) протокола 6?
45. Отрицательные подтверждения вызывают немедленные ответные действия отправителя. При этом отсутствие положительных подтверждений запускает реакцию лишь по прошествии некоторого периода ожидания. Можно ли создать надежный канал связи, используя только отрицательные подтверждения? Если это возможно, приведите пример. Если это невозможно, объясните почему.
46. Предположим, безошибочный спутниковый канал с пропускной способностью 64 Кбит/с используется для пересылки 512-байтных фреймов данных в одном направлении с очень короткими подтверждениями, идущими в обратном направлении. Какова будет максимальная скорость передачи данных

при размере окна, равном 1, 7, 15 и 127? Время распространения сигнала от Земли до спутника — 270 мс.

47. Кабель длиной в 100 км работает на скорости T1. Скорость распространения сигнала равна 2/3 от скорости света в вакууме. Сколько битов помещается в кабеле?
48. Назовите хотя бы одну причину, по которой в протоколе PPP применяется байт-стаффинг вместо бит-стаффинга (это предотвращает ошибку синхронизации фреймов из-за случайно встретившегося в поле данных флагового байта).
49. Каковы минимальные накладные расходы при пересылке IP-пакета по протоколу PPP? Учитывайте только накладные расходы самого PPP, а не заголовки IP. Каковы максимальные накладные расходы?
50. Следующий поток данных пересылается с помощью PPP по сетям SONET: ESC FLAG FLAG ESC. Какая последовательность байтов пересылается в качестве пользовательских данных? Напишите свой ответ в виде байтовых последовательностей, каждая из которых состоит из восьми единиц или нулей. ESC можно представить с помощью битовой последовательности 01111101, а FLAG — с помощью битовой последовательности 01111110.
51. IP-пакет длиной 100 байт передается по абонентскому шлейфу с использованием стека протоколов ADSL. Сколько ячеек ATM будет передано? Кратко опишите их содержимое.
52. Цель данного упражнения — реализация механизма обнаружения ошибок с помощью стандартного алгоритма циклического избыточного кода (CRC), описанного в тексте. Напишите две программы: генератор (*generator*) и верификатор (*verifier*). Программа-генератор считывает со стандартного устройства ввода $n$-битовое сообщение из нулей и единиц, представленных в виде строки ASCII-текста. Вторая строка является $k$-битовым многочленом (также в ASCII). На устройстве вывода мы получаем текст из $n + k$ нулей и единиц, представляющий собой сообщение, подлежащее пересылке. Затем выводится многочлен — в том же виде, в каком он был считан. Программа-верификатор считывает результат работы генератора и выводит сообщение, в котором обозначено, корректен ли данный результат. После этого напишите программу *alter*, вносящую сбой, а именно инвертирующую только один бит первой строки в зависимости от аргумента (например, порядкового номера бита, предполагая, что слева располагается бит с номером 1). Все остальные данные передаются без изменений. Набрав

```
generator <file | verifier
```

пользователь должен увидеть сообщение о том, что данные переданы корректно. Набрав

```
generator <file | alter arg | verifier
```

пользователь должен получить сообщение об ошибке при передаче.

# ГЛАВА 4

# Подуровень управления доступом к среде

Многие из протоколов канального уровня, о которых мы говорили в главе 3, рассчитаны на использование широковещательной среды для передачи данных. Они нуждаются в дополнительных механизмах, обеспечивающих эффективное и справедливое распределение канала между несколькими отправителями. Таким протоколам и посвящена эта глава.

Главная проблема любой широковещательной сети — кому предоставить канал, если его запрашивают несколько пользователей. К примеру, представьте себе конференц-звонок, в котором принимают участие шесть человек с разных телефонов. Связь организована таким образом, что каждый может слышать всех собеседников и разговаривать с ними. Когда один из них закончит свою речь, весьма вероятно, что сразу двое или трое заговорят одновременно, тем самым создав неловкую ситуацию. При личной встрече это предотвращается внешними средствами. Например, на совещании можно поднять руку и получить разрешение высказаться. Когда доступен лишь один канал, гораздо труднее определить, кому предоставить слово. Для решения этой проблемы разработано множество протоколов. В литературе широковещательные каналы иногда называют **каналами с множественным доступом (multiaccess channels)** или **каналами с произвольным доступом (random access channels)**.

Протоколы, определяющие очередность использования линии, относятся к канальному уровню, точнее, к подуровню **управления доступом к среде (MAC, Medium Access Control)**. MAC особенно важен в локальных сетях (LAN), в частности беспроводных, так как они по своей природе являются широковещательными. Глобальные сети (WAN) включают в себя как двухточечные компоненты (например, прямое подключение), так и компоненты с совместным использованием, осуществляемым с помощью MAC (например, кабельные сети общего доступа, предоставляемые интернет-провайдерами). В данной главе мы в общих чертах обсудим LAN (так как они тесно связаны с каналами множественного доступа), включая некоторые вопросы, которые не относятся к MAC напрямую. Главной темой будет управление каналом.

Технически подуровень MAC является нижней частью канального уровня, и логичнее было бы изучить сначала его, а затем протоколы «точка-точка»,

рассмотренные в главе 3. Но намного легче сначала рассмотреть протоколы с двумя участниками, а потом — с несколькими. По этой причине мы слегка отклонимся от строгого следования снизу вверх по иерархической лестнице при рассмотрении уровней.

## 4.1. ПРОБЛЕМА РАСПРЕДЕЛЕНИЯ КАНАЛА

Основная проблема, обсуждаемая в этой главе, — распределение одного широковещательного канала между претендующими на него многочисленными пользователями. Канал может быть частью беспроводного спектра в каком-нибудь регионе, отдельным проводом или оптоволоконным кабелем, к которому присоединено несколько узлов. Это не имеет значения. Во всех случаях он соединяет каждого пользователя со всеми остальными, и любой абонент, полностью нагружающий канал, мешает тем, кто тоже хочет передавать данные.

Сначала мы в общих чертах рассмотрим недостатки статических схем распределения канала в случае неравномерного трафика. Затем изложим ключевые предположения, применяемые для моделирования динамических схем. После этого обсудим несколько примеров таких схем.

### 4.1.1. Статическое распределение канала

Традиционный способ распределения канала (например, телефонной линии) между многочисленными конкурирующими пользователями — разделение его пропускной способности с помощью одной из схем мультиплексирования, рассмотренных нами в разделе 2.4.4 (например, FDM). При наличии $N$ пользователей полоса пропускания делится на $N$ диапазонов одинаковой ширины, и каждому пользователю предоставляется один из них. Поскольку при такой схеме у каждого пользователя своя личная частотная полоса, то конфликтов не возникает. При небольшом и постоянном числе абонентов, передающих стабильный поток или большие партии трафика, это простой и эффективный механизм распределения. Аналогичный пример для беспроводной связи — радиостанции FM-диапазона. Каждая станция получает часть FM-полосы и большую часть времени передает по ней свой сигнал.

Однако при большом и постоянно меняющемся числе пользователей или неравномерном трафике FDM не обеспечивает достаточно эффективного распределения. Если в какой-то момент отправителей меньше, чем частей спектра, то значительная часть полосы пропускания простаивает. И наоборот, если пользователей больше, то некоторым из них придется отказать в доступе, даже если уже подключенные абоненты почти не используют канал.

Предположим, что количество пользователей можно каким-то способом удерживать на постоянном уровне. Но и в этом случае разделение канала на статические подканалы неэффективно. Основная проблема в том, что если какие-то пользователи не передают данные, их часть спектра просто пропадает. При этом они занимают линию, и остальные абоненты не могут ее использовать. Статическое разделение не подходит для большинства компьютерных систем

с чрезвычайно неравномерным потоком данных (вполне обычным является отношение пикового трафика к среднему как 1000:1). В результате большая часть каналов будет часто простаивать.

Низкую эффективность статического FDM можно увидеть на примере простых вычислений теории массового обслуживания. Для начала подсчитаем среднее время задержки $T$ для отправки фрейма по каналу емкостью $C$ бит/с. Предполагается, что фреймы приходят в случайном порядке со средней скоростью $\lambda$ фреймов в секунду. Длина фреймов варьируется и в среднем равна $1/\mu$ бит. При таких параметрах скорость обслуживания канала равна $\mu C$ фреймов в секунду. Теория массового обслуживания говорит о том, что

$$T = \frac{1}{\mu C - \lambda}.$$

(Для любознательных: это результат для очереди «M/M/1». Требуется, чтобы случайность длительности промежутков между фреймами и длины фреймов соответствовала экспоненциальному распределению или, что эквивалентно, являлась результатом пуассоновского процесса.)

В нашем примере $C$ равно 100 Мбит/с, средняя длина фрейма $1/\mu$ = 10 000 бит, скорость входного потока $\lambda$ = 5000 фреймов в секунду. Тогда $T$ = 200 мкс. Обратите внимание: если бы мы не учли задержки при формировании очереди и просто посчитали, сколько времени нужно на передачу фрейма длиной 10 000 бит по сети с пропускной способностью 100 Мбит/с, то получили бы неправильный ответ: 100 мкс. Этот результат верен лишь при отсутствии конкуренции за канал.

Теперь разделим канал на $N$ независимых подканалов, каждый из которых имеет пропускную способность $C/N$ бит/с. Средняя входная скорость в подканале теперь равна $\lambda/N$ фреймов в секунду. Вычислив новое значение средней задержки $T$, получим:

$$T_N = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT.$$

В разделенном канале средняя задержка в $N$ раз хуже, чем в канале, в котором все фреймы волшебным образом организованы в одну общую очередь. Тот же вывод можно сделать на следующем примере: предоставляя доступ к банкоматам в холле банка, лучше организовать посетителей в одну общую очередь. Отдельные очереди могут привести к тому, что одни банкоматы будут простаивать, в то время как перед другими выстроится много людей.

Аргументы, применимые к FDM, относятся и к другим способам статического распределения канала. Можно использовать схему TDM и выделять каждому пользователю $N$-й слот. Но если абонент не будет передавать данные, этот слот просто пропадет. С тем же успехом можно разделить сети физически. Если взять 100-мегабитную сеть и сделать из нее десять 10-мегабитных, статически распределив по ним пользователей, то в результате средняя задержка возрастет с 200 мкс до 2 мс.

Таким образом, ни один статический метод распределения каналов не подходит для неравномерного трафика, поэтому далее мы рассмотрим динамические методы.

### 4.1.2. Допущения, связанные с динамическим распределением каналов

Прежде чем приступить к рассмотрению многочисленных методов распределения каналов, следует тщательно сформулировать задачу. В основе всех разработок в данной области лежат следующие пять допущений.

1. **Независимый трафик.** Модель состоит из $N$ независимых **станций** (компьютеров, телефонов, персональных средств связи и т. д.), в каждой из которых программа или пользователь формируют фреймы для передачи. Ожидаемое число фреймов в интервале времени $\Delta t$ равно $\lambda \Delta t$, где $\lambda$ — постоянная величина (скорость прибытия новых фреймов). Как только фрейм сформирован, станция блокируется и ничего не делает, пока он не будет успешно передан.
2. **Единый канал.** Он доступен для всех. Все станции могут передавать и принимать по нему данные. Они обладают одинаковой производительностью, хотя программно протокол может устанавливать для них различные роли (например, приоритеты).
3. **Наблюдаемые коллизии.** Если два фрейма передаются одновременно, они перекрываются по времени, и в результате сигнал искажается. Такое событие называется **коллизией (collision)**. Обнаруживать их могут все станции. Фрейм, искаженный из-за коллизии, должен быть передан повторно. Других ошибок, кроме тех, которые вызваны коллизиями, нет.
4. **Непрерывное/дискретное время.** Если допустить, что время непрерывно, то передача фреймов может начаться в любой момент. В противном случае время разделяется на дискретные интервалы (слоты). Отправка фрейма происходит только с началом слота. Один слот может содержать 0 (свободный интервал), 1 (успешная передача) или более фреймов (коллизия).
5. **Контроль (опрос) несущей/его отсутствие.** При контроле несущей станции определяют, свободна ли линия, прежде чем начать ее использовать. Если канал занят, станции не будут пытаться передавать по нему фреймы, пока тот не освободится. Если контроля несущей нет, то станции не могут заранее получить эту информацию. Они просто начинают передачу и только потом выясняют, была ли она успешной.

О приведенных выше допущениях следует сказать несколько слов. Первое утверждает, что фреймы приходят независимо друг от друга (как на разные станции, так и в пределах одной) и что они формируются непредсказуемо, но с постоянной скоростью. В действительности это не слишком хорошая модель сетевого трафика, поскольку давно известно, что пакеты прибывают целыми последовательностями в определенные диапазоны временной шкалы (см. работу

Паксона и Флойд; Paxson, Floyd, 1995). Последние исследования подтверждают данную закономерность (Фонтюнь и др.; Fontugne et al., 2017). При этом **пуассоновские модели**, как их часто называют, находят широкое применение, в том числе потому, что они легко поддаются математическому описанию. Они позволяют анализировать протоколы, чтобы составить общее представление об изменении производительности с течением времени и о разнице между реализациями.

Единый канал — центральное допущение в данной модели. Никаких дополнительных средств связи нет. Станции не могут тянуть руки в надежде, что учитель их спросит, поэтому приходится искать лучшее решение.

Оставшиеся три допущения зависят от инженерной реализации системы, поэтому при изучении конкретных протоколов мы будем указывать, какие допущения следует считать верными.

Допущение о коллизиях является базовым. Станциям необходим способ обнаружения коллизий, если они собираются повторно пересылать фреймы, а не мириться с их потерей. Для проводных каналов можно использовать оборудование, умеющее определять коллизии. В этом случае станции заранее обрывают передачу, чтобы не засорять канал. В беспроводных каналах распознавать коллизии намного сложнее; об их возникновении приходится узнавать по факту, когда не прибывает ожидаемое подтверждение. Также возможно успешное получение некоторых фреймов, попавших в коллизию, — это зависит от типа сигнала и оборудования на получающей стороне. Подобные ситуации встречаются нечасто, поэтому будем предполагать, что все эти фреймы теряются. Кроме того, есть протоколы, специально предназначенные для предотвращения коллизий, а не решения создаваемых ими проблем.

Причина существования двух альтернативных допущений для времени заключается в том, что дискретное время иногда помогает повысить производительность. Однако использующие его станции должны синхронизироваться с тактовым генератором или друг с другом, а это не всегда возможно. Мы рассмотрим оба варианта. В каждой конкретной системе работает только одно из возможных допущений.

Контроль несущей также реализован не во всех системах. В проводных сетях такой контроль обычно присутствует, но беспроводные не всегда могут эффективно его использовать, поскольку не каждая станция может слышать все остальные из-за разницы частотных диапазонов. Аналогично, когда станция не может напрямую взаимодействовать с другими (например, ей приходится пересылать информацию через кабельный модем, играющий роль центрального узла), контроль несущей бывает недоступен. Обратите внимание, что слово «несущая» здесь означает электрический сигнал, распространяющийся по каналу.

Чтобы избежать недопонимания, стоит заметить, что ни один протокол коллективного доступа не гарантирует надежную доставку. Даже при отсутствии коллизий получатель мог по каким-то причинам неправильно скопировать часть фрейма. Надежность обеспечивают другие составляющие канального или более высоких уровней.

## 4.2. ПРОТОКОЛЫ КОЛЛЕКТИВНОГО ДОСТУПА

Существует множество алгоритмов коллективного доступа. В следующих разделах будут рассмотрены наиболее интересные из них и даны примеры их применения на практике.

### 4.2.1. Система ALOHA

История первого протокола подуровня MAC начинается на нетронутых цивилизацией Гавайях в 1970-х годах. В данном случае «нетронутые цивилизацией» означает «не имеющие рабочей телефонной системы». Это не упрощало жизнь Нормана Абрамсона (Norman Abramson) и его коллег из Гавайского университета, которые пытались подключить пользователей на удаленных островах к главному компьютеру в Гонолулу. Идея протянуть кабели на громадное расстояние по дну Тихого океана даже не рассматривалась, так что исследователи искали другой способ.

Решение было основано на использовании радиосистемы ближнего радиуса действия. Терминал каждого пользователя передавал фреймы на центральный компьютер в пределах общей полосы частот. Также был найден простой и элегантный метод решения проблемы распределения каналов. Результаты работы этой группы ученых впоследствии стали основой многих исследований (Шварц и Абрамсон; Schwartz and Abramson, 2009). Несмотря на то что в разработанной Абрамсоном сети ALOHA использовалась широковещательная радиосвязь со стационарными передатчиками, ее общий принцип применим к любой системе, в которой независимые пользователи соревнуются за право использования одного общего канала.

В данном разделе мы рассмотрим две версии системы ALOHA: чистую и дискретную. Различие между ними состоит в том, что в чистой версии время является непрерывным, а в дискретной делится на дискретные интервалы, в которые должны помещаться все фреймы.

#### *Чистая система ALOHA*

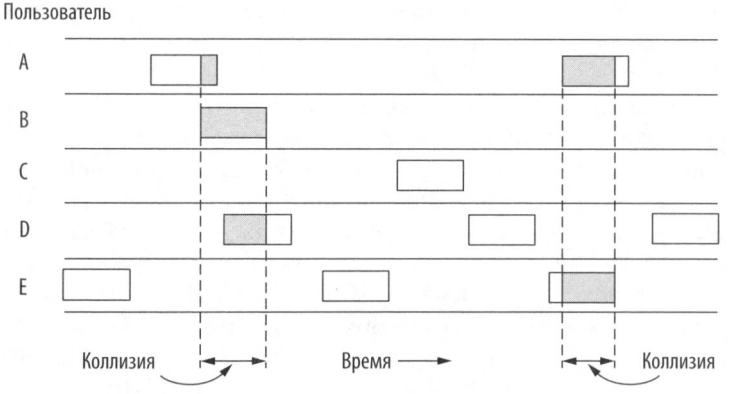
В основе ALOHA лежит простая идея: разрешить пользователям передачу, как только у них появляются данные для отправки. Конечно, будут возникать коллизии, приводящие к повреждению фреймов. Отправителям необходимо уметь обнаруживать такие ситуации. Каждая станция отправляет свой фрейм центральному компьютеру, а тот передает его на все остальные станции. Отправитель прослушивает широковещательную передачу, чтобы понять, насколько она была успешной. В других системах, например проводных LAN, отправитель имеет возможность распознавать коллизии во время передачи.

Если фрейм был уничтожен, отправитель выжидает в течение случайного временного интервала и пытается переслать его снова. Время ожидания должно быть случайным, иначе одни и те же фреймы будут синхронно попадать в коллизию снова и снова. Системы, в которых несколько пользователей используют

один общий канал так, что время от времени возникают конфликты, называются **системами с конкуренцией (contention systems)**.

На илл. 4.1 показан пример формирования фреймов в ALOHA. Все они имеют единый фиксированный размер, так как за счет этого пропускная способность системы становится максимальной.

Когда два фрейма одновременно пытаются занять канал, происходит коллизия (как видно на илл. 4.1). Оба фрейма искажаются. Даже если только один первый бит второго фрейма перекроет последний бит первого, оба фрейма полностью теряются (то есть их контрольные суммы не совпадут с правильными значениями). Позже они должны быть переданы повторно. В контрольной сумме нет (и не должно быть) различия между полным и частичным искажением информации. Потеря есть потеря.




**Илл. 4.1.** В чистой системе ALOHA фреймы передаются в абсолютно произвольное время

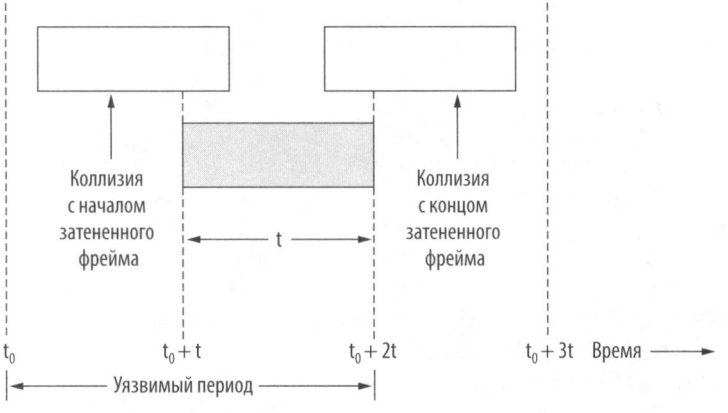
Интересно, какова эффективность канала ALOHA? Другими словами, какая часть передаваемых фреймов избежит коллизии при таком беспорядке? Представьте бесконечное множество пользователей, сидящих за своими компьютерами (станциями). Пользователь всегда находится в одном из двух состояний: ввод с клавиатуры или ожидание. Вначале все находятся в состоянии ввода. Закончив набор строки, пользователь перестает вводить текст, ожидая ответа. В это время станция передает фрейм, содержащий набранную строку, по общему каналу на центральный компьютер и опрашивает канал, проверяя успешность передачи. Если фрейм передан успешно, пользователь видит ответ и продолжает набор. В противном случае он ждет повторной передачи, которая происходит раз за разом, пока данные не будут успешно отправлены.

Пусть «время фрейма» означает интервал, требуемый для отправки стандартного фрейма фиксированной длины (это длина фрейма, деленная на скорость передачи). Допустим, новые фреймы, порождаемые станциями, хорошо распределены по Пуассону со средним значением $N$ фреймов за интервал. (Допущение о бесконечном количестве пользователей необходимо для того, чтобы гарантировать, что $N$ не станет уменьшаться по мере их блокирования.) Если

$N > 1$, это означает, что сообщество пользователей формирует фреймы с большей скоростью, чем может быть передано по каналу, и почти каждый фрейм будет искажаться. Разумнее предположить, что $0 < N < 1$.

Помимо новых, станции повторно отправляют старые фреймы, пострадавшие от столкновений. Предположим, что старые и новые фреймы хорошо распределены по Пуассону со средним значением $G$ фреймов за интервал. Очевидно, что $G \geq N$. При малой загрузке канала (то есть при $N \approx 0$) коллизий возникает мало, как и повторных передач, то есть $G \approx N$. При большой загрузке коллизий много, а следовательно, $G > N$. Какая бы ни была загрузка, производительность канала $S$ будет равна предлагаемой загрузке $G$, умноженной на вероятность успешной передачи $P_0$, то есть $S = GP_0$, где $P_0$ — вероятность того, что фрейм не повредится в результате коллизии.

Фрейм не пострадает, если в течение интервала времени его передачи больше ничего не отправлять, как показано на илл. 4.2. При каких условиях фрейм, затененный на рисунке, будет передан без повреждений? Пусть $t$ — это время, требуемое для передачи фрейма. Если пользователь сформирует фрейм в интервале времени между $t_0$ и $t_0 + t$, то его конец столкнется с началом затененного фрейма. При этом судьба затененного фрейма предрешена еще до того, как будет послан его первый бит. Но в чистой ALOHA станции не прослушивают линию до начала передачи и у них нет способа узнать, что канал занят и по нему уже передается фрейм. Аналогичным образом, любой другой фрейм, передача которого начнется в интервале от $t_0 + t$ до $t_0 + 2t$, столкнется с концом затененного фрейма.




**Илл. 4.2.** Уязвимый период времени для затененного фрейма

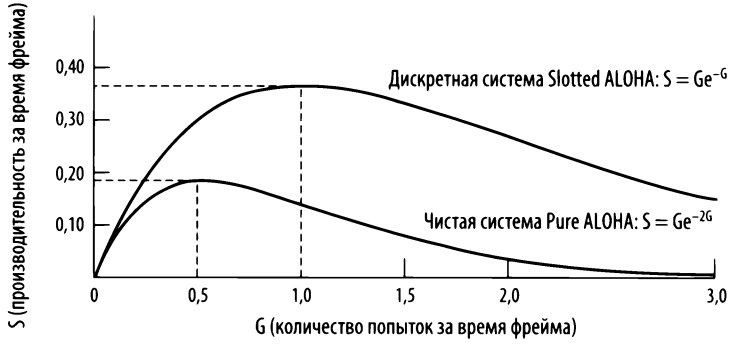
Вероятность того, что за время фрейма вместо $G$ будет сформировано $k$ фреймов, можно вычислить по формуле распределения Пуассона:

$$\Pr[k] = \frac{G^k e^{-G}}{k!}. \tag{4.1}$$

Таким образом, вероятность генерации нуля фреймов в течение этого интервала равна $e^{-G}$. Среднее количество фреймов, сформированных за интервал длиной в два фрейма, равно $2G$. Вероятность того, что никто не начнет передачу в течение всего уязвимого периода, равна $P_0 = e^{-2G}$. Учитывая, что $S = GP_0$, получаем:

$$S = Ge^{-2G}.$$

Соотношение между предоставляемым трафиком и пропускной способностью показано на илл. 4.3. Максимальная производительность достигает значения $S = 1/(2e)$, что приблизительно равно 0,184 при $G = 0{,}5$. Другими словами, лучшее, на что мы можем надеяться, — это использовать канал на 18 %. Этот результат несколько разочаровывает, но если каждый передает данные тогда, когда хочет, трудно ожидать стопроцентного успеха.




**Илл. 4.3.** Зависимость производительности канала от предоставляемого трафика для систем ALOHA

## *Дискретная система ALOHA*

Вскоре после появления системы ALOHA Робертс (Roberts, 1972) опубликовал метод, позволяющий удвоить ее производительность. Он предложил разделять время на дискретные интервалы, соответствующие одному фрейму. Их называют **слотами (slots)**. При таком подходе пользователи должны согласиться с определенными временными ограничениями. Одним из способов достижения синхронизации является установка специальной станции, которая генерирует синхронизирующий сигнал в начале каждого интервала.

**Дискретная ALOHA** Робертса отличается от **чистой ALOHA** Абрамсона тем, что станция не может начинать передачу сразу после ввода пользователем строки. Вместо этого она должна дождаться начала нового слота. Таким образом, система ALOHA с непрерывным временем превращается в дискретную. Уязвимый временной интервал теперь в два раза короче. Чтобы понять это, взгляните на илл. 4.3 и представьте, какие теперь возможны коллизии. Вероятность отсутствия трафика в течение того же интервала, в котором передается тестовый фрейм, равна $e^{-G}$. В результате получаем:

$$S = Ge^{-G}.$$

Как видно из илл. 4.3, дискретная ALOHA имеет пик при $G = 1$. Производительность канала составляет $S = 1/e$, что приблизительно равно 0,368, то есть в два раза больше, чем в чистой ALOHA. Если система работает при $G = 1$, вероятность появления пустого слота равна 0,368 (согласно уравнению 4.1). Для дискретной системы ALOHA в оптимальной ситуации 37 % интервалов будут пустыми, 37 % — с успешно переданными фреймами и 26 % — с коллизией. При более высоких значениях $G$ количество пустых интервалов уменьшается, но коллизий становится значительно больше. Чтобы увидеть, как быстро растет их число, рассмотрим передачу тестового фрейма. Вероятность того, что он избежит коллизии, равна $e^{-G}$. Фактически это означает, что все остальные станции будут молчать в этом слоте. Таким образом, шанс возникновения коллизии равен $1 - e^{-G}$. Вероятность передачи фрейма ровно за $k$ попыток (то есть после $k - 1$ коллизий, за которыми следует успешная отправка) равна

$$P_k = e^{-G}\left(1 - e^{-G}\right)^{k-1}.$$

Ожидаемое число попыток передачи $E$ для одной строки, введенной на терминале, равно

$$E = \sum_{k=1}^{\infty} kP_k = \sum_{k=1}^{\infty} ke^{-G}\left(1 - e^{-G}\right)^{k-1} = e^{G}.$$

Поскольку $E$ экспоненциально зависит от $G$, небольшое увеличение нагрузки в канале может серьезно снизить его производительность.

Дискретная система ALOHA заслуживает внимания по причине, которая на первый взгляд не кажется очевидной. Она появилась в 1970-е годы, применялась в некоторых экспериментальных системах и затем была практически забыта (если не считать упоминаний в работах чудаковатых авторов, считающих ее интересной). С появлением кабельного интернет-доступа вновь возникла проблема распределения общего канала между большим числом конкурирующих пользователей. Тогда дискретная ALOHA была вновь извлечена на белый свет и в сочетании с рядом новых идей обеспечила решение проблемы. Часто вполне работоспособные протоколы оказывались невостребованными по политическим причинам (например, если какая-нибудь корпорация требовала, чтобы все использовали исключительно ее продукцию) или из-за постоянно меняющихся технологий. Однако по прошествии многих лет какой-нибудь мудрый человек вспоминал о существовании древнего метода, способного решить современную проблему. Именно поэтому в данной главе мы изучим ряд элегантных, но непопулярных протоколов, которые вполне могут оказаться востребованными в будущем при условии, что о них будет знать достаточное количество разработчиков сетей. Разумеется, мы также обсудим множество актуальных протоколов.

## 4.2.2. Протоколы множественного доступа с контролем несущей

В дискретной системе ALOHA максимальный коэффициент использования канала равен $1/e$. Такой скромный показатель неудивителен, поскольку станции

передают данные, когда хотят, не считаясь с тем, что делают остальные. В такой системе неизбежно возникает большое количество коллизий. Однако в локальных сетях можно организовать процесс так, что станции будут учитывать поведение друг друга. За счет этого можно достичь намного большего значения, чем $1/e$. В данном разделе представлены некоторые протоколы, позволяющие улучшить производительность канала.

Протоколы, в которых станции прослушивают среду передачи данных и действуют в соответствии с этим, называются **протоколами с контролем (опросом) несущей (carrier sense protocols)**. Было разработано множество таких протоколов, и они давно подробно проанализированы, например, в работе Клейнрока и Тобаги (Kleinrock and Tobagi, 1975). Ниже мы рассмотрим несколько версий протоколов с контролем несущей.

### *Настойчивый и ненастойчивый CSMA*

Мы начнем с **протокола CSMA с настойчивостью 1 (Carrier-Sense Multiple Access — множественный доступ с контролем несущей)**. Длинноватое название для простейшей схемы CSMA. Когда у станции появляются данные для передачи, она сначала прослушивает канал, проверяя, свободен ли он. Когда канал простаивает, станция сразу отправляет данные. Если же он занят — она ждет освобождения линии, а затем передает фрейм. В случае коллизии станция ждет в течение случайного интервала времени, затем снова прослушивает канал, и если он свободен, повторяет передачу. Такой протокол называется CSMA с настойчивостью 1, поскольку станция передает фрейм с вероятностью 1, как только обнаружит, что канал свободен.

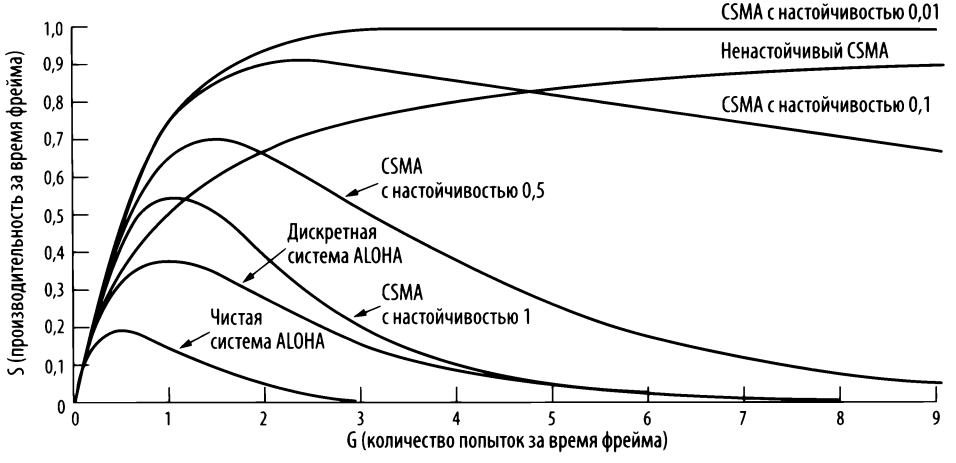
Может показаться, что в этой схеме отсутствуют коллизии (за исключением редких случаев одновременной отправки), но это не так. Допустим, две станции пришли в состояние готовности во время работы третьей. Обе они ждут, пока та не закончит отправку, после чего начинают передавать одновременно, и в результате происходит коллизия. Если бы станции не были столь нетерпеливы, коллизий было бы меньше.

Если копнуть чуть глубже, можно увидеть, что на число коллизий значительное влияние оказывает задержка распространения сигнала. Существует небольшая вероятность того, что как только одна станция начнет передачу, другая также придет в состояние готовности и опросит канал. Если сигнал от первой станции еще не успел достичь второй, она решит, что канал свободен, и также начнет передачу, в результате произойдет коллизия. Эта вероятность зависит от числа фреймов, помещающихся в канал, то есть от показателя **«полоса пропускания, умноженная на задержку» (bandwidth-delay product)** для данного канала. Если канал вмещает лишь небольшую часть фрейма, как бывает в большинстве LAN, где задержка распространения невелика, то и шанс коллизии мал. Чем больше время распространения сигнала, тем выше вероятность коллизий и ниже производительность протокола. Однако даже такая система значительно лучше чистой ALOHA, так как обе станции воздерживаются от передачи, пока передает третья станция, в результате чего ее фрейм остается неповрежденным. То же самое можно сказать о дискретной системе ALOHA.

Далее мы рассмотрим **ненастойчивый протокол CSMA**. В нем предпринята попытка сдержать стремление станций начинать передачу, как только канал освобождается. Как и в случае с предыдущим протоколом, прежде чем начать передачу, станция опрашивает канал. Если в данный момент никто ничего не передает, станция сразу же начинает передачу сама. Однако если канал занят, станция не ждет его освобождения, постоянно прослушивая его и пытаясь захватить сразу, как только он освободится, как в предыдущем протоколе. Вместо этого она ждет в течение случайного интервала времени, а затем снова прослушивает линию. В результате количество коллизий уменьшается и повышается эффективность использования канала. При этом увеличивается интервал ожидания по сравнению с протоколом CSMA с настойчивостью 1.

Третий протокол с контролем несущей — **CSMA с настойчивостью p**. Он применяется в дискретных каналах и работает следующим образом. Когда станция готова передавать, она опрашивает канал. Если канал свободен, она с вероятностью $p$ начинает передачу. С вероятностью $q = 1 - p$ она отказывается от передачи и ждет начала следующего слота. Этот процесс повторяется до тех пор, пока фрейм не будет передан или какая-либо другая станция не начнет передачу. В последнем случае станция ведет себя так же, как в случае коллизии. Она ждет в течение случайного интервала времени, после чего начинает все заново. Если при первом прослушивании канала выясняется, что он занят, станция ждет следующий слот, а затем применяет тот же алгоритм IEEE 802.1 с использованием более совершенного протокола CSMA с настойчивостью $p$, который мы обсудим далее в разделе 4.4.

На илл. 4.4 показана расчетная зависимость производительности канала от предлагаемого потока фреймов для всех трех протоколов, а также для чистой и дискретной систем ALOHA.



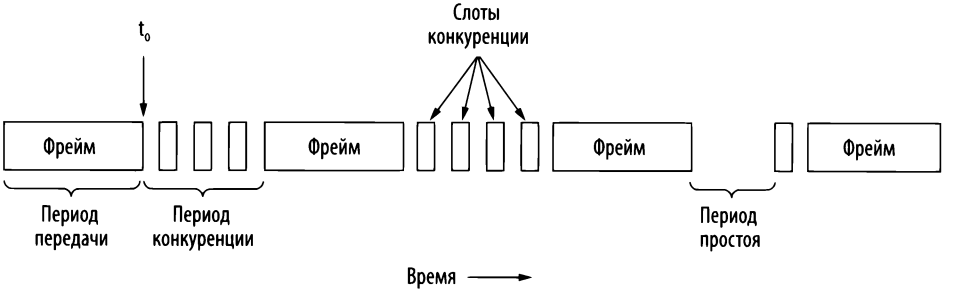

**Илл. 4.4.** Сравнение использования канала в зависимости от его загрузки для различных протоколов коллективного доступа

### *Протокол CSMA с обнаружением коллизий*

Настойчивый и ненастойчивый протоколы CSMA, несомненно, более эффективны по сравнению с системой ALOHA, поскольку гарантируют, что ни одна станция не начнет передачу, если канал уже занят. Однако если две станции, обнаружив, что канал свободен, одновременно начали передачу, коллизия все равно произойдет. Еще одно улучшение – станции способны быстро распознать коллизию и немедленно прекратить передачу (а не доводить ее до конца), так как данные все равно искажены. Эта стратегия обеспечивает более экономное использование времени и пропускной способности канала.

**Протокол CSMA с обнаружением коллизий (CSMA/CD)** лежит в основе классических локальных сетей Ethernet и потому заслуживает подробного рассмотрения. Важно понимать, что распознавание коллизий представляет собой аналоговый процесс. Оборудование станции должно прослушивать канал во время передачи. Если считываемый сигнал отличается от пересылаемого, становится понятно, что произошла коллизия. Полученный сигнал не обязательно должен идеально совпадать с отправленным. Это затруднительно для беспроводных сетей, в которых принятый сигнал нередко в 1 000 000 раз слабее отправленного. Необходимо выбрать такую модуляцию, которая позволит распознавать коллизии (например, коллизию двух 0-вольтовых сигналов распознать практически невозможно).

В протоколе CSMA/CD, как и во многих других протоколах LAN, применяется концептуальная модель, показанная на илл. 4.5. В момент времени $t_0$ одна из станций заканчивает передачу фрейма. Все остальные станции, готовые к передаче, теперь могут попытаться отправить свои фреймы. Если две или несколько станций одновременно начнут передачу, то произойдет коллизия. Обнаружив ее, станция прекращает передачу, ждет в течение случайного периода времени, после чего пытается снова, при условии, что к этому моменту не начала передачу другая станция. Таким образом, простая модель протокола CSMA/CD состоит из чередующихся периодов конкуренции и передачи, а также периодов простоя (когда все станции молчат).




**Илл. 4.5.** Протокол CSMA/CD может находиться в одном из трех состояний: передачи, конкуренции и простоя

Рассмотрим более подробно алгоритм борьбы за право использования канала. Предположим, две станции одновременно начали передачу в момент $t_0$. Сколько

понадобится времени на то, чтобы они поняли, что произошла коллизия? От ответа на этот вопрос зависит длина периода конкуренции, а следовательно, величина задержки и производительность канала.

Минимальное время обнаружения коллизии равно времени распространения сигнала от одной станции до другой. Исходя из этого, можно предположить, что станция, которая не фиксирует коллизию за время, требуемое для прохождения сигнала по всему кабелю, может быть уверена, что ей удалось захватить канал. «Захват» означает, что все остальные станции знают, что она осуществляет передачу, и не мешают ей. Однако такое заключение неверно.

Рассмотрим наихудший сценарий. Пусть время, необходимое для прохождения сигнала между двумя самыми дальними станциями, равно $\tau$. В момент времени $t_0$ одна из станций отправляет сигнал. В момент времени $t_0 + \tau - \varepsilon$, за мгновение до того, как сигнал достигнет самой дальней станции, та станция также начинает передачу. Конечно, почти мгновенно она обнаруживает коллизию и останавливается. Однако всплеск шума, вызванный коллизией, достигает передающей станции только к моменту $2\tau - \varepsilon$. То есть станция не может быть уверена в том, что захватила канал, пока не пройдет $2\tau$ с момента начала передачи.

С учетом этого конкуренцию CSMA/CD можно рассматривать как дискретную систему ALOHA с шириной интервала $2\tau$. В коаксиальном кабеле длиной 1 км $\tau \approx 5$ мкс. Различие между CSMA/CD и дискретной ALOHA состоит в том, что в первом случае за слотом, в котором передачу осуществляет только одна станция (то есть когда канал захвачен), следует передача оставшейся части фрейма. Это позволяет значительно улучшить производительность, если время фрейма намного больше времени распространения сигнала по каналу.

## 4.2.3. Протоколы без коллизий

Хотя в протоколе CSMA/CD коллизии не могут происходить после того, как станция захватывает канал, они могут случаться в период конкуренции. Это снижает производительность системы, особенно когда произведение полосы пропускания на значение задержки велико, то есть при большой длине кабеля (и больших $\tau$) и коротких фреймах. Коллизии не только уменьшают пропускную способность, но и делают время пересылки фрейма непостоянным, что очень плохо для трафика, передаваемого в режиме реального времени (например, голосовых данных по IP). Метод CSMA/CD не является универсальным.

В данном разделе мы рассмотрим протоколы, которые решают проблему борьбы за канал, причем делают это даже без периода конкуренции. Большинство из них сегодня не используются в крупных системах, но в такой изменчивой отрасли всегда хорошо иметь про запас несколько отличных протоколов, которые можно применить в будущем.

В описанных ниже протоколах предполагается наличие $N$ станций. Для каждой из них запрограммирован постоянный уникальный адрес в пределах от 0 до $N - 1$. Некоторые станции могут часть времени оставаться пассивными, но это не имеет значения. Мы будем исходить из того, что задержка распространения сигнала пренебрежимо мала. Главный вопрос остается неизменным:

какой станции будет предоставлен канал после успешной передачи? Мы будем по-прежнему использовать модель, изображенную на илл. 4.5, с ее дискретными интервалами конкуренции.

### Протокол битовой карты

Начнем с протокола без коллизий, который называется **базовым методом битовой карты (basic bit-map method)**. Каждый период конкуренции состоит ровно из $N$ слотов. Если у станции 0 есть фрейм для передачи, она передает единичный бит в слоте 0. Другим станциям не разрешается осуществлять передачу в этом интервале. Независимо от действий станции 0, станция 1 получает возможность передать единичный бит в слоте 1, но только если у нее имеется поставленный в очередь фрейм. В целом станция $j$ может сообщить о наличии у нее готового к передаче фрейма, отправив единичный бит во время интервала $j$. В результате к окончанию интервала $N$ все $N$ станций знают, кто хочет передавать. В этот момент они начинают передачу фреймов в соответствии с порядковыми номерами (илл. 4.6).

**Илл. 4.6.** Базовый протокол битовой карты

Поскольку все знают, чья очередь осуществлять передачу, коллизий не возникает. После того как последняя станция передает свой фрейм (все станции могут это отследить), начинается новый период подачи заявок из $N$ интервалов. Если станция переходит в состояние готовности (получает фрейм для отправки) сразу после того, как она отказалась от передачи, значит, ей не повезло и она должна ждать следующего цикла.

Протоколы, в которых намерение передавать объявляется до самой передачи, называются **протоколами с резервированием (reservation protocols)**. Они заранее резервируют канал для определенной станции, предотвращая коллизии. Оценим производительность такого протокола. Для удобства будем измерять время в однобитных интервалах периода подачи заявок, при этом фрейм данных состоит из $d$ единиц времени.

При слабой загрузке канала битовая карта просто будет повторяться снова и снова из-за недостатка фреймов с данными. Рассмотрим эту ситуацию с точки зрения станции с небольшим номером, например 0 или 1. Обычно в тот момент, когда у нее возникает потребность в передаче, текущий слот уже находится где-то в середине битовой карты. В среднем станция будет ждать $N/2$ слотов до окончания текущего периода резервирования и еще $N$ слотов следующего периода, прежде чем она сможет начать передачу.

Перспективы станций с большими номерами более радужные. В среднем время ожидания передачи составит половину цикла ($N/2$ однобитовых слотов). Таким станциям редко приходится ждать следующего цикла. Поскольку станциям с небольшими номерами приходится ждать в среднем $1{,}5N$ слота, а с большими — $N/2$ слотов, среднее время ожидания для всех станций составляет $N$ слотов.

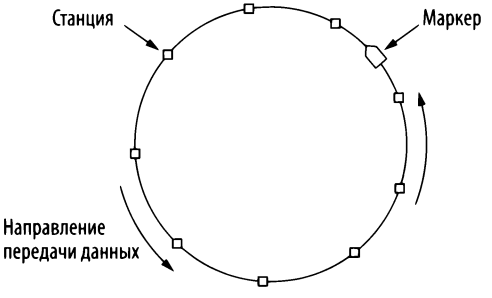
Если канал слабо загружен, его производительность легко вычислить. Накладные расходы на фрейм составляют $N$ бит, и при длине фрейма в $d$ бит производительность равна $d/(d + N)$.

При сильной загруженности канала, когда все станции хотят что-то передать, период подачи заявок из $N$ бит чередуется с $N$ фреймами. При этом накладные расходы на передачу одного фрейма составляют всего один бит, а производительность равна $d/(d + 1)$. Среднее время задержки для фрейма равно сумме времени ожидания в очереди на станции и дополнительных $(N - 1)d + N$ однобитовых интервалов, когда он попадет в начало своей внутренней очереди. Этот интервал указывает, как долго станции приходится ожидать завершения передачи всеми остальными станциями и очередного получения битовой карты.

### *Передача маркера*

Смысл протокола битовой карты в том, что он позволяет каждой станции передавать данные по очереди в предопределенном порядке. Другой способ добиться этого основан на передаче небольшого сообщения, **маркера (token)**, от одной станции к следующей в той же заранее заданной очередности. Маркер является разрешением на отправку. Допустим, у станции имеется поставленный в очередь фрейм, готовый к пересылке. Когда станция получает маркер, она имеет право отправить фрейм, прежде чем передавать маркер следующей станции. Если фреймов для отправки нет, то она просто передает маркер.

В протоколе **маркерного кольца (token ring)** для определения порядка, в котором станции отправляют данные, используется топология сети. Станции подключены одна к другой, образуя простое кольцо. Таким образом, передача маркера заключается в получении его с одной стороны и пересылке в противоположную, как показано на илл. 4.7. Фреймы передаются в том же направлении, что и маркер. Они передаются по кольцу, проходя по всем станциям, которые оказываются на их пути. Но чтобы фрейм не циркулировал вечно (как маркер), какая-то станция должна извлечь его из кольца. Это может быть либо




**Илл. 4.7.** Маркерное кольцо

первоначальный отправитель (если фрейм прошел полный цикл), либо станция-получатель.

Обратите внимание, что для реализации передачи маркера физическое кольцо не требуется. Достаточно иметь логическое кольцо, в котором каждая станция знает, какие станции находятся перед ней и после нее. Канал, соединяющий станции, может представлять собой одну длинную шину.

В этом случае каждая станция передает маркер по шине соседям в предопределенном порядке. Получив маркер, станция может использовать шину для отправки одного фрейма. Такой протокол называется **маркерной шиной (token bus)**. Он описан в стандарте IEEE 802.4, который оказался настолько неудачным, что был отменен институтом IEEE. Стандарты не вечны.

Производительность протокола с передачей маркера схожа с производительностью протокола с битовой картой, хотя периоды конкуренции и фреймы одного цикла здесь перемешаны. После отправки фрейма каждая станция должна подождать, пока все $N$ станций (включая ее саму) передадут маркер своим соседям и пока $N - 1$ станции отправят фреймы (если они имеются). Тонкое различие в том, что поскольку все позиции в цикле эквивалентны, никаких отклонений для сильно или слабо загруженных станций нет. Кроме того, в маркерном кольце каждая станция отправляет маркер только соседней станции, прежде чем протокол перейдет к следующему этапу. Маркер не должен посещать все станции, чтобы протокол продвинулся на шаг вперед.

Протоколы MAC на базе маркерных колец появляются с определенной периодичностью. Один из ранних протоколов такого рода — Token Ring, то есть «Маркерное кольцо», — описан в стандарте IEEE 802.5. В 1980-х он был популярен в качестве альтернативы классическому Ethernet. В 1990-е годы намного более быстрое маркерное кольцо, **интерфейс передачи распределенных данных по волоконно-оптическим каналам (Fiber Distributed Data Interface, FDDI)**, потерпело поражение от коммутируемого Ethernet. В 2000-х **отказоустойчивое пакетное кольцо (Resilient Packet Ring, RPR)** было описано в IEEE 802.17, чтобы стандартизировать множество вариантов кольцевых сетей, используемых провайдерами в городских условиях. Интересно, какие протоколы появятся после 2020 года.

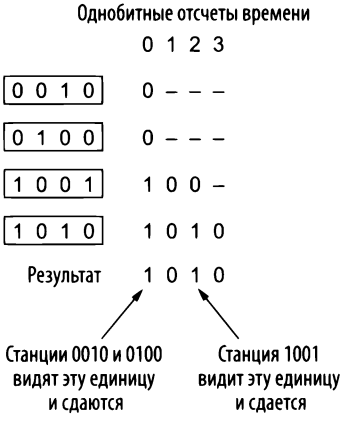
### *Двоичный обратный отсчет*

Недостатком двух представленных выше протоколов являются накладные расходы в один бит на станцию. Из-за этого они плохо масштабируются на большие сети с сотнями или даже тысячами станций. Можно добиться лучших результатов, используя двоичные адреса станций и канал, способный определенным образом комбинировать передаваемые данные. Станция, желающая занять канал, объявляет свой адрес в виде битовой строки, начиная со старшего бита. Предполагается, что все адреса станций содержат одинаковое количество битов. Будучи отправленными одновременно, биты адреса в каждой позиции логически складываются (логическое ИЛИ) средствами канала. Мы будем называть этот метод **протоколом с двоичным обратным отсчетом (binary countdown)**. Он использовался в сети Datakit (Фрейзер; Fraser, 1983). Подразумевается,

что задержки распространения сигнала пренебрежимо малы, поэтому станции слышат утверждаемые номера практически мгновенно.

Во избежание конфликтов следует применить правило арбитража: как только станция с 0 в старшем бите адреса видит, что в суммарном адресе этот 0 заменился единицей, она сдается и ждет следующего цикла. Например, если станции 0010, 0100, 1001 и 1010 конкурируют за канал, то в первом битовом интервале они передают биты 0, 0, 1 и 1 соответственно. В этом случае суммарный первый бит адреса будет равен 1. Следовательно, станции с номерами 0010 и 0100 считаются проигравшими, а станции 1001 и 1010 продолжают борьбу.

Следующий бит у обеих оставшихся станций равен 0 — таким образом, обе продолжают. Третий бит равен 1, поэтому станция 1001 сдается. Победителем оказывается станция 1010, так как ее адрес наибольший. Выиграв торги, она может начать передачу фрейма, после чего начнется новый цикл торгов. Схема протокола показана на илл. 4.8. Данный метод предполагает, что приоритет станции напрямую зависит от ее номера. В некоторых случаях такое жесткое правило может играть положительную, а порой — отрицательную роль.




**Илл. 4.8.** Протокол с двоичным обратным отсчетом. Прочерк означает молчание

Эффективность использования канала при этом методе составляет $d/(d + \log_2 N)$. Однако можно так хитро выбрать формат фрейма, что его первое поле будет содержать адрес отправителя. Тогда даже эти $\log_2 N$ бит не пропадут зря и эффективность составит 100 %.

Двоичный обратный отсчет является примером простого, элегантного и эффективного протокола. Разработчикам будущих сетей еще предстоит заново его открыть. Хочется надеяться, что когда-нибудь он займет свою нишу в сетевых технологиях.

## 4.2.4. Протоколы с ограниченной конкуренцией

Итак, мы рассмотрели две основные стратегии предоставления доступа к каналу в широковещательных сетях: конкуренцию как в CSMA и протоколы без коллизий. Каждую стратегию можно оценить по двум важным параметрам: времени

задержки при низкой загрузке канала и эффективности канала при большой загрузке. В условиях низкой загрузки протоколы с конкуренцией (то есть чистая или дискретная системы ALOHA) предпочтительнее, потому что время задержки (и число коллизий) в таких системах меньше. По мере роста загруженности канала эти системы становятся все менее привлекательными, поскольку возрастают накладные расходы, связанные с коллизиями. Для протоколов без коллизий справедливо обратное. При низкой нагрузке у них относительно высокое время задержки, но по мере роста загруженности эффективность использования канала возрастает (накладные расходы фиксированные).

Очевидно, что было бы неплохо объединить лучшие свойства обеих стратегий и получить протокол, использующий разные стратегии в зависимости от загруженности канала. Такие протоколы мы будем называть **протоколами с ограниченной конкуренцией (limited-contention protocols)** — они на самом деле существуют. На них мы завершим изучение сетей с контролем несущей.

До сих пор мы рассматривали только симметричные протоколы коллективного доступа, в которых каждая станция пытается получить доступ к каналу с равной вероятностью $p$. Интересно, что производительность всей системы может быть увеличена при использовании асимметричного протокола, в котором станциям назначаются различные вероятности.

Прежде чем приступить к асимметричным протоколам, давайте кратко рассмотрим производительность в симметричном случае. Предположим, что $k$ станций конкурируют за доступ к каналу. Вероятность передачи каждой станции в каждый интервал времени равна $p$. Шанс получения станцией доступа к каналу в данном слоте включает вероятность того, что любая станция передает данные (с вероятностью $p$), а остальные $k - 1$ станций воздерживаются от передачи (каждая с вероятностью $1 - p$). Итоговое значение равно $kp(1 - p)^{k-1}$. Чтобы найти оптимальное значение $p$, продифференцируем данное выражение по $p$, приравняем результат к нулю и решим полученное уравнение относительно $p$. В результате наилучшее значение $p$ равно $1/k$. Заменив в формуле $p$ на $1/k$, мы получим вероятность успеха при оптимальном значении $p$:

$$\Pr[\text{успех при оптимальной вероятности } p] = \left(\frac{k-1}{k}\right)^{k-1}.$$

Зависимость этой вероятности от количества готовых станций графически представлена на илл. 4.9. Для небольшого числа станций значение вероятности успеха неплохое, но как только количество станций достигает хотя бы пяти, вероятность снижается почти до асимптотической величины, равной $1/e$.

Из илл. 4.9 очевидно, что вероятность получения доступа к каналу для какой-либо станции можно увеличить, только снизив конкуренцию за канал. Этим занимаются протоколы с ограниченной конкуренцией. Сначала они делят все станции на группы (необязательно непересекающиеся). Состязаться за интервал 0 разрешается только членам группы 0. Если кто-то из них выигрывает, он получает канал и передает по нему фрейм. Если никто из них не хочет передавать или происходит коллизия, члены группы 1 состязаются за интервал 1, и т. д. При соответствующем разбиении на группы конкуренция за каждый слот

уменьшается, что повышает вероятность его успешного использования (см. левую часть графика).

**Илл. 4.9.** Вероятность получения доступа к каналу в симметричном протоколе

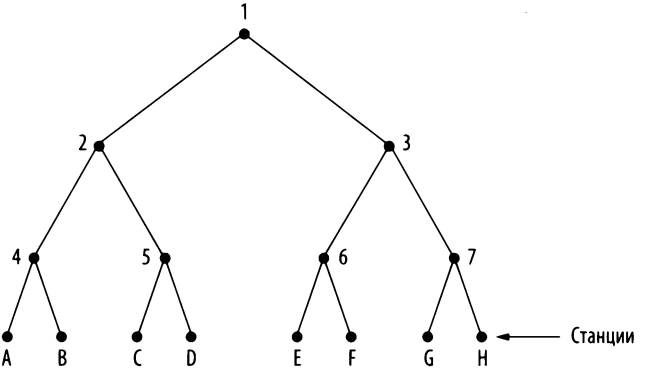
Вопрос в том, как разбивать станции на группы. Прежде чем обсуждать общий случай, рассмотрим несколько частных. Для начала возьмем крайний случай, когда каждая группа включает только одну станцию. Такое разбиение гарантирует полное отсутствие коллизий, так как на каждый интервал времени будет претендовать только один участник. Подобные протоколы уже рассматривались ранее (например, протокол с двоичным обратным отсчетом). Еще одним особым случаем является разбиение на группы, состоящие из двух станций. Вероятность того, что обе станции одновременно начнут передачу в течение одного интервала, равна $p^2$, и при малых значениях $p$ этим значением можно пренебречь. По мере увеличения количества станций в группах вероятность столкновений будет возрастать, однако длина битовой карты, необходимой, чтобы перенумеровать все группы, будет уменьшаться. Другим предельным случаем будет одна группа, в которую войдут все станции (то есть дискретная система ALOHA). Нам требуется механизм динамического разбиения станций, с небольшим количеством крупных групп при слабой загруженности канала и большом количестве мелких групп (может быть, даже из одной станции), когда нагрузка на канал высока.

### *Протокол адаптивного прохода по дереву*

Одним из простых способов динамического разбиения на группы является алгоритм, разработанный во время Второй мировой войны в армии США для проверки солдат на сифилис (Дорфман; Dorfman, 1943). У $N$ солдат брался анализ крови. Часть каждого образца помещалась в одну общую пробирку. Этот смешанный образец проверялся на наличие антител. Если антитела не обнаруживались, все солдаты в данной группе объявлялись здоровыми. В противном же случае группа делилась пополам, и каждая половина группы проверялась

отдельно. Подобный процесс продолжался до тех пор, пока размер группы не уменьшался до одного солдата.

В компьютерной версии данного алгоритма (Капетанакис; Capetanakis, 1979) станции рассматриваются в виде листьев двоичного дерева, как показано на илл. 4.10. В первом после успешной передачи периоде конкуренции (слот 0) могут участвовать все станции. Если одной из них удается получить канал, то на этом работа алгоритма заканчивается. В случае коллизии ко второму этапу (слот 1) допускается только половина станций, те, которые относятся к узлу 2 дерева. Если одна из этих станций успешно захватывает канал, то следующее состязание устраивается для второй половины станций (относящихся к узлу 3 дерева). Если же две или более станции узла 2 конфликтуют в слоте 1, то в конкуренции в слоте 2 участвуют станции узла 4.




**Илл. 4.10.** Дерево из восьми станций

Таким образом, если коллизия происходит во время слота 0, то все дерево сканируется на единичную глубину для поиска станций, готовых к передаче данных. Каждый однобитный слот ассоциируется с определенным узлом дерева. В случае коллизии поиск продолжается для левого и правого дочерних узлов. Если количество станций, претендующих на передачу, равно нулю или единице, поиск в данном узле дерева прекращается, поскольку все готовые станции уже обнаружены.

При сильной загруженности канала вряд ли стоит начинать поиск с узла 1 — маловероятно, что из всех станций претендовать на канал будет всего одна. По той же причине могут быть пропущены узлы 2 и 3. На каком уровне дерева следует запускать алгоритм в общем случае? Очевидно, что чем сильнее загруженность канала, тем более низкий уровень выбирается для начала поиска готовых станций. Мы будем предполагать, что каждая станция может точно оценить $q$ (количество готовых на данный момент станций), например, отслеживая недавний трафик.

Пронумеруем уровни дерева на илл. 4.10 — узел 1 на уровне 0, узлы 2 и 3 на уровне 1 и т. д. Обратите внимание, что каждый узел на уровне $i$ включает в себя $2^{-i}$ часть от всех нижележащих станций. Если $q$ распределяется равномерно, то ожидаемое число готовых станций ниже узла на уровне $i$ равно $2^{-i}q$. Вполне

очевидно, что оптимальным уровнем для начала поиска будет тот, на котором среднее число конкурирующих в интервале станций равно 1, то есть уровень, на котором $2^{-i}q = 1$. Отсюда получаем $i = \log_2 q$.

Были разработаны многочисленные усовершенствования базового алгоритма, — в частности, некоторые детали обсуждаются в работе Бертсекаса и Галлагера (Bertsekas and Gallager, 1992). Идея оказалась настолько удачной, что исследователи продолжают ее оптимизировать до сих пор (см. работу Де Марко и Ковальски; De Marco and Kowalski, 2017). Например, рассмотрим случай, при котором передавать хотят только станции *G* и *H*. На узле 1 произойдет коллизия, поэтому будет проверен узел 2. Он окажется пустым. Узел 3 проверять нет смысла, так как там гарантированно будет коллизия. (Нам известно, что под узлом 1 находятся две или более станции, а так как под узлом 2 нет ни одной станции, то все они должны быть под узлом 3.) Поэтому проверку узла 3 можно пропустить и сразу проверить узел 6. Поскольку под узлом 6 ничего не оказалось, то проверку узла 7 также можно пропустить и проверить узел *G*.

## 4.2.5. Протоколы беспроводных локальных сетей

Систему, состоящую из ноутбуков, взаимодействующих по радио, можно рассматривать как беспроводную локальную сеть — мы уже обсуждали это в разделе 1.4.3. Такая LAN — пример сети на базе широковещательного канала. Она имеет другие характеристики, нежели проводная LAN, поэтому здесь требуются специальные протоколы управления доступом к среде (MAC). В данном разделе мы познакомимся с некоторыми из них. Далее в разделе 4.4 мы подробно рассмотрим стандарт 802.11 (Wi-Fi).

Распространенная конфигурация беспроводных LAN подразумевает наличие офисного здания с заранее размещенными в нем точками доступа. Все точки соединены друг с другом медным проводом или оптоволоконным кабелем; они рассылают данные на пользовательские станции. Если мощность передатчиков точек доступа и ноутбуков настроена так, что диапазон приема составляет около десятка метров, то соседние комнаты становятся единой сотой. Тогда все здание превращается в большую сотовую систему, подобную мобильной телефонии, описанной в главе 2. В отличие от обычной сотовой системы, у каждой соты в данном случае всего один канал, работающий со всеми станциями, находящимися в нем, включая точку доступа. Обычно пропускная способность такого канала измеряется мегабитами или даже гигабитами в секунду. Теоретически стандарт IEEE 802.11ac обеспечивает пропускную способность до 7 Гбит/с, однако в реальности она гораздо ниже.

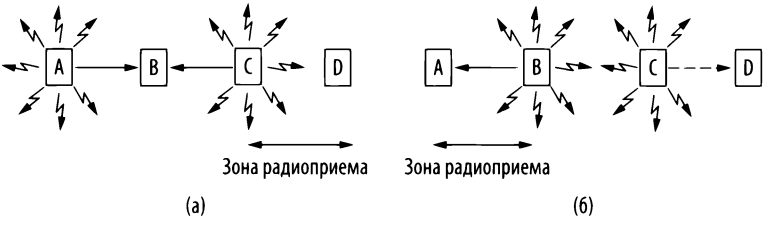
Мы уже упоминали, что обычно беспроводные системы не имеют возможности распознавать коллизии в тот момент, когда они происходят. Принимаемый сигнал на станции может быть очень слабым, возможно, в миллион раз слабее излучаемого. Обнаружить его так же трудно, как найти иголку в стоге сена. Для выявления уже случившихся коллизий и других ошибок используются подтверждения.

Есть еще одно очень важное различие между проводными и беспроводными LAN. В беспроводной сети у станций иногда нет возможности передавать или

получать фреймы с других станций из-за ограниченного диапазона радиопередачи. В проводных сетях, если одна станция отправляет фрейм, все остальные его получают. Эта особенность приводит к различным сложностям.

Для простоты мы допустим, что каждый передатчик работает в некой фиксированной области, которую можно представить как регион покрытия, имеющий форму круга. Внутри него другая станция может слышать и принимать данные с этой станции. Важно понимать, что на практике регион покрытия будет неправильной формы, так как распространение радиосигналов зависит от среды. Стены и другие препятствия, ослабляющие и отражающие сигналы, приводят к тому, что сила сигнала в разных направлениях меняется. Однако модель с окружностью для наших целей вполне подходит.

В беспроводных LAN можно просто попытаться применить протокол CSMA — прослушивать эфир и осуществлять передачу только тогда, когда он никем не занят. Но проблема в том, что в действительности имеет значение интерференция на приемнике, а не на передатчике, поэтому этот протокол не слишком подходит для беспроводных сетей. Чтобы наглядно увидеть суть проблемы, рассмотрим илл. 4.11, где показаны четыре беспроводные станции. В нашем случае не имеет значения, какая из них является точкой доступа, а какая — портативным компьютером. Мощность передатчиков такова, что взаимодействовать могут только соседние станции, то есть *A* может слышать *B*, *C* — *B* и *D* (но не *A*).




**Илл. 4.11.** Беспроводная локальная сеть. (а) A и C — скрытые станции во время пересылки данных на B. (б) B и C — засвеченные станции во время пересылки данных на A и D

Сначала рассмотрим, что происходит, когда станции *A* и *C* передают данные станции *B*, как изображено на илл. 4.11 (а). Если *A* отправляет данные, а *C* сразу же опрашивает канал, то она не будет слышать *A*, поскольку та расположена слишком далеко, и может прийти к неверному выводу о том, что канал свободен и можно посылать данные на станцию *B*. Если станция *C* начнет передачу, она будет конфликтовать со станцией *B* и исказит фрейм, передаваемый станцией *A*. (Мы предполагаем, что никакая схема по типу CDMA не используется для предоставления нескольких каналов, поэтому из-за коллизий сигналы искажаются и оба фрейма разрушаются.) Нам необходим MAC-протокол, который предотвратит коллизии такого рода, ведь это лишняя трата полосы пропускания. Проблема, при которой одна станция не слышит возможного конкурента, поскольку он расположен слишком далеко от нее, называется **проблемой скрытой станции (hidden terminal problem)**.

Теперь рассмотрим другую ситуацию: *B* передает данные *A* в то же время, когда *C* хочет начать передачу *D*, как показано на илл. 4.11 (б). Станция *C* при опросе канала слышит выполняемую передачу и может ошибочно предположить, что она не может передавать данные станции *D* (пунктирная стрелка на рисунке). В действительности такая передача создала бы помехи только в зоне от станции *B* до станции *C*, где в данный момент не ведется прием. Нам необходим MAC-протокол, который предотвратит такой тип задержек, ведь это лишняя трата полосы пропускания. Такая ситуация называется **проблемой засвеченной станции (exposed terminal problem)**.

Сложность в том, что перед тем, как начать передачу, станции необходимо знать, есть ли какая-нибудь активность в радиодиапазоне вблизи приемника. Протокол CSMA может лишь сообщить об активности вокруг передатчика путем опрашивания несущей. В случае проводной передачи все сигналы достигают всех станций, поэтому такой проблемы нет. При этом во всей системе одновременно только одна станция может вести передачу. В системе на основе радиосвязи ближнего действия несколько станций могут передавать данные одновременно, если принимающие станции находятся достаточно далеко. Нам нужно, чтобы даже в растущей соте одновременная передача не прекращалась. Так, гости на вечеринке не ждут, пока все замолчат, чтобы начать говорить, — в большом помещении может происходить несколько разговоров одновременно (если только все гости не пытаются пообщаться с одним и тем же собеседником).

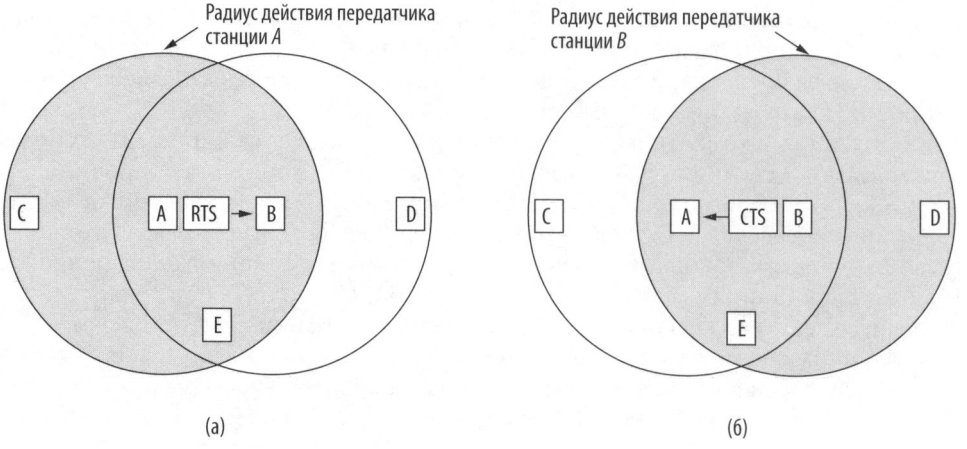
Одним из первых значительных протоколов, разработанных для беспроводных LAN и умеющих справляться с этими проблемами, является протокол **MACA (Multiple Access with Collision Avoidance — множественный доступ с предотвращением коллизий)** (Карн; Karn, 1990; Гарсиа-Луна-Асевес; Garcia-Luna-Aceves, 2017). Идея, лежащая в основе этого протокола, заключается в том, что отправитель заставляет получателя передать короткий фрейм. Соседние станции слышат эту передачу и воздерживаются от действий на время, требуемое для приема большого фрейма данных. Этот метод заменяет контроль несущей.

Протокол MACA проиллюстрирован на илл. 4.12. Рассмотрим ситуацию, в которой станция *A* передает станции *B*. Станция *A* начинает с того, что посылает станции *B* фрейм **RTS (Request To Send — запрос на передачу)**, как показано на илл. 4.12 (а). Этот короткий фрейм (30 байт) содержит длину фрейма данных, который последует за ним. Затем *B* отвечает фреймом **CTS (Clear To Send — разрешение передачи)**, см. илл. 4.12 (б). Он также содержит длину фрейма данных (скопированную из фрейма RTS). Приняв фрейм CTS, *A* начинает передачу.

Теперь выясним, как реагируют станции, фиксирующие передачу одного из этих фреймов. Любая станция, которая фиксирует RTS, находится близко к станции *A* и поэтому должна молчать, пока та не примет CTS. Станции, слышащие CTS, находятся рядом с *B*, следовательно, должны воздержаться от передачи, пока станция *B* не получит фрейм данных (его длину они могут узнать из CTS).

На илл. 4.12 станция *C* находится в зоне станции *A*, но не входит в зону станции *B*. Поэтому она фиксирует RTS от *A* (но не CTS от *B*). Поскольку она не интерферирует с CTS, она не обязана воздерживаться от передачи в то время, пока пересылается фрейм с данными. *D*, напротив, находится близко от *B*, но далеко от *A*. Она не слышит RTS, но фиксирует CTS, а это означает, что она

находится вблизи станции, собирающейся принять фрейм с данными. Поэтому ей нельзя вести передачу, пока этот фрейм не будет получен. *E* слышит оба управляющих сообщения и так же, как и *D*, должна хранить молчание, пока не будет завершена передача фрейма данных.




**Илл. 4.12.** Протокол MACA. (а) Станция A посылает фрейм RTS станции B. (б) станция B отвечает фреймом CTS станции A

Несмотря на все меры предосторожности, коллизии все равно могут произойти. Например, станции *B* и *C* могут одновременно послать RTS станции *A*. Возникнет коллизия, и фреймы не будут приняты. В этом случае передатчики, не услышав CTS в установленный срок, ждут случайное время и повторяют попытку.

## 4.3. СЕТЬ ETHERNET

Теперь, когда мы рассмотрели общие вопросы, касающиеся протоколов распределения канала, пришло время обсудить их практическое применение в реальных системах. Большое число технологий для персональных (PAN), локальных (LAN) и общегородских (MAN) сетей описано в серии стандартов IEEE 802. Некоторые из них используются до сих пор, но многие утратили актуальность, как показано на илл. 1.37. Те, кто верит в реинкарнацию, считают, что один из членов Ассоциации стандартов IEEE — вновь родившийся Чарльз Дарвин, отбраковывающий слабые технологии. И действительно, выжили сильнейшие: среди них наиболее важными являются стандарты 802.3 (Ethernet) и 802.11 (беспроводные LAN). Технология Bluetooth (беспроводные PAN) применяется очень широко, но ее описывают другие стандарты помимо 802.15.

Мы начнем изучение реальных систем с Ethernet, вероятно, наиболее распространенной в мире технологии для объединения устройств в сеть. Существует два типа Ethernet: **классический Ethernet (classic Ethernet)**, который решает

проблему множественного доступа с помощью методов, представленных в этой главе; и **коммутируемый Ethernet (switched Ethernet)**, в котором для соединения компьютеров используются **коммутаторы (switches)**. Важно понимать, что хотя в обоих названиях присутствует слово Ethernet, между этими технологиями много различий. Классический Ethernet — изначальный вариант, достигавший скорости 3–10 Мбит/с. Коммутируемый Ethernet — результат развития классического. Различают быстрый, гигабитный, 10-гигабитный, 40-гигабитный и 100-гигабитный коммутируемый Ethernet; он работает на скоростях 100, 1000, 10 000, 40 000 или 100 000 Мбит/с соответственно. На сегодняшний день используется только такой тип Ethernet.

Мы рассмотрим этапы развития Ethernet в хронологическом порядке. Так как Ethernet и IEEE 802.3 — это одно и то же (за исключением двух небольших деталей, кратко описанных ниже), оба названия употребляются равнозначно, в том числе и в этой книге. Дополнительную информацию, касающуюся Ethernet, можно найти в книге Сперджена и Циммермана (Spurgeon and Zimmerman, 2014).

## 4.3.1. Физический уровень классического Ethernet

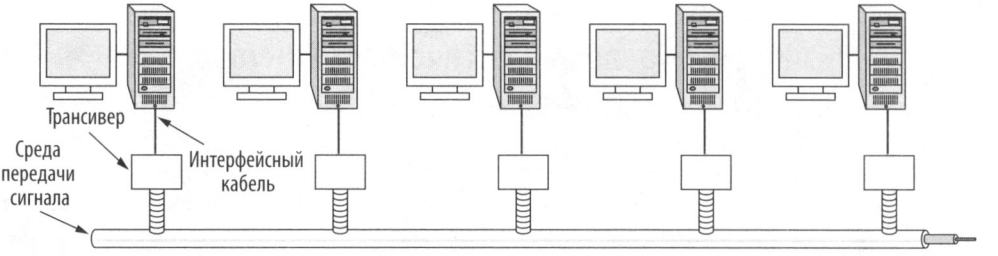
История Ethernet начинается приблизительно во времена системы ALOHA, когда студент Боб Меткалф получил магистерскую степень в Массачусетском технологическом университете. Позже он защитил докторскую в Гарварде, где ознакомился с наработками Абрамсона по системе ALOHA. Эта тема так заинтересовала Меткалфа, что после выпуска из Гарварда он решил провести лето на Гавайях, работая с Абрамсоном. После он перешел в исследовательский центр Xerox, где стал свидетелем разработки и создания устройств, которые впоследствии назовут «персональными компьютерами». Однако эти устройства были изолированы друг от друга. Используя результаты исследований Абрамсона, Меткалф вместе со своим коллегой Дэвидом Боггсом разработал и реализовал первую локальную сеть (Metcalfe and Boggs, 1976). В ней использовался длинный толстый коаксиальный кабель, а пропускная способность составляла 3 Мбит/с.

Система получила название **«Ethernet»**, в честь *люминофорного эфира*, через который, как когда-то считалось, распространяются электромагнитные лучи. (В XIX веке британский физик Джеймс Клерк Максвелл обнаружил, что электромагнитное излучение можно описать волновым уравнением. Ученые предположили, что пространство должно быть заполнено некой эфирной средой, по которой излучение распространяется. Только после знаменитого эксперимента Майкельсона — Морли в 1887 году физики поняли, что оно способно распространяться в вакууме.)

Система Xerox Ethernet оказалась настолько успешной, что в 1978 году DEC Intel и Xerox разработали стандарт 10-мегабитного Ethernet — **стандарт DIX (DIX standard)**. С небольшими изменениями в 1983 году DIX превратился в стандарт IEEE 802.3. К несчастью, у компании Xerox на тот момент была длинная история значительных изобретений (например, персональный компьютер), которые она не смогла успешно выпустить на рынок. Об этом рассказано в книге «Fumbling the Future» Смита и Александера (Smith and Alexander, 1988). Когда

стало понятно, что Xerox не заинтересован в Ethernet и может предложить разве что помощь в его стандартизации, Меткалф основал собственную компанию, 3Com. Компания занималась продажей адаптеров Ethernet для персональных компьютеров. Были проданы миллионы таких устройств.

Классический Ethernet — это проходящий по всему зданию длинный кабель, к которому подключаются компьютеры. Эта архитектура показана на илл. 4.13. Первый вариант, называемый в народе **толстым Ethernet (thick Ethernet)**, напоминал желтый садовый шланг с маркировкой каждые 2,5 м — в этих местах подключались компьютеры. (По стандарту 802.3 *не требовалось*, чтобы кабель был желтым, но это *подразумевалось*.) Ему на смену пришел **тонкий Ethernet (thin Ethernet)**; эти кабели лучше гнулись, а соединения выполнялись с помощью стандартных разъемов BNC. Тонкий Ethernet был намного дешевле и проще в установке, но длина сегмента не превышала 185 м (вместо 500 м для толстого Ethernet), а каждый сегмент поддерживал не более 30 компьютеров (вместо 100).




**Илл. 4.13.** Архитектура классического Ethernet

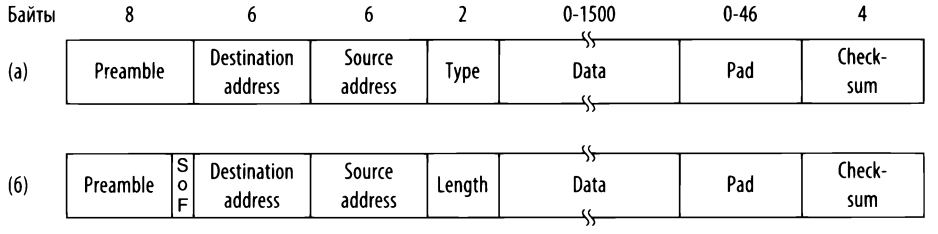
Все версии Ethernet имеют ограничения по длине кабеля в сегменте, то есть участкам без усилителя. Для построения сетей больших размеров несколько кабелей соединяются **повторителями (repeaters)**. Повторитель — это устройство физического уровня. Он принимает, усиливает (регенерирует) и передает сигналы в обоих направлениях. С точки зрения программного обеспечения ряд кабелей, соединенных таким образом, не отличается от сплошного кабеля (за исключением небольшой временной задержки, связанной с повторителями).

Информация по этим кабелям передается с использованием манчестерского кода, о котором мы говорили в разделе 2.4.3. Ethernet может состоять из большого количества сегментов кабеля и повторителей, однако два трансивера могут располагаться на расстоянии не более 2,5 км и между ними должно быть не более четырех повторителей. Эти ограничения нужны для того, чтобы протокол MAC, о котором мы поговорим далее, работал корректно.

## 4.3.2. Протокол MAC в классическом Ethernet

Формат фрейма, применяемый для отправки данных, показан на илл. 4.14. Сначала идет поле `Preamble` (Преамбула) длиной 8 байт, которое содержит последовательность 10101010 (за исключением последнего байта, в котором значения

двух последних битов равны 11). Последний байт в стандарте 802.3 называется разделителем `Start of Frame`, `SoF` (начало фрейма). Манчестерское кодирование такой последовательности битов в результате дает меандр с частотой 10 МГц и длительностью 6,4 мкс. Это позволяет получателю синхронизировать свой таймер с таймером отправителя. Два последних бита 11 сообщают получателю, что сейчас начнется передача остальной части фрейма.




**Илл. 4.14.** Форматы фреймов. (a) DIX Ethernet; (б) IEEE 802.3

Затем следуют два адресных поля: `Destination address` (Адрес получателя) и `Source address` (Адрес отправителя). Каждый занимает по 6 байт. Первый передаваемый бит адреса получателя содержит 0 для обычных адресов и 1 для групповых. Групповые адреса позволяют нескольким станциям принимать информацию от одного отправителя. Фрейм, отправляемый групповому адресату, может быть получен всеми станциями, входящими в эту группу. Этот механизм называется **групповой рассылкой (multicasting)**. Если адрес состоит только из единиц, то фрейм могут принять абсолютно все станции сети. Таким способом осуществляется **широковещание (broadcasting)**. Групповая рассылка более избирательна и предусматривает управление группами, чтобы определять, какие станции в них входят. При широковещании, напротив, никакой разницы между станциями нет, поэтому управление группами не требуется.

Интересной особенностью исходных адресов станций является их абсолютная уникальность. Они централизованно назначаются IEEE, и это гарантирует, что нигде в мире нет двух станций с одинаковым адресом. Идея заключается в том, что каждая станция может быть однозначно идентифицирована по ее 48-битному номеру. Для этого первые 3 байта поля адреса используются для **уникального идентификатора организации (Organizationally Unique Identifier, OUI)**. Значения этого поля определяются IEEE и являются индикатором производителей (каждый из них получает блок из $2^{24}$ адресов). Производитель назначает последние 3 байта адреса и программирует весь адрес в сетевой карте перед тем, как она поступает в продажу.

Затем следует поле `Type` (Тип) или `Length` (Длина), в зависимости от того, к какому стандарту относится фрейм – Ethernet или IEEE 802.3. В Ethernet поле `Type` показывает получателю, что делать с фреймом. Дело в том, что одновременно на одном и том же компьютере может работать несколько протоколов сетевого уровня. Поэтому операционная система должна понимать, какому протоколу передать полученный фрейм Ethernet. Поле `Type` определяет процесс,

для которого предназначается фрейм. Например, код типа 0x0800 означает, что данные содержат пакет IPv4.

Создатели IEEE 802.3 благоразумно решили, что в этом поле должна передаваться длина фрейма. Для определения этого показателя в Ethernet необходимо было заглянуть внутрь данных, а это нарушение правил использования сетевых уровней. Разумеется, это означало, что получатель никак не мог выяснить, что же ему делать с входящим фреймом в IEEE 802.3. Эту проблему решили путем добавления в данные еще одного заголовка для протокола управления логическим каналом (Logical Link Control, LLC), который мы рассмотрим позже. Он занимает 8 байт и передает 2 байта информации о типе протокола.

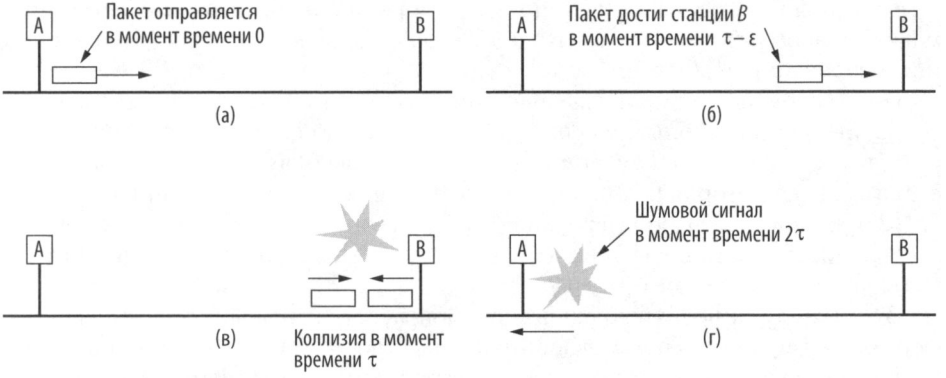
К сожалению, к моменту публикации стандарта IEEE 802.3 использовалось так много оборудования и программного обеспечения для DIX Ethernet, что многие производители и пользователи были не в восторге от необходимости реорганизации полей `Type` и `Length`. В 1997 году IEEE признал поражение и допустил оба способа. К счастью, значения всех полей `Type`, использовавшиеся до 1997 года, были больше 1500 (тогда это был максимальный размер данных). Теперь правило таково, что любое значение меньше 0x600 (1536) можно интерпретировать как `Length`, а больше 0x600 — как `Type`. Теперь в IEEE спокойны: все используют их стандарт, при этом продолжая делать то же, что и раньше, не утруждая себя мыслями о протоколе LLC и не чувствуя вины за нарушение стандарта. Вот что происходит, когда политика (в данном случае отраслевая) и технологии идут рука об руку.

Наконец, за полем `Type` следует поле `Data` (Данные), размер которого ограничен 1500 байтами. При официальном закреплении стандарта Ethernet это число было выбрано, в общем-то, произвольно. Основной причиной послужило то, что трансиверу нужно довольно много оперативной памяти для хранения всего фрейма, а память в далеком 1978 году еще была очень дорогой. Соответственно, увеличение верхней границы размера поля данных привело бы к необходимости установки большего объема памяти и удорожанию всего трансивера.

Между тем кроме верхней границы размера поля данных очень важна и нижняя. Поле данных, содержащее 0 байт, вызывает определенные проблемы. Дело в том, что когда трансивер обнаруживает коллизию, он обрезает текущий фрейм, а это означает, что отдельные куски фреймов постоянно блуждают по кабелю. Чтобы легче отличать нормальные фреймы от мусора, сети Ethernet требуется фрейм размером не менее 64 байт (от поля адреса получателя до поля контрольной суммы включительно). Если во фрейме содержится меньше 46 байт данных, в него вставляется специальное поле `Pad` (Заполняющие биты), с помощью которого его размер доводится до необходимого минимума.

Есть и другая (и даже более важная) цель установки нижней границы размера фрейма: предотвращение ситуации, когда станция успевает передать короткий фрейм раньше, чем его первый бит дойдет до самого дальнего конца кабеля, где он может столкнуться с другим фреймом. Эта ситуация показана на илл. 4.15. В момент времени 0 станция *A* на одном конце сети посылает фрейм. Пусть время прохождения фрейма по кабелю равно $\tau$. За мгновение до того, как он достигнет конца кабеля (то есть в момент времени $\tau - \varepsilon$), самая дальняя станция *B* начинает передачу. Когда *B* замечает, что получает бóльшую мощность, нежели

передает сама, она понимает, что произошла коллизия. Тогда она прекращает передачу и выдает 48-битный шумовой сигнал, предупреждающий остальные станции. Примерно в момент времени 2τ отправитель замечает шумовой сигнал и также прекращает передачу, затем выжидает случайное время и пытается возобновить ее.




**Илл. 4.15.** Обнаружение коллизии может занять 2τ

Если размер фрейма слишком маленький, не исключено, что отправитель закончит передачу прежде, чем получит шумовой сигнал в момент 2τ. В этом случае он может ошибочно предположить, что его фрейм был успешно принят. Для предотвращения этой ситуации все фреймы должны быть такой длины, чтобы время их передачи было больше 2τ. В LAN со скоростью передачи 10 Мбит/с при максимальной длине кабеля 2500 м и наличии четырех повторителей (требование спецификации 802.3) время передачи одного фрейма должно составлять в худшем случае около 50 мкс. Следовательно, длина фрейма должна быть такой, чтобы время передачи было не меньше этого минимума. При скорости 10 Мбит/с на передачу одного бита тратится 100 нс, значит, минимальный размер фрейма должен быть равен 500 бит. Из соображений надежности это число было увеличено до 512 бит, или 64 байт.

Последнее поле фрейма — `Checksum` (Контрольная сумма). По сути, это 32-битный код CRC того же типа, какой мы обсуждали в разделе 3.2. Если точнее, он определяется при помощи того же порождающего многочлена, что используется для PPP, ADSL и других типов каналов. Этот CRC позволяет выявлять ошибки: он проверяет, правильно ли приняты биты фрейма. Исправления не происходит — при обнаружении ошибки фрейм удаляется.

### *CSMA/CD с алгоритмом двоичной экспоненциальной выдержки*

В классическом Ethernet используется алгоритм CSMA/CD с настойчивостью 1, который мы рассматривали в разделе 4.2. Это означает, что станция прослушивает среду передачи, когда у нее появляется фрейм для отправки, и передает данные, если канал освобождается. Затем она проверяет, не произошла ли коллизия. Если

это случилось, станция прерывает передачу, посылая короткий сигнал о наличии коллизии, и повторяет отправку данных через случайный интервал времени.

На примере модели на илл. 4.5 рассмотрим, как определяется случайная длина интервала ожидания после коллизии, так как это новый метод. Когда возникает проблема, время делится на дискретные слоты. Их длительность равна максимальному времени обращения сигнала (то есть его прохождения по кабелю туда и обратно) — 2τ. Для удовлетворения потребностей Ethernet при максимальном размере сети необходимо, чтобы один слот составлял 512 битовых интервалов, или 51,2 мкс.

После первой коллизии каждая станция ждет или 0, или 1 слот, прежде чем снова предпринять попытку передачи. Если после коллизии две станции выберут одно и то же псевдослучайное число, они снова будут конфликтовать друг с другом. После второй коллизии каждая станция выбирает случайным образом 0, 1, 2 или 3 слота из набора и ждет снова. При возникновении третьей коллизии (вероятность такого события после предыдущих двух равна 1/4) слоты будут выбираться в диапазоне от 0 до $2^3 - 1$.

В общем случае после $i$ столкновений выбирается случайный номер в диапазоне от 0 до $2^i - 1$ и станция пропускает это количество слотов. Но после 10 коллизий подряд интервал рандомизации фиксируется на отметке 1023 слота. После 16 коллизий подряд контроллер признает свое поражение и возвращает компьютеру ошибку. Дальнейшим восстановлением занимаются более высокие уровни.

Это **алгоритм двоичной экспоненциальной выдержки (binary exponential backoff)**. Он был выбран для динамического учета количества станций, пытающихся осуществить передачу. Если выбрать интервал рандомизации равным 1023, то вероятность повторной коллизии будет пренебрежимо мала, но среднее время ожидания составит сотни слотов, в результате среднее время задержки будет слишком велико. С другой стороны, если каждая станция будет выбирать время ожидания всего из двух вариантов, 0 и 1, то в случае коллизии сотни станций они будут продолжать конфликтовать вновь и вновь до тех пор, пока 99 из них не выберут 1, а одна станция — 0. Это может длиться годами. Алгоритм экспоненциально увеличивает интервал рандомизации по мере возникновения повторных коллизий. Тем самым он обеспечивает низкую задержку при коллизии небольшого числа станций и одновременно гарантирует, что при коллизии большого числа станций проблема будет решена за разумное количество времени.

Если коллизии не произошло, отправитель предполагает, что фрейм успешно доставлен. Таким образом, ни в CSMA/CD, ни в Ethernet подтверждения не применяются. Такой вариант подходит для кабельных и оптоволоконных каналов с низким числом ошибок. Они распознаются с помощью кода CRC и исправляются более высокими уровнями. Как мы увидим далее, в зашумленных беспроводных каналах подтверждения используются.

## 4.3.3. Производительность Ethernet

Оценим производительность классического Ethernet в условиях большой постоянной загрузки, то есть когда $k$ станций постоянно готовы к передаче. Строгий анализ алгоритма двоичной экспоненциальной выдержки довольно

сложен. В качестве альтернативы мы предположим (согласно Меткалфу и Боггсу (Metcalfe and Boggs, 1976)), что вероятность повторной передачи в каждом слоте постоянна. Если каждая станция осуществляет передачу в слоте конкуренции с вероятностью $p$, то вероятность того, что одной из них удастся завладеть каналом ($A$), равна

$$A = kp(1 - p)k^{-1}.$$

Значение $A$ будет максимальным, когда $p = 1/k$. При $k$, стремящемся к бесконечности, $A$ будет стремиться к $1/e$. Вероятность того, что период соревнования за канал будет состоять ровно из $j$ слотов, равна $A(1 - A)^{j-1}$, следовательно, среднее число слотов за период конкуренции равно

$$\sum_{j=0}^{\infty} jA(1-A)^{j-1} = \frac{1}{A}.$$

Так как длительность каждого слота равна $2\tau$, средняя продолжительность периода конкуренции $w$ составит $2\tau/A$. При оптимальном значении вероятности $p$ среднее количество слотов конкуренции никогда не превысит $e$; таким образом, $w$ будет равна $2\tau e \approx 5{,}4\tau$.

Если среднее время передачи фрейма составляет $P$ секунд, то эффективность канала при его сильной загруженности будет равна:

$$\text{Эффективность канала} = \frac{P}{P + 2\tau/A}. \tag{4.2}$$

Из этой формулы мы видим, как максимальная длина кабеля влияет на производительность. Чем длиннее кабель, тем дольше период конкуренции за канал. Становится понятно, почему стандарт Ethernet накладывает ограничение на максимальное расстояние между станциями.

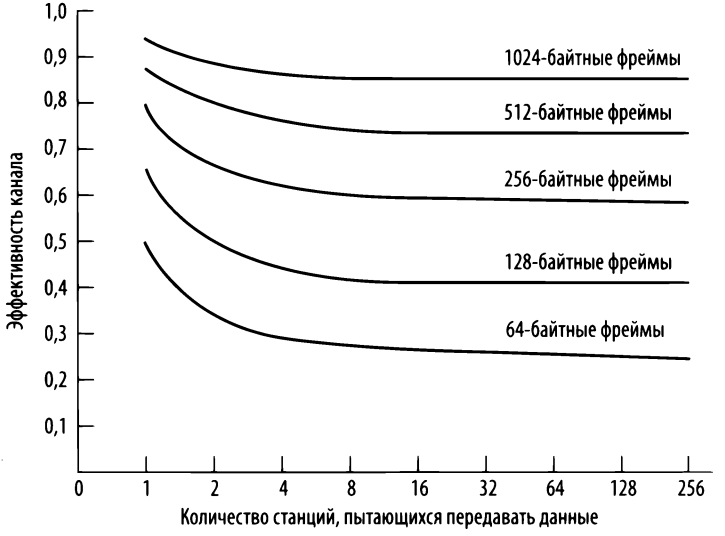
Будет полезно переформулировать уравнение (4.2) в терминах длины фрейма $F$, пропускной способности сети $B$, длины кабеля $L$ и скорости распространения сигнала $c$ для оптимального случая: $e$ слотов конкуренции на фрейм. При $P = F/B$ уравнение (4.2) примет вид:

$$\text{Эффективность канала} = \frac{1}{1 + 2BLe/cF}. \tag{4.3}$$

Если второе слагаемое делителя велико, эффективность сети будет низкой. В частности, увеличение пропускной способности или размеров сети (произведение $BL$) уменьшит эффективность при заданном размере фрейма. К сожалению, основные исследования в области сетевого оборудования нацелены как раз на увеличение этого произведения. Пользователи хотят высокой скорости при больших расстояниях (что обеспечивают, например, оптоволоконные MAN), следовательно, в таких случаях стандарт Ethernet будет не лучшим решением. Другие реализации Ethernet мы увидим в следующем разделе.

На илл. 4.16 показана зависимость эффективности канала от числа готовых станций при $2\tau = 51{,}2$ мкс и скорости передачи данных 10 Мбит/с. Для расчетов используется уравнение (4.3). При 64-байтном временном слоте 64-байтные

фреймы оказываются неэффективными, и это неудивительно. С другой стороны, при длине фреймов 1024 байта и при асимптотическом значении $e$ 64-байтных слотов на период конкуренции этот период равен 174 байтам, а эффективность канала составит 85 %. Этот результат намного лучше, чем 37 % в дискретной системе ALOHA.



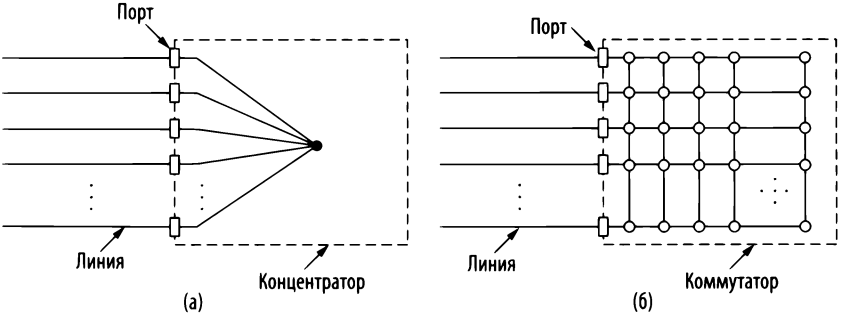

**Илл. 4.16.** Эффективность Ethernet на скорости 10 Мбит/с, 512-битные временные слоты

Теоретическому анализу производительности Ethernet (и других стандартов) было посвящено много работ. Большинство результатов следует воспринимать с долей (или даже тонной) скептицизма по двум причинам. Прежде всего, практически во всех этих теоретических исследованиях предполагается, что трафик подчиняется пуассоновскому распределению. Когда же ученые рассмотрели реальные потоки данных, они обнаружили, что сетевой трафик редко распределен по Пуассону и часто включает множество пиков; см. Паксон и Флойд (Paxson and Floyd, 1995); Фонтюнь и др. (Fontugne et al., 2017). Это означает, что при увеличении периода усреднения трафик не сглаживается. Помимо использования сомнительных моделей, многие из этих работ фокусируются на «интересных» случаях невероятно высокой загрузки канала. Боггс и др. (Boggs et al., 1988) на практике доказали, что Ethernet хорошо работает в реальных условиях, даже когда загрузка относительно высока.

## 4.3.4. Коммутируемый Ethernet

Очень скоро Ethernet стал отходить от архитектуры с одним длинным кабелем, которая использовалась в классическом варианте. Проблема поиска обрывов или ведущих в пустоту соединений привела к новому способу подключения,

в котором каждая станция соединяется с центральным **концентратором (hub)** отдельным кабелем. Концентратор просто соединяет все провода в электрическую схему, как если бы они были спаяны вместе. Такая конфигурация показана на илл. 4.17 (а).




**Илл. 4.17.** Конфигурация Ethernet. (а) Концентратор; (б) Коммутатор

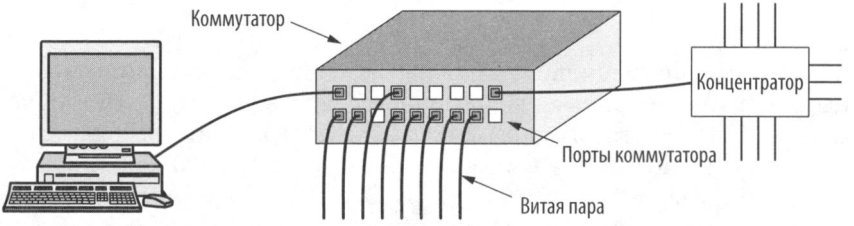
Для соединения применялись витые пары — они и так уже были проложены телефонными компаниями в большинстве офисных зданий. Повторное использование было весьма выгодным, но максимальная длина кабеля между компьютером и концентратором была ограничена до 100 м (или 200 м при условии качественной витой пары категории 5). В подобной конфигурации было легко удалять и добавлять станции, а также находить разрывы кабеля. Благодаря преимуществам использования существующей кабельной разводки и простоте обслуживания концентраторы на витой паре вскоре стали ведущей формой реализации сетей Ethernet.

Однако концентраторы не увеличивают пропускную способность, так как логически они эквивалентны одному длинному кабелю классической сети Ethernet. При добавлении станций доля каждой из них в общей фиксированной емкости канала уменьшается. Рано или поздно локальная сеть переполнится. Чтобы этого избежать, можно увеличить скорость передачи данных — например, с 10 на 100 Мбит/с, 1 Гбит/с или даже больше. Однако с ростом объемов мультимедийных данных и мощности серверов даже гигабитные версии Ethernet перестанут справляться.

К счастью, существует другой способ решения проблемы высокой загрузки — коммутируемая сеть Ethernet. Основой системы является **коммутатор (switch)**, который включает высокоскоростную плату, объединяющую все порты (илл. 4.17 (б)). Внешне коммутатор ничем не отличается от концентратора. Оба представляют собой обычные коробки, оборудованные несколькими (от 4 до 48) стандартными разъемами RJ-45 для подключения витой пары. Каждый кабель соединяет коммутатор или концентратор с одним компьютером, как показано на илл. 4.18. У коммутатора есть все преимущества концентратора. Новую станцию легко добавить или удалить, подключив или отключив один провод. Большинство сбоев кабеля или портов легко выявляется по неправильной работе всего

лишь одной станции. Общий компонент все же может подвести систему — речь идет о самом коммутаторе, — но если сеть пропадет на всех станциях, инженеры сразу поймут, в чем дело, и заменят устройство.

Однако внутри коммутатор и концентратор существенно различаются. Коммутаторы отдают фреймы только на порты, для которых те предназначены. Когда со станции на порт коммутатора приходит фрейм Ethernet, коммутатор проверяет адреса Ethernet и узнает, на какой порт этот фрейм нужно отдать. Для данного шага требуется, чтобы устройство могло сопоставлять номера портов и адреса. Этот процесс мы обсудим в разделе 4.8, где будет рассматриваться общий случай соединения нескольких коммутаторов друг с другом. Пока что предположим, что коммутатор знает порт получателя фрейма. Он пересылает фрейм на порт получателя через высокоскоростную плату. Скорость платы составляет несколько гигабит в секунду, а используемый протокол стандартизировать не требуется, так как он не выходит за пределы коммутатора. Затем порт получателя отправляет фрейм станции назначения по соединяющему их проводу. Другие порты об этом фрейме даже не подозревают.




**Илл. 4.18.** Коммутатор Ethernet

Что произойдет, если два компьютера или два порта станут передавать фреймы одновременно? Как мы помним, поведение коммутаторов отличается от концентраторов. Внутри концентратора все станции находятся в одной и той же **области коллизий (collision domain)**. Для планирования пересылки фреймов требуется алгоритм CSMA/CD. У коммутатора каждый порт находится в своей области коллизий. Обычно передача по кабелю осуществляется в дуплексном режиме, а значит, и станция, и порт могут одновременно отправлять фреймы, не беспокоясь о других станциях и портах. Коллизии при этом невозможны, и CSMA/CD не нужен. Однако если кабель полудуплексный, то станция и порт должны договариваться о передаче с помощью обычного CSMA/CD.

Что касается производительности, у коммутатора два преимущества перед концентратором. Во-первых, поскольку коллизии отсутствуют, пропускная способность используется более эффективно. Во-вторых, что еще более важно, благодаря коммутатору разные станции могут посылать фреймы одновременно. Достигнув портов коммутатора, они перейдут по внутренней плате устройства на правильные выходные порты. Но так как на один выходной порт может быть одновременно отправлено два фрейма, внутри коммутатора должен быть буфер для их временного хранения, если моментальная доставка на выходной порт невозможна. В целом эти усовершенствования дают большое преимущество

в производительности по сравнению с концентратором. Общую пропускную способность системы можно увеличить на порядок, в зависимости от числа портов и схем пересылки трафика.

Изменения в технологии портов, на которые пересылаются фреймы, также дают преимущества, связанные с безопасностью. Большинство интерфейсов LAN (сетевых адаптеров) могут работать в **«неразборчивом режиме» (promiscuous mode)**, когда *все* фреймы передаются на все компьютеры, а не только адресату. При использовании концентратора каждый подключенный к нему компьютер может видеть трафик между всеми остальными устройствами (что очень радует мошенников). Коммутатор передает трафик только на порты адресатов. Это обеспечивает лучшую изоляцию и защиту от утечки данных: трафик не попадет в чужие руки. Однако если вопрос безопасности в организации стоит очень серьезно, в дополнение к этому лучше применять шифрование.

Поскольку коммутатор ожидает фреймы Ethernet на каждом входном порте, некоторые из этих портов можно использовать в качестве концентраторов. На илл. 4.18 порт в правом верхнем углу соединен не со станцией, а с 12-портовым концентратором. Полученные концентратором фреймы конкурируют как обычно, с коллизиями и двоичной выдержкой. Победители попадают в коммутатор через концентратор и подвергаются там той же процедуре, что и все остальные входящие фреймы. Коммутатор не знает о том, что им пришлось с боем прорываться к нему. Он переправляет их на нужные выходные линии через высокоскоростную системную плату. Возможна ситуация, когда адресатом является одна из линий, подключенных к концентратору; это означает, что фрейм уже был доставлен, так что коммутатор удаляет его. В современных сетях в основном применяется коммутируемый Ethernet. Тем не менее устаревшие концентраторы все еще встречаются.

## 4.3.5. Fast Ethernet

Одновременно с широким распространением коммутаторов скорость Ethernet 10 Мбит/с перестала быть чем-то необычным. Поначалу казалось, что 10 Мбит/с — это просто фантастически высокая скорость. Примерно так же ощущался переход с 56-килобитных телефонных модемов на кабельные. Однако мир меняется очень быстро. Известный закон Паркинсона («Работа занимает все отведенное на нее время») можно перефразировать так: «Данные занимают всю предоставленную пропускную способность канала».

Многим приложениям требовалась высокая пропускная способность, и поэтому появились 10-мегабитные LAN, связанные лабиринтами повторителей, концентраторов и коммутаторов. Сетевым администраторам иногда казалось, что система едва держится и может развалиться от любого прикосновения. Но даже с коммутаторами Ethernet максимальная полоса пропускания одного компьютера ограничивалась кабелем, которым тот соединялся с портом коммутатора.

Учитывая обстоятельства, в 1992 году институт IEEE начал пересмотр стандартов и дал заказ комитету 802.3 выработать спецификацию более быстрых сетей. Одни предлагали сохранить 802.3 без изменений и просто увеличить

скорость работы, другие — полностью его переделать и снабдить новым набором функций: например, трафик в реальном времени и оцифрованную речь. При этом предлагалось сохранить старое название стандарта (из соображений маркетинга). После некоторых колебаний комитет решил просто изменить скорость работы 802.3, а все остальные параметры оставить прежними. Такая стратегия позволила бы решить проблему прежде, чем технология изменится, избежать непредвиденных проблем с совершенно новыми разработками и обеспечить обратную совместимость с существующими Ethernet LAN. Сторонники отвергнутого предложения поступили так, как в этой ситуации поступил бы любой человек, связанный с компьютерной индустрией: они хлопнули дверью, организовали собственный комитет и разработали свой стандарт (802.12), который, впрочем, с треском провалился.

Работа шла довольно быстро (по меркам комитета стандартизации), и уже в июне 1995 года институт IEEE официально утвердил стандарт 802.3u. С технической точки зрения в нем нет ничего нового по сравнению с предыдущей версией. Честнее было бы назвать это не новым стандартом, а расширением 802.3 (чтобы еще больше подчеркнуть обратную совместимость с ним). Такой прием применялся часто. Большинство называет этот стандарт **«быстрый Ethernet» (Fast Ethernet)**, и мы не будем исключением.

Основная идея Fast Ethernet довольно проста: оставить без изменений все старые форматы фреймов, интерфейсы, процедуры и лишь уменьшить время передачи одного бита с 100 до 10 нс. Как это технически осуществить? Можно скопировать принцип, применяемый в классическом 10-мегабитном Ethernet, но в 10 раз уменьшить максимальную длину сегмента. Однако преимущества витой пары были столь неоспоримы, что практически все системы Fast Ethernet в результате были построены именно на этом типе кабеля. Таким образом, в Fast Ethernet используются исключительно концентраторы (хабы) и коммутаторы; никаких моноканалов с ответвителями типа «зуб вампира» или с BNC-коннекторами здесь нет.

Однако некоторые технические решения все же необходимо было принять. Самый важный вопрос — какие типы кабелей поддерживать. Одним из претендентов была витая пара категории 3. Главным аргументом в ее пользу было то, что практически все западные офисы уже были оборудованы по крайней мере кабелем с четырьмя витыми парами категории 3 (или выше). Они использовались в телефонных линиях, и их длина (до ближайшего телефонного щита) составляла не более 100 м. Иногда можно было встретить два таких кабеля. Таким образом, установка Fast Ethernet не требовала смены кабеля во всем здании. Для многих организаций это было очень существенно.

Главный недостаток витой пары категории 3 — неспособность передавать сигналы 100-мегабитной сети на расстояние 100 м (именно таково максимальное расстояние между компьютером и концентратором, установленное стандартом для 10-мегабитных концентраторов). Витые пары категории 5 с такой задачей справились бы без проблем, а для оптоволокна это и вовсе смешная цифра. Нужно было найти компромисс. В итоге комитет 802.3 разрешил применять все три типа кабелей, как показано на илл. 4.19, при условии, что решения на

основе витой пары категории 3 будут усилены и смогут обеспечить необходимую пропускную способность канала.

| Название | Тип | Длина сегмента, м | Преимущества |
|---|---|---|---|
| 100Base-T4 | Витая пара | 100 | Использование неэкранированной витой пары категории 3 |
| 100Base-TX | Витая пара | 100 | Полный дуплекс при 100 Мбит/с (витая пара категории 5) |
| 100Base-FX | Оптоволокно | 2000 | Полный дуплекс при 100 Мбит/с; большая длина сегмента |

**Илл. 4.19.** Основные типы кабелей для сетей Fast Ethernet

В схеме **100Base-4T**, использующей витую пару категории 3, сигнальная скорость составляет 25 МГц, что лишь на 25 % больше, чем 20 МГц стандарта Ethernet. (Следует помнить, что при манчестерском кодировании, о котором мы говорили в разделе 2.4.3, требуется два тактовых интервала для каждого из 10 млн битов, отправляемых каждую секунду.) Чтобы достичь требуемой пропускной способности, в схеме 100Base-4T применяются четыре витые пары. Одна из них всегда направляется на концентратор, одна — от него, а две оставшиеся переключаются в зависимости от текущего направления передачи данных. Чтобы достигнуть 100 Мбит/с на трех витых парах в направлении передачи, для каждой пары применяется довольно сложная схема. Она состоит в отправке троичных цифровых сигналов с тремя разными уровнями напряжения. Вряд ли эта схема выиграет приз за элегантность, так что мы избавим читателя от деталей.

В любом случае стандартная телефонная проводка десятилетиями включала четыре витые пары в каждом кабеле. Поэтому большинство организаций может использовать уже существующую инфраструктуру. Для этого придется отказаться от офисного телефона, но не такая уж это большая цена за быструю электронную почту.

Система 100Base-T4 утратила актуальность, поскольку во многих офисных зданиях проложили витую пару категории 5 для сетей **100Base-TX**, которые в итоге завоевали рынок. Эта схема проще, поскольку кабели такого типа могут работать с сигналами на частоте 125 МГц. Поэтому для каждой станции используются только две витые пары: одна к концентратору, другая от него. Не применяется ни прямое битовое кодирование (NRZ), ни манчестерское. Вместо них имеется специальная система кодирования **4B/5B** (см. раздел 2.4.3). Четыре бита данных кодируются в форме пяти сигнальных битов и передаются на частоте 125 МГц, обеспечивая скорость 100 Мбит/с. Это схема проста, но в ней выполняется достаточное число переходов для обеспечения синхронизации, и полоса пропускания расходуется довольно эффективно. Система 100Base-TX является полнодуплексной: станции могут одновременно передавать по одной витой паре и принимать по другой с одинаковой скоростью 100 Мбит/с.

В последнем варианте, **100Base-FX**, используется два оптических многомодовых кабеля, по одному для каждого направления передачи. Таким образом,

это также полный дуплекс со скоростью 100 Мбит/с в обе стороны. При такой схеме расстояние между станцией и коммутатором может достигать 2 км.

Fast Ethernet поддерживает соединение с помощью концентраторов либо коммутаторов. Чтобы алгоритм CSMA/CD работал, необходимо соблюдать соотношение между минимальным размером фрейма и максимальной длиной кабеля, учитывая возрастание скорости от 10 до 100 Мбит/с. Таким образом, нужно либо увеличить минимальный размер фрейма (более 64 байт), либо пропорционально уменьшить максимальную длину кабеля (менее 2500 м). Самый простой способ — сократить максимальное расстояние между двумя станциями в 10 раз, поскольку концентратор с кабелями 100 м длиной точно попадает в эти границы. Однако кабели 100Base-FX в 2 км слишком длинны для 100-мегабитного концентратора с обычным алгоритмом управления коллизиями в сетях Ethernet. Их нужно подключать к коммутатору, чтобы они могли работать в полнодуплексном режиме без коллизий.

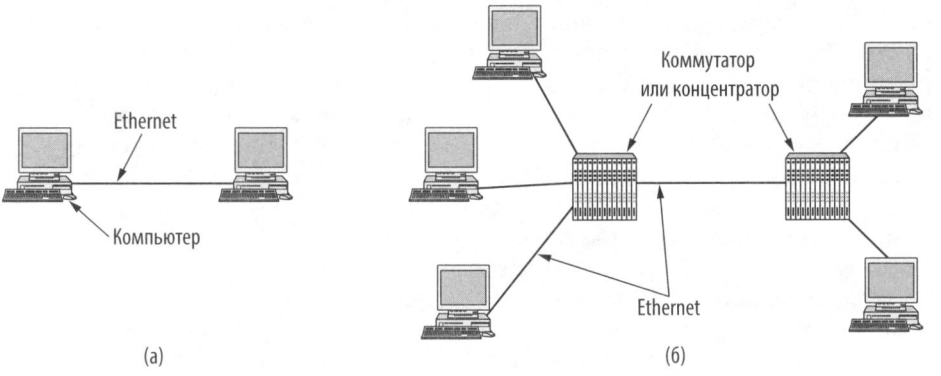
Пользователям очень полюбился Fast Ethernet, но они не собирались так просто выбрасывать 10-мегабитные платы Ethernet со старых компьютеров. В результате практически все коммутаторы могут поддерживать и 10-мегабитные, и 100-мегабитные станции. Для упрощения перехода на новое оборудование сам стандарт предусматривает механизм, названный **автоматическим согласованием (auto-negotiation)**, который позволяет двум станциям автоматически договориться об оптимальной скорости (10 или 100 Мбит) и дуплексном режиме (полный дуплекс или полудуплекс). Обычно он работает без проблем, однако известны случаи возникновения ситуаций, в которых дуплексный режим не совпадает. Одна станция применяет автоматическое согласование, а на другой оно не работает и сразу же устанавливается полнодуплексный режим (Шалунов и Карлсон; Shalunov and Carlson, 2005). Большая часть оборудования Ethernet использует эту функцию для самонастройки.

## 4.3.6. Gigabit Ethernet

Как говорится, еще не высохли чернила на только что созданном стандарте Fast Ethernet, как комитет 802 приступил к работе над новой версией. Вскоре она получила название **«гигабитный Ethernet» (Gigabit Ethernet)**. IEEE ратифицировал наиболее популярную форму сети в 1999 году под названием 802.3ab. Ниже мы обсудим некоторые ключевые свойства Gigabit Ethernet. Более подробную информацию можно найти в работе Сперджена и Циммермана (Spurgeon and Zimmerman, 2014).

Главные цели при создании Gigabit Ethernet были, по сути, такими же, что и для Fast Ethernet: увеличить производительность в 10 раз и сохранить обратную совместимость со старыми сетями Ethernet. В частности, Gigabit Ethernet должен был обеспечить дейтаграммную службу без подтверждений, как при одноадресной, так и при широковещательной передаче. При этом необходимо было сохранить неизменными 48-битную схему адресации и формат фрейма, включая нижние и верхние границы его размера. Новый стандарт соответствовал всем этим требованиям.

Как и в случае Fast Ethernet, все сети Gigabit Ethernet строятся по принципу «точка-точка». Простейшая конфигурация, показанная на илл. 4.20 (а), состоит из двух компьютеров, напрямую соединенных друг с другом. Однако чаще всего используется вариант с коммутатором или концентратором, к которому подключается множество компьютеров; также возможна установка дополнительных коммутаторов или концентраторов (илл. 4.20 (б)). В обеих конфигурациях к каждому отдельному кабелю Gigabit Ethernet всегда присоединяются два устройства, ни больше ни меньше.




**Илл. 4.20.** Сеть Ethernet, состоящая: (а) из двух станций; (б) из множества станций

Как и Fast Ethernet, Gigabit Ethernet может работать в двух режимах: полнодуплексном и полудуплексном. «Нормальным» считается полнодуплексный, при этом трафик может идти одновременно в обоих направлениях. Этот режим используется, когда имеется центральный коммутатор, соединенный с периферийными компьютерами или другими коммутаторами. В такой конфигурации сигналы всех линий буферизируются, поэтому абоненты могут отправлять данные, когда захотят. Отправитель не прослушивает канал, потому что ему не с кем конкурировать. На линии между компьютером и коммутатором компьютер — единственный потенциальный отправитель. Передача произойдет успешно, даже если коммутатор одновременно отправляет фрейм на компьютер (так как линия полнодуплексная). Конкуренции нет, и протокол CSMA/CD не применяется. Поэтому максимальная длина кабеля определяется исключительно мощностью сигнала, а не временем, за которое шумовой всплеск доходит обратно к отправителю. Коммутаторы могут работать на смешанных скоростях; более того, они автоматически выбирают оптимальную скорость. Самонастройка поддерживается так же, как и в Fast Ethernet, но теперь можно выбирать 10, 100 или 1000 Мбит/с.

Полудуплексный режим работы используется тогда, когда компьютеры соединены не с коммутатором, а с концентратором. Концентратор не буферизирует входящие фреймы. Вместо этого он электрически соединяет все линии, симулируя моноканал классического Ethernet. В этом режиме возможны коллизии, поэтому применяется CSMA/CD. Фрейм минимального размера (64 байта)

может передаваться в 100 раз быстрее, чем в классическом Ethernet. Поэтому максимальная длина кабеля должна быть соответственно уменьшена в 100 раз. Она составляет 25 м — именно при таком расстоянии между станциями шумовой всплеск гарантированно достигнет отправителя до окончания его передачи. Если бы кабель имел длину 2500 м, то отправитель 64-байтного фрейма в системе со скоростью 1 Гбит/с завершил бы передачу задолго до того, как фрейм прошел бы только десятую часть пути в одну сторону (не говоря уже о том, что сигнал должен еще и вернуться обратно).

Такое строгое ограничение побудило комитет добавить в стандарт два дополнительных свойства, что позволило увеличить максимальную длину кабеля до 200 м. Это должно было устроить большинство организаций. Первое свойство — **расширение носителя (carrier extension)**. Оно сообщает аппаратному обеспечению, что нужно добавить собственное поле заполнения после обычного фрейма, чтобы расширить его до 512 байт. Поскольку это поле добавляется отправителем и изымается получателем, программному обеспечению нет до него никакого дела. Конечно, тратить 512 байт полосы на передачу 46 байт пользовательских данных (именно столько полезной нагрузки содержится в 64-байтном фрейме) несколько расточительно. Эффективность такой передачи составляет всего 9 %.

Второе свойство, позволяющее увеличить допустимую длину сегмента, — **пакетная передача фреймов (frame bursting)**. Отправитель может посылать не единичный фрейм, а пакет, объединяющий в себе сразу несколько фреймов. Если полная длина пакета оказывается менее 512 байт, то производится аппаратное заполнение (как в предыдущем случае). Если же фреймов, готовых к передаче, достаточно, эта схема оказывается весьма эффективной и применяется вместо расширения носителя.

Честно говоря, трудно представить себе организацию, которая сначала потратит немало средств на установку современных компьютеров с платами для гигабитной сети Ethernet, а потом соединит их древними концентраторами, имитирующими работу классического Ethernet со всеми его коллизиями. Сетевые платы и коммутаторы Gigabit Ethernet когда-то были довольно дорогими, но как только спрос на них возрос, цены быстро упали. Однако обратная совместимость — это «священная корова» в компьютерной индустрии, поэтому, несмотря ни на что, комитету необходимо было ее обеспечить. Сегодня большинство компьютеров поставляются с интерфейсом Ethernet, способным работать на скоростях 10, 100 и 1000 Мбит/с (а иногда и более высоких) и совместимым с каждым из этих режимов.

Gigabit Ethernet поддерживает как медные, так и волоконно-оптические кабели, что отражено на илл. 4.21. Работа на скорости около 1 Гбит/с означает необходимость кодирования и отправки бита каждую наносекунду. Первоначально это достигалось за счет коротких экранированных медных кабелей (версия 1000Base-CX) и оптоволокна. Оно допускает две длины волны, и, следовательно, существуют две разные версии: 0,85 мкм (короткие волны, для 1000Base-SX) и 1,3 мкм (длинные, для 1000Base-LX).

Передача с помощью коротких волн возможна с дешевыми светодиодами. Такой вариант применяется с многомодовым волокном для соединения станций внутри здания, так как для 50-мкм волокна допустимая длина составляет не более

500 м. Для передачи сигналов на длинных волнах требуются лазеры. С другой стороны, при использовании одномодового (10 мкм) волокна длина кабеля может достигать 5 км. Это позволяет подключать здания друг к другу (например, в студенческом городке) аналогично связям «точка-точка». Последующие вариации стандарта допускали даже более длинные связи на одномодовом волокне.

| Название | Тип | Длина сегмента, м | Преимущества |
|---|---|---|---|
| 1000Base-SX | Оптоволокно | 550 | Многомодовое волокно (50; 62,5 мкм) |
| 1000Base-LX | Оптоволокно | 5000 | Одномодовое (10 мкм) или многомодовое (50; 62,5 мкм) волокно |
| 1000Base-CX | Экранированный кабель с 2 витыми парами | 25 | Экранированная витая пара |
| 1000Base-T | Неэкранированный кабель с 4 витыми парами | 100 | Стандартная витая пара категории 5 |

**Илл. 4.21.** Кабели Gigabit Ethernet

Для отправки битов с помощью этих версий Gigabit Ethernet используется система кодирования **8B/10B**, заимствованная из другой сетевой технологии, Fibre Channel (оптоволоконный канал), и упомянутая в разделе 2.4.3. В этой системе 8 бит данных кодируются в кодовые слова из 10 бит, которые отправляются по проводу или оптическому волокну, — отсюда и название 8B/10B. Кодовые слова выбираются так, чтобы они могли быть сбалансированы (то есть иметь равное число нулей и единиц) и чтобы переход осуществлялся достаточное число раз для восстановления синхронизации. Отправка битов, закодированных с помощью NRZ, требует на 25 % больше полосы пропускания, чем передача незакодированных битов, — значительное преимущество по сравнению с манчестерским кодом, предполагающим стопроцентное расширение полосы.

Однако все это требовало новых медных или оптоволоконных кабелей, поддерживающих более быструю передачу сигналов. Ни одна из этих технологий не совместима с витой парой категории 5, которая была в огромных количествах проложена для сетей Fast Ethernet. В течение года потребность была закрыта благодаря 1000Base-T, и с тех пор это самая популярная форма Gigabit Ethernet. Очевидно, людям не слишком нравится заново прокладывать кабели в зданиях.

Чтобы сеть Ethernet могла работать по проводам категории 5 со скоростью 1000 Мбит/с, требуется более сложная схема передачи сигналов. Используются все четыре витые пары в кабеле; каждая пересылает данные одновременно в обоих направлениях, применяя цифровую обработку сигналов для их разделения. Для обеспечения скорости 125 мегасимволов/с в каждом проводе используется пять уровней напряжения, которые переносят по 2 бита. Схема преобразования

битов в символы не так проста. Она включает скремблинг (для безопасной передачи) и код исправления ошибок, в котором четыре значения внедряются в пять сигнальных уровней.

Скорость 1 Гбит/с — это довольно быстро. Если получатель отвлечется на другую задачу хотя бы на 1 мс и не освободит входной буфер, он может пропустить до 1953 фреймов. А если один компьютер передает данные по гигабитной сети, а другой принимает их по классическому Ethernet, буфер получателя переполнится очень быстро. Исходя из этих рисков было принято решение о внедрении в систему Gigabit Ethernet контроля потока. Для его реализации получатель посылает служебный фрейм, сообщающий, что отправитель должен на некоторое время приостановиться. Служебные фреймы PAUSE — это на самом деле обычные фреймы Ethernet с записью 0x8808 в поле Type. Продолжительность паузы определяется в единицах времени передачи минимального фрейма. Для Gigabit Ethernet такая единица равна 512 нс, а паузы могут длиться до 33,6 мс.

Вместе с Gigabit Ethernet было добавлено и еще одно расширение — **Джамбо-пакеты (Jumbo frames)**. Они допускают фреймы длиной более 1500 байт, обычно до 9 Кбайт.

Это расширение защищено патентом. Оно не описано в стандарте, иначе Ethernet уже не будет совместим с предыдущими версиями. Тем не менее большинство производителей его поддерживают. Причина в том, что 1500 байт — это слишком маленькая единица данных для гигабитных скоростей. Манипулируя большими блоками информации, можно уменьшить частоту пересылки фреймов и снизить нагрузку из-за необходимой обработки (например, не придется прерывать процессор, чтобы сообщить о прибытии фрейма, или разбивать и заново соединять сообщения, не поместившиеся в одном фрейме Ethernet).

## 4.3.7. 10-гигабитный Ethernet

Как только Gigabit Ethernet был стандартизован, комитет 802 заскучал и захотел продолжить работу. Тогда IEEE предложил ему начать разработку 10-гигабитного Ethernet (10-Gigabit Ethernet). Работа шла по тому же принципу, что и при стандартизации предыдущих версий. Первые стандарты для оптоволоконного и экранированного медного кабеля появились в 2002 и 2004 годах, а для медной витой пары — в 2006 году.

10 Гбит/с — это поистине колоссальная скорость. В 1000 раз быстрее первоначального стандарта Ethernet! Где она может понадобиться? Ответ — в датацентрах и точках обмена трафиком с высококлассными маршрутизаторами, коммутаторами и серверами, а также в сильно загруженных магистральных каналах, соединяющих офисы компаний в разных городах. Весь город можно охватить единой сетью на базе оптоволокна и Ethernet. Связь на больших расстояниях требует прокладки оптического волокна, тогда как более короткие соединения можно выполнять с помощью медных кабелей.

Все версии 10-гигабитного Ethernet поддерживают только полнодуплексную передачу данных. CSMA/CD больше не входит в архитектуру, и стандарты фокусируются на деталях физического уровня, которые обеспечивают высокую

скорость. Однако совместимость не потеряла своего значения, поэтому интерфейсы 10-гигабитного Ethernet выполняют автоматическое согласование скорости и выбирают максимально возможное значение для обоих концов линии.

Основные типы 10-гигабитного Ethernet перечислены на илл. 4.22. На средних расстояниях применяется многомодовое волокно с длиной волны 0,85 мкм, а на больших — одномодовое с длиной волны 1,3 и 1,5 мкм. Сеть 10GBase-ER может охватывать до 40 км, что хорошо подходит для использования на больших территориях. Все эти версии отправляют последовательный поток информации, образованный путем скремблинга битов данных и последующего их кодирования по схеме **64B/66B** (она требует меньше накладных расходов, чем 8B/10B).

| Название | Тип | Длина сегмента | Преимущества |
|---|---|---|---|
| 10GBase-SR | Оптоволокно | До 300 м | Многомодовое волокно (0,85 мкм) |
| 10GBase-LR | Оптоволокно | 10 км | Одномодовое волокно (1,3 мкм) |
| 10GBase-ER | Оптоволокно | 40 км | Одномодовое волокно (1,5 мкм) |
| 10GBase-CX4 | 4 пары биаксиального кабеля | 15 м | Биаксиальный медный кабель |
| 10GBase-T | Неэкранированный кабель с 4 витыми парами | 100 м | Неэкранированная витая пара категории 6а |

**Илл. 4.22.** Кабели 10-гигабитного Ethernet

Первая версия, разработанная для медного кабеля, 10GBase-CX4, работает на базе кабеля с четырьмя парами биаксиального медного провода. В каждой паре используется кодирование 8B/10B, они работают на скорости 3,125 гигасимволов/с, обеспечивая скорость передачи 10 Гбит/с. Эта версия дешевле волоконной и первой вышла на рынок, но еще неясно, сумеет ли она вытеснить с рынка 10-гигабитный Ethernet на базе витой пары.

10GBase-T — это версия, работающая на неэкранированной витой паре. Несмотря на то что официально она требует прокладки кабеля категории 6а, пока что можно использовать и более старые категории (включая пятую), то есть уже проложенные во множестве зданий по всему миру кабели. Неудивительно, что для достижения скорости 10 Гбит/с на витой паре во многом задействуется физический уровень. Мы лишь слегка коснемся работы более высоких уровней. Каждая из четырех витых пар используется для передачи данных в обоих направлениях на скорости 2500 Мбит/с. Это достигается за счет скорости пересылки сигналов 800 мегасимволов/с на 16 уровнях напряжения. Символы создаются путем скремблинга данных, защиты их с помощью кода LDPC (Low Density Parity Check) и последующего кодирования для исправления ошибок.

Различные варианты 10-гигабитного Ethernet получили широкое распространение на рынке, а комитет 802.3 продолжил работу. В конце 2007 года IEEE создала группу по стандартизации сетей Ethernet, работающих на скоростях

40 и 100 Гбит/с. Такой рывок позволит Ethernet стать серьезным соперником альтернативным технологиям в областях, требующих высокой производительности, — междугородних соединениях в магистральных сетях и коротких соединениях через системные платы устройств. Описание стандарта еще не завершено, однако некоторые патентованные продукты уже доступны.

### 4.3.8. 40- и 100-гигабитный Ethernet

Завершив стандартизацию 10-гигабитной сети Ethernet, комитет 802.11 приступил к работе над новыми стандартами сети Ethernet для скоростей 40 и 100 Гбит/с. Первый стандарт предназначен для внутренних соединений в центрах обработки данных и не рассчитан на обычных провайдеров, а тем более на конечных пользователей. Второй стандарт — для магистральных интернет-каналов — должен работать на оптических сетевых трассах длиной в тысячи километров. Он также может использоваться в виртуальной частной LAN для соединения двух центров обработки данных с миллионами процессоров.

Сначала был принят стандарт 802.3ba (2010), а затем — стандарты 802.3bj (2014) и 802.3cd (2018). Каждый из этих трех стандартов описывает и 40-гигабитный, и 100-гигабитный Ethernet. При их разработке ставились следующие цели:

1. Обратная совместимость со стандартами 802.3 вплоть до скорости 1 Гбит/с.
2. Минимальный и максимальный размеры фреймов должны остаться прежними.
3. Вероятность ошибки не более $10^{-12}$.
4. Хорошая работа в оптических сетях.
5. Скорость передачи данных — 40 или 100 Гбит/с.
6. Возможность использования одномодового или многомодового волокна, а также специализированных магистральных шин.

Новые стандарты позволят постепенно заменить медный провод оптическим волокном и высокопроизводительными (медными) магистральными шинами, рассчитанными на центры обработки данных с поддержкой облачных вычислений. Поддерживается полдюжины схем модуляции, включая схему 64B/66B (она аналогична схеме 8B/10B, но требует больше битов). Кроме того, можно использовать до 10 параллельных полос с пропускной способностью 10 Гбит/с, что в сумме дает до 100 Гбит/с. Эти полосы, как правило, представляют собой различные диапазоны частот, передаваемые по оптическому волокну. Интеграция с имеющимися оптическими сетями производится в соответствии с рекомендациями G.709.

Начиная с 2018 года несколько компаний стали внедрять 100-гигабитные коммутаторы и сетевые адаптеры. Для тех, кто не хочет ограничиваться полосой 100 Гбит/с, уже начата работа над стандартами сетей со скоростями до 400 Гбит/с — сетей класса 400GbE. Для подробного ознакомления смотрите стандарты 802.3cd, 802.3ck, 802.3cm и 802.3cn. Заметим, что при скорости 400 Гбит/с обычный (сжатый) фильм в формате 4K можно полностью скачать примерно за 2 с.

### 4.3.9. Ретроспективный взгляд на Ethernet

Ethernet существует вот уже более 40 лет, и никаких серьезных конкурентов у него не появилось. Скорее всего, он не потеряет актуальности в течение долгого времени. Не многие микропроцессорные архитектуры, операционные системы и языки программирования могут похвастаться таким продолжительным и уверенным лидерством. Вероятно, Ethernet выгодно отличается от всех остальных систем. Чем же?

Возможно, основной причиной столь длительного успеха является простота и гибкость системы. Простота в данном случае означает прежде всего надежность, невысокую цену и легкость обслуживания. С тех пор как были созданы архитектуры на базе концентраторов и коммутаторов, чисто технические поломки стали чрезвычайно редкими. Человек так устроен, что с трудом может отказаться от хорошо работающей системы в пользу чего-то нового. Нужно принять во внимание и тот факт, что огромное количество кое-как собранного компьютерного оборудования работает не слишком надежно. Именно по этой причине так называемые апгрейды часто дают противоположный ожидаемому результат, и системы после них работают не лучше, а, наоборот, хуже.

Вторая причина популярности Ethernet — это низкая цена. Витая пара сравнительно недорога, так же как аппаратные компоненты. Затрат может потребовать, например, переход на новые платы Gigabit Ethernet или коммутаторы, но это всего лишь дополнения к существующей сети (а не замена всего имеющегося оборудования), к тому же оптовые цены значительно выгоднее розничных.

Сети Ethernet не доставляют большой головной боли системным администраторам — они обслуживаются без особых проблем. Не нужно устанавливать никакое программное обеспечение (кроме драйверов), и очень мало конфигурационных таблиц (в которых так просто ошибиться). Новые узлы добавляются очень просто.

Еще одно достоинство Ethernet заключается в хорошем взаимодействии с TCP/IP — доминирующим протоколом сети интернет. IP — протокол без установления соединения, поэтому он без проблем внедряется в локальных сетях Ethernet, работающих по тому же принципу. IP имеет довольно плохую совместимость с сетями ATM, ориентированными на установление соединения. Этот факт крайне негативно сказывается на популярности ATM.

И что важнее всего, разработчикам Ethernet удалось добиться хороших показателей по самым главным направлениям. Скорости выросли на несколько порядков, в систему были внедрены коммутаторы и концентраторы, но эти изменения никак не коснулись программного обеспечения. Помимо этого, часто допускается временное использование существующей кабельной разводки. Если продавец скажет: «Вот отличная новая сетевая система! Она работает просто фантастически быстро и надежно! Вам необходимо только выкинуть весь ваш старый железный хлам и стереть все старые программы», — у него возникнут проблемы с объемами продаж.

Многие альтернативные технологии (о которых вы, вероятно, даже не слышали) в момент своего появления были даже быстрее тогдашнего Ethernet.

Помимо ATM, этот список включает FDDI и волоконно-оптический канал (FC, Fibre Channel)[1] — две оптические локальные сети на базе кольца. Обе были несовместимы с Ethernet, и обе канули в Лету. Они были слишком запутанными, что вело к усложнению микросхем и повышению цен. Главный урок — не забывать принцип KISS (Keep It Stupid Simple), то есть «будьте проще». Позже Ethernet догнал и перегнал конкурентов по скорости, по пути заимствуя элементы их технологий (например, кодирование 4B/5B у FDDI и 8B/10B у FC). У соперников не осталось никаких преимуществ, и они либо исчезли, либо стали применяться в узкоспециализированных сферах.

Создается впечатление, что области применения Ethernet продолжат некоторое время расширяться. 10-гигабитный Ethernet освободился от ограничений максимального расстояния, накладываемых CSMA/CD. Много внимания уделяется **Ethernet операторского класса (carrier-grade Ethernet)**, который позволит сетевым провайдерам предоставлять услуги, основанные на Ethernet, своим клиентам в MAN и WAN (Хокинс; Hawkins, 2016). Эта система способна передавать Ethernet-фреймы на большие расстояния по оптоволоконному кабелю и требует усовершенствования возможностей управления, чтобы операторы смогли предлагать пользователям надежные высококачественные услуги. Сверхбыстродействующие сети класса 100GbE также находят применение в системных платах, соединяющих компоненты больших маршрутизаторов и серверов. Эти варианты использования представляют собой дополнение к передаче фреймов между компьютерами в офисах. Следующим и, вероятно, не последним шагом являются сети класса 400GbE.

## 4.4. БЕСПРОВОДНЫЕ ЛОКАЛЬНЫЕ СЕТИ

Популярность беспроводных локальных сетей постоянно растет. Все больше домов, офисных зданий, кафе, библиотек, аэровокзалов, зоопарков и других общественных мест оснащаются соответствующим оборудованием для подключения настольных компьютеров, ноутбуков, планшетов и смартфонов к интернету. Кроме того, беспроводные LAN позволяют двум или нескольким расположенным неподалеку компьютерам обмениваться данными и без выхода в интернет.

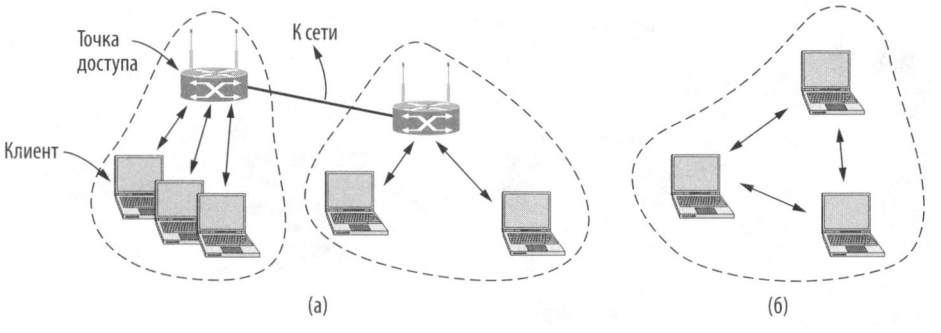
Больше двух десятилетий основным стандартом беспроводных LAN был 802.11. Общие сведения о нем уже были даны в разделе 1.4.3. Пришло время рассмотреть этот стандарт более детально. В последующих разделах речь пойдет о стеке протоколов, методах радиопередачи (на физическом уровне), протоколе подуровня MAC, структуре фрейма и службах. Более подробные сведения о стандарте 802.11 можно найти в работах Бинга (Bing, 2017) и Дэвиса (Davis, 2018). Чтобы получить информацию из первых рук, обратитесь к официальному техническому описанию стандартов IEEE.

---

[1] Данная технология получила название «Fibre Channel», а не «Fiber Channel», так как документ с ее описанием редактировал британец.

### 4.4.1. Стандарт 802.11: архитектура и стек протоколов

Сети 802.11 можно использовать в двух режимах. Наиболее популярный — **инфраструктурный режим (infrastructure mode)**. Это подключение клиентов (например, ноутбуков и смартфонов) к другой сети, например внутренней сети компании или интернету. Данная схема представлена на илл. 4.23 (а). Каждый клиент связывается с **точкой доступа (Access Point, AP)**, которая, в свою очередь, подключена к сети. Клиент отправляет и получает пакеты через AP. Несколько AP можно объединять, например, в проводную сеть под названием **«распределительная сеть» (distribution system)**. Так формируется расширенная сеть 802.11. В этом случае клиенты могут отправлять фреймы другим клиентам через их точки доступа.




**Илл. 4.23.** Архитектура сети стандарта 802.11. (а) Инфраструктурный режим. (б) Произвольный режим

Второй режим, показанный на илл. 4.23 (б), называется **децентрализованной сетью (ad hoc network)**. Это набор компьютеров, которые связаны таким образом, чтобы напрямую отправлять фреймы друг другу. Точка доступа не используется. Поскольку доступ в интернет — революционная технология в беспроводных соединениях, децентрализованные сети не слишком популярны.

Теперь рассмотрим протоколы. Все протоколы, используемые семейством стандартов 802.x, включая 802.11 и Ethernet, схожи по структуре. Часть стека протоколов, соответствующая основным вариантам стандарта 802.11, изображена на илл. 4.24. И для клиентов, и для точек доступа применяется один стек. Физический уровень почти такой же, как и в модели OSI, а вот канальный уровень во всех протоколах 802.x разбит на два или более подуровня. Что касается 802.11, то MAC (подуровень управления доступом к среде) отвечает за распределение канала, то есть за то, какая станция будет передавать следующей. Выше находится LLC (подуровень управления логической связью), цель которого в том, чтобы скрыть различия стандартов 802.x от сетевого уровня. Это ответственная задача, но, помимо этого, LLC сегодня является связующим уровнем, отвечающим за идентификацию протокола (например, IP), информация о котором передается во фрейме 802.11.

С момента появления в 1997 году физический уровень обзавелся несколькими методами передачи. Два первоначальных метода, инфракрасная передача (как в пульте дистанционного управления телевизором) и режим скачкообразного изменения частоты в диапазоне 2,4 ГГц, сегодня не используются. Третий из исходных методов, широкополосный сигнал с прямой последовательностью на скорости 1 или 2 Мбит/с в диапазоне 2,4 ГГц, был расширен и завоевал популярность со скоростями до 11 Мбит/с. Этот стандарт известен под названием 802.11b.

**Илл. 4.24.** Часть стека протоколов стандарта 802.11

Чтобы предоставить поклонникам беспроводных сетей желанное увеличение скорости, в 1999 и 2003 годах были разработаны новые методы передачи на основе мультиплексирования с ортогональным частотным разделением (Orthogonal Frequency Division Multiplexing, OFDM), описанного в разделе 2.5.3. Первый метод, 802.11a, работает в другом диапазоне частот — 5 ГГц. Второй, 802.11g, остался в диапазоне 2,4 ГГц для обеспечения совместимости. Оба работают на скорости до 54 Мбит/с.

В октябре 2009 года окончательно сформировался стандарт 802.11n. Он содержит методы передачи данных, дающие значительный прирост скорости за счет одновременного использования нескольких антенн на приемнике и передатчике.

Дойдя до конца алфавита, в декабре 2013 года институт IEEE опубликовал стандарт под названием 802.11ac. Следует заметить, что члены комитета 802.11 используют пропущенные буквы для внесения небольших технических улучшений, часто служащих для уточнения или исправления ошибок (например, в случае стандарта 802.11r). Стандарт 802.11ac предназначен для диапазона 5 ГГц, поэтому он несовместим со старым оборудованием, работающим только в полосе 2,4 ГГц. Сегодня наиболее продвинутые мобильные устройства используют 802.11ac. Недавно принятый 802.11ax обеспечивает еще более высокую скорость.

Далее мы вкратце рассмотрим все эти методы передачи и подробно изучим самые актуальные из них, отбросив устаревшие стандарты 802.11. Формально они относятся к физическому уровню, которому была посвящена глава 2.

Но из-за того, что они тесно связаны с беспроводными LAN вообще и с LAN стандарта 802.11 в частности, мы познакомимся с ними здесь.

## 4.4.2. Стандарт 802.11: физический уровень

Все представленные ниже методы позволяют передать фрейм подуровня MAC с одной станции на другую по радиоканалу. Отличаются они используемыми технологиями и скоростями, достижимыми на практике. Детальное рассмотрение этих методов выходит за рамки нашей книги, мы лишь дадим краткое описание, которое, возможно, заинтересует читателей и снабдит их необходимыми терминами для поиска более подробной дополнительной информации (см. также главу 2).

Все методы стандарта 802.11 используют радиосигналы ближнего радиуса действия в диапазонах 2,4 ГГц или 5 ГГц. Преимущество этих полос в том, что они не требуют лицензирования, то есть доступны для любого передатчика, отвечающего небольшому числу ограничений, например излучаемой мощности до 1 Вт (хотя для большинства передатчиков в беспроводных LAN характерна мощность 50 мВт). К сожалению, этот факт также известен производителям автоматических гаражных дверей, беспроводных телефонов, микроволновых печей и множества других устройств, конкурирующих за спектр частот с ноутбуками и смартфонами, использующими Wi-Fi. Полоса 2,4 ГГц более заполнена, поэтому в некоторых случаях 5 ГГц предпочтительнее (несмотря на меньший радиус действия из-за более высокой частоты). К сожалению, радиоволны 5 ГГц короче, чем волны 2,4 ГГц, и не так хорошо проходят сквозь стены, поэтому этот диапазон не является безоговорочным победителем.

Все методы позволяют передавать сигнал на разной скорости в зависимости от текущих условий. Если беспроводной сигнал слабый, выбирается низкая скорость, если сильный — ее можно повысить. Такая корректировка называется **адаптацией скорости (rate adaptation)**. Скорости могут различаться в десятки раз, поэтому хорошая адаптация важнее производительности соединения. Поскольку для совместимости это значения не имеет, в стандартах не говорится, как именно корректировать скорость.

Первый метод передачи, который мы рассмотрим, — **802.11b**. Это технология расширенного спектра, поддерживающая скорости 1, 2, 5,5 и 11 Мбит/с (на практике рабочая скорость почти всегда близка к максимальной). Данный метод похож на систему CDMA (см. раздел 2.4.4), однако в нем есть только один код расширения спектра, применяемый всеми пользователями. Расширение необходимо для выполнения требования FCC: мощность должна распределяться по диапазону ISM. Для стандарта 802.11b используется **последовательность Баркера (Barker sequence)**. Ее отличительная особенность — в низкой автокорреляции (за исключением случаев, когда последовательности выровнены). Благодаря этому получатель может захватить начало передачи. Для достижения скорости 1 Мбит/с последовательность Баркера комбинируется с модуляцией BPSK, и с каждым набором из 11 элементарных сигналов (чипов) передается 1 бит. Сигналы пересылаются со скоростью 11 мегачипов/с. Чтобы достичь

скорости 2 Мбит/с, последовательность комбинируется с модуляцией QPSK, и на каждые 11 чипов приходится 2 бита. На более высоких скоростях дело обстоит по-другому. Вместо последовательности Баркера для конструирования кодов применяется **дополнительная кодовая манипуляция (Complementary Code Keying, CCK)**. На скорости 5,5 Мбит/с в каждом коде из 8 элементарных сигналов отправляется 4 бита, а на скорости 11 Мбит/с — 8 бит.

Перейдем к **802.11a**. Он поддерживает скорости до 54 Мбит/с в 5-гигагерцевом диапазоне ISM. Можно подумать, что 802.11a появился раньше 802.11b, но это не так. Хотя группа 802.11a была основана раньше, стандарт 802.11b первым получил одобрение, а продукты на его основе вышли на рынок раньше продуктов 802.11a (в том числе из-за сложностей работы в более высоком диапазоне 5 ГГц).

Метод 802.11a основан на **мультиплексировании с ортогональным частотным разделением каналов OFDM (Orthogonal Frequency Division Multiplexing)**, так как оно эффективно использует спектр и устойчиво к искажению беспроводного сигнала, например, из-за многолучевого распространения. Биты параллельно отправляются по 52 поднесущим, из которых 48 содержат данные и 4 служат для синхронизации. Каждый символ передается в течение 4 мкс и состоит из 1, 2, 4 или 6 бит. Биты кодируются для исправления ошибок с применением сверточного кода. Поэтому только 1/2, 2/3 или 3/4 битов не являются избыточными. В разных комбинациях 802.11a может обеспечивать восемь разных показателей скорости, от 6 до 54 Мбит/с. Это значительно выше, чем у 802.11b, к тому же в диапазоне 5 ГГц гораздо меньше помех. Однако радиус действия 802.11b примерно в семь раз больше, чем у 802.11a, что во многих ситуациях крайне важно.

Несмотря на неплохую дальность действия, разработчики 802.11b не собирались давать этому неожиданному фавориту шанс на победу в соревновании скоростей. К счастью, в мае 2002 года FCC отменила давнее правило, требующее, чтобы все беспроводное коммуникационное оборудование, работающее в США в диапазонах ISM, применяло расширение спектра. Это позволило начать разработку стандарта **802.11g**, который был одобрен комитетом IEEE в 2003 году. Он копирует методы модуляции OFDM стандарта 802.11a, но, как и 802.11b, используется в ограниченном диапазоне ISM 2,4 ГГц. 802.11g предлагает те же скорости, что и 802.11a (6–54 Мбит/с) и, разумеется, совместимость с любыми устройствами 802.11b, которые могут оказаться поблизости. Все эти различия зачастую сбивают с толку обычных пользователей, поэтому продукты обычно поддерживают 802.11a/b/g в одной сетевой карте.

Не останавливаясь на достигнутом, комитет IEEE начал работу над физическим уровнем **802.11n** с очень высокой производительностью. Он был одобрен в 2009 году. Цель 802.11n — обеспечить пропускную способность не менее 100 Мбит/с, устранив все накладные расходы беспроводной связи. Для этого требуется увеличить базовую скорость как минимум в четыре раза. Комитет удвоил ширину каналов с 20 до 40 МГц и снизил накладные расходы на передачу, разрешив совместную отправку целой группы фреймов. Что еще важнее, в стандарте 802.11n предусмотрено использование до четырех антенн для пересылки до четырех потоков информации одновременно. Сигналы потоков смешиваются на стороне получателя, но их можно разделить с помощью коммуникационных

методов **MIMO (Multiple Input Multiple Output — «несколько входов, несколько выходов»)**. Наличие нескольких антенн либо повышает скорость, либо увеличивает радиус действия и надежность. MIMO, как и OFDM, — одна из тех удачных идей в сфере коммуникаций, которые в корне меняют дизайн беспроводных сетей и наверняка нередко станут применяться и в будущем. Краткое описание метода использования нескольких антенн в стандарте 802.11 вы найдете в работе Халперина и др. (Halperin et al., 2010).

В 2013 году институт IEEE опубликовал стандарт 802.11ac. Он использует более широкие каналы (80 и 160 МГц), модуляцию 256-QAM и **многопользовательскую систему MIMO (Multiuser MIMO, MU-MIMO)**, включающую до восьми потоков, а также другие приемы в попытках обеспечить теоретически максимальный битрейт — 7 Гбит/с. Однако на практике не удается даже приблизиться к этому пределу. Стандарт 802.11ac сегодня используется большинством массово выпускаемых мобильных устройств.

Еще одной недавно появившейся версией стандарта 802.11 является **802.11ad**. Этот стандарт работает в полосе 60 ГГц (57–71 ГГц), то есть использует очень короткие радиоволны, длина которых составляет лишь 5 мм. Поскольку они не могут проходить сквозь стены или другие преграды, 802.11ad может применяться только внутри одного помещения. Это одновременно и минус, и плюс. Пользователь в соседнем офисе или квартире не создаст никаких помех для вашей работы. Сочетание высокой пропускной способности с низкой проницаемостью идеально подходит для потоковой передачи несжатых фильмов в формате 4K или 8K от базовой станции к мобильным устройствам, находящимся в том же помещении. Стандарт **802.11ay** пошел еще дальше, увеличив пропускную способность в четыре раза.

Наконец, мы подошли к **802.11ax**, который иногда называют **высокоэффективным беспроводным стандартом (high-efficiency wireless)**.

Данный стандарт получил понятное для потребителя название **Wi-Fi 6**. (Если вы думаете, что не заметили наименований от «Wi-Fi 1» до «Wi-Fi 5» по невнимательности, то это не так. Предыдущие названия давались в соответствии с нумерацией стандартов IEEE. Однако эту версию группа Wi-Fi Alliance решила назвать «Wi-Fi 6» с учетом того, что это шестая версия Wi-Fi.) 802.11ax позволяет использовать более эффективный метод QAM-модуляции в сочетании с новой схемой OFDMA. Она (теоретически) может работать в нелицензируемых частях спектра вплоть до 7 ГГц, обеспечивая скорость передачи данных до 11 Гбит/с. Вы можете попытаться достигнуть этой скорости у себя дома, однако, не располагая идеально обустроенной тестовой лабораторией, вряд ли добьетесь успеха. В то же время вы вполне можете получить скорость 1 Гбит/с.

В схеме модуляции OFDMA стандарта 802.11ax центральный планировщик выделяет каждой из передающих станций единицы ресурса фиксированной длины, тем самым снижая степень конкуренции в зашумленном эфире. Также 802.11ax позволяет повторно использовать пространственный спектр за счет метода **«окрашивания» (coloring)**: отправитель помечает начало своей передачи так, чтобы остальные могли определить, возможно ли совместное использование спектра. В некоторых случаях отправитель может осуществлять одновременную передачу, уменьшив свою мощность соответствующим образом.

Кроме того, в отличие от 802.11ac, где используется модуляция 256-QAM, позволяющая передавать 8 бит на символ, стандарт 802.11ax использует модуляцию 1024-QAM (10 бит на символ). Данный стандарт также обеспечивает более интеллектуальное планирование за счет использования функции **целевого времени пробуждения (TWT, target wake time)**. С ее помощью маршрутизатор минимизирует количество коллизий путем составления графика передачи для устройств пользователя. Эта функция, вероятно, будет наиболее полезной в «умном доме», где все больше подключенных приборов периодически отправляют контрольные сигналы на домашний маршрутизатор.

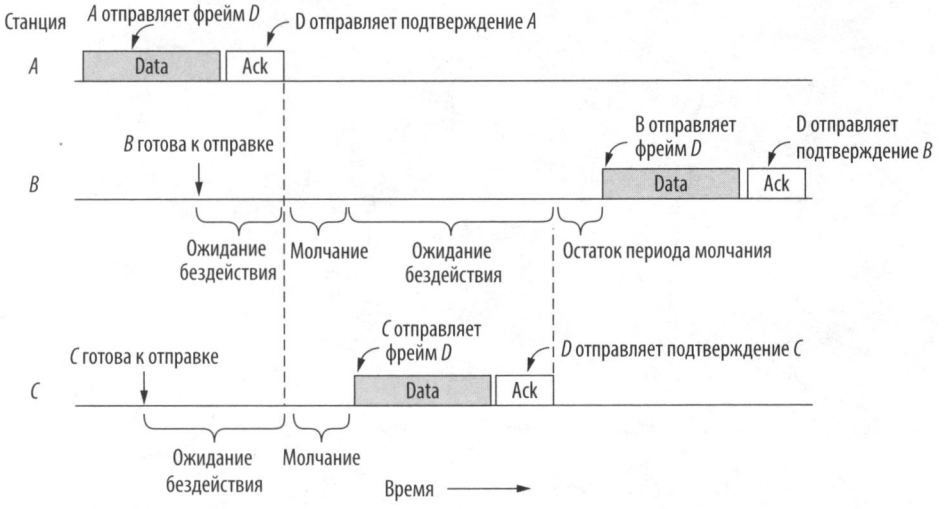
## 4.4.3. Стандарт 802.11: протокол подуровня управления доступом к среде

Однако вернемся из области электротехники к компьютерным наукам. Протокол подуровня MAC в стандарте 802.11 довольно сильно отличается от аналогичного протокола Ethernet вследствие двух фундаментальных факторов, характерных для беспроводного обмена данными.

Прежде всего радиопередатчики почти всегда работают в полудуплексном режиме. Это означает, что они не могут одновременно передавать сигналы и прослушивать всплески шума на одной и той же частоте. Получаемый сигнал может быть в миллион раз слабее передаваемого, и его можно не зафиксировать при одновременной передаче. В Ethernet станция ожидает, пока в канале настанет тишина, и тогда начинает отправку. Если шумовой всплеск не приходит обратно в течение времени, необходимого на пересылку 64 байт, то можно утверждать, что фрейм почти наверняка доставлен корректно. В беспроводных сетях такой механизм распознавания коллизий не работает.

Вместо этого 802.11 пытается избегать коллизий за счет протокола **CSMA с предотвращением коллизий (CSMA with Collision Avoidance, CSMA/CA)**. Концептуально он аналогичен CSMA/CD для Ethernet, где канал прослушивается перед началом отправки, а период молчания после коллизии вычисляется экспоненциально. Однако если у станции есть фрейм для пересылки, то она начинает цикл с периода молчания случайной длины (за исключением случаев, когда она давно не использовала канал и он простаивает). Станция не ожидает коллизий. Число слотов, в течение которых она молчит, выбирается в диапазоне от 0 до, скажем, 15 в случае физического уровня OFDM. Станция дожидается бездействия канала в течение короткого периода времени (называемого DIFS; подробнее о нем ниже) и отсчитывает свободные слоты, приостанавливая отсчет на время отправки фреймов. Свой фрейм она отправляет, когда счетчик достигает нуля. При успешной передаче адресат немедленно отправляет обратно короткое подтверждение. Если подтверждения нет, делается вывод, что произошла ошибка — будь то коллизия или любая другая. В таком случае отправитель удваивает период молчания и повторяет попытку, продолжая экспоненциально наращивать длину паузы (как в случае Ethernet), пока фрейм не будет успешно передан или пока не будет достигнуто максимальное число повторов.

Пример некоторой временной шкалы приводится на илл. 4.25. Станция *A* отправляет фрейм первой. Пока она передает, станции *B* и *C* переходят в режим готовности к отправке. Они видят, что канал занят, и дожидаются его освобождения. Вскоре после получения подтверждения станцией *A* канал переходит в режим бездействия. Но вместо того чтобы сразу отправлять фреймы (что привело бы к коллизии), станции *B* и *C* начинают свои периоды молчания. *C* выбирает короткий период молчания, поэтому ей удается отправить данные первой. *B* приостанавливает обратный отсчет, когда видит, что канал занят станцией *C*, и возобновляет только после получения станцией *C* подтверждения. Вскоре период молчания *B* завершается, и она также отправляет фрейм.



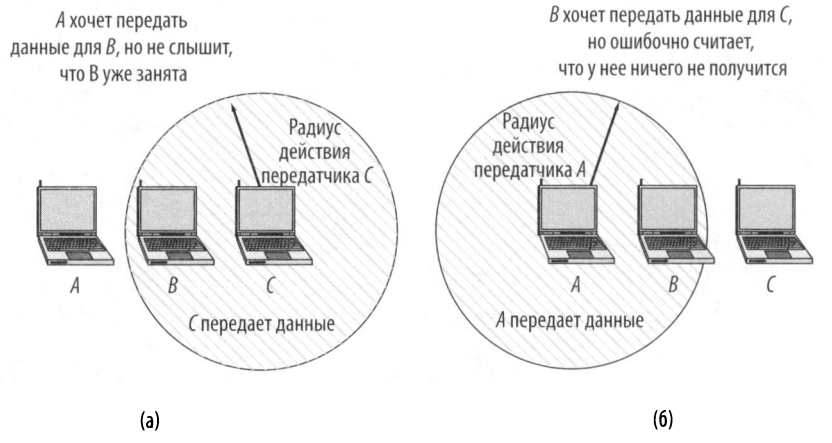

**Илл. 4.25.** Отправка фрейма с протоколом CSMA/CA

По сравнению с Ethernet здесь два основных отличия. Во-первых, раннее начало периодов молчания помогает избегать конфликтов. Это важное преимущество, так как коллизии обходятся дорого, ведь даже если происходит столкновение, фрейм все равно отправляется целиком. Во-вторых, чтобы станции могли «догадываться» о коллизиях, которые распознать невозможно, применяется схема с подтверждениями.

Такой режим называется **распределенной координацией (Distributed Coordination Function, DCF)**. Все станции действуют независимо, нет централизованного контроля. Стандарт также включает необязательный режим **сосредоточенной координации (Point Coordination Function, PCF)**, при котором все процессы в ячейке контролирует точка доступа — как базовая станция сотовой сети. Однако на практике PCF не применяется, так как невозможно запретить станциям из соседней сети передавать конкурирующий трафик.

Вторая проблема заключается в том, что области передачи разных станций могут не совпадать. В проводной сети система спроектирована таким образом,

чтобы все станции могли слышать друг друга. Особенности передачи радиосигналов не позволяют обеспечить такое постоянство для беспроводных станций. Следовательно, может возникнуть упомянутая ранее проблема скрытой станции (илл. 4.26 (а). Поскольку не все станции слышат друг друга, передача в одной части ячейки может быть не воспринята станцией, находящейся в другой ее части. В приведенном на рисунке примере станция *C* передает данные станции *B*. Если станция *A* прослушает канал, она не обнаружит ничего подозрительного и сделает ложный вывод о том, что она имеет право начать передачу станции *B*. Это решение приведет к коллизии.



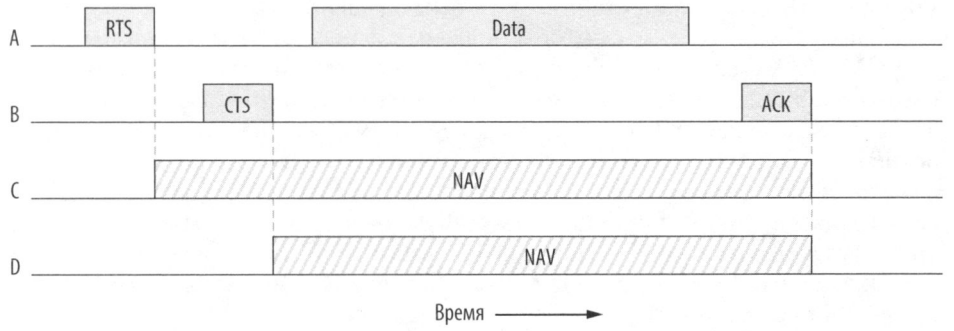

**Илл. 4.26.** Проблема (а) скрытой станции; (б) засвеченной станции

Кроме того, есть и обратная проблема, показанная на илл. 4.26 (б). Станция *B* хочет отправить данные для станции *C* и прослушивает канал. Услышав, что в нем уже осуществляется какая-то передача, *B* делает ложный вывод, что отправка для *C* сейчас невозможна. Между тем станция *A* — источник сигнала, который смутил станцию *B*, — на самом деле отправляет данные станции *D* (на рисунке ее нет). Таким образом, теряется возможность передать информацию.

Чтобы решить проблему очередности передачи данных станциями, в стандарте 802.11 прослушивание канала происходит и на физическом, и на виртуальном уровне. При физическом прослушивании среда просто проверяется на наличие сигнала. Виртуальное прослушивание заключается в том, что станции ведут логический журнал использования канала, отслеживая **вектор распределения сети (Network Allocation Vector, NAV)**. Каждый фрейм содержит поле NAV, которое сообщает, как долго будет передаваться последовательность, в которую он входит. Станции, услышавшие этот фрейм, понимают, что канал будет занят в течение указанного в NAV периода, даже если физический сигнал в канале отсутствует. Например, NAV для фреймов данных включает также время, необходимое для отправки подтверждения. Все станции, фиксирующие этот фрейм, воздерживаются от передачи в течение периода отправки подтверждения, слышали они его или нет. По сути, поле NAV служит для отсчета времени ожидания, когда

отправитель предполагает, что канал занят. В стандарте 802.11 интервал поля NAV отсчитывается в микросекундах. В ситуациях, когда в эфире множество беспроводных устройств, поле NAV, установленное одним отправителем, может сбрасываться другими передающими в том же диапазоне станциями. Это порождает коллизии и снижает производительность. Для устранения такого эффекта в версии 802.11ax используется не одно, а два поля NAV. Первое модифицируется фреймами, которые соответствуют фреймам, привязанным к станции, второе — фреймами, которые могут улавливаться станцией, но исходят из перекрывающихся сетей.

Дополнительный механизм RTS/CTS использует NAV, чтобы запрещать станциям отправлять фреймы одновременно со скрытыми станциями (илл. 4.27). В этом примере станция *A* хочет передать данные станции *B*. Станция *C* находится в зоне действия *A* (а также, возможно, в зоне действия *B*, но это не имеет значения). Станция *D* входит в зону действия *B*, но не входит в зону действия *A*.




**Илл. 4.27.** Использование прослушивания виртуального канала в протоколе CSMA/CA

Протокол начинает работать тогда, когда станция *A* решает, что ей необходимо послать данные станции *B*. *A* посылает станции *B* фрейм RTS, запрашивая разрешение на передачу. Если станция *B* может принять данные, она отсылает обратно подтверждение о том, что канал чист, — фрейм CTS. После приема CTS станция *A* отправляет фрейм и запускает таймер подтверждения. В случае корректного приема *B* генерирует фрейм подтверждения, завершающий передачу. Если интервал таймера на станции *A* истекает до получения подтверждения, то считается, что произошла коллизия, и весь алгоритм работы протокола повторяется с самого начала после периода молчания.

Теперь рассмотрим этот же процесс с точки зрения станций *C* и *D*. Станция *C* находится в зоне действия *A*, поэтому она также принимает фрейм RTS и понимает, что скоро по каналу будут передаваться какие-то данные. Исходя из информации, содержащейся в RTS, *C* может предположить, сколько времени займет передача последовательности, включая завершающее подтверждение. Поэтому, чтобы не мешать другим, она воздерживается от передачи данных, пока обмен не будет завершен. Для этого она обновляет свою запись NAV, указывая, что канал занят, как показано на илл. 4.27. Станция *D* не слышит RTS,

зато фиксирует CTS и также выставляет NAV. Обратите внимание: сигналы NAV не передаются, а являются лишь внутренними напоминаниями станций о том, что нужно хранить молчание в течение определенного промежутка времени.

Однако, несмотря на теоретическую привлекательность модели RTS/CTS, это один из тех методов, практическая реализация которых провалилась. Есть несколько причин, почему она используется так редко. Она не рассчитана на короткие фреймы (которые отправляются вместо RTS) и на присутствие точек доступа (которые по определению должны быть слышны всем). В других ситуациях она также замедляет работу. RTS/CTS в стандарте 802.11 немного отличается от протокола MACA, с которым мы познакомились в разделе 4.2, поскольку каждый, кто получает RTS или CTS, сохраняет молчание в течение какого-то промежутка, чтобы подтверждение (ACK) сумело пройти по каналу без коллизий. По этой причине проблема засвеченной станции не решается (как в случае MACA), устраняется только проблема скрытых станций. Чаще всего скрытых станций совсем немного, к тому же технология CSMA/CA и так помогает им. Она замедляет станции, которым по какой-либо причине не удается успешно отправить данные, чтобы повысить вероятность удачной передачи.

CSMA/CA с физическим и виртуальным прослушиванием составляет суть протокола 802.11. Однако есть несколько других механизмов, разработанных для того же стандарта. Создание каждого из них обусловлено определенными потребностями при реальной эксплуатации, так что мы кратко их рассмотрим.

Первая потребность — это надежность. В противоположность проводным каналам, беспроводные полны шума и ненадежны, во многом из-за влияния других устройств (например, СВЧ-печей, работающих в тех же нелицензируемых диапазонах ISM). Подтверждения и повторная отправка не помогут, если вероятность успешной передачи фрейма мала.

Основная стратегия увеличения числа успешных передач заключается в снижении скорости передачи. На низких скоростях используются более надежные методы модуляции сигнала, который с большей вероятностью будет правильно получен при заданном отношении «сигнал/шум». При ощутимой потере фреймов станция понижает скорость. Если фреймы приходят с небольшой потерей, станция периодически повышает скорость, проверяя, можно ли ее использовать.

Еще один способ повысить шанс передачи неповрежденного фрейма состоит в том, чтобы посылать более короткие фреймы. Если вероятность ошибки в одном бите равна $p$, то вероятность того, что $n$-битовый фрейм будет принят корректно, равна $(1 - p)^n$. Например, при $p = 10^{-4}$ шанс корректной передачи полного фрейма Ethernet длиной 12 144 бита составляет менее 30 %. Большая часть фреймов будет потеряна. Но если их длина составит только одну треть (4048 бит), то две трети фреймов (то есть большинство) будут получены правильно, а число повторных передач снизится.

Уменьшения длины фреймов можно добиться, сократив максимальный размер сообщения, которое принимается от сетевого уровня. Кроме того, 802.11 позволяет разделять фреймы на **фрагменты (fragments)**, каждый из которых снабжается отдельной контрольной суммой. Размер фрагмента не фиксирован, а является параметром, который может быть скорректирован точкой доступа. Фрагменты нумеруются и подтверждаются индивидуально с использованием

протокола с ожиданием (то есть отправитель не может передать фрагмент с номером $k + 1$, пока не получит подтверждение о доставке фрагмента с номером $k$). Они идут один за другим с подтверждением (и возможно, с повторной отправкой) между ними, до тех пор пока весь фрейм не будет успешно передан или пока время передачи не достигнет заданного максимума. Представленный выше механизм NAV удерживает станции от передачи только до прихода первого подтверждения о доставке. Но есть и другой механизм (описанный далее), который позволяет получателю принять всю пачку фрагментов, без фреймов от других станций между ними.

Вторая потребность, которую мы обсудим, — экономия энергии. Время работы от аккумулятора для мобильных беспроводных устройств всегда представляет проблему. Стандарт 802.11 решает вопрос управления электропитанием, чтобы клиенты не тратили энергию впустую в отсутствие передачи или приема информации.

Экономия энергии обеспечивается главным образом за счет использования **фреймов-маяков (beacon frames)**. Это периодические широковещательные сообщения, отправляемые точкой доступа (AP), например, каждые 100 мс. Фреймы сообщают клиентам о присутствии AP и содержат системные параметры: идентификатор AP, время, интервал до следующего маяка и настройки безопасности.

Клиенты могут вставить бит управления электропитанием во фреймы, с помощью которых они сообщают точке доступа о переходе в **энергосберегающий режим (power-save mode)**. В этом режиме клиент может «дремать», а AP будет буферизовать предназначенный для него трафик. Чтобы проверить наличие входящего трафика, клиент «просыпается» при каждом приходе маяка и проверяет содержащуюся в нем карту трафика. Эта карта говорит клиенту о наличии буферизованного трафика. Если он есть, клиент посылает сообщение опроса в точку доступа и она передает буферизованный трафик. Затем клиент может вернуться в спящий режим до следующего маяка.

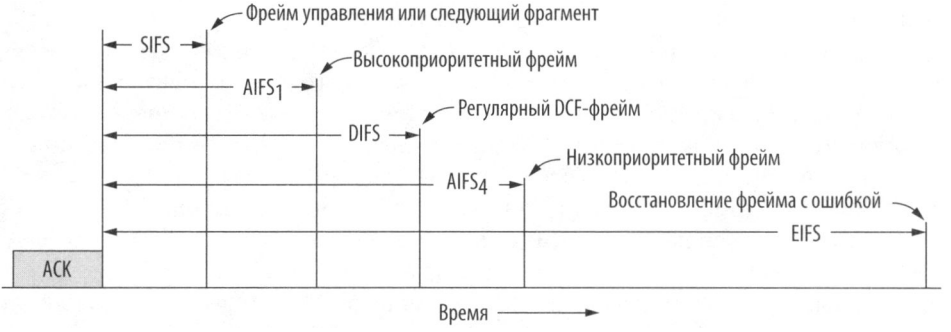
В 2005 году к 802.11 был добавлен другой энергосберегающий механизм, **автоматический переход в режим сохранения энергии (Automatic Power Save Delivery, APSD)**. Точка доступа буферизирует фреймы и посылает их клиенту сразу после того, как он передает ей фреймы. Клиент может перейти в спящий режим, пока у него нет большего количества трафика для отправки (и получения). Этот механизм хорошо работает, например, в IP-телефонии (VoIP), где трафик часто идет в обоих направлениях. Например, беспроводной IP-телефон мог бы использовать APSD, чтобы посылать и получать фреймы каждые 20 мс (это намного чаще, чем интервал маяка в 100 мс), а в промежутках находиться в спящем режиме.

Третья, и последняя, потребность, которую мы исследуем, — это QoS. При конфликте высокоскоростного одноранговой трафика и VoIP-трафика из предыдущего примера пострадает последний. Он будет передаваться с задержками, даже при том, что требования к пропускной способности у IP-телефонии невелики. Эти задержки, вероятно, понизят качество голосовых вызовов. Чтобы предотвратить это, нужно предоставить трафику IP-телефонии более высокий приоритет.

В IEEE 802.11 есть умный механизм, обеспечивающий этот вид QoS. Он был введен в 2005 году как набор расширений под названием 802.11e. Он расширяет CSMA/CA с помощью точно определенных интервалов между фреймами. После отправки фрейма, прежде чем любая станция сможет начать передачу, требуется определенное количество времени простоя, чтобы проверить, что канал больше не занят. Эта уловка должна определить различные временные интервалы для разных видов фреймов.

На илл. 4.28 изображено пять интервалов. Интервал между регулярными фреймами данных называется **DIFS (DCF InterFrame Spacing — межфреймовый интервал DCF)**. Любая станция может попытаться захватить канал, чтобы послать новый фрейм после того, как среда была неактивна для DIFS. При этом действуют обычные правила конкуренции, включая двоичную экспоненциальную выдержку в случае коллизии. Самый короткий интервал — это **SIFS (Short InterFrame Interval — короткий межфреймовый интервал)**. Он используется для того, чтобы одна из сторон в диалоге могла получить шанс начать первой.

Например, можно разрешить получателю отправить ACK или другие последовательности фреймов управления, такие как RTS и CTS, или разрешить отправителю передать пакет фрагментов. Отправка следующего фрагмента только после ожидания SIFS препятствует вмешательству другой станции во время обмена данными.




**Илл. 4.28.** Межфреймовые интервалы в стандарте 802.11

Два интервала **AIFS (Arbitration InterFrame Space — арбитражный межфреймовый интервал)** представляют собой примеры двух различных уровней приоритета. Короткий интервал $AIFS_1$ короче DIFS, но длиннее SIFS. Он может использоваться точкой доступа, чтобы переместить голос или другой приоритетный трафик в начало очереди. AP ждет более короткого интервала, прежде чем отправить голосовой трафик. Таким образом, она передает его раньше регулярного трафика. Длинный интервал $AIFS_4$ больше, чем DIFS. Он используется для фонового трафика, который может быть задержан до окончания передачи регулярного. Прежде чем отправить этот трафик, AP ждет в течение более длинного интервала, позволяя сначала передать регулярный трафик. Полный

механизм QoS определяет четыре приоритетных уровня с различными параметрами выдержки и времени ожидания.

Последний временной интервал называется **EIFS (Extended InterFrame Spacing — расширенный межфреймовый интервал)**. Он используется только той станцией, которая только что получила поврежденный или неопознанный фрейм и хочет сообщить о проблеме. Идея в том, что приемник может не сразу понять, что происходит, и ему нужно выждать какое-то время, чтобы не прервать текущий диалог между станциями.

Еще одна составляющая расширений QoS — понятие **возможности передачи (TXOP, transmission opportunity)**. Первоначальный механизм CSMA/CA позволял станциям посылать один фрейм за раз. Это всех устраивало, пока диапазон скоростей не увеличился. В 802.11a/g одна станция могла отправлять фреймы со скоростью 6 Мбит/с, а другая — 54 Мбит/с. Каждая из них передает один фрейм, но первой станции нужно для отправки в 9 раз больше времени (не считая фиксированных накладных расходов), чем второй. У этого неравенства есть неприятный побочный эффект замедления быстрого отправителя, который конкурирует с медленным отправителем, примерно до скорости последнего. Например, если станции работают по отдельности, их собственные скорости — 6 и 54 Мбит/с (снова без учета накладных расходов). Но работая вместе, они обе получают среднюю скорость 5,4 Мбит/с, что является большой неприятностью для быстрого отправителя. Эта проблема известна как **аномалия скорости (rate anomaly)** (Хойс и др.; Heusse et al., 2003).

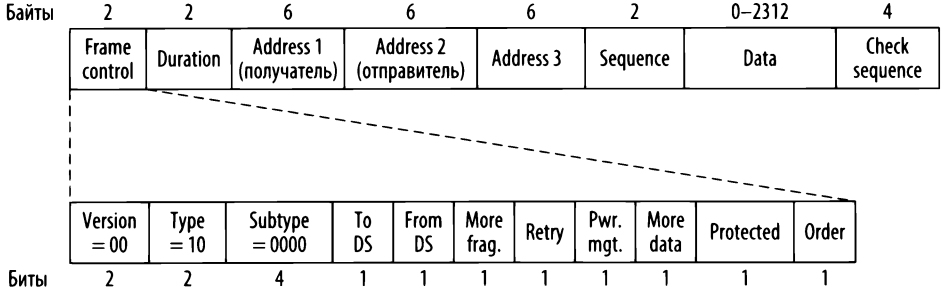
При использовании TXOP все станции получают равное количество времени передачи, а не одинаковое число фреймов. Станции с более высокой скоростью в этом периоде работают с большей пропускной способностью. В нашем примере отправители, совместно работающие со скоростями 6 и 54 Мбит/с, теперь достигнут 3 и 27 Мбит/с соответственно.

## 4.4.4. Стандарт 802.11: структура фрейма

Стандарт 802.11 определяет три класса фреймов, передаваемых по радиоканалу: информационные (фреймы данных), служебные и управляющие. Все они имеют заголовки с множеством полей, используемых подуровнем MAC. Кроме того, существуют поля, предназначенные для физического уровня, но они в основном относятся к методам модуляции, поэтому здесь не представлены.

В качестве примера мы рассмотрим формат фрейма данных. Он показан на илл. 4.29. Вначале идет поле `Frame Control` (Управление фреймом). Оно содержит 11 вложенных полей. Первое — `Protocol version` (Версия протокола), установлено в 00 (2 бита). Именно оно позволит будущим версиям 802.11 работать одновременно в одной ячейке сети. Далее следует поле `Type` (Тип): информационный, служебный или управляющий; затем `Subtype` (Подтип) — например, RTS или CTS. Для обычного фрейма данных (без QoS) они установлены как бинарные 10 и 0000. Биты `To DS` (К DS) и `From DS` (От DS) говорят о направлении движения фрейма: в сеть или из сети, соединенной с точкой доступа; эта сеть называется распределительной. Поле `More fragments` (Дополнительные

фрагменты) сообщает, что далее следует еще один фрагмент. `Retry` (Повтор) маркирует заново отправленный фрейм. `Power management` (Управление питанием) используется станцией-отправителем для указания на переход в режим пониженного энергопотребления или на выход из него. `More data` (Дополнительные данные) говорит о том, что у отправителя имеются еще фреймы для пересылки. Бит `Protected Frame` (Шифрование) является индикатором использования шифрования в теле фрейма в целях безопасности. Вопросы безопасности будут рассмотрены в следующем разделе. Наконец, установленный бит `Order` (Порядок) говорит приемнику о том, что фреймы с этим битом должны обрабатываться строго по порядку.




**Илл. 4.29.** Фрейм данных стандарта 802.11

Второе основное поле фрейма данных — `Duration` (Длительность). В нем задается время (в микросекундах), которое будет потрачено на передачу фрейма и подтверждения. Это поле присутствует во всех типах фреймов, в том числе в служебных, и именно в соответствии с ним станции выставляют признаки NAV.

Далее следуют адреса. Фрейм данных включает три адреса в формате, соответствующем стандарту IEEE 802. Очевидно, что сюда входят адреса отправителя и получателя, но что же содержится в третьем? Следует помнить, что точка доступа — это просто пункт ретрансляции фреймов, когда они движутся между клиентом и другой точкой сети, возможно, удаленным клиентом или интернет-порталом. Третий адрес указывает на эту удаленную конечную точку.

Поле `Sequence` (Последовательность) нумерует фрагменты, что позволяет выявлять дубликаты. Из 16 доступных битов 4 идентифицируют фрагмент, а 12 содержат число, которое растет с каждой новой передачей. В поле `Data` (Данные) находится передаваемая по каналу информация, его длина может достигать 2312 байт. Первые байты этой полезной нагрузки представлены в формате, известном как **подуровень управления логическим соединением (Logical Link Control, LLC)**. LLC — связующий элемент, идентифицирующий протокол более высокого уровня (например, IP), которому нужно передать полезную нагрузку. В конце, как обычно, расположено поле `Frame check sequence` (Контрольная последовательность фрейма). Это такой же 32-битный CRC, который мы встречали в разделе 3.2.2.

Управляющие фреймы имеют тот же формат, что и информационные, но к нему добавляется еще один — для той части данных, которая меняется в зависимости от подтипа (например, параметры во фреймах-маяках). Служебные фреймы короткие. Как и во всех других фреймах, в них содержится `Frame control`, `Duration` и `Frame check sequence`. При этом они могут иметь только один адрес и не иметь поля `Data`. Ключевой здесь является информация, содержащаяся в поле `Subtype` (RTS, CTS или ACK).

## 4.4.5. Службы

Стандарт 802.11 определяет службы, чтобы клиенты, точки доступа и соединяющие их сети могли сформировать согласованную беспроводную LAN. Их можно разделить на несколько категорий.

### *Сопоставление и доставка данных*

**Служба сопоставления (association service)** используется мобильными станциями для подключения к точкам доступа. Обычно она применяется сразу же после вхождения в зону действия AP. По прибытии станция узнает (либо от фреймов-маяков, либо напрямую у AP) идентификационную информацию и характеристики точки доступа (предоставляемая скорость передачи данных, меры безопасности, возможности энергосбережения, поддержка QoS и т. д.). Помимо этого, сообщения-маяки от AP содержат **идентификатор набора служб (Service Set IDentifier, SSID)**. Многие думают, что это имя сети. Станция посылает запрос на сопоставление с точкой доступа, которая может принять или отвергнуть его. В то время как маяки рассылаются всегда, идентификаторы SSID могут не передаваться. Если идентификатор SSID не передается, то станция должна каким-то образом узнать (или обнаружить) имя, закрепленное за точкой доступа.

**Пересопоставление (reassociation)** позволяет станции сменить точку доступа. Оно применяется, когда мобильная станция переходит от одной AP к другой в той же расширенной LAN стандарта 802.11, по аналогии с передачей в сотовой сети. При корректном пересопоставлении такой переход не ведет к потере данных. (Однако, как и в сети Ethernet, в стандарте 802.11 все службы предоставляются лишь с обязательством приложения максимальных усилий к их исполнению.)

Никаких гарантий доставки не дается. По инициативе мобильной станции или точки доступа может быть произведено **снятие сопоставления (disassociate)**, то есть разрыв соединения. Оно требуется при выключении станции или ее уходе из зоны действия AP. Точка доступа может инициировать разрыв соединения, например, если она временно выключается для проведения технического обслуживания. В стандарте 802.11w во фреймы снятия сопоставления была добавлена аутентификация.

Когда фреймы достигают точки доступа, **служба распределения (distribution service)** определяет их маршрутизацию. Если адрес назначения является локальным для данной AP, то фреймы следуют напрямую по радиоканалу. В противном случае их необходимо пересылать по проводной сети. **Служба интеграции**

**(integration service)** обеспечивает любую передачу, если фрейм нужно выслать за пределы сети стандарта 802.11 или если он получен из сети другого стандарта. Типичный случай — соединение между беспроводной LAN и интернетом.

Поскольку главным назначением сетей стандарта 802.11 является обмен данными, они, разумеется, обеспечивают и **службу доставки данных (data delivery service)**. Она позволяет станциям передавать и получать данные по протоколам, которые мы рассмотрели ранее в этой главе. Поскольку стандарт 802.11 основан на стандарте Ethernet, где доставка данных не является гарантированной на 100 %, то для беспроводных сетей это тем более верно. Верхние уровни должны заниматься обнаружением и исправлением ошибок.

### *Безопасность и конфиденциальность*

Прежде чем станции смогут посылать фреймы через точку доступа, они должны пройти **аутентификацию**. В зависимости от выбора схемы безопасности она происходит по-разному. Если сети 802.11 «открыты», их разрешено использовать любому, если нет — для аутентификации нужны параметры учетной записи.

Широко распространенная схема аутентификации **WPA2 (Wi-Fi Protected Access 2 — защищенный доступ Wi-Fi 2)** обеспечивает безопасность, заданную стандартом 802.11i. (WPA — промежуточная схема, которая реализует подмножество 802.11i. Мы пропустим ее и перейдем сразу к полной схеме.) При использовании WPA2 точка доступа может взаимодействовать с сервером аутентификации, у которого есть имя пользователя и база данных паролей, чтобы определить, разрешено ли станции получить доступ к сети. Также может быть сконфигурирован предустановленный ключ (pre-shared key); это необычное название сетевого пароля. Станция и AP обмениваются несколькими фреймами с запросом и ответом, что позволяет станции доказать, что у нее есть правильные учетные данные. Этот обмен происходит после сопоставления.

В корпоративных сетях широко используется другой метод аутентификации, описанный в стандарте **802.1X**, — **аутентификация на основе портов (port-based authentication)**. Стандарт 802.1X подразумевает централизованную аутентификацию (например, когда аутентификация устройств выполняется на центральном сервере), что обеспечивает более тщательный контроль доступа, учет ресурсов, биллинг и идентификацию. Для аутентификации в сети станция (или «проситель») обращается к аутентификатору, который связывается с сервером аутентификации. Стандарт 802.1X использует **расширенный протокол аутентификации (Enhanced Authentication Protocol, EAP)**. Набор EAP содержит более 50 различных методов аутентификации. Отметим наиболее популярные из них: **EAP-TLS** выполняет аутентификацию на основе сертификатов; **EAP-TTLS** и **PEAP** позволяют клиенту использовать различные способы сопоставления, включая аутентификацию на основе паролей; **EAP-SIM** дает мобильному телефону возможность выполнить аутентификацию с помощью SIM-карты. Стандарт 802.1X имеет целый ряд преимуществ по сравнению с простой схемой WPA. В частности, он позволяет более тщательно контролировать доступ на уровне отдельных пользователей, однако для этого необходимо администрировать инфраструктуру сертификатов.

Предшественницей WPA была схема **приватности на уровне проводной связи (Wired Equivalent Privacy, WEP)**. Она подразумевает выполнение аутентификации с предустановленным ключом перед сопоставлением. Позже выяснилось, что схема WEP небезопасна, и в настоящее время она практически не используется. Первая практическая демонстрация взлома WEP произошла, когда Адам Стабблфилд стажировался в AT&T (Stubblefield и др., 2002). Он смог написать код и протестировать атаку за одну неделю, при этом большая часть времени ушла на получение разрешения от администрации на покупку карт Wi-Fi, необходимых для эксперимента. Программное обеспечение для взлома паролей WEP теперь есть в свободном доступе.

После того как схему WEP взломали, а схему WPA признали устаревшей, была предпринята следующая попытка в виде схемы WPA2. В ней используется служба конфиденциальности, которая управляет параметрами кодирования и декодирования. Алгоритм кодирования для WPA2 основан на **улучшенном стандарте шифрования (Advanced Encryption Standard, AES)**. Это американский правительственный стандарт, одобренный в 2002 году. Ключи для шифрования определяются во время процедуры аутентификации. К сожалению, в 2017 году схема WPA2 также была взломана (Ванхоф и Писсенс; Vanhoef and Piessens, 2017). Обеспечение хорошего уровня безопасности — непростая задача даже при наличии не поддающихся взлому алгоритмов шифрования, поскольку самым слабым звеном является управление ключами.[1]

### *Приоритизация и управление питанием*

Для обработки трафика с различными приоритетами используется служба **планирования трафика QoS (QoS traffic scheduling)**. Она применяет протоколы, которые мы описали, чтобы дать голосовому и видеотрафику преимущество перед негарантированным и фоновым трафиком. Сопутствующая служба также обеспечивает синхронизацию более высокого уровня. Это позволяет станциям координировать свои действия, что может быть полезным для обработки мультимедиа.

Наконец, есть две службы, помогающие станциям управлять использованием спектра. **Регулирование мощности передатчика (transmit power control)** дает станциям необходимую информацию для соблюдения установленных нормативных пределов мощности передачи (они варьируются в зависимости от региона). Служба **динамического выбора частоты (dynamic frequency selection)** предоставляет данные, благодаря которым станции могут избежать передачи в диапазоне 5 ГГц (используется радарами).

С помощью этих служб стандарт 802.11 обеспечил богатый набор возможностей, чтобы соединить близко расположенных мобильных клиентов с интернетом. Это был огромный успех. Позже в стандарт неоднократно вносились изменения и добавлялись новые возможностей. Ознакомиться с его историей и перспективами развития можно в работе Херца и др. (Hiertz et al., 2010).

[1] В 2018 году был опубликован стандарт WPA3, обеспечивающий более высокий уровень безопасности. — *Примеч. науч. ред.*

## 4.5. BLUETOOTH

В 1994 году шведская компания Ericsson заинтересовалась вопросом беспроводной связи между мобильными телефонами и другими устройствами (например, ноутбуками). Совместно с четырьмя другими компаниями (IBM, Intel, Nokia и Toshiba) в 1998 году была сформирована «Специальная рабочая группа» (SIG, Special Interest Group). Она занялась развитием стандарта беспроводного соединения компьютеров и устройств связи, а также созданием аксессуаров с недорогими маломощными радиоустройствами ближнего действия. Проект был назван **Bluetooth («Синий зуб»)** в честь великого короля викингов по имени Харальд Синезубый II, который объединил (читай: завоевал) Данию и Норвегию (провода ему действительно не пригодились).

Bluetooth 1.0 появился в июле 1999 года, и с тех пор SIG ни разу не пожалела об этом. Сегодня Bluetooth используется в самых разнообразных пользовательских электронных устройствах — от мобильных телефонов и ноутбуков до наушников, принтеров, клавиатур, мышей, игровых приставок, часов, аудиоплееров, навигационных устройств и т. д. Протоколы Bluetooth позволяют этим устройствам находить друг друга и соединяться с помощью **сопряжения (pairing)**, а затем надежно передавать данные.

За прошедшее десятилетие протоколы претерпели значительные изменения. В 2004 году, после стабилизации первоначальных протоколов, был выпущен Bluetooth 2.0 с более высокими скоростями передачи данных. Версия Bluetooth 3.0 2009 года может использоваться для сопряжения устройств в комбинации с 802.11 для высокоскоростной передачи. В версии 4.0, выпущенной в 2010 году, был введен режим пониженного энергопотребления. Он будет полезен всем, кто не хочет регулярно менять батарейки в устройствах по всему дому.

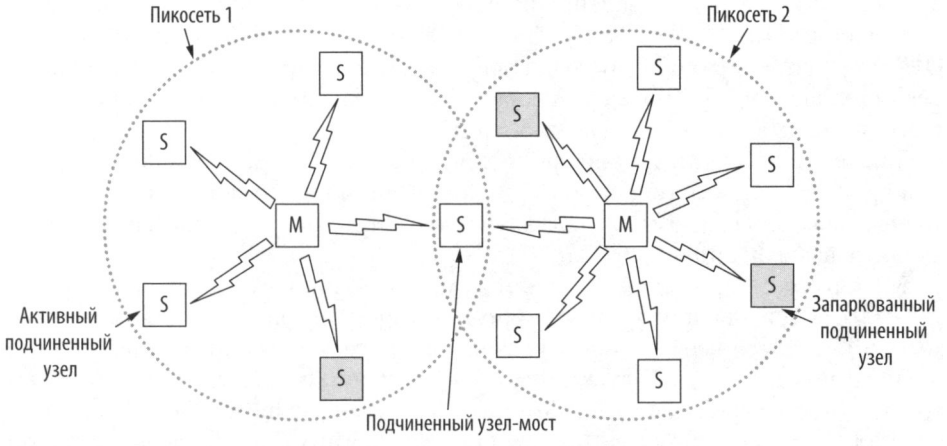
Ниже мы рассмотрим основные аспекты Bluetooth 4.0, поскольку эта версия по-прежнему является наиболее распространенной. Далее мы обсудим версию Bluetooth 5 и ее отличия (в основном незначительные) от предыдущей версии.

### 4.5.1. Архитектура Bluetooth

Начнем изучение системы Bluetooth с краткого обзора ее компонентов и задач. В основе Bluetooth лежит **пикосеть (piconet)**, состоящая из одного главного узла и нескольких (до семи) подчиненных узлов, расположенных в радиусе 10 м. В одной и той же комнате, если она достаточно большая, могут располагаться несколько пикосетей. Более того, они могут даже связываться друг с другом посредством моста (специального узла), как показано на илл. 4.30. Несколько объединенных вместе пикосетей составляют **рассеянную сеть (scatternet)**.

Помимо семи активных подчиненных узлов, пикосеть может включать до 255 так называемых запаркованных узлов. Это устройства, переведенные в режим пониженного энергопотребления главным узлом, — за счет этого продлевается ресурс их источников питания. В таком режиме узел может только отвечать на запросы активации или на сигнальные последовательности от главного узла. Также существует два промежуточных режима энергопотребления — приостановленный и анализирующий.

Такая архитектура оказалась очень простой и дешевой в реализации (микросхема Bluetooth стоила менее $5), а именно этого разработчики и добивались. В результате подчиненные узлы получились довольно примитивными — они лишь выполняют то, что им прикажет главный узел. По сути, пикосеть — это централизованная система TDM, в которой главный узел контролирует временные слоты для передачи данных и распределяет их между подчиненными узлами. Обмен происходит только между подчиненным и главным узлами. Прямой связи между подчиненными узлами нет.




**Илл. 4.30.** Две пикосети могут объединиться в рассеянную сеть

## 4.5.2. Применение Bluetooth

Большинство сетевых протоколов просто предоставляют каналы для обмена данными между сущностями и оставляют их прикладное использование на усмотрение разработчиков. Например, в стандарте 802.11 не говорится, что пользователи должны использовать свои ноутбуки для чтения электронной почты, работы в интернете и т. п. Bluetooth SIG, напротив, специфицирует отдельные сценарии применения и для каждого из них предоставляет свой набор протоколов. На момент написания данного раздела было 25 таких сценариев, называемых **профилями (profiles)**. К сожалению, это значительно усложняет систему. Мы опустим детали и кратко рассмотрим профили, чтобы понять, чего пыталась достичь с их помощью группа Bluetooth SIG.

Шесть профилей предназначены для различного использования аудио и видео. К примеру, профиль Intercom (ICP) позволяет двум телефонам соединяться друг с другом по принципу рации. Профили Headset (HSP) и Hands-free (HFP) обеспечивают голосовую связь беспроводной гарнитуры с базовой станцией. Это позволяет, например, совершать звонки за рулем автомобиля. Существуют профили для потоковой передачи стереозвука и видео, например, с портативного аудиоплеера на наушники или с цифровой камеры на телевизор.

Профиль HID (human interface device — человеко-машинный интерфейс) предназначен для подключения клавиатуры и мыши к компьютеру. Другие профили позволяют мобильному телефону или другому устройству получать изображение с камеры или отправлять их на принтер. Пожалуй, более интересен профиль, с помощью которого можно использовать мобильный телефон в качестве пульта дистанционного управления для телевизора с поддержкой Bluetooth.

Следующая группа профилей имеет отношение к сетям. Профиль доступа к персональной сети (PAN) позволяет устройствам Bluetooth сформировать произвольную сеть или удаленно подключиться через точку доступа к другой сети, например к 802.11 LAN. Профиль удаленного доступа (dial-up networking, DUN) представляет собой то, ради чего изначально был задуман весь проект: с его помощью ноутбук может установить беспроводное соединение с мобильным телефоном, имеющим встроенный модем.

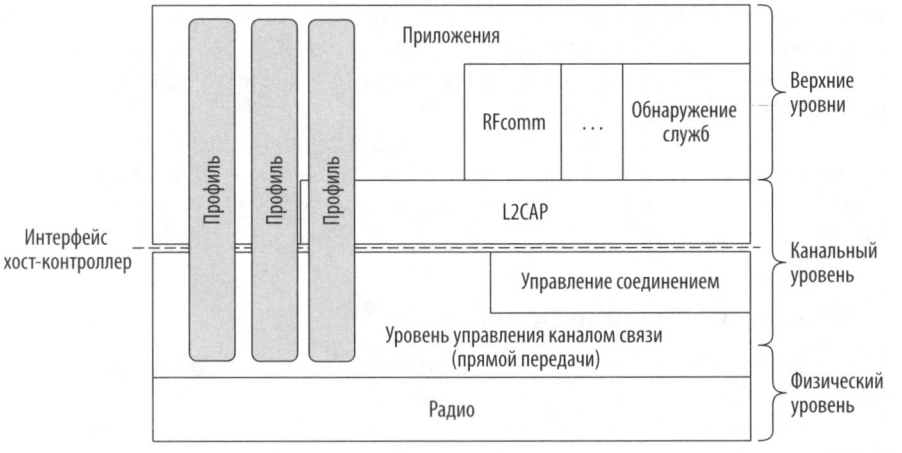
Также были определены профили для обмена информацией на более высоком уровне. В частности, профиль синхронизации предназначен для загрузки данных в мобильный телефон, когда его владелец выходит из дома, и извлечения их после возвращения.

Мы опустим остальные профили, заметим только, что некоторые из них служат основой для профилей, упомянутых выше. Профиль общего доступа (на котором строятся все остальные) устанавливает и обслуживает защищенные соединения (каналы) между главным и подчиненным узлами. Другие базовые профили задают основы обмена объектами и передачи аудио и видео. Служебные профили широко используются, например, для эмуляции последовательной шины данных, что особенно полезно при работе со многими устаревшими приложениями.

Неужели действительно так необходимо подробно описывать в стандарте все сценарии применения и предоставлять наборы протоколов для каждого из них? Вероятно, нет. Однако было создано множество рабочих групп, которые рассматривали различные стороны стандарта. Каждая группа разработала свой профиль. Считайте это демонстрацией закона Конвея в действии. (В апреле 1968 года в журнале *Datamation* была опубликована статья Мелвина Конвея (Melvin Conway), в которой утверждалось, что если поручить написание компилятора $n$ программистам, то получится $n$-проходный компилятор. В более общем виде эта мысль звучит так: структура программного обеспечения отражает структуру группы разработчиков.) Наверное, можно было обойтись двумя наборами протоколов (вместо 25) — один для передачи файлов и один для обмена данными в реальном времени.

## 4.5.3. Стек протоколов Bluetooth

Стандарт Bluetooth содержит множество протоколов, условно разбитых на уровни, как показано на илл. 4.31. Сразу можно заметить, что структура не соответствует ни OSI, ни TCP/IP, ни 802, ни какой-либо другой известной модели.




**Илл. 4.31.** Архитектура протоколов Bluetooth

Внизу находится физический уровень радиосвязи (что вполне соответствует моделям OSI и 802), на котором описывается радиопередача и методы модуляции. Многое здесь направлено на то, чтобы удешевить систему, сделав ее доступной для массового покупателя.

Уровень управления каналом связи (немодулированной передачи) чем-то напоминает подуровень MAC, но содержит и некоторые элементы физического уровня. Здесь описывается, как главный узел контролирует временные слоты и как они группируются во фреймы.

Далее следуют два протокола, использующие протокол управления каналом связи. Протокол управления соединениями устанавливает логические каналы между устройствами, контролирует энергопотребление, сопряжение и шифрование, а также QoS. Он находится ниже линии интерфейса хост-контроллера. Этот интерфейс нужен для удобства реализации: как правило, протоколы ниже линии выполняются на чипе Bluetooth, а выше — на устройстве Bluetooth, где чип размещен.

Над линией находится протокол канального уровня — **L2CAP (Logical Link Control and Adaptation Protocol — протокол управления логическими связями и сопоставлениями)**. Он формирует сообщения переменной длины и при необходимости обеспечивает надежность передачи. L2CAP используется множеством протоколов, в том числе и двумя описанными ранее служебными протоколами. Протокол обнаружения служб используется для определения их местонахождения в пределах сети. Протокол RFcomm (Radio Frequency communication — радиочастотная передача) эмулирует работу стандартного последовательного порта ПК, к которому обычно подключаются клавиатура, мышь, модем и другие устройства.

На самом верхнем уровне находятся приложения. Профили представлены вертикальными прямоугольниками, потому что каждый из них определяет часть стека протокола для конкретной цели. Специфические профили, например

профили для гарнитур, используют только те протоколы, которые необходимы для их работы. Например, профили могут включать L2CAP, если у них есть пакеты для отправки, но пропустить его, если имеется только постоянный поток звуковых сэмплов.

В следующих разделах мы рассмотрим уровень радиосвязи и различные протоколы канального уровня Bluetooth, поскольку они пусть грубо, но все-таки соответствуют физическому уровню и подуровню MAC в других изученных нами стеках протоколов.

## 4.5.4. Bluetooth: уровень радиосвязи

Уровень радиосвязи переносит информацию бит за битом от главного узла к подчиненным и обратно. Это маломощная приемопередающая система с радиусом действия порядка 10 м. Она работает в ISM-диапазоне 2,4 ГГц, как и 802.11. Диапазон разделен на 79 каналов по 1 МГц в каждом. Чтобы обеспечить сосуществование с другими сетями в ISM-диапазоне, применяется расширенный спектр со скачкообразной перестройкой частоты. Возможны до 1600 скачков частоты в секунду, то есть минимальный временной слот для передачи равен 625 мкс. Все узлы пикосетей перестраивают частоты одновременно, в соответствии с синхронизацией слотов и псевдослучайной последовательностью скачков, генерируемой главным узлом.

К сожалению, оказалось, что ранние версии Bluetooth и 802.11 интерферируют настолько, что нарушают передачи друг друга. По этой причине некоторые компании полностью отказались от Bluetooth, но в конечном итоге техническое решение было найдено. Оно заключалось в том, чтобы адаптировать последовательность скачков для исключения каналов, на которых есть другие радиосигналы. Этот процесс, названный **адаптивной перестройкой рабочей частоты (adaptive frequency hopping)**, снижает помехи.

Для отправки битов по каналу используются три формы модуляции. В базовой схеме применяется кодирование со сдвигом частоты, чтобы отправлять 1-битный символ каждую микросекунду. Это дает общую скорость передачи данных 1 Мбит/с. Начиная с версии Bluetooth 2.0, скорость увеличилась благодаря кодированию со сдвигом фазы. Общая скорость передачи достигает 2 или 3 Мбит/с за счет отправки 2 или 3 бит за символ. Увеличенная скорость применяется только для фреймов данных.

## 4.5.5. Bluetooth: канальный уровень

Уровень управления каналом связи (немодулированной передачи) — это наиболее близкий к MAC-подуровню элемент иерархии Bluetooth. Он трансформирует простой поток битов во фреймы и определяет некоторые ключевые форматы. В самом простом варианте главный узел каждой пикосети задает последовательности временных интервалов по 625 мкс, причем передача данных со стороны главного узла начинается в четных слотах, а со стороны подчиненных узлов — в нечетных. Эта схема, по сути, представляет собой традиционный TDM,

где главный узел получает одну половину слотов, а подчиненные делят между собой вторую. Фреймы могут быть длиной 1, 3 или 5 слотов. В каждом из них 126 служебных битов отведено на код доступа и заголовок. Время установления сигнала занимает 250–260 мкс на скачок частоты, чтобы позволить дешевым радиоустройствам стабилизироваться. Полезные данные фрейма могут быть зашифрованы в целях конфиденциальности. Ключ шифрования выбирается, когда ведущее устройство соединяется с ведомым. Переключение частоты происходит только между фреймами, но не в момент передачи. В результате передача пятислотового фрейма намного более эффективна, чем однослотового, потому что при тех же затратах отправляется больше данных.

Протокол управления соединениями устанавливает логические каналы — **соединения (links)**. В них происходит обмен фреймами между обнаружившими друг друга устройствами (главным и подчиненным). Прежде чем использовать соединение, два устройства проходят процедуру сопряжения. Первоначальный метод сопряжения заключается в настройке обоих устройств с помощью одного и того же четырехзначного PIN-кода (Personal Identification Number — личный идентификационный номер). Совпадение PIN-кода показывает, что соединение установлено с нужным удаленным устройством. Но лишенные воображения пользователи и заданные по умолчанию коды типа «0000» или «1234» делают этот метод небезопасным на практике.

Новый метод, **безопасное простое сопряжение (secure simple pairing)**, позволяет пользователям подтвердить, что оба устройства показывают один и тот же ключ, или увидеть ключ на одном устройстве и ввести его на втором. Этот метод более надежный, поскольку пользователи не должны выбирать или устанавливать PIN. Они просто подтверждают ключ, более длинный и сгенерированный самим устройством. Конечно, этот метод не может использоваться на некоторых устройствах с ограниченным вводом/выводом, таких как беспроводные гарнитуры.

Когда сопряжение завершено, протокол устанавливает соединения. Для передачи полезной нагрузки (пользовательских данных) используются два основных типа соединений. Первый тип — **синхронный с установлением связи (Synchronous Connection Oriented, SCO)**. Он предназначен для передачи данных в реальном времени, например при телефонных разговорах. Такой тип канала получает фиксированный временной слот для передачи в каждом направлении. У подчиненного узла может быть до трех соединений SCO с главным узлом; каждое из них представляет собой аудиоканал PCM с пропускной способностью 64 000 бит/с. Из-за ограниченного времени передачи фреймы, переданные по каналу SCO, никогда не отправляются заново. Вместо этого для повышения надежности может быть использована FEC.

Другой тип соединения — **асинхронный без установления связи (Asynchronous Connectionless, ACL)**. Он используется для коммутации пакетов данных, которые могут появиться в произвольный момент времени. Трафик ACL доставляется по принципу максимальных усилий без гарантий. Фреймы могут теряться и пересылаться повторно. У подчиненного узла может быть только одно ACL-соединение с главным узлом.

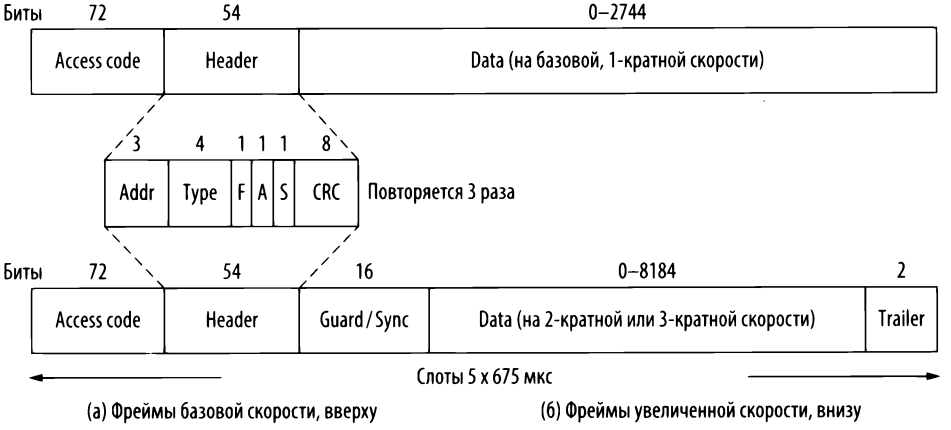
Данные, отправляемые по ACL-каналу, приходят с уровня L2CAP. Этот уровень выполняет четыре основные функции. Во-первых, он принимает пакеты размером до 64 Кбайт с верхних уровней и разбивает их на фреймы для передачи по физическому каналу. На противоположном конце этот же уровень используется для обратного действия — объединения фреймов в пакеты. Во-вторых, он мультиплексирует и демультиплексирует многочисленные источники пакетов. После сборки пакета L2CAP определяет, куда его следует направить (например, протоколу RFcomm или протоколу обнаружения служб). В-третьих, он обеспечивает контроль ошибок и повторную передачу фреймов. L2CAP выявляет ошибки и заново отправляет неподтвержденные пакеты. Наконец, L2CAP следит за соблюдением требований QoS между несколькими соединениями.

## 4.5.6 Bluetooth: структура фрейма

Существует несколько форматов фреймов Bluetooth, наиболее важный из них показан в двух вариантах на илл. 4.32. В начале фрейма указывается `Access code` (Код доступа), который обычно служит идентификатором главного узла. С его помощью подчиненные узлы, находящиеся в радиусе действия двух главных, различают, кому из них предназначаются данные. Затем следует `Header` (Заголовок) из 54 бит, в котором содержатся поля, характерные для подуровня MAC. Если фрейм отправляется с базовой скоростью, далее расположено поле данных. Его размер ограничен 2744 битами (для передачи за пять слотов). Если фрейм соответствует одному слоту, то формат остается таким же, но поле данных в этом случае составляет 240 бит.

При увеличенной скорости блок данных, `Data`, может быть в 2 или 3 раза больше, потому что каждый символ переносит 2 или 3 бита вместо одного. Данным предшествуют охранное поле `Guard` и паттерн синхронизации `Sync`, для переключения на более высокую скорость передачи. Таким образом, код доступа и заголовок передаются на базовой скорости, и только полезные данные передаются на большей скорости. Такие фреймы заканчиваются коротким полем `Trailer`.

Рассмотрим, из чего состоит обычный заголовок фрейма. Поле `Address` указывает на одно из восьми активных устройств, которому предназначена информация. Поле `Type` определяет тип передаваемого фрейма (ACL, SCO, опрос или пустой фрейм), метод коррекции ошибок и количество слотов, из которых состоит фрейм. Бит потока `F (Flow)` добавляется подчиненным узлом — так он сообщает о том, что его буфер заполнен. Этот бит обеспечивает примитивную форму управления потоком. Бит подтверждения `A (Acknowledgement)` используется для отправки вместе с фреймом вложенного подтверждения (ACK). Бит последовательности `S (Sequence)` применяется для нумерации фреймов, чтобы обнаруживать повторные передачи. Это протокол с ожиданием, поэтому одного бита действительно оказывается достаточно. Далее следует `Checksum` — 8-битная контрольная сумма заголовка. Весь 18-битный заголовок фрейма повторяется трижды, что в итоге составляет 54 бита, как показано на илл. 4.32. На принимающей стороне простая схема анализирует все три копии каждого бита. Если они совпадают, бит принимается. В противном случае все решает большинство.




**Илл. 4.32.** Типичный информационный фрейм Bluetooth. (а) На базовой скорости. (б) На увеличенной скорости

Таким образом, на передачу 10 бит заголовка тратится 54 бита пропускной способности. Причина проста: за передачу данных с помощью дешевых, маломощных устройств (2,5 мВт) с низкой вычислительной способностью приходится платить большой избыточностью.

Для ACL- и SCO-фреймов применяются различные форматы поля данных. Самый простой пример — SCO-фрейм с базовой скоростью: длина его поля данных всегда равна 240 бит. Возможны три варианта: 80, 160 или 240 бит полезной информации. При этом оставшиеся биты поля данных используются для коррекции ошибок. В самой надежной версии (80 бит полезной информации) одно и то же содержимое повторяется три раза (что и составляет 240 бит), как и в заголовке фрейма.

Пропускная способность вычисляется следующим образом. Подчиненный узел может использовать только нечетные временные слоты, поэтому ему достается 800 слотов в секунду. Столько же получает и главный узел. При 80 битах полезных данных, передающихся в одном фрейме, емкость канала подчиненного узла (так же, как и главного) равна 64 000 бит/с. Этого как раз хватает для организации полнодуплексного PCM-канала голосовой связи (именно поэтому 1600 скачков в секунду было выбрано в качестве скорости перестройки частот). Другими словами, несмотря на изначальную пропускную способность в 1 Мбит/с, полнодуплексный несжатый голосовой канал может вызвать перегрузку пикосети. Эффективность 13 % — результат затрат 41 % емкости на стабилизацию, 20 % на заголовки и 26 % на повторное кодирование. Этот недостаток подчеркивает значение ускорения и фреймов, содержащих более одного слота.

## 4.5.7. Bluetooth 5

В июне 2016 года группа Bluetooth SIG опубликовала версию Bluetooth 5. В январе 2019 года был выпущен Bluetooth 5.1. Это довольно небольшие изменения

по сравнению с Bluetooth 4. И все же между Bluetooth 4 и Bluetooth 5 имеется ряд различий. Вот список ключевых обновлений в версии Bluetooth 5.0:

1. Поддержка IoT.
2. Скорость возросла с 1 до 2 Мбит/с.
3. Размер сообщений увеличился с 31 до 255 байт.
4. Радиус действия в помещениях увеличился с 10 до 40 м.
5. Немного уменьшилось потребление энергии.
6. Слегка увеличился радиус действия маяков.
7. Несколько повысился уровень безопасности.

Нельзя назвать это существенными переменами, но их и не следовало ожидать, учитывая необходимость в обеспечении обратной совместимости. Стандарт Bluetooth 5.1 внес еще несколько небольших обновлений, касающихся отслеживания устройств, кэширования и некоторых других незначительных моментов.

## 4.6. DOCSIS

Хотя изначально сети кабельного телевидения предназначались для трансляции телевизионных программ, теперь они также широко используются в качестве альтернативного способа подключения квартир и домов к интернету вместо телефонных сетей. Далее мы рассмотрим «подуровень MAC» стандарта DOCSIS, который используется большинством провайдеров кабельного телевидения.

### 4.6.1. Общие сведения

Спецификация DOCSIS также предусматривает наличие подуровня MAC, однако он не так явно отделен от канального уровня по сравнению с другими протоколами, которые мы изучали в предыдущих главах. В то же время в DOCSIS выполняется целый ряд стандартных задач подуровня MAC, включая распределение канала (путем удовлетворения запросов), настройку QoS и уникальную модель пересылки. В данном разделе мы коснемся каждой из этих трех задач. В последних полнодуплексных версиях стандарта DOCSIS 3.1 и DOCSIS 4.0 были введены новые методы планирования и исключения помех.

DOCSIS использует стандартный формат фреймов MAC. Помимо прочего, он содержит поля с данными о длине фрейма и контрольной сумме, а также расширенное поле заголовка для поддержки множества функций, включая защиту канального уровня. Некоторые заголовки поддерживают специальные функции, например синхронизацию входящего потока, регулирование мощности исходящей передачи, запрашивание пропускной способности и конкатенацию фреймов. Один из особых типов фрейма — фрейм запроса; с его помощью кабельный модем запрашивает пропускную способность (как будет описано далее).

## 4.6.2. Пристрелка

Кабельный модем передает **запрос на пристрелку (ranging request)**. Это позволяет **головной станции (CMTS)** определить величину сетевой задержки для кабельного модема, а также выполнить необходимые настройки мощности. Пристрелка — это, по сути, периодическая настройка различных параметров передачи, в частности синхронизации, частоты и мощности. Получив от CMTS сигнал опроса, кабельный модем отправляет ей запрос на пристрелку. На основе этого сообщения CMTS выдает ответ, позволяющий модему настроить параметры синхронизации и мощности сигнала. По умолчанию запрос производится с интервалом в 30 с, но его можно уменьшить; обычно его величина составляет от 10 до 20 с.

## 4.6.3. Распределение пропускной способности каналов

Для распределения пропускной способности между всеми кабельными модемами головная станция DOCSIS использует процесс удовлетворения запросов. Обычно каждый исходящий или входящий поток трафика получает служебный поток, для которого CMTS выделяет некоторую полосу.

### *Служебные потоки*

Выделение каналов в DOCSIS обычно предполагает их распределение между CMTS и одним или несколькими кабельными модемами в домах абонентов. CMTS должна обслуживать все исходящие и входящие каналы и отбрасывать любые фреймы, исходный MAC-адрес которых не принадлежит модемам в группе. Центральную роль в MAC-уровне стандарта DOCSIS играет концепция **служебного потока (service flow)**. Он позволяет обеспечить управление QoS для исходящих и входящих потоков. С каждым кабельным модемом ассоциируются идентификаторы одного или нескольких служебных потоков, которые выбираются на этапе регистрации модема. На каждый поток могут накладываться разные ограничения в зависимости от типа трафика. Так, служебный поток может иметь ограничение по размеру пакета или предназначаться только для определенных приложений, например, с постоянной скоростью потока. Любой кабельный модем должен поддерживать минимум по одному исходящему и входящему служебному потоку, которые называются «основными».

### *Процесс удовлетворения запросов и DOCSIS с низкой задержкой*

Когда у кабельного модема появляются данные для передачи, он отправляет CMTS короткий запрос с информацией о количестве этих данных. Затем он ждет сообщения о выделении пропускной способности с описанием доступных отправителю возможностей для исходящей передачи.

Распределение полосы обеспечивается путем разделения исходящей передачи на дискретные интервалы, называемые **мини-слотами (minislots)**. Мини-слот — это просто подходящая в данном случае единица времени для исходящего

трафика, обычно с шагом увеличения 6,25 мкс. В первых версиях стандарта DOCSIS размер мини-слота должен был равняться произведению минимального приращения на степень двойки, однако в последних версиях это ограничение было снято. Регулируя размер мини-слотов, предоставленных конкретному служебному потоку, CMTS может тем самым обеспечивать QoS и приоритизацию для различных потоков трафика.

Как правило, QoS позволяет CMTS выделять больше пропускной способности отдельным кабельным модемам (предоставляя более качественный сервис абонентам, относящимся к более высокому классу обслуживания). В последних версиях стандарта DOCSIS также появилась возможность предоставления дифференцированного обслуживания для приложений, чувствительных к задержкам. В частности, новая версия протокола DOCSIS позволяет снизить величину задержки за счет использования спецификации под названием **DOCSIS с низкой задержкой (Low-Latency DOCSIS, LLD)**. LLD учитывает то, что для многих интерактивных приложений (например, игр и видеоконференций) низкая задержка столь же важна, как и высокая пропускная способность. В существующих сетях DOCSIS значение задержки для некоторых потоков часто может быть достаточно большим, что может объясняться затратами времени на получение доступа к среде передачи данных и на формирование очереди.

LLD решает эти проблемы, уменьшая циклическую задержку, связанную с процессом удовлетворения запросов, а также используя две очереди (одна для чувствительного к задержкам трафика приложений, вторая — для нечувствительного). Уменьшение задержки процесса удовлетворения запросов позволяет CMTS сократить длительность планирования с 2–4 мс до 1 мс. Для полного устранения задержки, вызываемой удовлетворением запросов, LLD использует механизмы проактивного планирования выделения каналов сервисным потокам. LLD позволяет приложениям указать, имеются ли у них пакеты, которые нельзя поставить в очередь, используя маркировку поля дифференцированного обслуживания во фрейме DOCSIS. Более подробные сведения о LLD можно найти в работе Уайта (White, 2019).

## 4.7. КОММУТАЦИЯ НА КАНАЛЬНОМ УРОВНЕ

У многих организаций есть несколько LAN, которые желательно объединить. Может быть, удобно соединить их в одну большую LAN? Это можно сделать с помощью специальных устройств, называемых **мостами (bridges)**. Коммутаторы Ethernet, описанные в разделе 4.3.4, — это современное название мостов. Они обеспечивают функциональность, которая выходит за рамки классического Ethernet и концентраторов Ethernet, — дают возможность соединить несколько LAN в большую и быструю сеть. Мы будем использовать термины «мост» и «коммутатор» как синонимы.

Мосты работают на канальном уровне. Они анализируют адреса, содержащиеся во фреймах этого уровня, и в соответствии с ними осуществляют маршрутизацию. Поскольку мосты не проверяют у фрейма поле `Payload`, они одинаково хорошо справляются как с пакетами IP, так и с другими типами пакетов, например

AppleTalk. В отличие от мостов, *маршрутизаторы* анализируют адреса в пакетах и действуют, основываясь на этой информации, поэтому они могут работать только с теми протоколами, для которых предназначены.

В этом разделе мы изучим работу мостов и объединение с их помощью нескольких физических LAN в одну логическую. Кроме того, мы рассмотрим обратный процесс — разделение одной физической LAN на несколько логических, так называемых виртуальных LAN. Обе технологии предоставляют полезную гибкость в управлении сетями. Подробную информацию о мостах, коммутаторах и связанных с ними вопросами можно найти в работах Перлман (Perlman, 2000) и Юй (Yu, 2011).

## 4.7.1. Применение мостов

Прежде чем перейти к обсуждению мостов, рассмотрим несколько распространенных случаев их применения. Есть три причины, по которым в организации может появиться несколько LAN.

Во-первых, у многих подразделений университетов или корпораций есть свои LAN, соединяющие персональные компьютеры, серверы и другие устройства (например, принтеры). У разных факультетов и отделов свои цели, поэтому они создают собственные сети независимо друг от друга. Однако рано или поздно возникает потребность в коммуникации, и здесь на помощь приходят мосты. В данном примере несколько отдельных LAN образовалось из-за автономности их владельцев.

Во-вторых, организация может размещаться в нескольких зданиях на значительном расстоянии друг от друга. Скорее всего, выгоднее создать несколько отдельных LAN и затем соединить их с помощью мостов и нескольких магистральных оптоволоконных кабелей, вместо того чтобы протягивать все кабели к одному центральному коммутатору. Даже если кабели легко проложить, их длина ограниченна (например, максимум для витой пары Gigabit Ethernet составляет 200 м). Сеть не будет работать с более длинными кабелями из-за чрезмерного ослабления или задержки сигнала. Единственное решение — разделить LAN и соединить ее части с помощью мостов, увеличив общее физическое расстояние, на котором может работать сеть.

В-третьих, иногда необходимо логически разделить одну LAN на несколько отдельных сетей, соединенных мостами, чтобы снизить нагрузку. Например, во многих крупных университетах в сети объединены тысячи рабочих станций, на которых работают студенты и сотрудники. В компании также могут работать тысячи людей. Огромные масштабы не позволяют объединить все рабочие станции в одну локальную сеть — компьютеров больше, чем портов в любом Ethernet-концентраторе, а станций больше, чем может быть в одной классической сети Ethernet.

Даже если удастся соединить друг с другом все рабочие станции, добавление дополнительных станций в концентратор Ethernet или классическую сеть Ethernet не приведет к увеличению емкости. Все они будут совместно использовать ту же фиксированную полосу пропускания. Чем больше станций подключится, тем меньше будет средняя пропускная способность каждой из них.

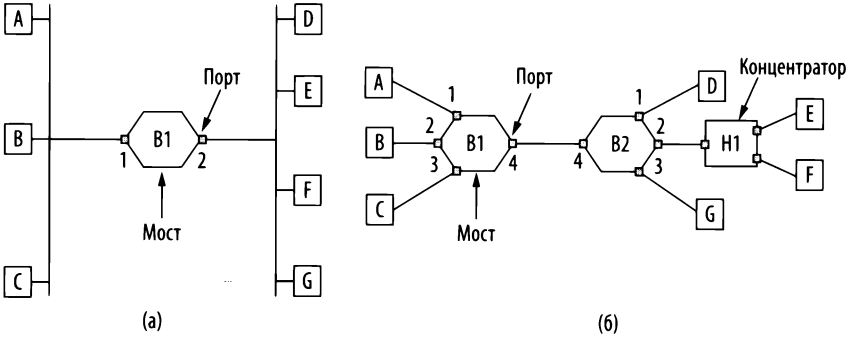
С другой стороны, емкость двух отдельных LAN в два раза больше, чем одной сети. Мосты позволяют объединять сети, сохраняя их пропускную способность. Идея в том, чтобы не передавать трафик на порты, на которых он не требуется, тогда каждая LAN сможет работать на максимальной скорости. Данный принцип также повышает надежность. Поврежденный узел, непрерывно передающий поток бессмысленных данных, может нарушить работу всей сети. Решая, что пересылать, а что нет, мост действует как пожарные двери в здании, защищая всю систему от разрушения из-за одного ненормального узла.

Для наилучшего результата мосты должны быть полностью прозрачными. В идеале мы покупаем их, подключаем кабели LAN, и все сразу же идеально работает — без каких-либо изменений оборудования или ПО, без присвоения коммутаторам адресов, скачивания таблиц маршрутизации и т. п. Мы просто все соединяем и уходим по своим делам. Более того, необходимо, чтобы наличие мостов вообще не влияло на работу существующих LAN. Не должно быть заметной разницы, какой LAN (с мостами или без) принадлежат станции, — они должны легко перемещаться в любой из них.

Удивительно, но создать такие мосты возможно — для этого используются два алгоритма. Алгоритм обратного обучения останавливает отправку трафика туда, где он не нужен; алгоритм связующего дерева прерывает циклы, которые могут возникнуть при непроизвольном соединении коммутаторов. Далее мы разберем эти алгоритмы по очереди, чтобы понять, как они достигают целей.

## 4.7.2. Обучающиеся мосты

Топология двух LAN, соединенных мостом, показана на илл. 4.33 для двух случаев. Слева к двум многоточечным LAN (таким, как классический Ethernet) присоединяется специальная станция — мост, он является элементом обеих сетей, справа — несколько LAN с соединениями «точка-точка» и один подключенный концентратор. Мосты — устройства, к которым присоединены станции и концентратор. Если LAN является сетью Ethernet, мосты называются Ethernet-коммутаторами.




**Илл. 4.33.** Мосты. (а) Мост, соединяющий две многоточечные LAN. (б) Мосты (и концентратор), соединяющие семь станций по двухточечной схеме

Мосты были разработаны во времена классического Ethernet, поэтому они часто изображаются в топологии с многоточечными кабелями, как на илл. 4.33 (а). Однако все топологии, которые можно встретить сейчас, состоят из двухточечных кабелей и коммутаторов. Мосты работают одинаково в обоих случаях. Все станции, присоединенные к одному порту моста, принадлежат к одной области коллизий, которая отличается от областей коллизий других портов. Если система содержит более одной станции (как в классическом Ethernet) и концентратор или полудуплексный канал, для отправки фреймов используется протокол CSMA/CD.

Однако LAN с мостовыми соединениями строятся по-разному. Чтобы объединить многоточечные LAN, мост добавляется в качестве новой станции для каждой из них, как показано на илл. 4.33 (а). В случае двухточечных LAN мост соединяется с концентраторами либо (что более предпочтительно) заменяет их, чтобы увеличить производительность. На илл. 4.33 (б) мосты заменили все концентраторы, кроме одного.

К одному мосту могут быть присоединены разные виды кабелей. Например, мосты *B1* и *B2* на илл. 4.33 (б) могут соединяться магистральным оптоволоконным каналом, а кабель между мостами и станциями может оказаться витой парой ближней связи. Такая структура полезна для соединения LAN различных зданий.

Теперь обсудим, что же происходит в мостах. Каждый мост работает в неизбирательном режиме, то есть принимает каждый фрейм от станций, подключенных к его портам. При появлении фрейма мост должен решить, игнорировать его или передать, и если передать, то на какой порт. Выбор производится на основе адреса получателя. Например, рассмотрим топологию на илл. 4.33 (а). Если станция *A* отправит фрейм станции *B*, мост *B1* получит его на порте 1. От этого фрейма можно немедленно отказаться без дальнейших хлопот, потому что он уже находится на правильном порте. Теперь предположим, что в топологии на илл. 4.33 (б) *A* посылает фрейм станции *D*. Мост *B1* получит его на порте 1 и выведет его на порт 4. Затем мост *B2* примет фрейм на своем порте 4 и выведет его на порте 1.

Для простой реализации этой схемы нужно, чтобы мост имел большую хеш-таблицу. Она может включать все возможные пункты назначения и то, к какому порту каждый из них относится. Например, на илл. 4.33 (б) в таблице моста *B1* станция *D* была бы привязана к порту 4. Таким образом, мост *B1* знал бы, на какой порт следует отправить фреймы для *D*. Фактически дальнейшая пересылка происходит позже, когда достигший моста *B2* фрейм уже не представляет интереса для моста *B1*.

Когда мосты включаются первый раз, все их хеш-таблицы пусты. Ни один мост не знает, где находятся адресаты, поэтому применяется **алгоритм лавинной адресации (flooding)**: каждый полученный фрейм с неизвестным адресом переправляется сразу по всем направлениям, кроме того, откуда он пришел. Со временем мосты узнают расположение получателей. Если адрес известен, лавинная адресация не используется и фреймы направляются только на нужный порт.

Мосты используют алгоритм **обратного обучения (backward learning)**. Как уже упоминалось, они работают в неизбирательном режиме, поэтому видят все

фреймы, приходящие на их порты. Просматривая адреса отправителей, они определяют, какая станция относится к конкретному порту. Например, если мост *B1* на илл. 4.33 (б) получает фрейм от станции *C* на порт 3, то он понимает, что станция *C* привязана к порту 3, и записывает это в таблицу. Любой последующий фрейм для станции *C*, полученный мостом *B1* по любому другому порту, будет переправляться на порт 3.

Топология сети может меняться по мере того, как отдельные станции и мосты включаются, выключаются, а также перемещаются. Чтобы зафиксировать эту динамику, в таблице указывается время прибытия фрейма от станции. При получении новых фреймов это время обновляется. Таким образом, для каждой станции известно время последнего полученного от нее фрейма.

Время от времени процесс сканирует хеш-таблицу и удаляет все записи старше нескольких минут. Таким образом, если компьютер был выключен, перенесен на новое место и включен снова, уже через несколько минут он войдет в обычный режим работы, и для этого не потребуется никаких специальных действий. Это также означает, что если станция «молчит» в течение нескольких минут, адресованный ей трафик снова будет рассылаться методом лавинной адресации, пока она сама не отправит какой-нибудь фрейм.

Процедура маршрутизации входящего фрейма зависит от того, на какой порт он прибыл (порт-источник) и на какой адрес направляется (адрес назначения). Алгоритм выглядит следующим образом:

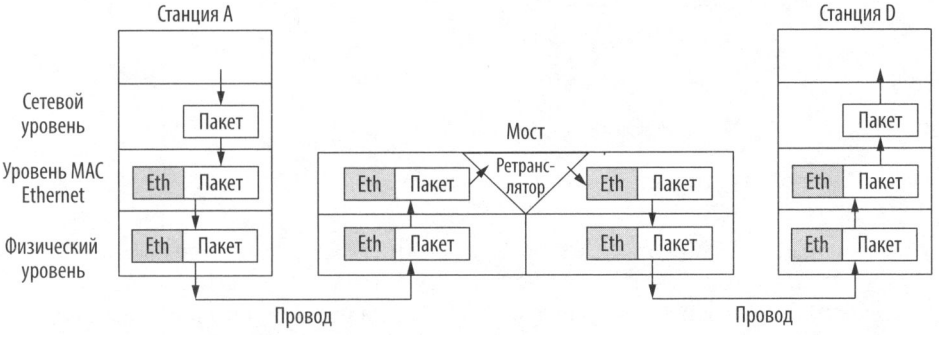
1. Если порт-источник и порт для адреса назначения совпадают, фрейм игнорируется.
2. Если порт-источник и порт для адреса назначения различаются, фрейм переправляется на порт назначения.
3. Если порт назначения неизвестен, используется алгоритм лавинной адресации и фрейм пересылается на все порты, кроме источника.

Может ли первая ситуация произойти с двухточечными линиями? Ответ — да, если для соединения группы компьютеров с мостом используются концентраторы. Пример показан на илл. 4.33 (б), где станции *E* и *F* соединены с концентратором *H1*, который, в свою очередь, подключен к мосту *B2*. Если станция *E* отправит фрейм станции *F*, то концентратор передаст его и *F*, и мосту *B2*. Именно это делают концентраторы — они связывают все порты вместе так, чтобы фрейм, полученный на одном порте, просто выводится на всех остальных. Фрейм достигнет моста *B2* на порте 2, который уже является правильным выходным портом для попадания в пункт назначения. Мост *B2* должен просто отказаться от фрейма.

Поскольку этот алгоритм должен применяться к каждому входящему фрейму, обычно он осуществляется с помощью специальных чипов СБИС. Чип производит поиск и обновляет записи таблицы за несколько микросекунд. Мосты проверяют только MAC-адреса, чтобы решить, как отправить фреймы. Поэтому можно начать отправку, как только появилось поле заголовка назначения — еще до того, как дошла остальная часть фрейма (конечно, если выходная линия доступна). Это сокращает время прохождения через мост, а также количество

фреймов, которые мост должен буферизовать. Такой способ называют **коммутацией без буферизации пакетов (cut-through switching)** или **маршрутизацией способом коммутации каналов (wormhole routing)**, и обычно он реализуется аппаратными средствами.

Можно взглянуть на работу моста с точки зрения стека протоколов и разобраться, что собой представляет устройство канального уровня. Рассмотрим фрейм, посланный от станции *A* станции *D* в конфигурации на илл. 4.33 (а), где в качестве LAN выступает сеть Ethernet. Фрейм пройдет через один мост. Стек используемых при этом протоколов показан на илл. 4.34.




**Илл. 4.34.** Протоколы, которые реализуются в мосте

Пакет приходит с более высокого уровня и спускается на уровень MAC Ethernet. Он получает заголовок Ethernet (а также трейлер, не показанный на рисунке). Далее фрейм передается на физический уровень, проходит по кабелю и принимается мостом.

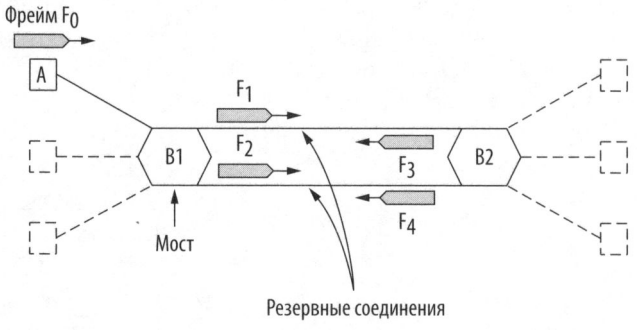
В мосте фрейм передается с физического уровня на уровень MAC Ethernet. Здесь фрейм обрабатывается дольше, чем на аналогичном уровне станции. Он передается на ретранслятор, все еще в пределах уровня MAC. Функция ретрансляции в мосте использует только заголовок MAC Ethernet, чтобы определить, как обработать фрейм. В нашем случае фрейм передается порту уровня MAC Ethernet, который связан со станцией *D*, и продолжает свой путь.

В общем случае ретрансляторы на конкретном уровне могут переписать для него заголовки. Позже будет показано, как это происходит в виртуальных LAN. Мост ни в коем случае не должен проверять содержимое фрейма и выяснять, что в нем находится IP-пакет. Для работы моста это значения не имеет, к тому же это нарушает иерархическое представление протокола. Обратите внимание, что мост, имеющий $k$ портов, также включает $k$ экземпляров MAC-уровня и физического уровня. В нашем простом примере $k = 2$.

## 4.7.3. Мосты связующего дерева

Для повышения надежности между мостами устанавливаются резервные соединения. На илл. 4.35 показаны два параллельных канала между *B1* и *B2*. Эта

конструкция гарантирует, что при разрыве одного соединения сеть не будет разделена на два набора компьютеров, которые не могут взаимодействовать друг с другом.




**Илл. 4.35.** Мосты с двумя параллельными соединениями

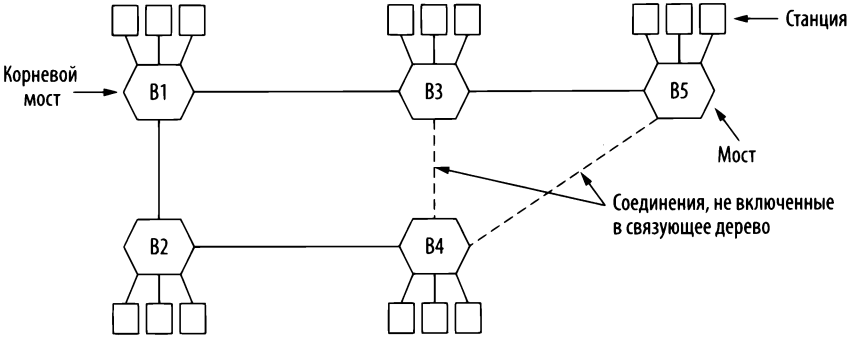
Впрочем, это создает некоторые дополнительные проблемы, поскольку в топологии возникают циклы. Рассмотрим следующий пример. Станция *A* отправляет фрейм в ранее неизвестный пункт назначения (илл. 4.35). Каждый мост, действуя по обычным правилам обработки фреймов с неизвестным получателем, использует метод лавинной адресации. Мост *B1* получает фрейм от станции *A*. Обозначим его $F_0$. Мост передает копии этого фрейма через все остальные порты. Мы рассмотрим только те из них, которые соединяют *B1* и *B2* (хотя фрейм будет отправлен и через другие). Так как между *B1* и *B2* имеется два соединения, в *B2* попадут две копии фрейма. Они обозначены на илл. 4.35 как $F_1$ и $F_2$.

Вскоре после этого мост *B2* получает их. Разумеется, он не знает (и не может знать), что это копии, а не два разных фрейма, отправленных друг за другом. Поэтому *B2* принимает $F_1$ и $F_2$ и отправляет копии каждого из них со всех остальных портов. Так возникают фреймы $F_3$ и $F_4$, которые по двум соединениям отправляются обратно в *B1*. Мост *B1* видит два новых фрейма с неизвестным адресом назначения и копирует их снова. Этот цикл продолжается бесконечно.

Эта проблема решается установлением связи между мостами и наложением на реальную топологию сети связующего дерева (spanning tree), которое охватывает оба моста. В результате некоторые потенциальные соединения между мостами игнорируются. Это позволяет создать фиктивную незацикленную топологию, которая является подмножеством реальной системы.

Для примера на илл. 4.36 показаны пять мостов, которые связаны между собой и имеют подключенные к ним станции. Каждая станция соединяется только с одним мостом. Есть несколько резервных каналов между мостами, так что если будут использоваться все соединения, фреймы будут зациклены. Эту систему можно представить в виде графа: мосты являются его вершинами, а соединения между ними — его ребрами. Такой граф можно редуцировать до связующего дерева, которое по определению не имеет циклов, удалив из него соединения, изображенные на илл. 4.36 пунктирными линиями. В получившемся

связующем дереве между каждыми двумя станциями существует только один путь. После того как мосты договорятся друг с другом о топологии связующего дерева, все коммуникации осуществляются только по его ветвям. Поскольку путь от отправителя к получателю единственный, зацикливание невозможно.




**Илл. 4.36.** Связующее дерево, соединяющее пять мостов. Пунктирными линиями показаны соединения, которые в него не входят

Чтобы построить связующее дерево, мосты применяют распределенный алгоритм. Каждый из них периодически рассылает по всем своим портам конфигурационное сообщение соседним мостам и обрабатывает полученные от них сообщения, как описано ниже. Эти сообщения дальше не отправляются, так как их цель — построение дерева, которое затем используется для передачи.

Сначала необходимо выбрать один мост, который будет корнем связующего дерева. Для этого каждый мост включает в конфигурационное сообщение идентификатор, основанный на своем MAC-адресе, а также идентификатор потенциального корневого моста. MAC-адреса устанавливаются изготовителем и являются уникальными, что гарантирует уникальность и удобство идентификаторов. Мосты выбирают в качестве корня мост с наименьшим идентификатором. После обмена достаточным числом сообщений, чтобы распространить эту новость, мосты принимают общее решение. На илл. 4.36 мост *B1* имеет наименьший идентификатор, он и становится корнем.

Далее создается дерево кратчайших путей от корня до каждого моста. На илл. 4.36 кратчайший путь от моста *B1* до мостов *B2* и *B3* преодолевается за один шаг непосредственно. Мост *B4* достигается за два шага, через *B2* или через *B3*. В этой ситуации выбирается мост с наименьшим идентификатором, таким образом, путь до *B4* лежит через *B2*. Путь до моста *B5* преодолевается за два шага через *B3*.

Чтобы найти эти кратчайшие пути, мосты включают в конфигурационные сообщения расстояние от корня. Каждый мост помнит кратчайший путь, который он находит к корню. Затем мосты отключают порты, которые не являются частью кратчайшего пути.

Дерево охватывает все мосты, но не все соединения (или даже не все мосты) обязательно присутствуют в нем. Это происходит, поскольку отключение портов

ликвидирует некоторые соединения в сети, чтобы предотвратить появление циклов. Алгоритм построения дерева продолжает работать постоянно, обнаруживая изменения в топологии и обновляя структуру дерева.

Алгоритм автоматического построения связующего дерева впервые предложила Радья Перлман (Radia Perlman). Она занималась проблемой объединения локальных сетей без циклов. На решение этой задачи была выделена неделя, но уже в первый день ей удалось придумать алгоритм связующего дерева. Благодаря этому у нее осталось время на то, чтобы изложить эту идею в стихотворной форме (Perlman, 1985):

*I think that I shall never see*
*A graph more lovely than a tree.*
*A tree whose crucial property*
*Is loop-free connectivity.*
*A tree which must be sure to span.*
*So packets can reach every LAN.*
*First the Root must be selected*
*By ID it is elected.*
*Least-cost paths from Root are traced*
*In the tree these paths are placed.*
*A mesh is made by folks like me*
*Then bridges find a spanning tree.*

Алгоритм связующего дерева описан в стандарте IEEE 802.1D и используется уже много лет. В 2001 году он был переработан, в результате новое связующее дерево после изменения топологии находится быстрее. Более подробную информацию о мостах вы найдете в работе Перлман (Perlman, 2000).

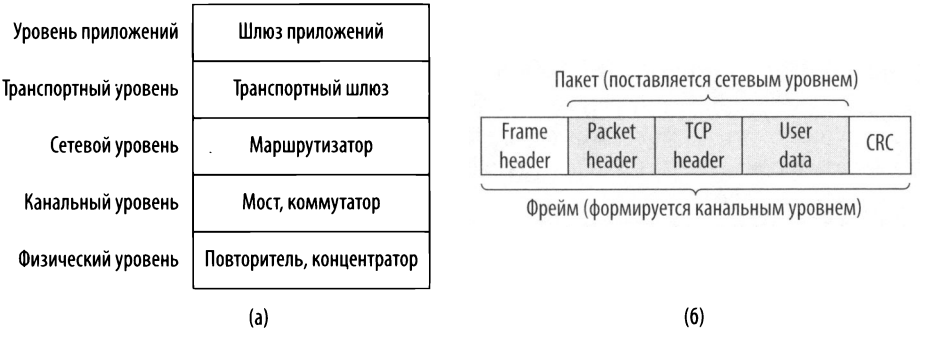
## 4.7.4. Повторители, концентраторы, мосты, коммутаторы, маршрутизаторы и шлюзы

В этой книге мы уже видели множество способов доставки фреймов и пакетов с одного компьютера на другой. Мы упоминали повторители, концентраторы, мосты, маршрутизаторы и шлюзы. Давайте рассмотрим все эти устройства вместе, отмечая их сходства и различия.

Чтобы разобраться, как работают упомянутые устройства, надо понять, что они задействованы на разных уровнях, как показано на илл. 4.37 (а). Уровень имеет значение, поскольку от него зависит, какую часть информации устройство использует для маршрутизации. Рассмотрим типичный сценарий. У пользователя появляются данные (`User data`), которые необходимо отправить на другой компьютер. Они передаются на транспортный уровень, который добавляет к ним свой заголовок (например, `TCP header`) и передает полученную единицу информации на сетевой уровень. Тот, в свою очередь, тоже добавляет свой заголовок (`Packet header`), в результате чего формируется пакет сетевого уровня (например, IP-пакет). На илл. 4.37 (б) IP-пакет выделен серым цветом. Пакет отправляется на канальный уровень, где дополняется еще одним заголовком

(Frame header) и контрольной суммой (CRC). Наконец, формируется фрейм, который спускается на физический уровень для передачи, например, по LAN.

Приступим к рассмотрению коммутирующих устройств и разберемся, как они соотносятся с пакетами и фреймами. На самом нижнем, физическом, уровне находятся повторители. Это аналоговые устройства, которые работают с сигналами, идущими по подключенным к ним кабелям.




**Илл. 4.37.** (а) Уровни и расположенные на них устройства. (б) Фреймы, пакеты и заголовки

Сигнал, появившийся в одном кабеле, очищается, усиливается повторителем и выдается на второй. Повторители не знают, что такое пакет, фрейм или заголовок. Они знают символы, кодирующие биты в напряжение. В классическом Ethernet, например, допускается установка четырех повторителей для усиления сигнала, чтобы увеличить максимальную длину кабеля с 500 до 2500 м.

Теперь перейдем к концентраторам. Концентратор имеет несколько входных линий, объединяемых с помощью электричества. Фреймы, приходящие на какой-либо вход, передаются на все остальные линии. Если одновременно по разным линиям придут два фрейма, они столкнутся, как в коаксиальном кабеле. Все линии, идущие в концентратор, должны работать с одинаковой скоростью. Концентраторы отличаются от повторителей тем, что, как правило, не усиливают входной сигнал и предназначены для нескольких входных линий. Впрочем, разница между ними незначительна. И те и другие являются устройствами физического уровня; они не анализируют адреса канального уровня и никак не используют их.

Далее мы поднимемся на канальный уровень. Здесь расположены мосты и коммутаторы. Только что мы довольно подробно обсудили мосты, поэтому знаем, что они соединяют две или несколько LAN. Как и в концентраторах, в современных мостах имеются несколько портов, рассчитанных обычно на 4–48 входящих линий определенного типа. В отличие от концентратора, каждый порт изолирован, чтобы быть своей собственной областью коллизий; если у порта есть полнодуплексная двухточечная линия, в алгоритме CSMA/CD нет необходимости. Когда приходит фрейм, мост извлекает из заголовка адрес назначения и анализирует его, сопоставляя с таблицей и определяя, куда нужно его передать. Для Ethernet этот адрес — 48-битный адрес назначения (см. илл. 4.14).

Мост только выводит фрейм на нужный порт; он может передавать несколько фреймов одновременно.

Мосты предлагают гораздо более высокую производительность, чем концентраторы, а благодаря изолированным портам входные линии могут работать на разных скоростях и даже с различными типами сетей. Распространенный пример — мост с портами, которые соединяются с 10-, 100 и 1000-Мбит/с Ethernet. Чтобы мост мог принять фрейм на одном порте и передать его на другой, необходима буферизация. Если фреймы приходят быстрее, чем передаются дальше, мост может исчерпать буферное пространство и начать отказываться от фреймов. Например, если Gigabit Ethernet заливает биты в 10-Мбит/с Ethernet на большой скорости, мост должен буферизовать их, надеясь, что памяти хватит. Эта проблема существует, даже если все порты работают на одной и той же скорости, потому что фреймы могут передаваться в порт назначения из нескольких портов.

Первоначально мосты предназначались для того, чтобы соединять разные виды LAN, например Ethernet и Token Ring. Однако из-за отличий между LAN это не сработало. Для фреймов различных форматов необходимо копирование и переформатирование. Это занимает время CPU, требует нового вычисления контрольной суммы и ведет к появлению необнаруженных ошибок из-за поврежденных битов в памяти моста. Еще одна серьезная проблема, не имеющая хорошего решения, — разная максимальная длина фреймов. По сути, слишком длинные для дальнейшей передачи фреймы должны быть отклонены. С идеей прозрачности можно распрощаться.

Еще две сферы, в которых LAN могут различаться, — это безопасность и QoS. У некоторых LAN (например, у 802.11) есть шифрование канального уровня и функции QoS (такие, как приоритеты), у других (например, у Ethernet) этого нет. Следовательно, при обмене между такими LAN трудно обеспечить безопасность или QoS, ожидаемые отправителем. По этой причине современные мосты обычно работают с сетями одного типа, а для соединения разных сетей используются маршрутизаторы, которые мы обсудим позже.

Коммутаторы — это другое название современных мостов. Различия больше связаны с маркетингом, чем с техническими особенностями, но есть несколько важных моментов. Мосты были разработаны, когда использовался классический Ethernet, поэтому они могут соединять относительно небольшое число LAN, а значит, имеют не так много портов. Сегодня чаще употребляется термин «коммутатор». Кроме того, все современные системы используют двухточечные линии, например витую пару. Отдельные компьютеры подключаются непосредственно к коммутатору, поэтому у него, как правило, много портов. Наконец, понятие «коммутатор» также используется в качестве общего термина. Функционал моста понятен. А вот слово «коммутатор» может относиться и к коммутатору Ethernet, и к совершенно другому устройству, принимающему решения о переадресации, например телефонному коммутатору.

Итак, мы кратко обсудили повторители и концентраторы, весьма схожие между собой, а также коммутаторы и мосты, между которыми еще меньше различий. Теперь перейдем к маршрутизаторам. Они существенно отличаются от рассмотренных выше устройств. Когда пакет приходит в маршрутизатор, заголовок и трейлер фрейма удаляются, а сам пакет, расположенный в поле данных

(выделены серым на илл. 4.37), передается программному обеспечению маршрутизатора. Далее анализируется заголовок пакета и в соответствии с ним выбирается его дальнейший путь. Если это IP-пакет, то в его заголовке содержится 32-битный (IPv4) или 128-битный (IPv6), но не 48-битный IEEE 802 адрес. ПО маршрутизатора не видит адреса фреймов и даже не знает, как они пришли, по LAN или по двухточечной линии. Более подробно мы изучим маршрутизаторы и принципы маршрутизации в главе 5.

На следующем уровне расположены транспортные шлюзы. Они служат для соединения компьютеров, которые применяют различные транспортные протоколы, ориентированные на установление соединения. Например, одно устройство использует TCP/IP, а другое — SCTP. Транспортный шлюз может копировать пакеты между соединениями, одновременно переводя их в нужный формат.

Наконец, шлюзы приложений различают форматы и содержимое данных и могут менять формат сообщений. Например, шлюз e-mail может переводить электронные письма в формат SMS-сообщений для мобильных телефонов. Как и «коммутатор», «шлюз» — своего рода общий термин. Он относится к процессу передачи, который выполняется на верхнем уровне.

### 4.7.5. Виртуальные локальные сети

На заре развития технологий локальных сетей толстые желтые кабели прокладывались во множестве офисов. Можно было подключить к сети каждый компьютер, рядом с которым проходил такой провод. Никто не задумывался над тем, к какой именно LAN подключался конкретный компьютер. Сотрудники соседних офисов пользовались одной сетью, даже если они работали в разных отделах. «География» брала верх над корпоративной структурой.

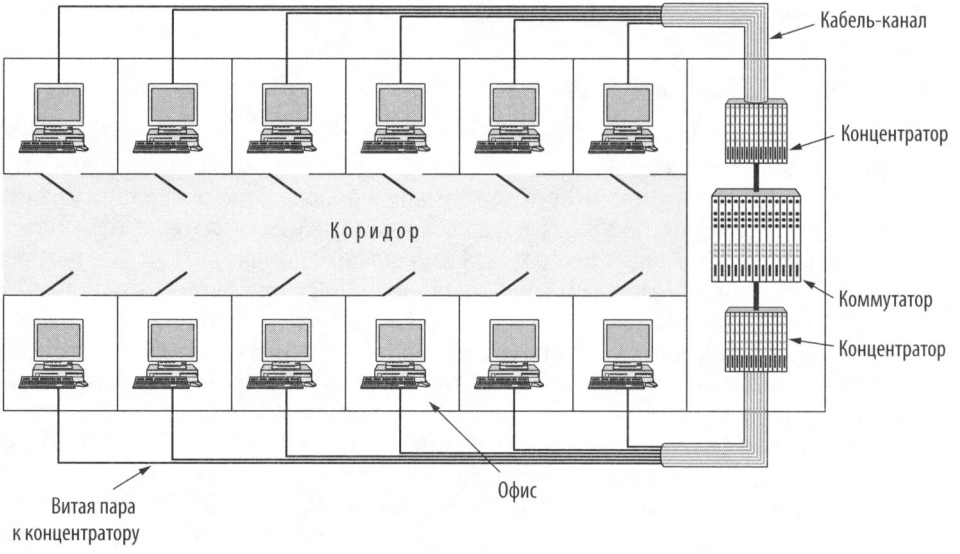
В 1990-х с появлением витой пары и концентраторов все изменилось, и офисные здания были переоборудованы (что обошлось недешево). На смену желтым проводам, напоминающим садовые шланги, пришла витая пара, идущая от каждого кабинета к монтажному шкафу в конце коридора или к центральному машинному залу (илл. 4.38). Если сотрудник, ответственный за переоборудование, был дальновидным человеком, то прокладывалась витая пара категории 5. Если же он был крохобором, то использовались уже существующие телефонные кабели категории 3. Через пару лет, с приходом сетей Fast Ethernet, их все равно заменили.

Сегодня кабели поменялись, а концентраторы стали коммутаторами, но общая схема осталась прежней. Она позволяет настраивать LAN не физически, а логически. Если компании требуется $k$ LAN, она может приобрести $k$ коммутаторов. Аккуратно собрав сеть (то есть вставив нужные коннекторы в нужные разъемы), можно распределить пользователей таким образом, чтобы это имело некий реальный организационный смысл и не зависело от расположения станций в здании.

А разве важно, к какой сети подключен конкретный пользователь? Все равно почти во всех организациях LAN объединены между собой. На самом деле это часто имеет значение. Сетевые администраторы по целому ряду причин любят группировать пользователей LAN в соответствии со структурой организации,

а не по принципу физического расположения компьютеров. Одной из таких причин является безопасность. Одна LAN может объединять веб-серверы и другие компьютеры для публичного доступа. Другая может содержать записи отдела кадров, которые не должны выходить за пределы компании. В этой ситуации наилучшим выходом является объединение компьютеров всех сотрудников отдела в одну LAN и запрет на передачу каких-либо данных за ее пределы. Обычно руководство не одобряет заявлений о том, что такое решение невозможно реализовать.

Второй причиной является нагрузка на сеть. Некоторые сети могут быть более загруженными, чем другие, в этом случае лучше их разделить. Например, если эксперименты отдела исследований приведут к перегрузке сети, то руководители вряд ли с энтузиазмом воспримут необходимость отказаться от части пропускной способности, которую планировалось использовать для видеоконференций. С другой стороны, это заставит их задуматься о необходимости внедрения более быстрой сети.




**Илл. 4.38.** Здание, где имеется централизованная проводка с концентраторами и коммутаторами

Еще одна причина — широковещание. Мосты применяют его, когда местонахождение адресата неизвестно, а протоколы верхних уровней также поддерживают эту технологию. Например, если пользователь хочет отправить пакет на IP-адрес *x*, как ему узнать, какой MAC-адрес указать во фрейме? Мы изучим этот вопрос более подробно в главе 5, а сейчас в двух словах опишем решение проблемы. Данная станция должна широковещательным методом разослать запрос: «Кто знает, какой MAC-адрес работает с IP-адресом *x*?» Затем она дожидается ответа. Поскольку компьютеров в LAN становится все больше, растет и число таких сообщений. Каждая широковещательная передача потребляет

больше ресурсов, чем обычный фрейм, потому что она предназначена для всех компьютеров в сети. Если сдерживать размер LAN, чтобы она не разрасталась без необходимости, то нагрузка от широковещательного трафика уменьшается.

С широковещанием связана еще одна проблема: время от времени сетевой интерфейс выходит из строя или нарушается его конфигурация. Тогда он начинает генерировать бесконечный поток фреймов, получаемый всеми станциями. Если не повезет, некоторые фреймы вызовут ответы, которые приведут к еще большему трафику. В результате этого **широковещательного шторма (broadcast storm)** вся пропускная способность сети будет занята бессмысленными фреймами, а все устройства объединенных LAN будут вынуждены заниматься исключительно обработкой и удалением мусора.

Сначала может показаться, что широковещательный шторм можно ограничить, разделив сеть с помощью мостов или коммутаторов. Но если нужно обеспечить прозрачность (то есть предоставить возможность перемещать компьютеры в другие LAN без каких-либо изменений конфигурации), то мосты должны передавать широковещательные фреймы.

Мы выяснили, почему компаниям нужны многочисленные LAN ограниченного размера. Теперь вернемся к проблеме разделения логической и физической топологии. Создание физической конфигурации, которая отразит организационную структуру, может потребовать дополнительных усилий и увеличить стоимость проекта даже при наличии централизованной проводки и коммутаторов. Например, если два сотрудника из одного отдела работают в разных зданиях, легче подключить их компьютеры к разным коммутаторам, принадлежащим разным LAN. Другой пример — сотрудник может перейти в пределах компании из одного подразделения в другое, но остаться в том же офисе или, наоборот, сменить офис, не меняя отдел. В результате пользователь будет работать в неправильной LAN, пока администратор не подключит его кабель вручную к нужному коммутатору. Помимо этого, количество компьютеров в различных отделах может не соответствовать числу портов на коммутаторах. Некоторые подразделения могут быть слишком маленькими, а другие, наоборот, слишком крупными, и им не хватает одного коммутатора. Это приводит к тому, что у коммутаторов используются не все порты.

Во многих компаниях организационные перестановки происходят постоянно, и сетевой администратор вынужден постоянно производить манипуляции с коннекторами и разъемами. Иногда это невозможно, поскольку витая пара от пользовательского компьютера не дотягивается до нужного коммутатора (который находится в другом здании) или же доступные порты коммутатора относятся к другой LAN.

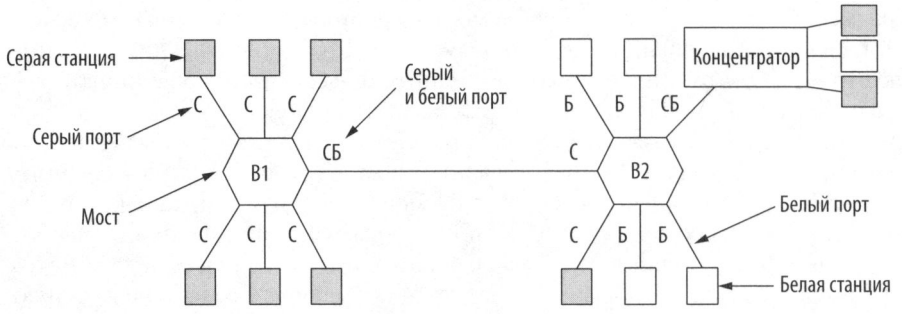
Пользователи нуждались в большей гибкости. В ответ на этот запрос были разработаны **виртуальные LAN (Virtual LAN, VLAN)**. Для их внедрения необходимо было изменить только программное обеспечение. Виртуальные сети были стандартизованы комитетом IEEE 802 и сегодня широко применяются во многих организациях. Далее мы рассмотрим эту разновидность LAN.

Виртуальные сети строятся на основе VLAN-совместимых коммутаторов. Чтобы создать такую систему, сетевой администратор должен решить, сколько всего будет VLAN, какие компьютеры будут в них входить, а также придумать

названия для этих VLAN. Часто их обозначают разными цветами — очень удобно распечатать цветную диаграмму, на которой наглядно показано расположение компьютеров. Пользователи красной сети будут изображены красным цветом, зеленой — зеленым и т. д. Таким образом, на одном рисунке можно отобразить физическую и логическую структуру одновременно.

В качестве примера рассмотрим LAN с мостом, изображенную на илл. 4.39. Девять компьютеров относятся к виртуальной сети «С» (серой), а еще пять — к сети «Б» (белой). Компьютеры серой сети распределены между двумя коммутаторами; два из них подключено к коммутатору через концентратор.

Чтобы VLAN функционировали корректно, в мостах должны быть конфигурационные таблицы. Они сообщают, через какие порты производится доступ к тем или иным VLAN. Например, когда фрейм прибывает из серой VLAN, его нужно разослать на все порты, помеченные буквой «С». Это справедливо как для ординарных (одноадресных) передач, при которых мосту не нужна информация о местоположении адресата, так и для групповых и широковещательных. Следует отметить, что порт может быть помечен сразу несколькими цветами VLAN.




**Илл. 4.39.** Две VLAN, серая и белая, в сети с мостом

Предположим, что одна из серых станций, подключенных к мосту *B1* (илл. 4.39), отправляет фрейм адресату, местонахождение которого пока неизвестно. Мост *B1* принимает фрейм и видит, что тот пришел с устройства с меткой «С». Поэтому его необходимо передать на все порты (кроме источника), принадлежащие серой VLAN. Фрейм будет направлен на остальные пять серых станций, подключенных к *B1*, а также по соединению *B1* с мостом *B2*. Мост *B2* аналогично перешлет фрейм на все порты с меткой «С», то есть одной оставшейся станции и концентратору (который передаст его всем своим станциям). Концентратор имеет обе метки («С» и «Б»), поскольку к нему подключены станции из обеих VLAN. Фрейм передается только на порты с меткой «С», поскольку мост знает, что другие порты не ведут к серым станциям.

В нашем примере фрейм передается от *B1* к *B2* только потому, что некоторые станции серой VLAN подключены к *B2*. Глядя на белую VLAN, можно увидеть, что порт *B2*, соединенный с *B1*, *не снабжен* меткой «Б». Это означает, что фрейм белой VLAN не будет отправлен от *B2* к *B1*. Это логично, поскольку к мосту *B1* не присоединено ни одной белой станции.

### Стандарт IEEE 802.1Q

Чтобы реализовать эту схему, мосты должны знать, к какой VLAN принадлежит входящий фрейм. Иначе мост *B2*, получив фрейм от моста *B1* (см. илл. 4.39), не поймет, куда его передавать — белой или серой VLAN. Если бы мы проектировали новый тип LAN, можно было бы просто добавить специальное поле заголовка. Но что делать с Ethernet — наиболее распространенной сетью, у которой нет запасных полей для идентификатора виртуальной сети?

Комитет IEEE 802 задался этим вопросом в 1995 году. После долгих дискуссий было сделано нечто невообразимое — изменен формат заголовка фрейма Ethernet. Новый формат был опубликован под названием 802.1Q в 1998 году. Он предусматривает тег VLAN; мы кратко обсудим его далее. Понятно, что вносить изменения в такую устоявшуюся систему, как Ethernet, непросто. Сразу возникает несколько вопросов:

1. И что, теперь надо выбросить несколько сотен миллионов уже существующих сетевых карт Ethernet?
2. Если нет, то кто создает новые поля фреймов?
3. Что произойдет с фреймами, которые уже имеют максимальный размер?

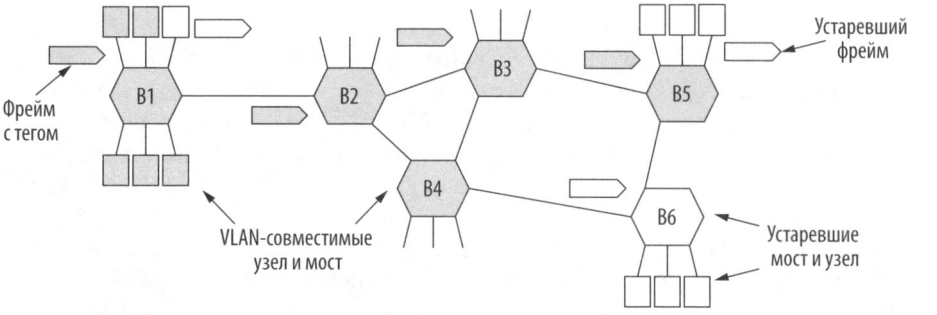
Конечно, комитет 802 знал об этих проблемах (даже слишком хорошо), и в итоге решение было найдено.

Оно состоит в том, что на самом деле поля VLAN реально используются только мостами и коммутаторами, а *не* компьютерами пользователей. Как видно на илл. 4.39, эти поля не так важны на линиях, ведущих к адресату; они необходимы, когда фреймы доходят до мостов. Кроме того, чтобы использовать VLAN, мосты должны быть VLAN-совместимыми. Это делает проект выполнимым.

Что касается старых сетевых карт Ethernet, то выкидывать их не пришлось. Комитет 802.3 даже не мог заставить пользователей изменить поле `Type` на `Lenght`. Можно представить реакцию общественности на заявление о том, что все существующие карты Ethernet больше не нужны. В любом случае новые карты Ethernet совместимы со стандартом 802.1Q и могут корректно заполнять поля идентификации VLAN.

Некоторые компьютеры (а также коммутаторы) могут быть несовместимы с VLAN. Поэтому первый встретившийся на пути фрейма VLAN-совместимый мост вставляет это поле, а последний — удаляет его. Пример смешанной топологии показан на илл. 4.40. Здесь VLAN-совместимые компьютеры генерируют снабженные тегами (то есть совместимые со стандартом 802.1Q) фреймы непосредственно, и последующие коммутаторы используют эти теги. Закрашенные элементы — VLAN-совместимые, пустые — нет.

В соответствии с 802.1Q, фреймы «окрашиваются» в зависимости от порта, на котором они получены. Чтобы этот метод работал, все станции одного порта должны принадлежать одной VLAN, что уменьшает гибкость. Например, на илл. 4.39 это свойство справедливо для всех портов, где отдельный компьютер соединяется с мостом, но не для порта, где концентратор соединяется с мостом *B2*.



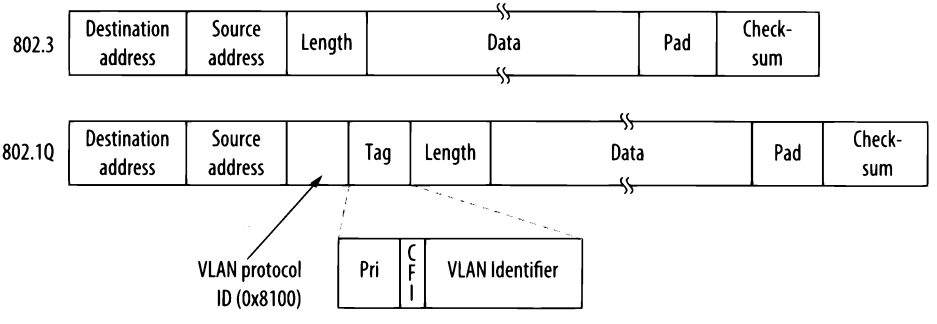

**Илл. 4.40.** LAN с мостами, частично совместимая с VLAN. Закрашенные элементы — VLAN-совместимые устройства. Все остальные несовместимы с VLAN

Также отметим, что мост может использовать протокол более высокого уровня, чтобы выбрать цвет. Таким образом, фреймы, которые приходят на порт, могут попадать в различные VLAN в зависимости от того, что они содержат — IP-пакеты или PPP-фреймы.

Возможны и другие методы, но они не поддерживаются в 802.1Q. Например, для выбора цвета VLAN может использоваться MAC-адрес. Этот метод применяется для фреймов из соседних 802.11 LAN, где ноутбуки передают данные через разные порты по мере перемещения. Один MAC-адрес будет закреплен за одной и той же VLAN, независимо от порта, через который он попадает в сеть.

Что касается фреймов, длина которых превышает 1518 байт, то в стандарте 802.1Q эта проблема решается путем повышения лимита до 1522 байт. К счастью, только VLAN-совместимые компьютеры и коммутаторы обязаны поддерживать такие длинные фреймы.

Теперь рассмотрим формат 802.1Q (илл. 4.41). Единственное заметное изменение — добавление пары 2-байтных полей. Первое называется `VLAN protocol ID` (Идентификатор протокола VLAN). Оно всегда имеет значение 0x8100. Поскольку оно превышает 1500, то все сетевые карты Ethernet интерпретируют его как `Type`, а не как `Lenght`. Неизвестно, что будет делать с такими фреймами устаревшая карта, поскольку они, по идее, не должны на нее передаваться.




**Илл. 4.41.** Форматы фреймов Ethernet-стандартов 802.3 (устарел) и 802.1Q

Во втором двухбайтном поле есть три вложенных поля. Главное из них — `VLAN Identifier` (Идентификатор VLAN), который занимает 12 младших битов. Оно содержит информацию, из-за которой и была затеяна смена формата: цвет виртуальной сети, к которой относится фрейм. Трехбитное поле `Priority` (Приоритет) не имеет отношения к VLAN. Но поскольку изменение формата Ethernet-фрейма производится раз в десять лет и занимает три года работы сотни людей, почему бы не предусмотреть в нем место для некоторой дополнительной полезной информации? Это поле позволяет различать трафик жесткого и мягкого реального времени и трафик, для которого время передачи некритично. Благодаря этому улучшаются показатели QoS в Ethernet. Поле `Priority` используется также при передаче голоса по Ethernet (хотя справедливости ради вот уже четверть века в IP имеется подобное поле, и никто никогда его не использовал).

Последнее поле, `CFI` (Canonical Format Indicator — Индикатор классического формата), следовало бы назвать `CEI` (Corporate ego indicator — Индикатор эгоизма компании). Изначально оно предназначалось для того, чтобы показывать порядок битов в MAC-адресе (от младшего к старшему или наоборот), однако в пылу дискуссий об этом как-то забыли. Его присутствие сейчас означает, что поле данных содержит сублимированный фрейм 802.5, который ищет еще одну сеть формата 802.5, и Ethernet для него лишь промежуточный этап. Все это, конечно, никак не связано с VLAN. Но политика комитета стандартизации не слишком отличается от обычной политики: если ты проголосуешь за мой бит, я проголосую за твой. Торг чистой воды.

Как уже упоминалось выше, когда VLAN-совместимый коммутатор получает фрейм с тегом VLAN, он использует идентификатор VLAN в качестве индекса таблицы, в которой ищет, на какие порты передать фрейм. Но откуда берется эта таблица? Если она разрабатывается вручную, это означает возврат в исходную точку: ручное конфигурирование мостов. Вся прелесть прозрачности мостов состоит в том, что они настраиваются автоматически и не требуют никакого вмешательства извне. Было бы ужасно жаль потерять это свойство. К счастью, мосты для виртуальных сетей также настраиваются сами в зависимости от тегов входящих фреймов. Если фрейм, помеченный как VLAN 4, приходит на порт 3, это означает, что одна из станций, подключенных к этому порту, находится внутри VLAN 4. Стандарт 802.1Q вполне четко поясняет, как строятся динамические таблицы, преимущественно ссылаясь на соответствующие части стандарта 802.1D.

Прежде чем завершить обсуждение маршрутизации VLAN, отметим следующее. Многие пользователи интернета и Ethernet весьма привязаны к сетям без установления соединения и активно выступают против любых систем, в которых есть хотя бы намек на соединение на сетевом или канальном уровне. Однако в виртуальных сетях есть кое-что, сильно напоминающее установку соединения. Работа VLAN невозможна без идентификатора в каждом фрейме. Он используется как индекс таблицы в коммутаторе, чтобы определить дальнейший путь фрейма. Именно это и происходит в сетях, ориентированных на установление соединения. В системах без установления соединения маршрут определяется с помощью адреса назначения, а не идентификатора. Мы подробнее обсудим эту тенденцию в главе 5.

## 4.8. РЕЗЮМЕ

В некоторых сетях для любой связи используется единственный канал. При их разработке основной проблемой является распределение этого канала между конкурирующими станциями, желающими им воспользоваться. Самые простые и эффективные схемы распределения при небольшом количестве станций и постоянном трафике — это FDM и TDM. Оба метода широко применяются, например, для разделения полосы пропускания телефонных магистралей. При большом и непостоянном числе станций или при трафике, создающем периодическую пиковую нагрузку (типичный случай для компьютерных сетей), FDM и TDM не подходят.

Существует несколько алгоритмов динамического распределения каналов. Многочисленные варианты протокола ALOHA (чистого или дискретного) используются в реальных системах (например, в сетях DOCSIS). Его усовершенствование — возможность прослушивать состояния канала. Станции могут отказываться от передачи, если слышат, что канал занят другой станцией. Применение метода контроля несущей привело к созданию различных CSMA-протоколов для LAN и MAN. Это основа сетей классического Ethernet и 802.11.

Уже много лет широко применяется класс протоколов, который полностью устраняет или, по крайней мере, снижает конкуренцию за канал. Протокол битовой карты, протокол с двоичным обратным отсчетом и протоколы на базе маркерных колец полностью исключают конкуренцию, а протокол адаптивного дерева — снижает. Он динамически делит станции на две непересекающиеся группы разного размера и разрешает состязание только внутри группы. В идеале группа формируется так, чтобы только одна станция была готова к передаче, когда это разрешено. Современные версии протоколов подуровня MAC, включая DOCSIS и Bluetooth, явно стремятся к устранению конкуренции, выделяя интервалы передачи каждому отправителю.

В беспроводных LAN возникают дополнительные проблемы. В них сложно выявлять коллизии, а у станций могут различаться зоны покрытия. Станции IEEE 802.11 (доминирующей беспроводной LAN) применяют CSMA/CA, оставляя небольшие интервалы, чтобы избежать коллизий и решить первую проблему. Для решения проблемы скрытой станции используется протокол RTS/CTS. Однако этот метод приводит к большим накладным расходам из-за проблемы засвеченной станции, которая особенно остро проявляется в ситуациях с большим числом радиоустройств.

Многие станции избегают конкуренции путем использования механизмов выбора канала. IEEE 802.11 обычно применяется для подключения ноутбуков и других устройств к беспроводным точкам доступа, а также для соединения самих устройств. Можно использовать любой из нескольких физических уровней, в том числе многоканальный FDM с несколькими антеннами (или без них), и расширение спектра. Современные версии стандарта 802.11 содержат функции безопасности на канальном уровне, в том числе поддержку аутентификации, а также расширенные возможности кодирования, позволяющие передавать данные с использованием технологии MIMO.

Ethernet является основной технологией проводных LAN. Классический Ethernet использовал CSMA/CD для распределения канала в желтом кабеле

толщиной с садовый шланг, который тянулся от одного компьютера к другому. Архитектура изменилась, скорости увеличились с 10 Мбит/с до 1 Гбит/с и продолжают расти. Теперь двухточечные линии, такие как витая пара, присоединяются к концентраторам и коммутаторам. Современные коммутаторы и полнодуплексные каналы исключают конкуренцию — коммутатор может передавать фреймы между различными портами параллельно.

Когда в здании много локальных сетей, нужен способ для их объединения. Для этого используются plug-and-play-устройства — мосты. При построении мостов применяются алгоритмы обратного обучения и связующего дерева. Как только этот функционал был добавлен в современные коммутаторы, термины «мост» и «коммутатор» стали синонимами. Чтобы упростить управление LAN с мостами, были созданы VLAN, которые позволили отделить физическую топологию от логической. В стандарте IEEE 802.1Q, разработанном для VLAN, был введен новый формат Ethernet-фреймов.

## ВОПРОСЫ И ЗАДАЧИ

1. Для решения задачи используйте формулу, приведенную в данной главе, записав ее в общем виде. Фреймы для передачи произвольно приходят на 100-мегабитный канал. Если канал занят, фрейм ставится в очередь. Длина фрейма распределяется по экспоненциальному закону с математическим ожиданием, равным 10 000 бит/фрейм. Для каждой из приведенных ниже скоростей получения фреймов вычислите задержку (включая время ожидания в очереди и время передачи) фрейма средней длины:

   а) 90 фреймов/с;

   б) 900 фреймов/с;

   в) 9000 фреймов/с.

2. $N$ станций совместно используют канал чистой системы ALOHA, работающий со скоростью 56 Кбит/с. Каждая станция передает 1000-битный фрейм в среднем каждые 100 с, даже если предыдущий фрейм еще не был передан (например, станции могут буферизовать исходящие фреймы). Каково максимальное значение $N$?

3. Десять тысяч систем бронирования авиабилетов конкурируют за доступ к каналу дискретной системы ALOHA. В среднем, каждая станция делает 18 запросов в час. Каждый слот составляет 125 мкс. Приблизительно оцените общую нагрузку канала.

4. Оценка загруженности канала дискретной системы ALOHA с бесконечным количеством пользователей показала, что 10 % слотов не используется.

   а) Какова при этом нагрузка канала $G$?

   б) Какова пропускная способность?

   в) Канал недогружен или перегружен?

5. Самая низкая максимальная пропускная способность достигается при использовании чистой системы ALOHA, а самая высокая — при использовании CSMA с настойчивостью 0,01 (см. илл. 4.4). Для ее повышения каждый протокол идет на некоторые компромиссы, например обеспечивает расширенную аппаратную поддержку или увеличивает время ожидания. Расскажите, на какие компромиссы идут протоколы, представленные на илл. 4.4.
6. Какова длина слота конкуренции в CSMA/CD для:
   а) 2-километрового двухпроводного кабеля (скорость распространения сигнала составляет 82 % скорости распространения сигнала в вакууме);
   б) 40-километрового многомодового оптоволоконного кабеля (скорость распространения сигнала составляет 65 % скорости распространения сигнала в вакууме)?
7. Как долго станция *s* должна ждать начала передачи в худшем случае, если в LAN применяется базовый протокол битовой карты?
8. Объясните, как станция с более низким номером может лишиться возможности отправки пакета в протоколе двоичного отсчета.
9. Посмотрите на илл. 4.10. Допустим, что станции знают, что к передаче готовы станции *B*, *D*, *G* и *H*. Как протокол адаптивного прохода по дереву будет обходить этот граф, чтобы каждая из этих четырех станций могла отправить свой фрейм? Сколько дополнительных коллизий произойдет, если поиск будет начат с корня дерева?
10. Компания друзей собралась, чтобы поиграть в видеоигры с высокой степенью интерактивности и большой нагрузкой на процессор и сеть. Друзья играют по беспроводной сети с высокой пропускной способностью. Беспроводной сигнал не может проходить сквозь стены, но все они находятся в одной комнате. Какой протокол лучше использовать при такой конфигурации: ненастойчивый протокол CSMA или протокол маркерного кольца? Обоснуйте свой ответ.
11. Группа из $2^n$ станций осуществляет арбитраж доступа к общему кабелю, используя протокол адаптивного прохода по дереву. В определенный момент две из них готовы к передаче. Назовите минимальное, максимальное и среднее количество слотов, необходимых для обхода дерева, если $2^n >> 1$?
12. Рассмотренные нами беспроводные LAN используют протоколы CSMA/CA и RTS/CTS вместо CSMA/CD. При каких условиях, если таковые имеются, можно использовать протокол CSMA/CD?
13. Шесть станций, обозначенных буквами *A*–*F*, взаимодействуют друг с другом по протоколу MACA. Могут ли одновременно произойти две передачи данных? Обоснуйте свой ответ.
14. В семиэтажном офисном здании на каждом этаже расположено по 15 офисов. В каждом офисе на стене установлен разъем для подключения терминала, так что в вертикальной плоскости эти разъемы образуют прямоугольную

сетку с расстоянием по 4 м между гнездами, как по горизонтали, так и по вертикали. Предполагая, что можно проложить кабель по прямой между любой парой гнезд, по горизонтали, вертикали или диагонали, сосчитайте, сколько метров кабеля потребуется для соединения всех гнезд при помощи:

а) звездообразной конфигурации с одним маршрутизатором посередине;

б) классической LAN стандарта 802.3.

15. Чему равна скорость в бодах классической 10-мегабитной сети Ethernet?
16. Как будет выглядеть манчестерский код для классической сети Ethernet при следующей двоичной последовательности: 0001110101?
17. В 10-мегабитной LAN длиной 1 км с протоколом CSMA/CD (не 802.3) скорость распространения сигнала составляет 200 м/мкс. Повторителей в этой системе нет. Длина фреймов данных равна 256 бит, включая 32 бита заголовка, контрольную сумму и другие накладные расходы. Первый слот после успешной передачи резервируется получателем, чтобы отправить 32-битный фрейм с подтверждением. Какова эффективная скорость передачи данных без учета накладных расходов, если при этом не возникает коллизий?
18. Представьте сеть CSMA/CD, которая работает со скоростью 1 Гбит/с в кабеле длиной 1 км без повторителей. Скорость сигнала составляет 200 000 км/с. Чему равен минимальный размер фрейма?
19. IP-пакет необходимо передать по сети Ethernet. Длина пакета — 60 байт, включая все служебные поля. Если не используется LLC, требуется ли заполнение Ethernet-фрейма? Если да, сколько байтов нужно добавить?
20. Фреймы Ethernet должны быть не короче 64 байт, чтобы в случае коллизии на дальнем конце провода передатчик продолжал передачу. В сетях типа Fast Ethernet минимальный размер фрейма также равен 64 байтам, однако биты могут выдаваться в 10 раз чаще, чем в классическом варианте Ethernet. Каким образом в системе удалось сохранить прежний минимальный размер фрейма?
21. Согласно спецификации 1000Base-SX, генератор синхронизирующего сигнала должен работать с частотой 1250 МГц, хотя гигабитный Ethernet обеспечивает максимальную скорость передачи, равную только 1 Гбит/с. Используется ли эта более высокая скорость для того, чтобы обеспечить дополнительный запас безопасности? Если нет, то зачем она нужна?
22. Сколько фреймов в секунду может обрабатывать Gigabit Ethernet? Хорошо подумайте перед тем, как отвечать. *Подсказка*: имеет значение тот факт, что это именно *Gigabit* Ethernet.
23. Назовите сетевую технологию, позволяющую паковать фреймы друг за другом. Чем выгодно такое свойство?
24. На илл. 4.27 показаны четыре станции: *A*, *B*, *C* и *D*. Какая из двух последних станций находится ближе к станции *A* и почему?
25. Приведите пример, показывающий, что RTS/CTS в протоколе 802.11 немного отличается от RTS/CTS в протоколе MACA.

26. Взгляните на илл. 4.33 (б). Допустим, что все станции, мосты и концентраторы, показанные на этом рисунке, являются беспроводными, а соединения между станциями показывают, что они находятся в пределах досягаемости друг от друга. Если *B2* передает данные *D*, в то время как *B1* хочет начать передачу для *A* и *H1* хочет начать передачу для *F*, какие пары станций являются скрытыми, а какие — засвеченными?
27. В беспроводной LAN с одной точкой доступа есть 10 клиентских станций. У четырех из них скорость передачи данных составляет 6 Мбит/с, еще у четырех — 18 Мбит/с и у оставшихся двух — 54 Мбит/с. Какова скорость передачи каждой станции, когда все 10 станций отправляют данные одновременно и при этом

    а) не используется TXOP;

    б) используется TXOP?
28. Пусть по 11-мегабитной локальной сети 802.11b передаются друг за другом по радиоканалу 64-байтные фреймы с вероятностью ошибки $10^{-7}$ на бит. Сколько фреймов в секунду в среднем будет искажаться при передаче?
29. Два устройства подключены к одной и той же сети стандарта 802.11, и оба загружают из интернета большие файлы. Объясните, каким образом одно устройство может получить более высокую скорость передачи данных по сравнению со вторым за счет использования (некорректного) предоставляемого стандартом 802.11 механизма QoS.
30. На илл. 4.28 показано, как в стандарте 802.11 используется разное время ожидания для фреймов с разными приоритетами. Этот подход исключает замедление такого высокоприоритетного трафика, как передача фреймов с данными реального времени, из-за передачи обычного трафика. В чем состоит недостаток данного подхода?
31. Назовите две причины, по которым в сетях могут использоваться корректирующие коды вместо кодов для обнаружения ошибок и повторной передачи.
32. Почему режим PCF лучше подходит для версий стандарта 802.11, работающих на более высоких частотах?
33. Недостатком профилей Bluetooth является то, что они существенно усложняют протокол. В каких ситуациях эти профили могут рассматриваться как преимущество с точки зрения приложений?
34. Представьте сеть, в которой станции обмениваются данными с помощью лазерных лучей, как показано на илл. 2.11. Объясните, в чем состоит ее сходство и отличие от Ethernet и 802.11 и как это учесть при проектировании протоколов канального уровня и подуровня MAC для этой сети.
35. На илл. 4.30 видно, что устройство Bluetooth может находиться одновременно в двух пикосетях. Почему одно и то же устройство не может являться главным сразу в обеих пикосетях?
36. Каков максимальный размер поля данных для фрейма Bluetooth с тремя слотами на базовой скорости? Обоснуйте свой ответ.

37. Bluetooth поддерживает два типа соединений между главным и подчиненным узлами. Что это за соединения и для чего они предназначены?
38. Как было упомянуто выше, эффективность однослотового фрейма с кодированием повторения составляет приблизительно 13 % на базовой скорости данных. Какова будет эффективность при использовании пятислотового фрейма?
39. Фреймы-«маяки» в версии стандарта 802.11 с методом частотных скачков содержат время задержки. Как вы думаете, аналогичные фреймы-«маяки» в Bluetooth также содержат время задержки? Поясните свой ответ.
40. Коммутатор, предназначенный для работы с Fast Ethernet, имеет системную плату, которая может передавать данные со скоростью 10 Гбит/с. Сколько фреймов в секунду может обработать такой коммутатор?
41. Рассмотрите расширенную LAN с мостами *B1* и *B2*, показанную на илл. 4.33 (б). Допустим, что хеш-таблицы обоих мостов пусты. Как будет выглядеть хеш-таблица моста *B2* после следующей последовательности передач:

    а) станция *B* отправляет фрейм станции *E*;

    б) станция *F* отправляет фрейм станции *A*;

    в) станция *A* отправляет фрейм станции *B*;

    г) станция *G* отправляет фрейм станции *E*;

    д) станция *D* отправляет фрейм станции *C*;

    е) станция *C* отправляет фрейм станции *A*.

    Предполагается, что каждый фрейм отправляется после получения предыдущего фрейма.

42. Рассмотрите расширенную LAN с мостами *B1* и *B2*, показанную на илл. 4.33 (б). Допустим, что хеш-таблицы обоих мостов пусты. Какие из этих передач приведут к широковещательной рассылке:

    а) станция *A* отправляет фрейм станции *C*;

    б) станция *B* отправляет фрейм станции *E*;

    в) станция *C* отправляет фрейм станции *B*;

    г) станция *G* отправляет фрейм станции *C*;

    д) станция *E* отправляет фрейм станции *F*;

    е) станция *D* отправляет фрейм станции *C*?

    Предполагается, что каждый фрейм отправляется после получения предыдущего фрейма.

43. Рассмотрите расширенную LAN с мостами *B1* и *B2*, показанную на илл. 4.33 (б). Допустим, что хеш-таблицы обоих мостов пусты. Перечислите все порты, в которые будет отправлен пакет при следующей последовательности передач:

    а) станция *A* отправляет пакет станции *C*;

    б) станция *E* отправляет пакет станции *F*;

в) станция *F* отправляет пакет станции *E*;

г) станция *G* отправляет пакет станции *E*;

д) станция *D* отправляет пакет станции *A*;

е) станция *B* отправляет пакет станции *F*.

44. Взгляните на илл. 4.36. Допустим, что к мостам *B4* и *B5* будет подключен дополнительный мост *B0*. Нарисуйте новое связующее дерево для этой топологии.

45. Рассмотрите сеть, изображенную на илл. 4.39. Если станция, подключенная к мосту *B1*, внезапно станет белой, нужно ли будет менять какие-либо метки? Если да, то какие?

46. Имеется локальная сеть Ethernet с семью мостами. Мост 0 подключен к мостам 1 и 2. Мосты 3, 4, 5 и 6 подключены и к мосту 1, и к мосту 2. Допустим, подавляющее большинство фреймов передается станциям, подключенным к мосту 2. Нарисуйте связующее дерево для протокола Ethernet, а затем – альтернативный вариант такого дерева, позволяющий уменьшить среднюю задержку фрейма.

47. Имеются две сети Ethernet. В сети 1 станции подключены к концентратору с помощью дуплексных кабелей. В сети 2 станции подключены к коммутатору с помощью полудуплексных кабелей. Почему в первом и втором случае следует (или нет) использовать протокол CSMA/CD?

48. Коммутаторы с ожиданием имеют преимущество перед сквозными коммутаторами при обработке поврежденных фреймов. Объясните почему.

49. В разделе 4.8.3 упоминалось, что некоторые мосты могут не входить в связующее дерево. Опишите в общих чертах ситуацию, при которой это возможно.

50. Чтобы VLAN заработала, мостам и коммутаторам нужны конфигурационные таблицы. А что, если в сетях, показанных на илл. 4.39, использовать концентраторы вместо коммутаторов? Понадобятся ли им конфигурационные таблицы? Ответ поясните.

51. На илл. 4.40 коммутатор обычного оконечного домена (справа) является VLAN-совместимым. Можно было бы поставить здесь обычный коммутатор? Если да, то как должна работать система? Если нет, то почему?

52. Перехватите трассировку сообщений вашего компьютера, несколько раз запустив на несколько минут неизбирательный режим. Создайте симулятор одного канала связи и реализуйте протоколы CSMA/CD. Оцените их эффективность, используя собственную трассировку для представления нескольких станций, конкурирующих за доступ к каналу. Насколько эта трассировка отражает загруженность канального уровня?

53. Напишите программу, симулирующую поведение протокола CSMA/CD в системе Ethernet с *N* станциями, готовыми к передаче во время отправки фрейма по каналу. Программа должна выдавать уведомление каждый раз, когда одна из станций успешно начинает передачу своего фрейма. Пусть такт системных часов совпадает с началом каждого слота (51,2 мкс),

а обнаружение коллизии с отправкой преднамеренных помех занимает один слот. Все фреймы имеют максимально допустимую длину.

54. Скачайте программу Wireshark с сайта www.wireshark.org. Это бесплатная программа с открытым исходным кодом, позволяющая отслеживать состояние сетей и детально просматривать, что именно происходит в трафике. Научитесь ее использовать, просмотрев одно из множества обучающих видео на YouTube. В интернете есть немало ресурсов с описанием того, какие эксперименты можно осуществить с помощью этой программы. Это позволит вам получить хороший практический опыт при работе с сетями.

# ГЛАВА 5
# Сетевой уровень

Сетевой уровень отвечает за передачу пакетов от отправителя получателю. Чтобы достичь пункта назначения, пакет может совершить множество скачков между маршрутизаторами. Это резко контрастирует с деятельностью канального уровня, цель которого намного скромнее — просто перемещать фреймы с одного конца «провода» (виртуального) на другой. Таким образом, сетевой уровень является самым нижним уровнем, имеющим дело с передачей данных по всему пути от начала до конца.

Для достижения этих целей сетевой уровень должен знать топологию сети (то есть весь набор маршрутизаторов и каналов) и рассчитывать подходящие маршруты, даже если она достаточно крупная. При выборе пути он также должен заботиться о равномерной нагрузке на маршрутизаторы и линии связи. Наконец, когда источник и адресат находятся в разных независимо управляемых сетях (иногда называемых автономными системами), приходится решать ряд дополнительных вопросов, включая проблему координации потоков трафика в рамках нескольких сетей и управление загруженностью сети. Эти проблемы обычно решаются на сетевом уровне; операторам сетей часто приходится делать это вручную. Обычно они самостоятельно перенастраивали сетевой уровень путем изменения низкоуровневой конфигурации. Однако с появлением программно-конфигурируемых сетей и программируемого оборудования появилась возможность настраивать сетевой уровень с помощью более эффективного ПО, и даже полностью переопределять его функции. В данной главе мы рассмотрим и проиллюстрируем все эти вопросы — в основном на примере интернета и протокола его сетевого уровня, IP (Internet Protocol).

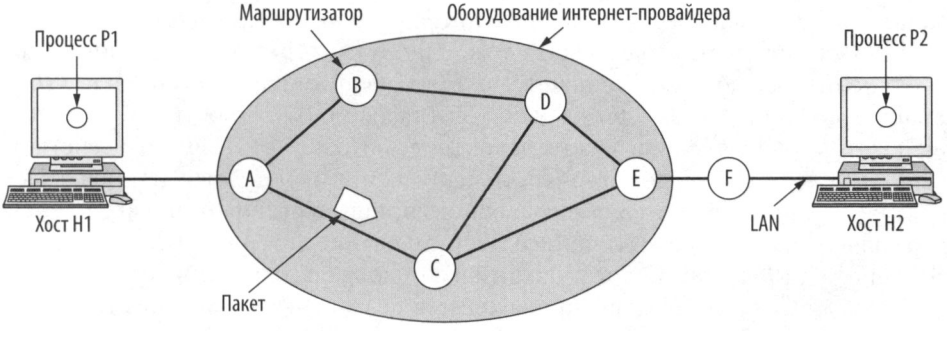
## 5.1. ВОПРОСЫ РАЗРАБОТКИ СЕТЕВОГО УРОВНЯ

В следующих разделах мы рассмотрим некоторые вопросы, с которыми приходится сталкиваться разработчикам сетевого уровня. К ним относятся службы, предоставляемые транспортному уровню, и внутреннее устройство сети.

### 5.1.1. Метод коммутации пакетов с ожиданием

Прежде чем начать подробное рассмотрение сетевого уровня, имеет смысл вспомнить окружение, в котором он функционирует (илл. 5.1). Основными

компонентами сети являются устройства интернет-провайдера (ISP) — маршрутизаторы, коммутаторы и промежуточные узлы, соединенные линиями связи (они показаны на рисунке в сером овале), а также устройства пользователя (находятся вне овала). Хост *H1* напрямую соединен с одним из маршрутизаторов провайдера *A* (это может быть домашний компьютер, подключенный к DSL-модему). Хост *H2*, напротив, входит в LAN (например, в офисную сеть Ethernet) с маршрутизатором *F*, принадлежащим клиенту, который с ним работает. Этот маршрутизатор связывается с провайдером по выделенной линии. Мы показали *F* вне овала, потому что он не принадлежит ISP. Однако в контексте данной главы мы будем считать маршрутизаторы клиента частью сети ISP, поскольку в них применяются те же самые алгоритмы, что и в маршрутизаторах интернет-провайдеров (именно алгоритмы будут основным предметом рассмотрения).




**Илл. 5.1.** Окружение, в котором функционируют протоколы сетевого уровня

Система работает следующим образом. Хост, у которого есть пакет для передачи, отправляет его на ближайший маршрутизатор — либо в своей LAN, либо провайдеру по двухточечному соединению (например, по каналу ADSL или по линии кабельного телевидения). Там пакет хранится до тех пор, пока не будет принят целиком и не пройдет полную обработку, включая верификацию контрольной суммы. Затем он передается по цепочке маршрутизаторов, которая в итоге приводит к пункту назначения. Такой механизм называется коммутацией пакетов с промежуточным хранением (store-and-forward), и мы уже рассматривали его в предыдущих главах.

## 5.1.2. Службы, предоставляемые транспортному уровню

Службы для транспортного уровня предоставляются сетевым уровнем посредством интерфейса между ними. Главное — определить, какими должны быть эти службы. Их разработка требует особой аккуратности, и при этом необходимо учитывать следующее:

1. Службы сетевого уровня не должны зависеть от технологии маршрутизатора.
2. Количество, тип и топология имеющихся маршрутизаторов не должны влиять на транспортный уровень.

3. Сетевые адреса, доступные транспортному уровню, должны использовать единую систему нумерации, даже между LAN и WAN.

С учетом этих условий разработчики абсолютно свободны в написании детальной спецификации служб для транспортного уровня. Эта свобода часто перерастает в яростную борьбу между двумя непримиримыми сторонами. В центре дискуссии — вопрос о том, какие службы должен предоставлять сетевой уровень: с установлением соединения или без него.

Один лагерь (представленный интернет-сообществом) заявляет, что работа маршрутизаторов сводится к перемещению пакетов. С этой точки зрения (основанной на сорокалетнем опыте работы с реальными компьютерными сетями) сеть по своей природе ненадежна, независимо от того, как она спроектирована. Хосты должны учитывать это и самостоятельно защищаться от ошибок (то есть заниматься их обнаружением и коррекцией) и управлять потоком.

Из этого следует, что сетевой службе не нужно требовать установления соединения, она должна в основном состоять из примитивов `SEND PACKET` (отправить пакет) и `RECEIVE PACKET` (получить пакет). В частности, сюда нельзя включать упорядочивание пакетов и контроль потока — все равно эти действия выполнит хост. Нет смысла выполнять одну и ту же работу дважды. Такое рассуждение — пример применения **сквозного принципа (end-to-end argument)**, оказавшего значительное влияние на формирование интернета (Зальцер и др.; Saltzer et al., 1984). Кроме того, каждый пакет должен содержать полный адрес получателя, так как передача производится независимо от предшествующих пакетов.

Другой лагерь, представленный телефонными компаниями, утверждает, что сеть должна предоставлять надежную службу с установлением соединения. Компании считают, что 100 лет успешного управления телефонными системами по всему миру — серьезный аргумент в их пользу. По их мнению, определяющим фактором является QoS, и без установления соединения в сети очень сложно добиться каких-либо приемлемых результатов, особенно когда дело касается трафика в реальном времени (например, передачи голоса и видео).

Этот спор актуален даже спустя несколько десятилетий. Когда-то самые популярные сети передачи данных (например, X.25 в 1970-х годах и Frame Relay в 1980-х) представляли собой системы, ориентированные на установление соединения. Но с появлением ARPANET и на ранних этапах развития интернета сетевые уровни без установления соединения получили широкое распространение. Сегодня неизменным символом успеха является протокол IP. На его популярность не повлияло даже появление требующей соединения технологии ATM, созданной в 1980-х годах с целью заменить IP; в настоящее время ATM применяется только в отдельных случаях, тогда как IP охватывает все телефонные сети. Однако требования QoS растут, и для интернета разрабатываются инструменты, предполагающие установление соединения. Примеры таких технологий — мультипротокольная коммутация по меткам (мы рассмотрим ее в этой главе) и VLAN (см. главу 4). Обе эти технологии сейчас широко используются.

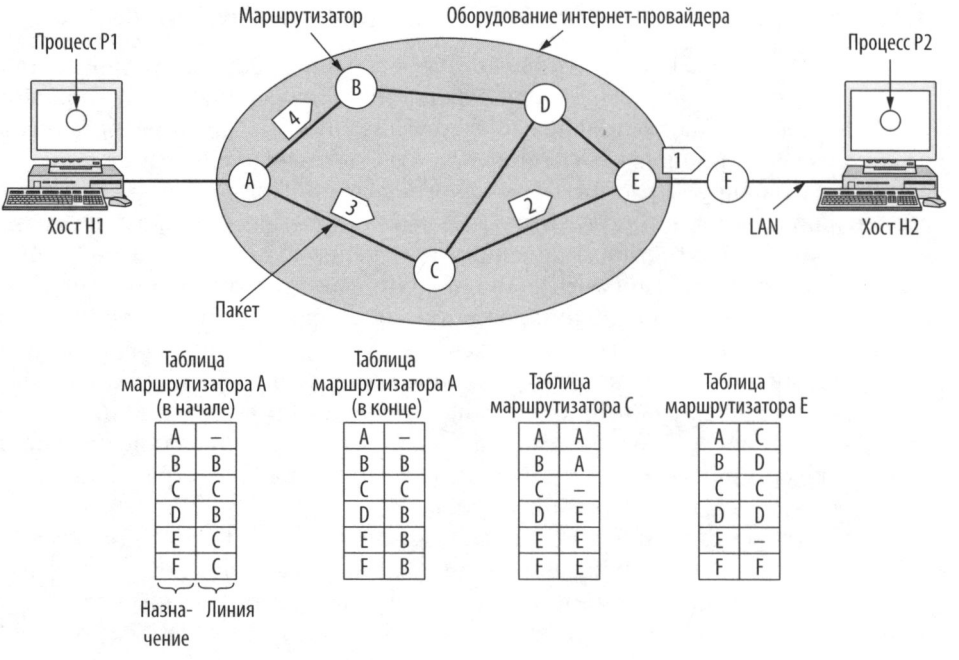
### 5.1.3. Реализация службы без установления соединения

Мы рассмотрели два класса служб, которые сетевой уровень может предоставлять своим пользователям. Теперь перейдем к обсуждению его устройства. Возможны два варианта в зависимости от типа службы. Если предоставляется служба без установления соединения, пакеты по отдельности передаются в сеть и их маршруты рассчитываются независимо. При этом никакой предварительной настройки не требуется. В этом случае пакеты часто называют **дейтаграммами (datagrams)**, по аналогии с телеграммами, а сети — **дейтаграммными (datagram network)**. При использовании службы, ориентированной на установление соединения, весь путь от маршрутизатора-отправителя до маршрутизатора-получателя должен быть определен до начала передачи. Такое соединение называется **виртуальным каналом (VC, Virtual Circuit)**, по аналогии с физическими каналами, устанавливаемыми в традиционных телефонных сетях. Сама система называется **сетью виртуального канала (virtual-circuit network)**. В этом разделе мы обсудим дейтаграммные сети; в следующем — сети виртуального канала.

Рассмотрим принцип работы дейтаграммных сетей. Пусть у процесса *P1* (илл. 5.2) есть длинное сообщение для *P2*. Он передает свое послание транспортному уровню и информирует его о том, что данные необходимо доставить процессу *P2* на хосте *H2*. Код транспортного уровня выполняется на хосте *H1*; более того, обычно он является частью операционной системы. Заголовок транспортного уровня вставляется в начало сообщения, и в таком виде оно передается на сетевой уровень. Как правило, это просто еще одна процедура операционной системы.

Предположим, что в нашем примере сообщение в четыре раза длиннее максимального размера пакета. Поэтому сетевой уровень должен разбить его на четыре части (1, 2, 3 и 4) и отправить их все поочередно на маршрутизатор *A* с использованием какого-нибудь протокола двухточечного соединения, например PPP. Здесь в игру вступает провайдер. Каждый маршрутизатор имеет свою внутреннюю таблицу, по которой он определяет дальнейший путь пакета до любого возможного пункта назначения. Каждая запись таблицы состоит из двух полей: адресат и ведущая к нему исходящая линия. Во втором поле могут использоваться только линии, непосредственно соединенные с данным маршрутизатором. Так, например, на илл. 5.2 у маршрутизатора *A* имеются только две исходящие линии: к *B* и к *C*. Поэтому все входящие пакеты передаются в какую-то из этих двух точек, даже если они не являются адресатами. Изначальная таблица маршрутизации *A* показана на илл. 5.2 с пометкой «в начале».

В маршрутизаторе *A* пакеты 1, 2 и 3, поступившие на вход, кратковременно сохраняются для верификации контрольной суммы. Затем в соответствии с таблицей *A* каждый пакет отправляется в новом фрейме по исходящему соединению на маршрутизатор *C*. Далее пакет 1 уходит на *E*, затем — на *F*. После этого он передается внутри фрейма по LAN на хост *H2*. Пакеты 2 и 3 следуют по тому же маршруту.




**Илл. 5.2.** Маршрутизация внутри дейтаграммной сети

Однако с пакетом 4 происходит нечто другое. После прибытия на *A* он пересылается на маршрутизатор *B*, несмотря на то что его адресат — *F*, как и у первых трех пакетов. По какой-то причине *A* решил отправить пакет 4 по новому пути. Может быть, в результате отправки трех пакетов на линии *ACE* возник затор и маршрутизатор решил обновить свою таблицу (она показана на илл. 5.2 с пометкой «в конце»). Алгоритмы, управляющие таблицами и принимающие решения о выборе маршрута, называются **алгоритмами маршрутизации (routing algorithms)**. Им будет уделено основное внимание в этой главе. Как мы увидим, существует несколько типов таких алгоритмов.

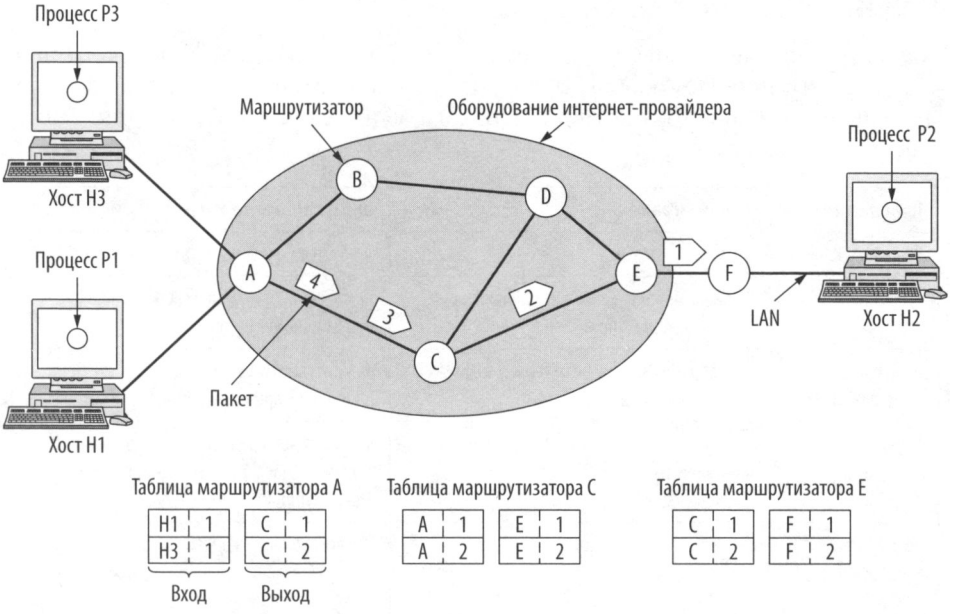
Протокол IP, составляющий основу всего интернета, является наиболее ярким примером сетевой службы без установления соединения. Каждый пакет содержит IP-адрес назначения, по которому маршрутизатор осуществляет индивидуальную отправку пакета. В пакетах IPv4 используются адреса длиной 32 бита, а в IPv6 — 128 бит. Далее мы подробно обсудим IP и его версии.

## 5.1.4. Реализация службы с установлением соединения

Службе, ориентированной на установление соединения, нужна сеть виртуального канала. Давайте разберемся, как она работает. Идея виртуальных каналов состоит том, чтобы не выбирать новый маршрут для каждого пакета, как на илл. 5.2. Вместо этого путь от источника до адресата выбирается в процессе установления соединения и хранится в специальных таблицах, встроенных

в маршрутизаторы. Один и тот же маршрут используется для всего трафика, проходящего через данное соединение. Именно так работает телефонная система. Когда соединение разрывается, виртуальный канал прекращает свое существование. При использовании службы с установлением соединения каждый пакет включает в себя идентификатор виртуального канала.

В качестве примера рассмотрим ситуацию, изображенную на илл. 5.3. Хост *H1* установил соединение с хостом *H2*. Это соединение запоминается и становится первой записью во всех таблицах маршрутизации. Так, первая строчка таблицы маршрутизатора *A* говорит о том, что если пакет с идентификатором соединения 1 пришел с хоста *H1*, то его нужно направить на *C* с идентификатором соединения 1. Точно так же первая запись *C* направляет пакет на *E* все с тем же идентификатором соединения 1.




**Илл. 5.3.** Маршрутизация в сети виртуального канала

Теперь выясним, что произойдет, если хост *H3* захочет установить соединение с *H2*. Он выбирает идентификатор соединения 1 (на данный момент это первое и единственное соединение) и просит сеть установить виртуальный канал. Таким образом, в таблице появляется вторая запись. Обратите внимание, что здесь возникает конфликт: *A* может отличить пакеты соединения 1, пришедшие с *H1*, от пакетов соединения 1, пришедших с *H3*, но *C* такой возможности не имеет. Поэтому *A* присваивает новый идентификатор соединения исходящему трафику и тем самым создает второе соединение. Для предотвращения таких конфликтов маршрутизаторы получили возможность менять идентификаторы соединения в исходящих пакетах.

Одним из примеров сетевой службы, ориентированной на соединение, является мультипротокольная коммутация по меткам (MultiProtocol Label Switching, MPLS). Она используется в сетях интернет-провайдеров; при этом IP-пакеты получают MPLS-заголовок, содержащий 20-битный идентификатор соединения или метку. Если провайдер устанавливает длительное соединение для передачи крупных объемов данных, MPLS часто остается невидимым для клиентов. Однако в последнее время эта служба применяется все чаще для обеспечения необходимого уровня QoS и решения других задач, связанных с управлением трафиком на уровне провайдера. К обсуждению MPLS мы еще вернемся далее в этой главе.

## 5.1.5. Сравнение сетей виртуальных каналов и дейтаграммных сетей

Как виртуальные каналы, так и дейтаграммы имеют своих сторонников и противников. Попробуем обобщить аргументы обеих сторон. Основные вопросы перечислены на илл. 5.4, хотя для каждого из этих пунктов наверняка можно найти контраргументы.

| Проблема | Дейтаграммы | Виртуальные каналы |
|---|---|---|
| Установка канала | Не требуется | Требуется |
| Адресация | Каждый пакет содержит полный адрес отправителя и получателя | Каждый пакет содержит короткий номер виртуального канала |
| Информация о состоянии | Маршрутизаторы не содержат информации о состоянии | Каждый виртуальный канал требует места в таблице маршрутизатора |
| Маршрутизация | Маршрут каждого пакета выбирается независимо | Маршрут выбирается при установке виртуального канала. Каждый пакет следует по этому маршруту |
| Эффект от выхода маршрутизатора из строя | Никакого, кроме потерянных пакетов | Все виртуальные каналы, проходившие через отказавший маршрутизатор, прекращают существование |
| QoS | Трудно реализовать | Легко реализуется при наличии достаточного количества ресурсов для каждого виртуального канала |
| Борьба с перегрузкой | Трудно реализовать | Легко реализуется при наличии достаточного количества ресурсов для каждого виртуального канала |

**Илл. 5.4.** Сравнение виртуальных каналов и дейтаграмм

Выбирая между виртуальными каналами и дейтаграммами, нужно сопоставить их плюсы и минусы в ряде вопросов. Во-первых, существует компромисс между временем установки соединения и временем обработки адреса.

Виртуальный канал требует определенных временных затрат на установку, однако в результате это существенно упрощает обработку пакетов данных. Чтобы понять, куда отправить пакет, маршрутизатору требуется всего лишь обратиться к таблице, используя номер канала. Дейтаграммная сеть не требует установки, но адрес получателя определяется с помощью более сложной процедуры поиска.

В связи с этим возникает другая проблема: адреса назначения в дейтаграммных сетях — глобальные, поэтому они намного длиннее, чем номера линий в сетях виртуальных каналов. При сравнительно небольшом размере пакетов включение полного адреса в каждый из них может привести к существенным издержкам, а следовательно, к снижению пропускной способности.

Еще одна проблема — объем памяти, выделяемый маршрутизатором для хранения таблиц. В дейтаграммной сети должно быть предусмотрено место для любого возможного адреса получателя, тогда как в системе виртуальных каналов — только для каждого канала. Но такое преимущество на деле обманчиво, поскольку пакеты, необходимые для установки соединения, используют адреса назначения так же, как и дейтаграммы.

Виртуальные каналы обладают некоторыми преимуществами в обеспечении QoS и предотвращении заторов в сети, так как ресурсы (например, буфер, пропускная способность, центральный процессор) можно зарезервировать заранее, во время установки соединения. Как только пакеты начнут приходить, необходимая пропускная способность и мощность маршрутизатора будут предоставлены. В дейтаграммной сети предотвращение заторов реализовать значительно сложнее.

В системах обработки транзакций (например, при запросе магазина на подтверждение оплаты картой) накладные расходы на установку соединения и удаление виртуального канала могут сильно снизить потребительские свойства сети. Если большая часть трафика будет такой, то использование виртуального канала не имеет особого смысла. Однако в случае длительных операций, таких как обмен данными через VPN внутри одной компании, постоянные виртуальные каналы (установленные вручную на месяцы и даже годы) могут оказаться полезными.

Недостаток виртуальных каналов — уязвимость в случае выхода из строя или временного выключения маршрутизатора. Даже если его включат через пару секунд, все проходившие через него каналы прервутся. Если же это произойдет в дейтаграммной сети, потеряются только те пакеты, которые находились на маршрутизаторе в данный момент (а скорее всего, даже они не пострадают, поскольку отправитель сразу же повторит передачу). Обрыв линии связи для виртуальных каналов является фатальным, но легко компенсируется в дейтаграммной системе. Кроме того, в ней маршрутизаторы могут обеспечить баланс трафика по всей сети, изменяя путь в процессе долгой передачи.

## 5.2. АЛГОРИТМЫ МАРШРУТИЗАЦИИ В РАМКАХ ОДНОЙ СЕТИ

Основная функция сетевого уровня заключается в маршрутизации пакетов от начальной до конечной точки. В этом разделе мы поговорим о том, как эта задача выполняется в рамках одного административного домена или одной автономной

системы. В большинстве сетей пакетам приходится проходить через несколько маршрутизаторов. Единственное исключение — широковещательные сети, но даже в них маршрутизация представляет собой проблему, если отправитель и получатель находятся в разных сегментах сети. Алгоритмы, выбирающие путь, и используемые ими структуры данных составляют основу проектирования сетевого уровня.

**Алгоритм маршрутизации (routing algorithm)** — это компонент программного обеспечения сетевого уровня, отвечающий за выбор выходной линии для отправки поступившего пакета. Если сеть использует дейтаграммную службу, выбор маршрута должен производиться заново для каждого пакета, так как оптимальный путь мог измениться. При использовании виртуальных каналов маршрут определяется только при создании нового канала, и после этого по нему следуют все пакеты данных. Этот подход называют **сеансовой маршрутизацией (session routing)**, так как путь существует на протяжении всего сеанса связи (например, пока вы подключены к VPN).

Важно различать маршрутизацию, при которой система выбирает путь для пакета, и пересылку, происходящую при его получении. Можно представить себе маршрутизатор как устройство, в котором выполняются два процесса. Один обрабатывает приходящие пакеты и выбирает для них исходящую линию по таблице маршрутизации; он называется **пересылкой (forwarding)**. Второй процесс отвечает за заполнение и обновление таблиц. Именно для этого нужен алгоритм маршрутизации.

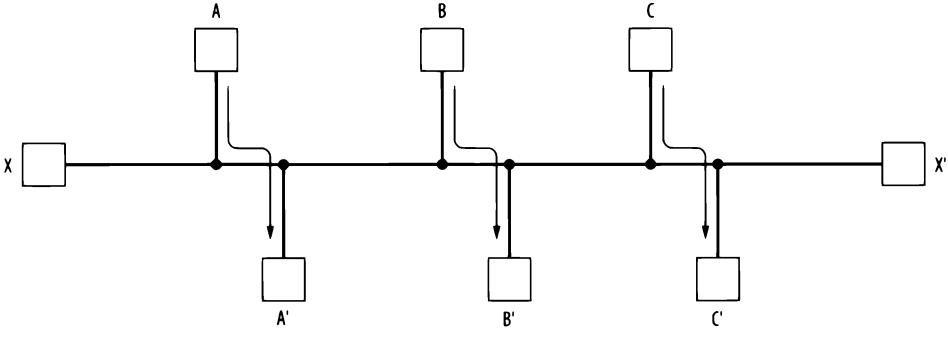
Вне зависимости от того, как выбирается путь — отдельно для каждого отправляемого пакета или только один раз при установлении нового соединения, — желательно, чтобы алгоритм маршрутизации обладал определенными свойствами: корректностью, простотой, надежностью, устойчивостью, равнодоступностью и эффективностью. Корректность и простота вряд ли требуют комментариев, а вот потребность в надежности не столь очевидна на первый взгляд. Крупные сети должны работать без системных сбоев непрерывно в течение многих лет. За это время в них будут происходить всевозможные отказы оборудования и программного обеспечения. Хосты, маршрутизаторы и линии связи будут постоянно выходить из строя, а топология неоднократно изменится. Алгоритм маршрутизации должен уметь с этим справляться, не прерывая выполнение задач на всех хостах. Представьте себе сеть, которая перезагружается при каждой поломке маршрутизатора!

Алгоритм маршрутизации также должен быть устойчивым. Существуют алгоритмы, которые никогда не сводятся к созданию фиксированного набора путей, независимо от того, как долго они работают. Устойчивый алгоритм достигает равновесия и остается в этом состоянии. Но он также должен быстро находить более подходящий набор маршрутов, поскольку соединение может прерваться до того, как будет достигнуто равновесие.

Такие свойства, как равнодоступность и эффективность, могут показаться очевидными — вряд ли кто-нибудь станет возражать против них, — однако они зачастую несовместимы. Для примера рассмотрим ситуацию на илл. 5.5. Предположим, что трафик между станциями $A$ и $A'$, $B$ и $B'$, а также $C$ и $C'$ настолько интенсивный, что горизонтальные линии связи полностью загружены. Чтобы

максимально увеличить общий поток данных, трафик между станциями $X$ и $X'$ нужно полностью остановить. К сожалению, станции $X$ и $X'$ могут с этим не согласиться. Очевидно, необходим компромисс между общей эффективностью и равным выделением канала для каждой отдельной станции.

Прежде чем начинать поиск приемлемого соотношения равнодоступности и эффективности, следует решить, что именно нужно оптимизировать. Для увеличения эффективности передачи данных по сети можно минимизировать среднее время задержки или увеличить общую пропускную способность. Однако эти действия также противоречат друг другу, поскольку работа любой системы с очередями ожидающих обработки пакетов, да еще и на максимуме производительности оборудования, приводит к увеличению времени ожидания. В качестве компромисса многие сети пытаются минимизировать расстояние, которое должен пройти пакет, или снизить количество пересылок для каждого пакета. В обоих случаях уменьшается задержка и объем полосы, требуемый для каждого пакета, благодаря чему повышается пропускная способность всей сети.




**Илл. 5.5.** Конфликт между справедливостью и эффективностью сети

Алгоритмы маршрутизации можно разделить на два основных класса: неадаптивные и адаптивные. **Неадаптивные алгоритмы (nonadaptive algorithms)** не учитывают текущую топологию и не измеряют трафик. Вместо этого путь для каждой пары станций определяется заранее, в автономном режиме, а список маршрутов загружается в маршрутизаторы во время загрузки сети. Эта процедура называется **статической маршрутизацией (static routing)**. Она не реагирует на сбои, поэтому обычно используется в тех случаях, когда выбор маршрута очевиден. Например, маршрутизатор $F$ на илл. 5.3 должен отправлять пакеты в сеть, на маршрутизатор $E$, независимо от конечного адреса назначения.

**Адаптивные алгоритмы (adaptive algorithms)**, напротив, реагируют на изменения топологии, а иногда и на загруженность линий. Такие алгоритмы **динамической маршрутизации (dynamic routing)** различаются по нескольким параметрам. Они могут получать информацию от соседних маршрутизаторов или от всех маршрутизаторов сети. Некоторые меняют путь при смене топологии, другие — через определенные равные интервалы времени по мере изменения

нагрузки. Наконец, для оптимизации они используют разные показатели: расстояние, количество транзитных участков (переходов) или ожидаемое время передачи.

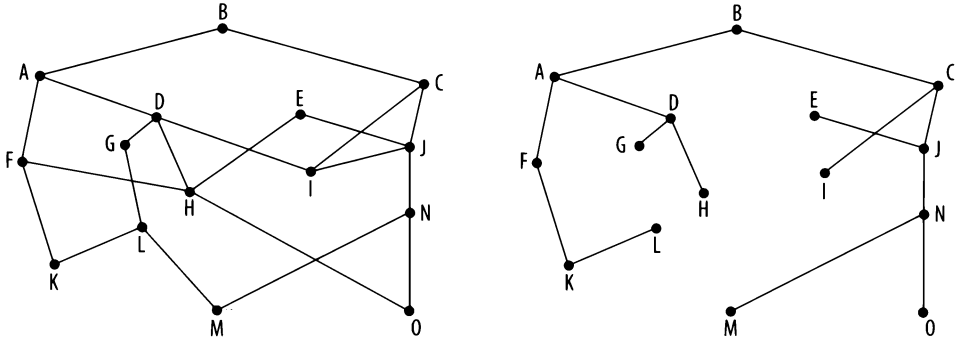
В следующих разделах мы рассмотрим множество алгоритмов маршрутизации. Помимо передачи пакета от источника к месту назначения, они определяют модель доставки. Пакет может отправляться нескольким получателям из списка, одному из них или всем. Представленные здесь алгоритмы принимают решения на основании топологии; вопрос об учете трафика при маршрутизации мы обсудим в разделе 5.3.

## 5.2.1. Принцип оптимальности

Прежде чем перейти к конкретным алгоритмам, сформулируем общее утверждение, описывающее оптимальные маршруты независимо от топологии или трафика, — **принцип оптимальности (optimality principle)**; см. работу Беллмана (Bellman, 1957).

Согласно этому принципу, если маршрутизатор $J$ расположен на оптимальном пути от маршрутизатора $I$ до маршрутизатора $K$, то оптимальный путь от $J$ до $K$ частично с ним совпадет. Чтобы убедиться в этом, обозначим часть пути от $I$ до $J$ как $r_1$, а остальную часть пути — $r_2$. Если бы существовал более выгодный путь от $J$ до $K$, чем $r_2$, то его можно было объединить с $r_1$, чтобы улучшить путь от $I$ до $K$, что противоречит первоначальному утверждению о том, что $r_1r_2$ — оптимальный путь.

Применив данный принцип, можно представить набор оптимальных маршрутов от всех источников ко всем обозначенным адресатам в виде дерева с получателем в качестве корня. Оно называется **входным деревом (sink tree)**. На илл. 5.6 (б) изображено такое дерево для сети, показанной на илл. 5.6 (а). Расстояние в этом случае измеряется количеством транзитных участков. Цель любого алгоритма маршрутизации состоит в том, чтобы обнаружить и использовать входные деревья для всех маршрутизаторов.




**Илл. 5.6.** (а) Сеть. (б) Входное дерево для маршрутизатора B

Обратите внимание, что входное дерево не всегда уникально; у одной сети может быть несколько деревьев с идентичной длиной пути. Если выбрать все возможные маршруты, получится более общая структура под названием **направленный ациклический граф (directed acyclic graph, DAG)**. В таких графах нет циклов. Для удобства мы также будем называть их входным деревом. В обоих случаях предполагается, что пути не мешают друг другу (к примеру, затор на одном маршруте не приводит к изменению другого).

Входные деревья (как и любые другие) не содержат циклов: каждый пакет доставляется за конечное и ограниченное число пересылок. Но на практике все не так просто. Маршрутизаторы (и соединения) могут отключаться и снова появляться в сети во время выполнения операции, поэтому у них может быть разное представление о текущей топологии. Кроме того, мы еще не обсудили, как маршрутизатор получает информацию для вычисления входного дерева. Он может самостоятельно собирать данные или получать их как-то иначе (мы рассмотрим этот вопрос чуть позже). В любом случае, принцип оптимальности и входное дерево — это ориентиры, по которым можно оценивать алгоритмы маршрутизации.

## 5.2.2. Алгоритм поиска кратчайшего пути

Начнем изучение алгоритмов маршрутизации с простого метода вычисления оптимальных путей с учетом полной информации о сети. Целью распределенного алгоритма является обнаружение таких путей, даже если маршрутизатор не располагает всеми сведениями о сети.

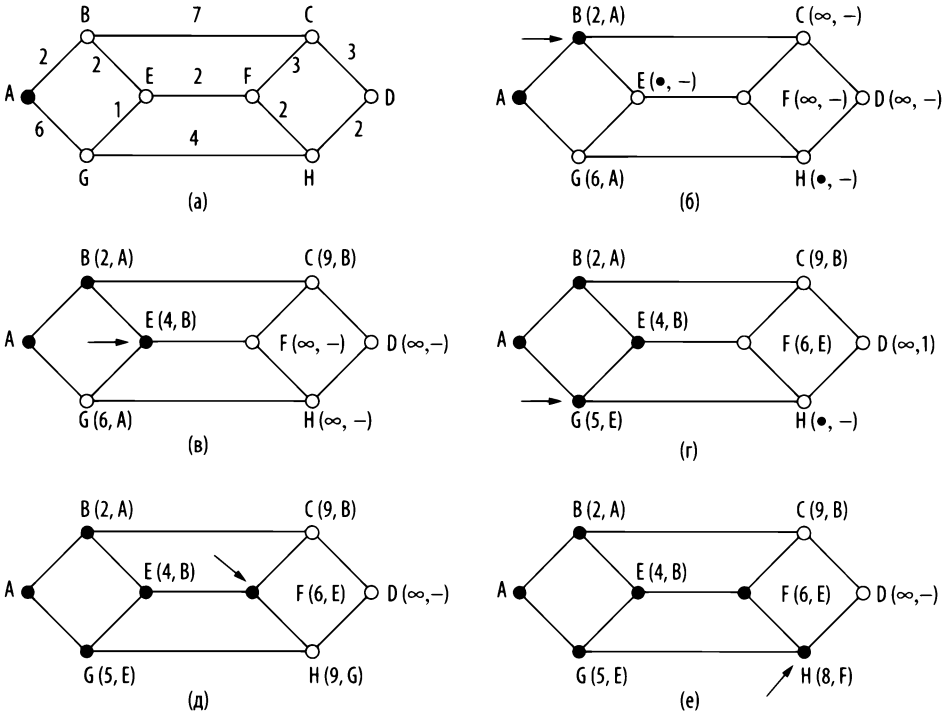
Идея заключается в построении графа сети, в котором каждый узел соответствует маршрутизатору, а каждое ребро — линии связи или ее участку. При выборе маршрута между двумя маршрутизаторами алгоритм просто находит кратчайший путь между ними на графе.

Концепция **кратчайшего пути (shortest path)** требует некоторого пояснения. Один из способов измерения длины пути состоит в подсчете количества транзитных участков. В этом случае пути *ABC* и *ABE* на илл. 5.7 имеют одинаковую длину. Можно измерять расстояния в километрах. Тогда получается, что *ABC* гораздо длиннее, чем *ABE* (предполагается, что рисунок изображен с соблюдением пропорций).

Однако помимо количества транзитных участков и физической длины линий есть и другие параметры. Например, каждому ребру можно присвоить метку средней задержки стандартного тестового пакета, которая измеряется каждый час. В таком графе кратчайшим является самый быстрый путь, а не путь с наименьшим количеством ребер или километров.

В общем случае метки ребер могут обозначать функции расстояния, пропускной способности, средней загруженности, стоимости связи, величины задержки и других факторов. Изменяя весовую функцию, алгоритм может вычислять «кратчайший» путь с учетом любого критерия или их комбинаций.

Известно несколько алгоритмов вычисления кратчайшего пути между двумя узлами графа. Один из них был создан знаменитым Дейкстрой (Dijkstra) в 1959 году. Алгоритм находит кратчайшие пути между отправителем и всеми




**Илл. 5.7.** Первые шесть шагов вычисления кратчайшего пути от A к D. Стрелка указывает на рабочий узел

возможными получателями в сети. Каждому узлу присваивается метка (в скобках), означающая длину наилучшего известного пути до узла отправителя. Эти расстояния должны быть неотрицательными; условие выполняется, поскольку расстояния основаны на реальных величинах — пропускной способности или времени задержки. Изначально маршруты неизвестны, поэтому все узлы помечаются символом бесконечности. По мере работы алгоритма и нахождения путей метки узлов меняются, показывая оптимальные маршруты. Метка может быть постоянной или временной. Сначала все они являются временными. Когда выясняется, что метка действительно соответствует кратчайшему пути, она становится постоянной и в дальнейшем не изменяется.

Чтобы показать, как работает этот алгоритм, рассмотрим взвешенный ненаправленный граф на илл. 5.7 (а), где весовые коэффициенты отражают, например, расстояние. Нужно найти кратчайший путь от *A* к *D*. Для начала мы черным кружком помечаем узел *A* как постоянный. Затем исследуем все соседние с ним узлы, указывая около них расстояние до *A*. При обнаружении более короткого пути к какому-либо узлу вместе с указанием расстояния в метке меняется и узел, через который этот путь проходит. Таким образом, позже можно восстановить весь маршрут. Если в сети несколько кратчайших путей от *A* до *D*, то, чтобы их найти, нужно запомнить все узлы, через которые можно пройти к узлу, преодолев одинаковое расстояние.

```
#define MAX_NODES 1024                          /* максимальное количество узлов */
#define INFINITY 1000000000                     /* число, превышающее длину максимального пути */
int n, dist[MAX_NODES][MAX_NODES];              /* dist[i][j] – это расстояние от i до j */

void shortest_path(int s, int t, int path[])
{ struct state {                                /* рабочий путь */
      int predecessor;                          /* предыдущий узел */
      int length;                               /* расстояние от источника до этого узла */
      enum {permanent, tentative} label;        /* метка состояния */
 } state[MAX_NODES];

 int i, k, min;
 struct state *p;

 for (p = &state[0]; p < &state[n]; p++) {      /* инициализация состояния */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
 }
 state[t].length = 0; state[t].label = permanent;
 k = t;                                         /* k – начальный рабочий узел */
 do {                                           /* Есть ли лучший путь от k? */
      for (i = 0; i < n; i++)                   /* У этого графа n узлов */
            if (dist[k][i] != 0 && state[i].label == tentative) {
                  if (state[k].length + dist[k][i] < state[i].length) {
                        state[i].predecessor = k;
                        state[i].length = state[k].length + dist[k][i];
                  }
            }

      /* Поиск узла, предварительно помеченного наименьшей меткой */
      k = 0; min = INFINITY;
      for (i = 0; i < n; i++)
            if (state[i].label == tentative && state[i].length < min) {
                  min = state[i].length;
                  k = i;
            }
      state[k].label = permanent;
 } while (k != s);

 /* Копирование пути в выходной массив */
 i = 0; k = s;
 do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

**Илл. 5.8.** Алгоритм Дейкстры для вычисления кратчайшего пути по графу

Проверив все соседние с $A$ узлы, мы просматриваем временно помеченные узлы во всем графе и делаем узел с наименьшей меткой постоянным (илл. 5.7 (б)). Он становится новым рабочим узлом.

Теперь мы повторяем ту же процедуру с узлом *B*, исследуя все ближайшие узлы. Сначала складываем расстояние от *B* до соседнего узла и значение метки *B* (то есть расстояние от *A* до *B*). Если полученная сумма меньше, чем метка исследуемого узла (то есть расстояние от *A*, уже найденное на предыдущем этапе), значит, обнаружен более короткий путь, поэтому метка узла меняется.

После того как все узлы, примыкающие к рабочему, исследованы и временные метки изменены (если это возможно), по всему графу ищется узел с наименьшей временной меткой. Он помечается как постоянный и становится текущим рабочим узлом. На илл. 5.7 показаны первые шесть этапов работы алгоритма.

Чтобы понять работу алгоритма, посмотрим на илл. 5.7 (в). На данном этапе узел *E* только что был отмечен как постоянный. Предположим, что существует более короткий путь, нежели *ABE*, например *AXYZE* (для некоторых *X* и *Y*). Существует две возможности – либо узел *Z* уже постоянный, либо еще нет. Если да, значит, узел *E* уже проверялся, когда узел *Z* был сделан постоянным и, следовательно, рабочим узлом. В этом случае путь *AXYZE* уже исследовался.

Теперь рассмотрим ситуацию, когда узел *Z* все еще помечен как временный. Если метка узла *Z* больше или равна метке узла *E*, путь *AXYZE* не может быть короче, чем путь *ABE*. Если же метка узла *Z* меньше метки узла *E*, тогда узел *Z* стал бы постоянным раньше узла *E* и узел *E* проверялся бы с узла *Z*.

Пример реализации алгоритма представлен на илл. 5.8. Глобальные переменные `n` и `dist` описывают граф и инициализируются до вызова функции `shortest_path`. Единственное отличие программы от описанного выше алгоритма заключается в том, что вычисление кратчайшего пути в программе начинается не с узла-источника *s*, а с конечного узла *t*.

Поскольку в однонаправленном графе кратчайшие пути от *t* к *s* те же, что и от *s* к *t*, не имеет значения, с какого конца начинать. Причина поиска пути в обратном направлении заключается в том, что каждый узел помечается предшествующим узлом, а не следующим. Когда найденный маршрут копируется в выходную переменную `path`, он инвертируется. В результате двух инверсий получается путь в нужном направлении.

## 5.2.3. Лавинная адресация

При выполнении алгоритма маршрутизации любой маршрутизатор должен принимать решения на основании локальных сведений, а не полной информации о сети. Одним из простых локальных методов является **лавинная адресация (flooding)**, при которой каждый входящий пакет отправляется на все исходящие линии, кроме той, по которой он пришел.

Очевидно, что этот алгоритм порождает огромное, даже бесконечное количество дублированных пакетов, если не принять меры. Например, можно поместить в заголовок пакета счетчик транзитных участков, который уменьшается при прохождении каждого маршрутизатора. Когда значение этого счетчика падает до нуля, пакет удаляется. В идеале счетчик устанавливается согласно длине

пути от источника до адресата. Если отправитель не знает это расстояние, он может установить значение счетчика в соответствии с наихудшим случаем, то есть диаметром сети.

С использованием счетчика переходов количество копий пакета может расти экспоненциально. Число переходов увеличивается, а маршрутизаторы заново отправляют пакеты, которые они уже видели. Чтобы остановить процесс, нужно учитывать пакеты, проходящие через маршрутизатор. Это позволяет не отправлять их повторно. Один из способов достижения этой цели состоит в следующем. Маршрутизатор-источник нумерует пакеты, полученные от хостов. Все маршрутизаторы ведут списки порядковых номеров пакетов, полученных от каждого источника. Если входящий пакет уже есть в списке, он далее не передается.

Чтобы предотвратить бесконечный рост списка, можно снабдить все списки счетчиком $k$, подразумевая, что все порядковые номера вплоть до $k$ уже встречались. Когда приходит пакет, можно легко проверить, был ли он уже передан, сравнив его порядковый номер с $k$; при положительном ответе такой пакет отвергается. Кроме того, не нужно хранить весь список до $k$, поскольку счетчик показывает всю нужную информацию.

В большинстве случаев использовать алгоритм лавинной адресации для отправки пакетов непрактично, но иногда он весьма полезен. Во-первых, он гарантированно доставляет пакет в каждый узел сети. Если пакет нужно доставить в одно конкретное место, этот метод может не оправдать себя, но он эффективен при широковещательной рассылке. В беспроводных сетях сообщения, передаваемые станцией, могут быть приняты любой другой станцией, находящейся в пределах радиуса действия передатчика. Фактически это является лавинной адресацией, и некоторые алгоритмы используют это свойство.

Во-вторых, этот метод чрезвычайно надежен. Даже если большинство маршрутизаторов окажутся полностью уничтоженными (например, если речь идет о военной сети связи в зоне боевых действий), алгоритм найдет путь (если он существует), чтобы доставить пакет по назначению. Кроме того, его почти не нужно настраивать. Маршрутизаторы должны лишь знать своих соседей. Это означает, алгоритм может использоваться в составе другого алгоритма маршрутизации — более эффективного, но требующего тщательной настройки. Также лавинная адресация может служить эталоном при тестировании других алгоритмов маршрутизации, так как она всегда находит все возможные пути в сети, в том числе кратчайшие. Ухудшить эталонные показатели времени доставки могут разве что накладные расходы, вызванные огромным количеством пакетов, формируемых самим алгоритмом.

## 5.2.4. Маршрутизация по вектору расстояний

Компьютерные сети обычно используют динамические алгоритмы маршрутизации. Они сложнее, чем лавинная адресация, но в то же время эффективнее, так как находят кратчайшие пути для текущей топологии. Самые популярные динамические методы — маршрутизация по вектору расстояний и маршрутизация

с учетом состояния каналов. В этом разделе мы изучим первый, в следующем — второй метод.

Алгоритмы **маршрутизации по вектору расстояний (distance vector routing)** работают, опираясь на таблицы (то есть векторы), поддерживаемые всеми маршрутизаторами. Эти таблицы содержат сведения об известных кратчайших путях к каждому из возможных адресатов и о том, какое соединение следует использовать. Для обновления содержимого таблиц производится обмен информацией с соседними маршрутизаторами. В результате любой маршрутизатор знает кратчайший путь до всех получателей.

Метод маршрутизации по вектору расстояний иногда называют в честь его авторов распределенным алгоритмом **Беллмана — Форда (Bellman — Ford)**; см. работы Беллмана (Bellman, 1957), а также Форда и Фалкерсона (Ford and Fulkerson, 1962). Изначально он применялся в сети ARPANET, а в интернете был известен под названием RIP.

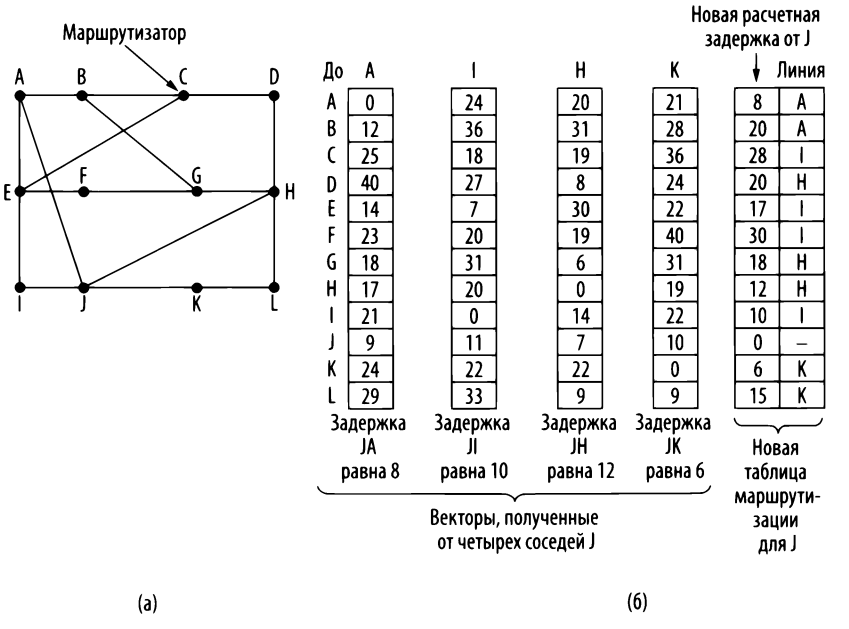
Все маршрутизаторы используют и обновляют таблицу с записями обо всех маршрутизаторах сети. Каждая запись состоит из двух частей: оптимальная исходящая линия для данного получателя и предполагаемое расстояние или время передачи пакета. В качестве меры расстояния используется число переходов или другие параметры, которые применяются при вычислении кратчайшего пути.

Предполагается, что маршрутизаторы знают расстояние до каждого соседа. Если единицей измерения являются переходы, то оно равно одному переходу. Если же расстояние измеряется временем задержки распространения, то маршрутизатор может измерить его с помощью специального пакета ECHO (эхо). Адресат добавляет в него время получения и отправляет его обратно как можно скорее.

Допустим, в качестве единицы измерения используется время задержки, и маршрутизатор знает значение этого параметра относительно каждого соседа. Через каждые $T$ мс все маршрутизаторы посылают своим соседям список с приблизительными задержками для каждого получателя. Они, разумеется, также получают подобный список от всех своих соседей. Допустим, одна из таких таблиц пришла от соседа $X$ и в ней указывается, что время распространения от $X$ до $i$ равно $X_i$. Если маршрутизатор знает, что время передачи до $X$ равно $m$, тогда задержка при передаче пакета маршрутизатору $i$ через $X$ составит $X_i + m$. Выполнив такие расчеты для всех своих соседей, маршрутизатор может выбрать наилучшие пути и записать это в новую таблицу. Обратите внимание, что старая таблица маршрутизации в расчетах не используется.

Процесс обновления таблицы проиллюстрирован на илл. 5.9. На илл. 5.9 (а) показана сеть. Первые четыре столбца на илл. 5.9 (б) показывают векторы задержек, полученные маршрутизатором $J$ от своих соседей. Маршрутизатор $A$ считает, что время передачи от него до $B$ равно 12 мс, 25 мс до $C$, 40 мс до $D$ и т. д. Допустим, $J$ измерил или оценил задержки до своих соседей $A$, $I$, $H$ и $K$ как 8, 10, 12 и 6 мс соответственно.

Теперь рассмотрим, как $J$ рассчитывает свой новый маршрут к маршрутизатору $G$. Он знает, что задержка до $A$ составляет 8 мс, и при этом $A$ думает, что от него до $G$ данные дойдут за 18 мс. Таким образом, $J$ знает, что если он станет




**Илл. 5.9.** (а) Сеть. (б) Полученные от A, I, H и K векторы и новая таблица маршрутизации для J

отправлять пакеты для *G* через *A*, то задержка составит 26 мс. Аналогично он вычисляет значения задержек для маршрутов от него до *G*, которые проходят через остальных его соседей (*I*, *H* и *K*), и получает, соответственно, 41 (31+10), 18 (6+12) и 37 (31+6). Лучшее значение — 18, поэтому *J* записывает в таблицу, что задержка до *G* составляет 18 мс, а оптимальный путь проходит через *H*. Данный расчет повторяется для всех остальных адресатов, и получается новая таблица (см. правый столбец на илл. 5.9).

## *Проблема счета до бесконечности*

Установление маршрутов согласно кратчайшим путям в сети называется **конвергенцией (convergence)**. Алгоритм маршрутизации по вектору расстояний — простой метод, позволяющий маршрутизаторам совместно вычислять оптимальные пути. Однако на практике он обладает серьезным недостатком: хоть он и находит правильный ответ, поиск длится очень долго. В частности, этот алгоритм быстро реагирует на хорошие новости и очень медленно — на плохие. Рассмотрим маршрутизатор, для которого наилучшее расстояние до *X* достаточно велико. Допустим, при очередном обмене векторами его сосед *A* сообщает ему, что от него до *X* совсем недалеко. Тогда наш маршрутизатор просто переключается на линию, проходящую через *A*, чтобы отправлять пакеты *X*. Таким образом, хорошая новость распространилась всего за один обмен информацией.

Чтобы увидеть, как быстро распространяются хорошие известия, рассмотрим линейную сеть из пяти узлов, показанную на илл. 5.10. Расстояние в ней

измеряется числом переходов. Предположим, что изначально маршрутизатор *A* выключен и все остальные маршрутизаторы об этом знают. То есть они считают, что расстояние до *A* равно бесконечности.

**Илл. 5.10.** Проблема счета до бесконечности

Когда *A* появляется в сети, остальные маршрутизаторы узнают об этом с помощью обмена векторами. Для простоты представим, что где-то в сети имеется гигантский гонг, в который периодически ударяют, чтобы инициировать одновременный обмен. После первого обмена *B* узнает, что у его соседа слева нулевая задержка при связи с *A*, а *B* помечает в своей таблице маршрутов, что *A* находится слева на расстоянии одного перехода. Все остальные маршрутизаторы в этот момент еще полагают, что *A* выключен. Значения задержек для *A* в таблицах на этот момент показаны во второй строке на илл. 5.10 (а). При следующем обмене информацией *C* узнает, что у *B* есть путь к *A* длиной 1, поэтому он обновляет свою таблицу, указывая длину пути до *A*, равную 2, но *D* и *E* об этом еще не знают. Таким образом, хорошие новости распространяются со скоростью один переход за один обмен векторами. Если самый длинный путь в сети состоит из *N* переходов, то через *N* обменов все маршрутизаторы подсети будут знать о включенных маршрутизаторах и заработавших линиях.

Теперь рассмотрим ситуацию на илл. 5.10 (б). Все линии и маршрутизаторы изначально находятся в рабочем состоянии. Маршрутизаторы *B*, *C*, *D* и *E* расположены на расстоянии 1, 2, 3 и 4 переходов от *A* соответственно. Внезапно либо *A* отключается, либо происходит обрыв линии между *A* и *B* (что с точки зрения *B* одно и то же).

При первом обмене пакетами *B* не слышит ответа от *A*. К счастью, *C* говорит: «Не волнуйся. У меня есть путь к *A* длиной 2». *B* вряд ли догадывается, что путь от *C* к *A* проходит через *B*. *B* может только предполагать, что у *C* около 10 выходных линий с независимыми путями к *A*, кратчайшая из которых имеет длину 2. Поэтому теперь *B* думает, что может связаться с *A* через *C* по пути длиной 3. При этом обмене маршрутизаторы *D* и *E* не обновляют свою информацию об *A*.

При втором обмене векторами *C* замечает, что у всех его соседей есть путь к *A* длиной 3. Он выбирает один из них случайным образом и устанавливает свое расстояние до *A* равным 4, как показано в третьей строке на илл. 5.10 (б). Результаты последующих обменов векторами также показаны на этом рисунке.

Теперь понятно, почему плохие новости медленно распространяются: ни один маршрутизатор не может установить расстояние, более чем на 1 превышающее минимальное значение, о котором сообщают его соседи. В итоге все маршрутизаторы будут бесконечно увеличивать значение расстояния до выключенного маршрутизатора. Число обменов, необходимых для завершения этого процесса, можно ограничить, если задать значение этой «бесконечности» равным длине самого длинного пути плюс 1.

Неудивительно, что эту проблему называют **счетом до бесконечности (count-to-infinity)**. Предпринималось много попыток решить ее. Например, было предложено запретить маршрутизатору сообщать о своих кратчайших путях тем соседям, от которых они получили эту информацию. Метод расщепления горизонта с «отравляющим» ответом обсуждался в RFC 1058. Однако на практике все эти эвристические правила с красивыми названиями оказались абсолютно бесполезными. Суть проблемы в том, что, когда *X* сообщает *Y* о том, что у него есть какой-то путь, *Y* никак не может узнать, является ли он сам частью этого пути.

## 5.2.5. Маршрутизация с учетом состояния линий

Маршрутизация на основе векторов расстояний использовалась в сети ARPANET вплоть до 1979 года, когда ее сменил алгоритм маршрутизации с учетом состояния линий. Отказаться от прежнего метода пришлось в первую очередь потому, что при изменении топологии сети он слишком долго стабилизировался (из-за проблемы счета до бесконечности). В результате ему на смену пришел совершенно новый алгоритм — **маршрутизация с учетом состояния линий (link state routing)**. Сегодня в крупных сетях и в интернете используются его варианты — алгоритмы маршрутизации IS-IS и OSPF.

В основе этого метода лежит относительно простая идея, которую можно изложить в пяти требованиях к маршрутизатору. Он должен сделать следующее:

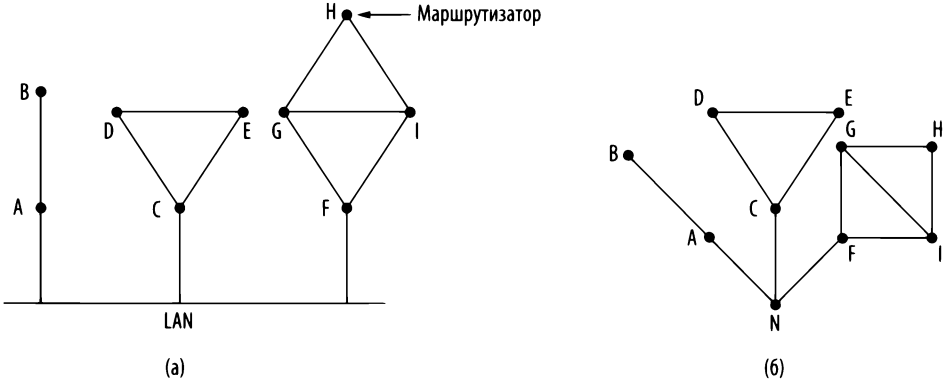
1. Обнаружить своих соседей и узнать их сетевые адреса.
2. Задать параметр расстояния или стоимости связи с каждым соседом.
3. Создать пакет, содержащий всю собранную информацию.
4. Разослать его на все маршрутизаторы и принять отправленные ими пакеты.
5. Вычислить кратчайший путь ко всем маршрутизаторам.

В результате каждый маршрутизатор получает полную топологию. После этого он может найти кратчайший путь ко всем остальным, применив алгоритм Дейкстры. Далее мы подробно изучим эти шаги.

### Знакомство с соседями

Когда маршрутизатор загружается, его первая задача — получить информацию о соседях. Для этого он посылает специальный пакет HELLO по всем линиям «точка-точка». Маршрутизатор на другом конце линии в ответ сообщает свое имя. Имена должны быть уникальными. Если маршрутизатор слышит, что три других маршрутизатора напрямую соединены с *F*, то должно быть абсолютно ясно — все они имеют в виду один и тот же *F*.

Если два или более маршрутизатора соединены с помощью широковещательной связи (например, коммутатора, кольцевой сети или классического Ethernet), ситуация несколько усложняется. На илл. 5.11 (а) изображена широковещательная LAN, к которой напрямую подключены три маршрутизатора: *A*, *C* и *F*. Каждый из них соединен с одним или несколькими дополнительными маршрутизаторами.




**Илл. 5.11.** Широковещательная LAN. (а) Девять маршрутизаторов и широковещательная LAN. (б) Графовая модель той же системы

Широковещательная LAN обеспечивает связь между каждой парой подключенных маршрутизаторов. Однако моделирование такой сети в виде системы связей «точка-точка» увеличивает топологию и ведет к неэкономной передаче. Существует более подходящая модель, в которой LAN представляет собой узел графа. На илл. 5.11 (б) она изображена в виде нового, искусственного узла *N*, с которым соединены *A*, *C* и *F*. Один из маршрутизаторов сети — **отмеченный маршрутизатор (designated router)** — выполняет роль *N* в протоколе маршрутизации. Маршрут от *A* до *C* по LAN в этом случае выглядит как путь *ANC*.

### Определение стоимости связи

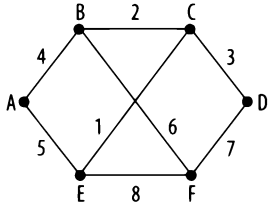
Алгоритм маршрутизации с учетом состояния линии требует, чтобы у каждого соединения был параметр расстояния или стоимости, необходимый для вычисления кратчайшего пути. Стоимость пути до соседних маршрутизаторов может быть задана автоматически или определена оператором сети. Чаще всего она

обратно пропорциональна пропускной способности. Так, сеть Ethernet со скоростью 1 Гбит/с может иметь стоимость 1, а Ethernet со скоростью 100 Мбит/с — 10. Благодаря этому в качестве наилучшего выбирается путь с более высокой пропускной способностью.

Если узлы сети находятся далеко друг от друга, определить стоимость пути можно на основе задержки соединения. В этом случае наилучшим считается более короткий путь. Самый простой способ определения задержки — отправка специального пакета ECHO. Другая сторона обязана немедленно на него ответить. Измерив время прохождения пакета в оба конца и разделив его на два, отправитель получает приблизительное значение задержки.

### Создание пакетов состояния линий

Следующий шаг после сбора информации, необходимой для обмена, — создание пакета, который содержит все эти данные. Пакет начинается с идентификатора отправителя, за ним следует порядковый номер и возраст (об этом ниже), а также список соседей. Также сообщается стоимость связи с каждым из них. На илл. 5.12 (а) представлен пример сети с указанием стоимости для всех соединений. Соответствующие пакеты состояния линий для шести маршрутизаторов показаны на илл. 5.12 (б).




(а)

Пакеты состояния линий

| A | |
|---|---|
| Поряд. номер | |
| Возраст | |
| B | 4 |
| E | 5 |

| B | |
|---|---|
| Поряд. номер | |
| Возраст | |
| A | 4 |
| C | 2 |
| F | 6 |

| C | |
|---|---|
| Поряд. номер | |
| Возраст | |
| B | 2 |
| D | 3 |
| E | 1 |

| D | |
|---|---|
| Поряд. номер | |
| Возраст | |
| C | 3 |
| F | 7 |

| E | |
|---|---|
| Поряд. номер | |
| Возраст | |
| A | 5 |
| C | 1 |
| F | 8 |

| F | |
|---|---|
| Поряд. номер | |
| Возраст | |
| B | 6 |
| D | 7 |
| E | 8 |

(б)

**Илл. 5.12.** (а) Сеть. (б) Пакеты состояния линий для этой сети

Создать пакет состояния линий несложно; труднее всего — правильно выбрать момент. Пакеты можно создавать периодически через равные временные интервалы либо когда происходит какое-то конкретное событие. Например, если линия или соседний маршрутизатор вышли из строя либо, наоборот, снова появились в сети или если они изменили свои свойства.

### Распределение пакетов состояния линий

Самая сложная часть алгоритма состоит в распространении пакетов состояния линий. Все маршрутизаторы должны принимать такие пакеты быстро и безотказно. Если маршрутизаторы используют разные версии топологии, это может привести к проблемам — зацикленным путям, недоступным устройствам и т. д.

Мы начнем с базового алгоритма распределения, а затем расскажем о некоторых его модификациях. В его основе лежит алгоритм лавинной адресации, с помощью которого происходит отправка пакетов состояния линии на все маршрутизаторы. Чтобы держать этот процесс под контролем, в каждый пакет добавляют порядковый номер, который увеличивается на единицу для каждого следующего пакета. Маршрутизаторы записывают все пары «источник + порядковый номер», которые им встречаются. Когда приходит новый пакет состояния линий, маршрутизатор ищет адрес отправителя и порядковый номер пакета в своем списке. Если это новый пакет, он рассылается дальше по всем линиям, кроме той, по которой он пришел. Дубликаты удаляются. Если номер нового пакета меньше, чем номер уже полученного от того же отправителя, он также удаляется как устаревший, поскольку очевидно, что у маршрутизатора есть более свежие данные.

С этим алгоритмом связано несколько проблем, но они решаемы. Во-первых, если порядковый номер обнулится, достигнув максимального значения, возникнет путаница. Чтобы этого избежать, используются 32-битные номера. Даже если рассылать пакеты каждую секунду, для обнуления понадобится 137 лет, поэтому о нем можно не беспокоиться.

Во-вторых, если маршрутизатор выйдет из строя, будет потерян его порядковый номер. Если он снова запустится с нулевым номером, следующий отправленный им пакет будет проигнорирован как устаревший.

В-третьих, может произойти искажение порядкового номера — например, вместо номера 4 будет принято число 65,540 (ошибка в одном бите); в этом случае пакеты с 5-го по 65,540-й будут игнорироваться маршрутизаторами как устаревшие.

Решение указанных проблем заключается в указании возраста пакета после его порядкового номера. Каждую секунду этот параметр уменьшается на единицу. Когда возраст снижается до нуля, информация от соответствующего маршрутизатора удаляется. В нормальной ситуации новый пакет приходит, скажем, каждые 10 с; таким образом, сведения о маршрутизаторе устаревают, только когда он выключается (или в случае потери шести пакетов подряд, что маловероятно). Кроме того, каждый маршрутизатор уменьшает поле *Возраст (Age)* на единицу во время первоначальной лавинной адресации, чтобы гарантировать, что ни один пакет не потеряется и не будет существовать вечно (пакет с нулевым возрастом удаляется).

Для повышения надежности данный алгоритм был слегка доработан. Когда пакет состояния линий приходит на маршрутизатор для рассылки, он не сразу ставится в очередь на отправку. Вместо этого он ненадолго помещается в область промежуточного хранения на случай появления новых связей или разрыва старых. Если за это время от того же отправителя успевает прийти еще один пакет, маршрутизатор сравнивает их порядковые номера и удаляет устаревший. Если номера одинаковые, то удаляется дубликат. Для защиты от ошибок получение всех пакетов состояния линий подтверждается.

Структура данных, используемая маршрутизатором *B* для работы с сетью на илл. 5.12 (а), показана на илл. 5.13. Каждая строка соответствует недавно полученному, но еще не полностью обработанному пакету состояния линий.

В таблицу записывается адрес отправителя, порядковый номер, возраст и данные. Кроме того, в ней содержатся флаги отправки и подтверждения для каждой из трех линий маршрутизатора *B* (к *A*, *C* и *F*). Флаги отправки означают, что пакет нужно отослать по указанной линии, а флаги подтверждения — что нужно сообщить о его получении.

Как видно из илл. 5.13, пакет состояния линий от маршрутизатора *A* пришел напрямую, поэтому он должен быть отправлен *C* и *F*, а подтверждение о его получении следует направить *A*, что и показывают флаговые биты. Аналогично пакет от *F* следует переслать маршрутизаторам *A* и *C*, а *F* отослать подтверждение.

| | | | Флаги отправки | | | Флаги подтверждения | | | |
|---|---|---|---|---|---|---|---|---|---|
| Источник | Порядковый номер | Возраст | A | C | F | A | C | F | Данные |
| A | 21 | 60 | 0 | 1 | 1 | 1 | 0 | 0 | |
| F | 21 | 60 | 1 | 1 | 0 | 0 | 0 | 1 | |
| E | 21 | 59 | 0 | 1 | 0 | 1 | 0 | 1 | |
| C | 20 | 60 | 1 | 0 | 1 | 0 | 1 | 0 | |
| D | 21 | 59 | 1 | 0 | 0 | 0 | 1 | 1 | |

**Илл. 5.13.** Буфер пакетов маршрутизатора B, показанного на илл. 5.12 (а)

Однако ситуация с третьим пакетом, полученным от маршрутизатора *E*, отличается. Он приходит дважды, по линиям *EAB* и *EFB*. Следовательно, его нужно отослать только *C*, но подтверждения необходимо выслать и *A*, и *F*, как указывают биты.

Если дубликат пакета приходит в тот момент, когда оригинал еще находится в буфере, значение битов должно измениться. Например, если копия пакета состояния *C* прибудет от *F*, прежде чем четвертая строка таблицы будет разослана, шесть флаговых битов примут значение 100011, и это будет означать, что нужно подтвердить получение пакета от *F*, но не пересылать его *F*.

### Вычисление новых маршрутов

Собрав весь комплект пакетов состояния линий, маршрутизатор может построить полный граф сети, так как у него есть данные обо всех линиях. На самом деле каждая линия представлена даже дважды, по одному значению для каждого направления. Иногда эти значения различаются, поэтому результаты вычисления кратчайшего пути от *A* до *B* и от *B* до *A* могут не совпадать.

Теперь для построения кратчайших путей ко всем возможным получателям локально применяется алгоритм Дейкстры. Так маршрутизатор узнает, какое соединение выбрать, чтобы отправить данные тому или иному адресату. Данные добавляются в таблицы маршрутизации, и возобновляется штатный режим работы.

В отличие от маршрутизации по вектору расстояния, маршрутизация с учетом состояния линий требует большего количества вычислений и памяти. В сети из $n$ маршрутизаторов, у каждого из которых $k$ соседей, количество памяти, необходимой для хранения входных данных, пропорционально $kn$. Это как минимум соответствует размеру таблицы маршрутизации, содержащей все адреса назначения. Кроме того, даже при использовании самых эффективных структур данных время вычисления растет быстрее, чем $kn$. В больших сетях это может стать проблемой. Тем не менее во многих практических ситуациях маршрутизация с учетом состояния линий вполне эффективна, поскольку ее не затрагивает проблема медленной конвергенции.

Этот вид маршрутизации широко применяется в современных сетях, поэтому стоит кратко коснуться темы протоколов. Многие интернет-провайдеры используют протокол маршрутизации с учетом состояния линий **IS-IS (Intermediate System-to-Intermediate System — связь между промежуточными системами);** см. работу Орана (Oran, 1990). Он был разработан для ранних сетей DECnet и впоследствии был принят ISO, чтобы использоваться вместе с протоколами OSI. После этого он был модифицирован для работы с другими протоколами, прежде всего IP. В разделе 5.7.6 мы обсудим еще один распространенный протокол маршрутизации с учетом состояния линий — **открытый протокол предпочтения кратчайшего пути (Open Shortest Path First, OSPF)**. Он был разработан Специальной комиссией интернет-разработок (IETF) через несколько лет после IS-IS и включал многие нововведения, разработанные для IS-IS. Сюда входит саморегулирующийся метод лавинной адресации для обновления информации о состоянии линий, концепция выделенного маршрутизатора в LAN, а также метод вычисления и поддержки расщепления пути и множества метрик. Соответственно, между протоколами IS-IS и OSPF нет почти никакой разницы. Самое большое различие состоит в том, что в IS-IS возможна одновременная поддержка нескольких протоколов сетевого уровня (например, IP, IPX и AppleTalk). OSPF не обладает таким свойством, что является преимуществом в больших многопротокольных средах.

Следует также сказать несколько общих слов об алгоритмах маршрутизации. Маршрутизация с учетом состояния линий или по вектору расстояния, а также другие алгоритмы предполагают, что маршрутизаторы вычисляют путь. Однако неисправности оборудования или программного обеспечения могут привести к очень серьезным проблемам во всей сети. Например, если маршрутизатор заявит о существовании линии, которой у него нет, или, наоборот, забудет об одной из своих линий, граф сети окажется неверным. Если маршрутизатор не сможет переслать пакеты или повредит их при передаче, маршрут не будет работать корректно. Наконец, различные неприятности могут возникнуть, если у маршрутизатора закончится свободная память или он неправильно рассчитает путь. Когда сеть достигает размера в несколько десятков или сотен тысяч маршрутизаторов, вероятность ошибки одного из них становится значительной. Единственное, что можно сделать, — попытаться свети ущерб к минимуму, когда случится неизбежное. Эти проблемы и методы их решения подробно обсуждаются в работе Перлман (Perlman, 1988).

## 5.2.6. Иерархическая маршрутизация внутри сети

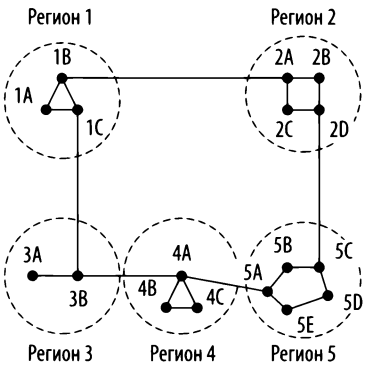
По мере роста сети постоянно увеличивается размер таблиц маршрутизации. Они занимают все больше памяти маршрутизатора, кроме того, возрастает время CPU, необходимое для их обработки. При этом растет число служебных пакетов, которыми обмениваются маршрутизаторы, что увеличивает нагрузку на линию. Даже если бы каждый маршрутизатор мог хранить всю топологию сети, повторное вычисление кратчайших путей при каждом ее изменении привело бы к чрезмерной трате ресурсов. Представьте, что будет, если мы станем заново рассчитывать оптимальные пути при каждом отказе или восстановлении соединения в очень крупной сети. Когда-нибудь сеть вырастет до таких размеров, при которых будет невозможно хранить на каждом маршрутизаторе всю необходимую информацию. Поэтому в больших сетях маршрутизация осуществляется иерархически, с применением **областей маршрутизации (routing areas)**.

При иерархической маршрутизации вся совокупность маршрутизаторов разбивается на отдельные **регионы (regions)**, или **области (areas)**. Каждый маршрутизатор знает все детали выбора пути в пределах своей области, но ему ничего не известно об устройстве других регионов. При объединении нескольких сетей логично рассматривать их как отдельные регионы, при этом маршрутизаторы одной сети не обязаны знать топологию других сетей.

Для очень больших сетей двухуровневой иерархии может оказаться недостаточно. Тогда регионы объединяются в кластеры, кластеры в зоны, зоны в группы и т. д., пока не иссякнет фантазия на названия для новых образований. В качестве примера простой многоуровневой иерархии рассмотрим маршрутизацию пакета, пересылаемого из Университета Беркли, штат Калифорния, в Малинди (Кения). Маршрутизатор в Беркли знает детали топологии в пределах Калифорнии, но трафик, направляющийся за пределы штата, он передает в Лос-Анджелес. Маршрутизатор в Лос-Анджелесе работает в пределах США, но все пакеты, направляемые за рубеж, переправляет в Нью-Йорк. Нью-йоркский маршрутизатор отправит пакет на маршрутизатор страны назначения, ответственный за прием трафика из-за границы. Он может располагаться, например, в Найроби. Наконец, направляясь вниз по дереву иерархии уже в пределах Кении, пакет попадет в Малинди.

На илл. 5.14 приведен пример количественной оценки маршрутизации в двухуровневой иерархии с пятью регионами. Полная таблица маршрутизатора *1A* состоит из 17 записей (см. илл. 5.14 (б)). При иерархической маршрутизации, показанной на илл. 5.14 (в), таблица, как и прежде, содержит сведения обо всех локальных маршрутизаторах, но записи об остальных регионах собраны в одном маршрутизаторе. Поэтому трафик в регион 2 идет по линии *1B–2A*, а во все остальные — по *1C–3B*. Благодаря этому размер таблицы сокращается с 17 до 7 строк. По мере роста соотношения между числом регионов и количеством маршрутизаторов на регион память расходуется все более экономно.

К сожалению, вместе с этим увеличивается длина пути. Например, наилучший маршрут от *1A* до *5C* проходит через регион 2, однако при иерархической маршрутизации весь трафик в регион 5 направляется через регион 3, поскольку так лучше для большинства адресатов в регионе 5.




(а)

Полная таблица для 1A

| Назначение | Линия | Переходы |
|---|---|---|
| 1A | – | – |
| 1B | 1B | 1 |
| 1C | 1C | 1 |
| 2A | 1B | 2 |
| 2B | 1B | 3 |
| 2C | 1B | 3 |
| 2D | 1B | 4 |
| 3A | 1C | 3 |
| 3B | 1C | 2 |
| 4A | 1C | 3 |
| 4B | 1C | 4 |
| 4C | 1C | 4 |
| 5A | 1C | 4 |
| 5B | 1C | 5 |
| 5C | 1B | 5 |
| 5D | 1C | 6 |
| 5E | 1C | 5 |

(б)

Иерархическая таблица для 1A

| Назначение | Линия | Переходы |
|---|---|---|
| 1A | – | – |
| 1B | 1B | 1 |
| 1C | 1C | 1 |
| 2 | 1B | 2 |
| 3 | 1C | 2 |
| 4 | 1C | 3 |
| 5 | 1C | 4 |

(в)

**Илл. 5.14.** Иерархическая маршрутизация

Когда единая сеть становится очень большой, возникает интересный вопрос: сколько уровней должна иметь иерархия? Для примера рассмотрим сеть с 720 маршрутизаторами. В системе без иерархии каждый из них должен поддерживать таблицу из 720 строк. Если сеть разбить на 24 региона по 30 маршрутизаторов, каждому из них потребуется 30 локальных записей плюс 23 записи об удаленных регионах, итого 53. При трехуровневой иерархии, состоящей из 8 кластеров по 9 регионов с 10 маршрутизаторами, каждому маршрутизатору понадобится 10 локальных записей, 8 — для других регионов в пределах своего кластера, плюс 7 для остальных кластеров, итого 25. Камоун и Клейнрок (Kamoun and Kleinrock) в 1979 году обнаружили, что оптимальное количество уровней иерархии для сети, состоящей из $N$ маршрутизаторов, равно $\ln N$. При этом потребуется $e \ln N$ записей для каждого маршрутизатора. Также они выяснили, что при иерархической маршрутизации эффективная средняя длина пути увеличивается незначительно и обычно остается в допустимых пределах.

## 5.2.7. Широковещательная маршрутизация

В некоторых сценариях применения хосты должны отправлять сообщения на множество других хостов или даже на все сразу. Можно привести такие примеры, как рассылка прогноза погоды или новостей фондового рынка, а также радиопередача в прямом эфире. Лучше всего передавать такие данные на все устройства, позволяя всем заинтересованным пользователям ознакомиться

с ними. Одновременная рассылка пакетов по всем адресам называется **широковещанием (broadcasting)**. Существует несколько методов ее реализации.

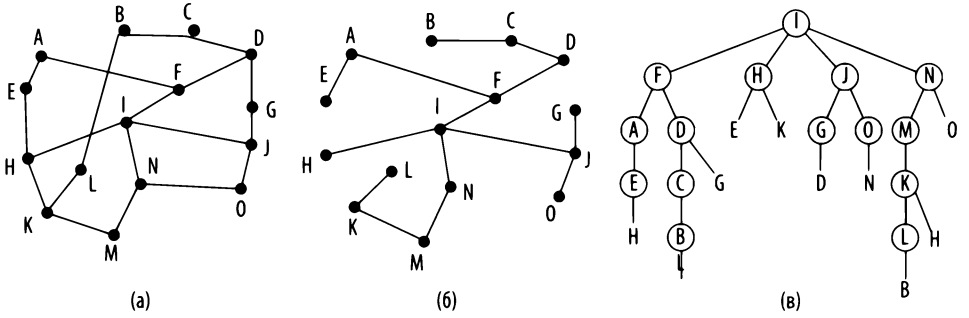
Один из способов широковещания не требует никаких специальных возможностей сети и заключается в простой рассылке отдельных пакетов по всем направлениям. Это медленно и неэффективно с точки зрения пропускной способности, к тому же предполагает, что у источника пакета есть полный список всех хостов. На практике такой метод нежелателен, хотя применяется довольно часто.

Более совершенный алгоритм называется **многоадресной маршрутизацией (multidestination routing)**. В каждом пакете содержится либо список адресатов, либо битовая карта с указанием нужных получателей. Когда приходит такой пакет, маршрутизатор проверяет список и определяет набор необходимых исходящих линий. (Линия выбирается, если это оптимальный путь хотя бы к одному адресату из списка.) Маршрутизатор создает копию пакета для каждой выбранной линии. В ней указаны только те адресаты, к которым ведет данный путь. Таким образом, весь список рассылки распределяется между исходящими линиями. После определенного числа передач каждый пакет будет содержать только один адрес назначения, как и обычный пакет. Многоадресная маршрутизация похожа на рассылку индивидуально адресованных пакетов. Разница в том, что в первом случае из нескольких пакетов, следующих по одному маршруту, только один «платит полную стоимость», а остальные «едут бесплатно». Таким образом, пропускная способность используется более эффективно. Однако этот метод все же требует начальных сведений обо всех получателях, а чтобы понять, куда отправить один многоадресный пакет, маршрутизатор должен выполнить столько же действий, сколько и при отправке набора отдельных пакетов.

Мы уже знакомы с еще одним улучшенным методом широковещательной маршрутизации: лавинной адресацией. Если этот алгоритм реализуется с помощью порядковых номеров, то каналы связи используются эффективно, поскольку в маршрутизаторах действует относительно простое правило принятия решения. Хотя этот метод не подходит для обычных двухточечных соединений, он заслуживает рассмотрения при широковещании. Впрочем, можно добиться большего, если заранее вычислить кратчайшие пути для стандартных пакетов.

Удивительно простая и изящная концепция **пересылки в обратном направлении (reverse path forwarding)** была впервые предложена в работе Далала и Меткалфа (Dalal and Metcalfe, 1978). Когда приходит широковещательный пакет, маршрутизатор проверяет, используется ли канал, по которому он прибыл, для обычной передачи данных *источнику широковещания*. Если это так, то, скорее всего, пакет пришел по наилучшему пути и является первой копией, полученной маршрутизатором. Тогда он рассылает этот пакет по всем линиям (кроме той, по которой он пришел). Но если пакет приходит от того же источника по другому каналу, он отвергается как вероятный дубликат.

Пример работы этого алгоритма представлен на илл. 5.15. Слева (а) изображена сеть, в центре (б) — входное дерево для маршрутизатора *I* этой сети, а справа (в) показано, как работает алгоритм пересылки в обратном направлении. На первом транзитном участке маршрутизатор *I* отсылает пакеты маршрутизаторам *F*, *H*, *J* и *N*, являющимся вторым ярусом дерева. Все пакеты приходят к ним




**Илл. 5.15.** Продвижение по встречному пути. (а) Сеть. (б) Входное дерево для маршрутизатора I. (в) Дерево, построенное методом пересылки в обратном направлении от маршрутизатора I

по приоритетной линии до *I* (по маршруту, совпадающему с входным деревом); на рисунке это обозначено буквами в кружках. Затем формируются 8 пакетов — по 2 каждым маршрутизатором, получившим пакет на первом этапе. Все они попадают к маршрутизаторам, еще не получавшим пакеты; 5 из них приходят по приоритетным линиям. Из 6 пакетов, формируемых на третьем транзитном участке, только 3 приходят по приоритетным линиям (на *C*, *E* и *K*). Остальные оказываются дубликатами. После 5 переходов широковещание заканчивается; общее число переданных пакетов — 23. При использовании входного дерева потребовалось бы 4 перехода и 14 пакетов.

Принципиальное преимущество этого метода в том, что он эффективен и одновременно прост в реализации. Как и в случае лавинной адресации, широковещательный пакет отправляется по всем каналам только один раз в каждом направлении. При этом маршрутизатор всего лишь должен знать, как достичь конкретного адресата. Он не обязан помнить порядковые номера (либо использовать другие механизмы остановки лавинной адресации) или включать в пакет список всех получателей.

Последний алгоритм широковещания, который мы рассмотрим, является развитием предыдущего метода. Он в явном виде использует входное дерево (или любое другое связующее дерево) для маршрутизатора, который начал широковещание. **Связующее дерево (spanning tree)** представляет собой подмножество сети, в котором находятся все маршрутизаторы и нет замкнутых путей. Если маршрутизатор знает, какие из его линий принадлежат связующему дереву, он может отправить пакет по всем его ветвям (кроме той, по которой пришли данные). Алгоритм обеспечивает отличную пропускную способность, так как для его выполнения требуется генерация абсолютного минимума пакетов. Например, если в качестве связующего дерева использовать входное дерево на илл. 5.15 (б), для рассылки потребуется минимальное число пакетов — 14. Единственный минус — у каждого маршрутизатора должна быть информация о связующем дереве. Иногда эти данные доступны (например, при маршрутизации с учетом состояния линий все маршрутизаторы обладают полными сведениями о топологии сети и могут вычислить связующее дерево),

но в некоторых случаях — недоступны (к примеру, при маршрутизации по вектору расстояний).

## 5.2.8. Многоадресная рассылка

Иногда пакеты отправляются группе получателей, например, в многопользовательских играх или при спортивных трансляциях на несколько точек просмотра. Если группа не слишком мала, передача отдельного пакета на каждый адрес — дорогостоящая операция. С другой стороны, в миллионной сети широковещание на группу из 1000 устройств также нерентабельно, особенно если большинство получателей не заинтересованы в данной информации (или, что еще хуже, заинтересованы, но не должны иметь к ней доступа, как в случае платных спортивных каналов). Таким образом, требуется способ рассылки сообщений строго определенным группам — довольно крупным, но небольшим по сравнению со всей сетью.

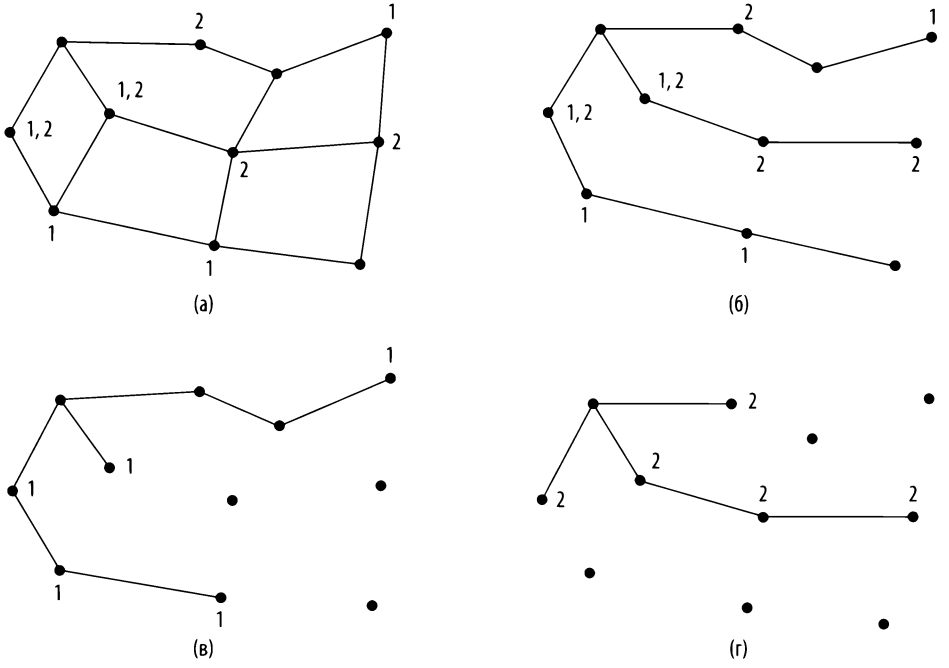
Передача сообщения членам такой группы называется **многоадресной рассылкой (multicasting)**, а используемый при этом алгоритм — **многоадресной маршрутизацией (multicast routing)**. Любая схема многоадресной рассылки предполагает возможность создания и удаления групп и определения списка маршрутизаторов, входящих в группу. Для алгоритма не имеет значения, как именно реализуются эти задачи. Пока что мы будем считать, что группа определяется по адресу рассылки, а каждый маршрутизатор знает, в какие группы он входит. Мы еще вернемся к вопросу принадлежности к группам, когда будем говорить о многоадресной интернет-рассылке в разделе 5.7.8.

Схемы многоадресной рассылки основаны на принципах широковещательной маршрутизации, о которой мы уже говорили: пакеты, предназначенные членам группы, пересылаются по связующему дереву, и при этом стоит задача эффективного использования пропускной способности сети. Однако выбор наилучшего связующего дерева зависит от того, является ли группа плотной (когда получатели занимают большую часть всей сети) или разреженной (когда большая часть сети не принадлежит группе). Мы рассмотрим оба случая.

Для плотных групп широковещание подходит — пакет будет успешно отправлен по всей сети. Минус этого алгоритма в том, что пакет получат маршрутизаторы вне группы. Диринг и Черитон (Deering and Cheriton, 1990) предложили удалять из связующего дерева ветви, не ведущие к членам группы. В результате получается эффективное многоадресное связующее дерево.

Для примера рассмотрим две группы, 1 и 2, в сети, изображенной на илл. 5.16 (а). Разные маршрутизаторы подключены к хостам, которые входят в одну или обе группы или не входят ни в одну из них. Связующее дерево для самого левого маршрутизатора показано на илл. 5.16 (б). Это дерево можно использовать для широковещания, однако (как это видно на примере двух усеченных вариантов дерева, приведенных далее) для многоадресной рассылки оно избыточно. На илл. 5.16 (в) удалены все линии, не ведущие к хостам, входящим в группу 1. В результате получается многоадресное связующее дерево, по которому самый левый маршрутизатор может отправлять пакеты группе 1. Пакеты передаются исключительно по нему — этот способ гораздо экономичнее,

чем широковещание, так как новое дерево использует 7 соединений вместо 10. На илл. 5.16 (г) показано усеченное многоадресное связующее дерево для группы 2. Оно использует 5 соединений, что также делает его эффективным. Следует обратить внимание на то, что для разных групп используются разные связующие деревья.




**Илл. 5.16.** Многоадресная рассылка. (а) Сеть. (б) Связующее дерево для самого левого маршрутизатора. (в) Многоадресное дерево для группы 1. (г) Многоадресное дерево для группы 2

Существует несколько способов усечения связующего дерева. Самый простой из них может применяться при маршрутизации с учетом состояния линий, когда каждому маршрутизатору известна полная топология сети, в том числе и состав групп. В этом случае каждый маршрутизатор может построить собственное усеченное связующее дерево для каждого отправителя. Сначала создается обычное входное дерево, а затем из него удаляются все связи, не соединяющие входной (корневой) узел с членами данной группы. Одним из протоколов маршрутизации с учетом состояния линий, работающих по такому принципу, является **MOSPF (Multicast OSPF — многоадресный OSPF)** (Мой; Moy, 1994).

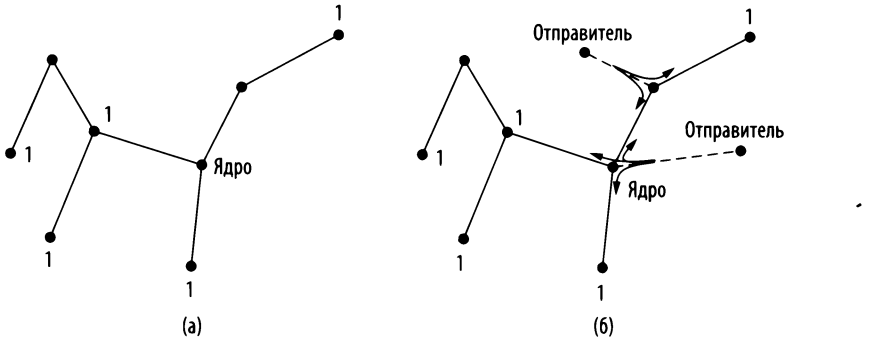
При маршрутизации по вектору расстояний может применяться другая стратегия усечения дерева. Для многоадресной рассылки здесь используется пересылка в обратном направлении. Когда многоадресное сообщение приходит на маршрутизатор, у которого нет хостов, входящих в группу (или соединений с другими маршрутизаторами, принимающими сообщения для группы), он может

ответить сообщением PRUNE (отсечь). Так он сообщает о том, что пакеты для данной группы ему больше отправлять не нужно. Такой же ответ может дать маршрутизатор, у которого нет хостов, входящих в группу, если он получил сообщение PRUNE по всем линиям, по которым осуществил многоадресную рассылку. В результате связующее дерево постепенно рекурсивно усекается. Примером протокола многоадресной маршрутизации, работающего по такому принципу, является **DVRMP (Distance Vector Multicast Routing Protocol — протокол многоадресной маршрутизации по вектору расстояний)** (Вайцман и др.; Waitzman et al., 1988).

Усечение позволяет строить эффективные связующие деревья, состоящие только из тех соединений, которые необходимы для связи с членами группы. Недостаток данного метода в том, что он требует от маршрутизаторов выполнения большого количества операций, особенно в крупных сетях. Предположим, что в сети есть $n$ групп, каждая из которых в среднем состоит из $m$ членов. Каждый маршрутизатор должен хранить $m$ усеченных связующих деревьев для каждой группы, то есть $mn$ деревьев для всей сети. К примеру, на илл. 5.16 (в) изображено связующее дерево, используемое самым левым маршрутизатором для связи с группой 1. Дерево, по которому самый правый маршрутизатор отправляет пакеты группе 1 (оно не показано на рисунке), выглядит по-другому, поскольку пакеты передаются непосредственно членам группы, а не через узлы в левой части графа. Это значит, что выбор направления, в котором маршрутизаторы должны передавать пакеты для группы 1, зависит от того, какой узел является отправителем. При большом количестве групп и отправителей для хранения всех деревьев потребуется много памяти.

Для построения отдельного связующего дерева для группы можно использовать **деревья с корнем в ядре (core-based trees)** (Балларди и др.; Ballardie et al., 1993). Согласно этому методу все маршрутизаторы выбирают общий корень, также называемый **ядром (core)** или **точкой встречи (rendezvouz point)**. Чтобы построить дерево, все члены группы передают в этот корень специальный пакет. Конечное дерево формируется из маршрутов, пройденных пакетами. На илл. 5.17 (а) показано дерево с корнем в ядре для группы 1. Для пересылки пакета этой группе отправитель передает сообщение ядру, откуда оно уже рассылается по дереву. Пример работы алгоритма продемонстрирован на илл. 5.17 (б) для отправителя, расположенного в правой части сети. Однако производительность этого метода может быть улучшена. Дело в том, что для многоадресной рассылки не требуется, чтобы пакеты для группы приходили в ядро. Как только пакет достигает дерева, он может быть передан как в направлении корня дерева, так и по любой его ветви. Так алгоритм работает для отправителя, расположенного вверху илл. 5.17 (б).

Общее дерево не является оптимальным вариантом для всех источников. На илл. 5.17 (б) пакет от отправителя в правой части сети достигает правого верхнего участника группы не напрямую, а через ядро, что требует трех переходов. Степень неэффективности зависит от взаимного расположения ядра и отправителей; чаще всего разумно располагать ядро посередине между отправителями. Если источник всего один (как, например, при видеотрансляции), лучше использовать в качестве ядра самого отправителя.




**Илл. 5.17.** (а) Дерево с корнем в ядре для группы 1. (б) Рассылка для группы 1

Следует отметить, что использование общего дерева позволяет существенно снизить затраты на хранение информации, а также уменьшить число отправленных сообщений и объем вычислений. Для каждой группы маршрутизатор должен хранить не $m$ деревьев, а лишь одно. Кроме того, маршрутизаторы, не являющиеся частью дерева, не участвуют в передаче сообщений группе. Поэтому алгоритмы на основе общих деревьев (в частности, деревьев с корнем в ядре) используются при широковещании для разреженных групп в сети интернет. Они входят в такие популярные протоколы, как PIM (Protocol Independent Multicast — многоадресная рассылка, не зависящая от протокола) (Феннер и др.; Fenner et al., 2006).

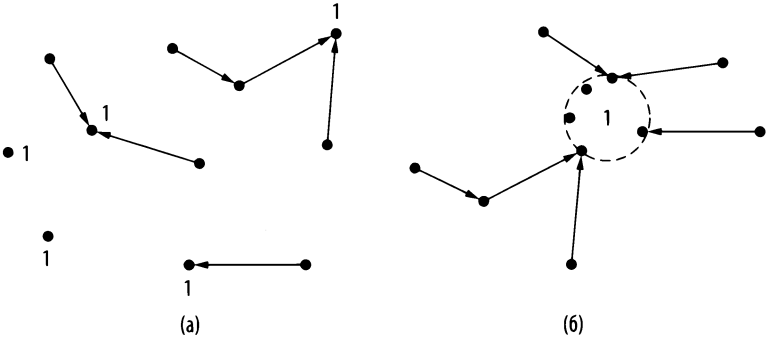
## 5.2.9. Произвольная маршрутизация

До сих пор мы рассматривали модели предоставления информации, в которых источник отправляет сообщение на один адрес (**одноадресная рассылка — unicast**), на все адреса (широковещание) или группе адресов (многоадресная рассылка). Существует еще одна модель под названием **произвольная рассылка (anycast)**, когда пакет отправляется ближайшему члену группы (Партридж и др.; Partridge et al., 1993). Методы поиска путей в этом случае называются **произвольной маршрутизацией (anycast routing)**.

Зачем нужна произвольная рассылка? Иногда узлы предоставляют услугу (например, сообщают время суток или передают контент), для которой важно одно: чтобы информация была правильной; при этом не имеет значения, какой узел ее предоставил — с этой задачей справится любой из них. Пример использования свободной рассылки в интернете — DNS, о которой мы поговорим в главе 7.

К счастью, нам не придется придумывать новые алгоритмы для произвольной маршрутизации: стандартные методы — маршрутизация по вектору расстояния и маршрутизация с учетом состояния линий — позволяют строить пути для произвольной рассылки. Предположим, что нам нужно передать данные группе 1. Вместо разных адресов все ее участники получат одинаковый адрес — «1». Алгоритм маршрутизации по вектору расстояния распределит векторы обычным способом, и узлы выберут кратчайший путь к адресу 1. В результате узлы

отправят данные на ближайшее устройство с таким адресом. Эти пути показаны на илл. 5.18 (а). Данный метод работает, поскольку протокол маршрутизации не знает о существовании нескольких устройств с адресом 1 и считает их одним узлом, как показано в топологии на илл. 5.18 (б).




**Илл. 5.18.** Произвольная маршрутизация. (а) Маршруты для свободной рассылки. (б) Топология с точки зрения протокола маршрутизации

Такой метод будет работать и для маршрутизации с учетом состояния линий. Но следует отметить, что протокол не обязан находить кажущиеся кратчайшими пути, проходящие через узел 1. Это привело бы к «прыжку через гиперпространство», потому что экземпляры узла 1 на самом деле расположены в различных частях сети. Впрочем, современные протоколы маршрутизации с учетом состояния линий различают маршрутизаторы и хосты (о чем мы не упоминали, поскольку в этом не было необходимости).

## 5.3. УПРАВЛЕНИЕ ТРАФИКОМ НА СЕТЕВОМ УРОВНЕ

Слишком большое количество пакетов в любой части сети может в конечном итоге привести к задержке и потере пакетов, что негативно скажется на производительности. Такая ситуация называется **перегрузкой (congestion)**.

### 5.3.1. Необходимость в управлении трафиком: перегрузка

За борьбу с перегрузкой отвечают сетевой и транспортный уровни. Поскольку она происходит в сети, именно сетевой уровень сталкивается с ней непосредственно и определяет, что делать с лишними пакетами. Самое эффективное решение — уменьшить нагрузку на сеть со стороны транспортного уровня; это значит, что оба уровня должны работать вместе. Сетевой уровень не может автоматически устранить перегрузку. Но операторы сетей могут настроить маршрутизаторы, коммутаторы и другие устройства этого уровня так, чтобы они смягчали ее последствия. Как правило, они заставляют источник снизить скорость отправки или направляют трафик по другим, менее загруженным

путям. В этой главе мы рассмотрим вопросы перегрузки, связанные с сетевым уровнем, и механизмы, которые он применяет для борьбы с ней. Для описания функций транспортного уровня в литературе часто используется более общее понятие «контроль перегрузок». Во избежание путаницы здесь мы будем говорить об **управлении перегрузками (congestion management)** или **управлении трафиком (traffic management)**, понимая под этим методы, применяющиеся на сетевом уровне. В главе 6 мы завершим рассмотрение темы, обсудив механизмы управления перегрузками на транспортном уровне.

На илл. 5.19 показано, как начинается перегрузка. Когда количество отправляемых хостами пакетов намного ниже пропускной способности сети, объем доставленного трафика пропорционален объему переданного (в два раза больше отправлено — в два раза больше получено). Но если число пакетов приближается к пределу емкости, при эпизодических всплесках трафика буферы маршрутизаторов переполняются; в итоге некоторые пакеты теряются. Потерянные пакеты расходуют часть пропускной способности, поэтому объем доставленного трафика оказывается ниже идеальной кривой. Таким образом, в сети возникает перегрузка.

**Илл. 5.19.** Перегрузка ведет к значительному падению производительности: возрастает частота потери пакетов, а также величина задержки, поскольку очереди маршрутизатора заполнены пакетами

В определенный момент может возникнуть **коллапс сети из-за перегрузки (congestion collapse)**, при котором производительность падает, поскольку абонентская нагрузка превышает пропускную способность. Коллапс сети происходит, когда увеличение нагрузки фактически ведет к уменьшению объема успешно доставленного трафика. При этом пакеты настолько задерживаются, что становятся бесполезными к моменту их доставки. Например, на ранних этапах существования интернета время, которое пакет проводил в очереди на отправку (при скорости 56 Кбит/с), зачастую превышало время его пребывания в сети. В результате пакет приходилось удалять. Еще один неприятный сценарий — когда отправители повторно передают сильно задержанные пакеты, полагая, что они утеряны. В этом случае пропускная способность используется неэффективно, поскольку по сети передаются копии одного и того же пакета. Чтобы продемонстрировать влияние этих факторов на производительность, мы отобразили на оси $y$ (см. илл. 5.19) **полезную пропускную способность (goodput)**, то есть скорость, с которой по сети передаются *полезные* пакеты.

В идеале сеть должна быть устроена так, чтобы перегрузки происходили как можно реже и чтобы в этих ситуациях не возникало коллапсов. К сожалению, в сети с коммутацией пакетов перегрузку невозможно полностью исключить. Если потоки пакетов внезапно начинают прибывать на маршрутизатор сразу по трем или четырем входным линиям и всем им нужна одна и та же выходная линия, то образуется очередь. Если у маршрутизатора заканчивается память для буферизации пакетов, они теряются. Увеличение объема памяти может в какой-то степени помочь, однако Нейгл (Nagle) в 1987 году установил, что при бесконечной памяти маршрутизаторов ситуация с перегрузкой часто не улучшается, а, наоборот, ухудшается. Последние исследования показали, что многие сетевые устройства имеют больше памяти, чем требуется. Это явление получило название **излишней сетевой буферизации (bufferbloat)**. Такие устройства могут снижать производительность сети, чему есть несколько объяснений. Во-первых, за то время, пока пакеты добираются до начала очереди, срок их ожидания истекает (и даже несколько раз) и производится отправка их дубликатов. Во-вторых, отправителей нужно своевременно уведомлять о перегрузках (о чем мы подробнее поговорим в главе 6). Если пакеты не будут удалены, а останутся в буферах маршрутизатора, то отправители продолжат отправку перегружающего сеть трафика. В результате ситуация ухудшится: произойдет коллапс сети. Линии с низкой пропускной способностью или маршрутизаторы, у которых скорость обработки пакетов ниже емкости канала, также могут быть перегружены. Если на других участках сети пропускная способность выше, проблему можно решить, направив туда часть трафика в обход узкого места. Но в конечном итоге увеличение трафика может привести к повсеместной перегрузке. В этом случае операторы сетей могут применить два подхода: сброс нагрузки (то есть игнорирование трафика) или увеличение пропускной способности.

Здесь уместно прояснить разницу между понятиями **«контроль перегрузок» (congestion control)**, **«управление трафиком» (traffic management)** и **«управление потоком» (flow control)**. Задача управления трафиком (или регулирования трафика) — гарантировать, что сеть справится с поступающим трафиком. Задача может решаться как устройствами в сети, так и отправителями (часто с использованием механизмов транспортного протокола, называемых методами контроля перегрузок). Управление перегрузками (или контроль перегрузок) касается поведения всех хостов и маршрутизаторов. Управление потоком, наконец, относится к трафику между конкретными отправителями и получателями. С его помощью регулируется скорость отправки данных: она не должна превышать максимально возможную скорость их получения. Задача управления потоком — не допустить потери данных из-за того, что отправитель обладает большей производительностью и его скорость превышает возможности адресата.

Чтобы понять разницу между этими понятиями, представьте сеть из 100-гигабитных оптоволоконных линий. Суперкомпьютер пытается передать большой файл персональному компьютеру, способному принимать данные со скоростью 1 Гбит/с. Хотя перегрузки в данной ситуации не наблюдается, алгоритм управления потоком периодически заставляет отправителя приостанавливать передачу, чтобы получатель успевал принимать трафик.

Приведем другой пример. Рассмотрим сеть из 1000 больших компьютеров, соединенных мегабитными линиями. Половина компьютеров пытается передать файлы остальным со скоростью 100 Кбит/с. Здесь проблема заключается уже не в том, что медленные получатели не успевают принимать данные от быстрых отправителей. Просто сеть не способна пропустить весь предлагаемый трафик.

Причина, по которой контроль перегрузок и управление потоком часто путают, заключается в том, что лучшее решение обеих проблем — добиться более медленной работы хоста. Таким образом, хост может получить просьбу снизить скорость в двух случаях: когда с потоком не справляется получатель или когда с ним не справляется вся сеть. Мы вернемся к этому вопросу в главе 6.

Мы начнем знакомство с управлением перегрузками с рассмотрения подходов, которые могут применяться сетевыми операторами в разных масштабах времени. Затем мы изучим методы предотвращения перегрузки, а также различные алгоритмы борьбы с ее последствиями.

## 5.3.2. Методы управления трафиком

Перегрузка означает, что нагрузка на сеть превышает (временно) возможности ресурсов сети (или некоторой ее части). Существует два возможных решения: увеличить объем ресурсов или снизить нагрузку. Как показано на илл. 5.20, эти решения обычно применяются в разных временных рамках в зависимости от задачи: предотвратить перегрузку или справиться с ней, если ее не удалось избежать.

**Илл. 5.20.** Временные рамки методов управления трафиком и перегрузками

Самый простой способ избежать перегрузки заключается в создании сети с достаточной пропускной способностью для передачи ожидаемого трафика. Если часть пути, по которому обычно пересылаются большие объемы данных, обладает низкой пропускной способностью, вероятность возникновения перегрузки довольно высока. Иногда при перегрузке возможно динамическое добавление ресурсов, например подключение свободных маршрутизаторов или резервных линий (для обеспечения устойчивости к ошибкам) либо же приобретение дополнительной пропускной способности на свободном рынке. Как правило, наиболее загруженные соединения и маршрутизаторы обновляются при первой возможности. Этот процесс называется **обеспечением (provisioning)** и происходит раз в несколько месяцев, в результате оценки долгосрочной динамики трафика.

Чтобы максимально использовать существующую пропускную способность сети, маршруты могут быть адаптированы к схемам трафика, которые меняются

в течение дня, поскольку пользователи сети просыпаются и засыпают в разных часовых поясах. Например, можно обходить загруженные линии, меняя весовые коэффициенты кратчайшего пути. Некоторые радиостанции следят за ситуацией на дорогах с вертолетов и передают полученные сведения радиослушателям, чтобы те могли объехать пробки. Это называется **маршрутизацией с учетом состояния трафика (traffic-aware routing)**. Так же можно разделять трафик, направляя его по разным линиям.

Иногда повысить пропускную способность невозможно, особенно за короткий срок. Тогда единственным выходом является снижение нагрузки.

В сети виртуальных каналов новые соединения отклоняются, если они могут привести к перегрузке сети. Это один из примеров **управления допуском (admission control)**, при котором отправители просто лишаются возможности передавать трафик, если это превышает возможности сети.

Когда перегрузка неизбежна, сеть может передать сообщение обратной связи отправителям, трафик которых вызывает проблему. Сеть может либо попросить этих отправителей снизить скорость, либо замедлить трафик самостоятельно, то есть произвести **троттлинг (throttling)**. Сложность второго подхода в том, что нужно обнаружить начало перегрузки и уведомить отправителя, что необходимо снизить скорость отправки. Для этого маршрутизаторы отслеживают среднюю нагрузку сети, время ожидания в очереди и частоту потери пакетов и дают отправителям обратную связь: явно или косвенно (например, путем удаления пакетов) сообщают о необходимости замедлиться.

В случае явного отклика маршрутизаторы должны участвовать в цикле обратной связи с отправителями. Чтобы данная схема работала, необходимо тщательно настроить временные параметры. Если каждый раз при одновременном получении двух пакетов маршрутизатор кричит «Стоп!», а простояв без работы 20 мкс, командует «Давай!», система будет находиться в состоянии постоянных незатухающих колебаний. И наоборот, если маршрутизатор для большей надежности станет ждать 30 минут, прежде чем что-либо сообщить, то механизм борьбы с перегрузкой будет реагировать слишком медленно, чтобы приносить хоть какую-то пользу. Своевременная доставка сообщений обратной связи не такая уж простая задача. Дополнительная проблема заключается в том, что маршрутизаторы отправляют больше сообщений, когда сеть уже и так перегружена.

Наконец, сеть может удалить те пакеты, которые она не в состоянии доставить. Такой подход получил общее название «сброс нагрузки». Существуют разные способы его реализации, включая формирование трафика (ограничение скорости конкретного отправителя) и соблюдение политики трафика (игнорирование трафика отправителя при превышении им некоторого порога). Правильно определив стратегию в отношении того, какие пакеты удалять, можно предотвратить коллапс сети. Далее мы подробно все это обсудим.

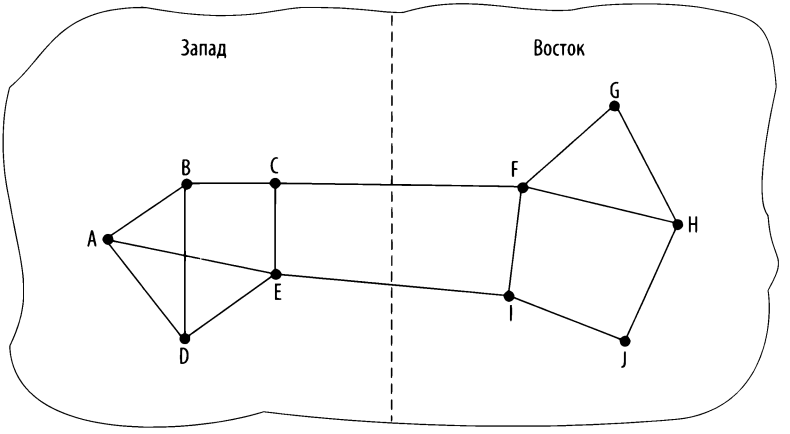
### *Маршрутизация с учетом состояния трафика*

Первый подход, который мы рассмотрим, называется маршрутизацией с учетом состояния трафика. Методы маршрутизации, представленные в разделе 5.2,

используют фиксированные весовые коэффициенты связей, которые адаптируются к изменениям топологии, но не к изменениям нагрузки трафика. Однако нагрузку также следует учитывать при вычислении маршрутов, чтобы направлять трафик в обход наиболее подверженных перегрузке участков.

Наиболее простой способ — сделать весовой коэффициент связи функцией от пропускной способности связи (фиксированной) и задержки распространения, а также измеренной (переменной) нагрузки или среднего времени ожидания в очереди. В результате пути с наименьшим весом будут при прочих равных параметрах наименее нагруженными.

Маршрутизация с учетом состояния трафика использовалась на ранних этапах развития интернета в рамках модели Кханны и Зинки (Khanna and Zinky, 1989). Однако здесь есть небольшая опасность. Рассмотрим сеть на илл. 5.21. Она разделена на две части — восток и запад, соединенные линиями *CF* и *EI*. Предположим, что основной трафик между западом и востоком проходит по *CF*, поэтому она слишком нагружена, что приводит к длительным задержкам. Учет времени ожидания в очереди при вычислении кратчайшего пути сделает линию *EI* более популярной. После внесения изменений в таблицы маршрутизации большая часть трафика пойдет по *EI* и слишком нагруженной окажется именно эта линия. При следующем обновлении таблиц кратчайшим путем снова станет *CF*. В конечном итоге значения в таблицах будут сильно колебаться, что приведет к ошибкам при выборе маршрутов и другим проблемам.




**Илл. 5.21.** Сеть, в которой восточная и западная части соединены двумя линиями связи

Если игнорировать нагрузку и учитывать только пропускную способность и задержку распространения, такой проблемы не возникнет. Попытки учесть нагрузку, используя узкий диапазон весовых значений, лишь замедляют колебания маршрутов. Удачное решение проблемы основывается на двух методах. Первый — это маршрутизация, при которой между отправителем и получателем существует несколько путей. В нашем примере это означает, что пакеты можно

передавать по обеим линиям между востоком и западом. Второй метод состоит в следующем: схема маршрутизации перемещает трафик по маршрутам настолько медленно, чтобы он конвергировался; так, например, работает схема Галлагера (Gallager, 1977).

С учетом этих трудностей протоколы маршрутизации интернета обычно не корректируют пути на основании уровня нагрузки. Вместо этого операторы сетей настраивают протоколы в более широких временных рамках, плавно меняя конфигурацию и параметры маршрутизации, другими словами, регулируют трафик. Это всегда было трудоемким ручным процессом, чем-то напоминающим черную магию. Несмотря на попытки его формализации, он оставался достаточно примитивным в силу непредсказуемого характера нагрузки в интернете и некоторой громоздкости параметров конфигурации протоколов. С появлением программно-конфигурируемых сетей часть задач была автоматизирована, а распространение по всей сети таких технологий, как MPLS-туннели, предоставило операторам большую гибкость при реализации многих составляющих регулирования трафика.

### *Управление допуском*

Популярный метод предотвращения перегрузки в сетях виртуальных каналов — **управление допуском (admission control)**. Идея проста: не создавать новый виртуальный канал до тех пор, пока сеть не сможет обработать дополнительный трафик без перегрузки. Таким образом, попытка добавить виртуальный канал может закончиться неудачно. Это лучшее решение, так как если позволить дополнительным пользователям подключиться к заполненной сети, ситуация только ухудшится. По аналогии, когда коммутатор в телефонной системе перегружен, вы поднимаете трубку и не слышите гудка.

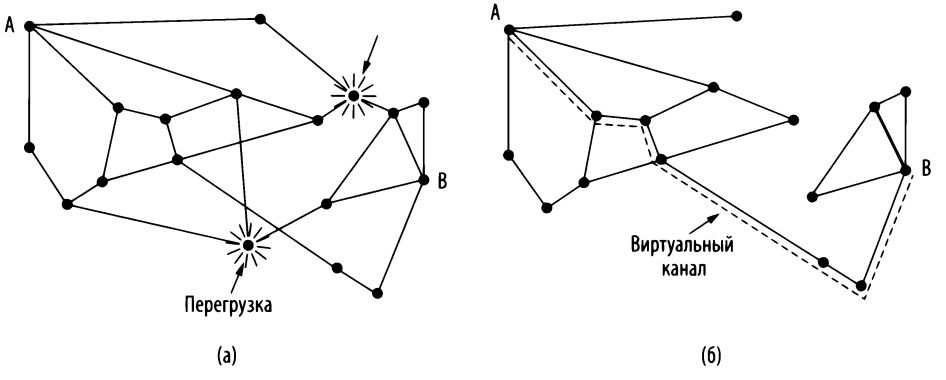
Суть этого подхода в том, чтобы определить, когда добавление нового виртуального канала приведет к перегрузке. В телефонных сетях эта задача решается просто благодаря фиксированной пропускной способности для вызовов (64 Кбит/с для несжатого аудио). Но в компьютерных сетях используются виртуальные каналы всевозможных типов. Поэтому если мы хотим прибегнуть к управлению допуском, каналы должны предоставлять информацию о характеристиках трафика.

Часто речь идет о скорости и форме. Возникает вопрос, как описать эти характеристики просто и содержательно. Обычно трафик неравномерен, поэтому нельзя полагаться только на среднюю скорость. Например, гораздо сложнее справиться с колебаниями трафика при поиске в интернете, чем при просмотре потокового видео того же объема, поскольку всплески интернет-трафика с большей вероятностью приводят к перегрузке маршрутизаторов сети. Этот эффект часто называют «дырявым ведром» (leaky bucket) или «маркерным ведром» (token bucket). «Дырявое ведро» обладает двумя параметрами, ведущими к ограничению средней скорости и мгновенному увеличению объема трафика. Поскольку это два наиболее распространенных механизма формирования трафика, мы рассмотрим их более подробно в данном разделе.

Располагая характеристиками трафика, сеть принимает решение о добавлении нового виртуального канала. Чтобы не произошло перегрузки, сеть может, например, зарезервировать часть пропускной способности для каждого виртуального канала. В этом случае характеристика трафика выступает в качестве договора об обслуживании, которое сеть обязуется предоставить пользователю. В разговоре о предотвращении перегрузки мы слишком рано отклонились в сторону смежной темы, QoS; вернемся к ней чуть позже.

Даже если сеть не берет на себя никаких обязательств, она может использовать характеристики трафика для управления доступом. При этом задача сводится к оценке числа виртуальных каналов, достаточного для обеспечения нужной пропускной способности сети и работы без перегрузок. Предположим, что все виртуальные каналы, которые могут передавать трафик со скоростью до 10 Мбит/с, используют один и тот же физический канал с пропускной способностью 100 Мбит/с. Сколько каналов можно использовать? Очевидно, что при 10 каналах нет никакого риска перегрузки, но это неэффективно в обычной ситуации, поскольку вряд ли все они будут одновременно работать в полную силу. В действующих сетях для оценки числа возможных каналов используются статистические данные. Таким образом, за счет допустимого увеличения риска сеть выигрывает в производительности.

Можно совместить принципы управления доступом и маршрутизации с учетом состояния трафика, направляя маршруты в обход проблемных участков. Для примера рассмотрим сеть на илл. 5.22 (а), в которой два маршрутизатора перегружены.




**Илл. 5.22.** (а) Перегруженная сеть. (б) Участок сети без перегрузки. Также показан виртуальный канал между A и B

Предположим, что хост, подключенный к маршрутизатору *A*, хочет установить соединение с хостом, соединенным с *B*. В других обстоятельствах это соединение прошло бы через один из перегруженных маршрутизаторов. Чтобы этого избежать, сеть усекается, как показано на илл. 5.22 (б). При этом исключаются перегруженные маршрутизаторы и все их линии связи. Пунктиром показан возможный путь виртуального канала в обход перегруженных маршрутизаторов.

Подробное описание маршрутизации, чувствительной к нагрузке, можно найти в работе Шейха и др. (Shaikh et al., 1999).

### Сброс нагрузки

Когда ни один из описанных методов не помогает в борьбе с перегрузкой, маршрутизаторы могут ввести в бой тяжелую артиллерию — **сброс нагрузки (load shedding)**. По сути, это игнорирование маршрутизаторами пакетов, которые они не могут обработать. Своим происхождением этот термин обязан сфере электроснабжения, где он означает отключение в случае перегрузок отдельных участков во избежание выхода из строя всей системы. Обычно это происходит в жаркие летние дни, когда спрос на электроэнергию (для питания кондиционеров) многократно превышает предложение.

Ключевой вопрос для маршрутизатора, заваленного пакетами, — какой из них исключить. Выбор зависит от типа приложений, использующих данную сеть. При передаче файлов более старые пакеты представляют большую ценность, чем новые. Например, если отбросить пакет 6, но сохранить пакеты 7–10, это заставит получателя выполнить лишнюю работу: поместить в буфер данные, которые он еще не может использовать. Для мультимедийных приложений, работающих в реальном времени, напротив, новый пакет важнее старого, поскольку пакеты становятся ненужными, если не приходят вовремя.

Первую стратегию (старое лучше нового) часто называют **винной стратегией (wine)**, а вторую (новое лучше старого) — **молочной стратегией (milk)**, так как большинство людей предпочтут пить свежее молоко и выдержанное вино, а не наоборот.

Более сложные алгоритмы сброса нагрузки требуют участия отправителей. В качестве примера можно привести пакеты, содержащие сведения о маршрутизации. Они намного важнее обычных пакетов с данными, поскольку устанавливают маршруты; если они будут утеряны, может пострадать связность сети. Другой пример — алгоритмы сжатия видеосигнала (например, MPEG), которые периодически передают полный фрейм, а далее отправляют его измененные версии. В этом случае потеря пакета с изменениями не так страшна, как потеря полного фрейма, так как от него зависят последующие пакеты.

Чтобы реализовать осмысленную стратегию отбрасывания части данных, приложения должны помечать свои пакеты, сообщая сети об их важности. Тогда маршрутизаторы смогут сначала исключить наименее важные пакеты, затем чуть более существенные, и т. д.

Конечно, при отсутствии определенного стимула все будут помечать свои пакеты не иначе как ОЧЕНЬ ВАЖНО — НИ В КОЕМ СЛУЧАЕ НЕ ВЫБРАСЫВАТЬ. Предотвращение неоправданного использования таких меток часто достигается за счет сетевых ресурсов и денежных средств. Например, сеть может разрешить отправителям пересылать пакеты с большей скоростью, чем указано в договоре на предоставление услуг, если пакет будет помечаться как низкоприоритетный. Такая стратегия весьма удачна, поскольку более эффективно распределяет свободные ресурсы: хостам разрешено пользоваться ими, пока это никому не мешает, но это право за ними не закреплено.

### *Формирование трафика*

Чтобы гарантировать пользователю определенную производительность, сеть должна знать примерный объем трафика. В телефонных сетях его оценить легко: обычный звонок (в несжатом формате) требует скорости 64 Кбит/с и одного 8-битного отсчета каждые 125 мкс. Однако в сетях передачи данных трафик **неравномерен (bursty)**. Он может быть неоднородным по трем причинам: непостоянная скорость передачи (например, при видеоконференциях со сжатием), взаимодействие пользователя и приложения (например, просмотр новой веб-страницы) и переключение компьютера между задачами. С неравномерным трафиком работать гораздо сложнее, чем с постоянным, так как при этом может произойти переполнение буферов и, как следствие, утеря пакетов.

**Шейпинг трафика (traffic shaping)** — способ регулировки средней скорости и равномерности потока входных данных. Приложения должны иметь возможность передавать тот трафик, который им нужен (включая неравномерный); при этом нужен простой и удобный способ описания возможных схем трафика. Когда устанавливается поток, пользователь и сеть (то есть клиент и оператор связи) договариваются об определенной схеме (то есть форме) трафика для данного потока. В результате клиент сообщает провайдеру: «Мой график передачи будет выглядеть следующим образом. Сможете ли вы это обеспечить?»

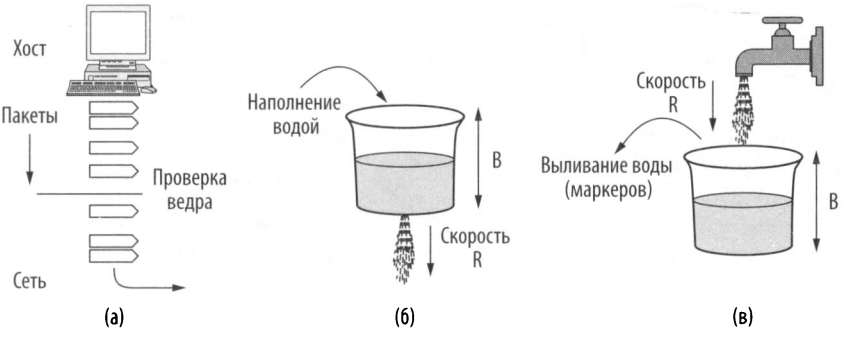
Иногда такой договор называется **соглашением об уровне обслуживания (SLA, Service Level Agreement)**, особенно если в нем оговариваются комплексные потоки и длительные периоды времени (например, весь трафик для данного клиента). До тех пор пока клиент выполняет свою часть условий соглашения и передает пакеты в пределах установленного графика, провайдер обязуется доставлять их в определенный срок.

Шейпинг трафика снижает перегрузку и таким образом помогает сети выполнять свои обязательства. Но чтобы это работало, необходимо ответить на вопрос: как провайдер узнает о том, что клиент соблюдает соглашение, и что делать, если клиент нарушит договор. Пакеты, отправка которых выходит за рамки условий, могут удаляться или помечаться как низкоприоритетные. Наблюдение за потоком трафика называется **политикой трафика (traffic policing)**.

Шейпинг трафика и его политика не так существенны для однорангового обмена и передач других типов, где используется вся доступная пропускная способность. Но они имеют важное значение для данных в реальном времени (например, при аудио- и видеосоединениях), обладающих строгими требованиями к QoS. Нам уже знаком один способ ограничения объема данных, отправляемого приложениями, — раздвижное окно. С помощью одного параметра он определяет, сколько данных передается в конкретный момент времени, что косвенно сдерживает скорость. Теперь мы обратимся к более общим методам описания трафика и рассмотрим алгоритмы дырявого и маркерного ведра. Хотя эти методы немного отличаются друг от друга, они дают одинаковый результат.

Представьте себе ведро с маленьким отверстием в донышке, как показано на илл. 5.23 (б). Независимо от скорости, с которой вода наливается в ведро, выходной поток обладает постоянной скоростью $R$, когда в ведре есть вода,

и нулевой скоростью, когда оно пустое. Кроме того, когда ведро наполняется и вода занимает весь объем $B$, вся лишняя вода выливается через край и теряется.

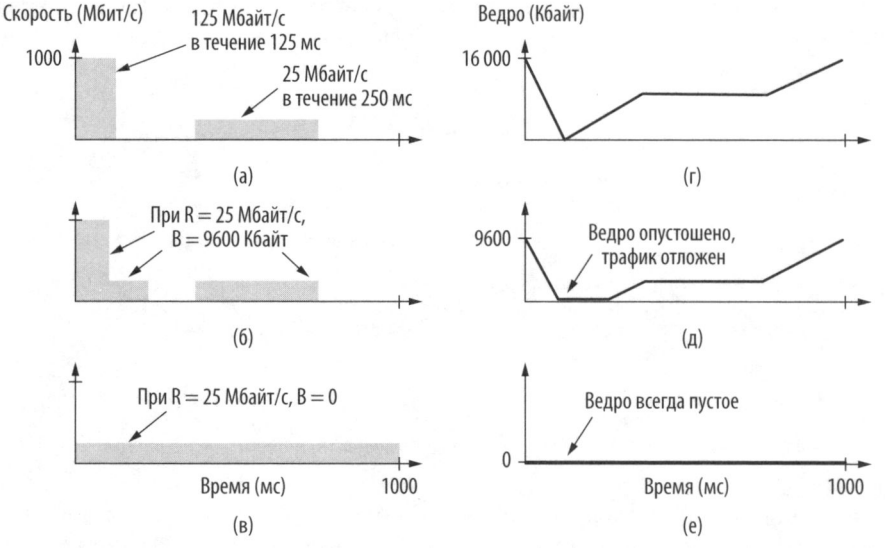



**Илл. 5.23.** (а) Формирование пакетов. (б) Алгоритм дырявого ведра. (в) Алгоритм маркерного ведра

С помощью такого ведра можно формировать или приводить в порядок пакеты, поступающие в сеть (илл. 5.23 (а)). Принцип следующий: каждый хост соединяется с сетью через интерфейс, содержащий «дырявое ведро». Чтобы пакет можно было отправить в сеть, в ведре должно быть достаточно места для воды. Если в момент поступления пакета ведро заполнено, пакет либо помещается в очередь до тех пор, пока в ведре не освободится достаточно места, либо отвергается. Первый вариант встречается, когда формирование трафика производится операционной системой хоста. Второй — когда проверка трафика, поступающего в сеть, осуществляется аппаратными средствами в сетевом интерфейсе интернет-провайдера. Этот метод был предложен Тернером (Turner, 1986) и называется **алгоритмом дырявого ведра (leaky bucket algorithm)**.

То же самое можно представить по-другому: в виде ведра, которое в данный момент наполняется (см. илл. 5.23 (в)). Вода вытекает из крана со скоростью $R$, а объем ведра, как и в предыдущем случае, равен $B$. Чтобы отправить пакет, необходимо, чтобы из ведра можно было вылить воду (или маркеры — так обычно называют содержимое ведра), а не налить ее туда. В ведре может содержаться ограниченное число маркеров (не более $B$); если ведро пустое, для отправки пакета необходимо подождать, пока не появятся новые маркеры. Данный метод называется **алгоритмом маркерного ведра (token bucket algorithm)**.

Алгоритмы дырявого и маркерного ведра ограничивают постоянную скорость потока, при этом пропуская кратковременные всплески трафика (также ограниченные максимальным значением) без искусственных задержек. Чтобы снизить нагрузку на сеть, шейпер «дырявого ведра» сглаживает крупные всплески трафика. Представьте компьютер, который производит данные со скоростью 1000 Мбит/с (125 млн байт в секунду); первая связь сети работает на той же скорости. Хост генерирует схему трафика, показанную на илл. 5.24 (а). Эта схема является неравномерной. Средняя скорость составляет 200 Мбит/с, хотя хост отправляет пиковый объем трафика в 16 000 Кбайт на максимальной скорости 1000 Мбит/с (за 1/8 с).




**Илл. 5.24.** (а) Трафик, передаваемый хостом. (б) Исходящий трафик, сформированный с помощью маркерного ведра со скоростью 200 Мбит/с и емкостью 9600 Кбайт. (в) Та же скорость, емкость 0 Кбайт. (г) Уровень маркерного ведра при формировании трафика со скоростью 200 Мбит/с и емкостью 16 000 Кбайт. (д) Та же скорость, емкость 9600 Кбайт. (е) Та же скорость, емкость 0 Кбайт

Теперь предположим, что маршрутизаторы могут принимать данные на максимальной скорости только в течение небольших временных интервалов — до тех пор, пока буфер не заполнится. Размер буфера составляет 9600 Кбайт — это меньше объема пикового трафика. При долгих интервалах маршрутизаторы лучше всего работают, если скорость не превышает 200 Мбит/с (именно такая пропускная способность указана в соглашении с клиентом). Из этого следует, что если для передачи используется эта схема, часть трафика будет удалена, так как не поместится в буферы маршрутизаторов.

Чтобы избежать потери пакетов, можно сформировать трафик на хосте по методу маркерного ведра. Если скорость $R$ равна 200 Мбит/с, а емкость $B$ равна 9600 Кбайт, трафик находится в пределах возможностей сети. Исходящий трафик такого маркерного ведра показан на илл. 5.24 (б). Хост может передавать на максимальной скорости 1000 Мбит/с в течение небольшого промежутка времени — до тех пор, пока ведро не опустеет. Затем он должен будет снизить скорость до 200 Мбит/с и отправить оставшуюся часть трафика. Суть в том, чтобы растянуть время передачи пикового трафика (пачки пакетов), если сеть не в состоянии обработать его в один прием. Уровень маркерного ведра показан на илл. 5.24 (д). Вначале ведро наполнено, но после первой порции трафика оно становится пустым. Когда уровень достигает нуля, новые пакеты могут передаваться только с той скоростью, с какой в буфер поступают маркеры; отправка крупных объемов трафика невозможна, пока ведро снова не будет полным. Оно постепенно наполняется, когда трафик не приходит, и остается пустым, пока данные приходят со скоростью его заполнения.

Чтобы трафик был равномерным, его можно сформировать. На илл. 5.24 (в) показан исходящий трафик маркерного ведра со скоростью 200 Мбит/с и емкостью 0. Это крайний случай — трафик полностью выровнен. Крупные пачки не принимаются; трафик приходит с постоянной скоростью. Уровень воды в ведре, соответственно, всегда равен нулю (см. илл. 5.24 (е)). Трафик помещается в очередь хоста, и в каждый момент времени какой-то пакет ожидает отправки.

Наконец, на илл. 5.24 (д) показан уровень маркерного ведра со скоростью $R$ = 200 Мбит/с и емкостью $B$ = 16 000 Кбайт. Это самое маленькое маркерное ведро, через которое данные проходят без изменений. Маршрутизатор может использовать его для проверки трафика, передаваемого хостом. Такое ведро можно расположить на одном из концов сети; если трафик будет соответствовать маркерному ведру, указанному в SLA, он сможет пройти через ведро. Если же хост будет отправлять данные слишком быстро или неравномерно, вода в маркерном ведре закончится и сеть узнает о том, что трафик не соответствует условиям договора. Далее в зависимости от конфигурации сети лишние пакеты будут либо удалены, либо помечены как низкоприоритетные. В нашем примере ведро пустеет лишь на короткое время — после пикового трафика. После этого оно восстанавливается и снова готово к новым всплескам.

Методы дырявого и маркерного ведра просты в реализации. Рассмотрим, как работает маркерное ведро. Хотя до этого мы говорили о непрерывно поступающей и вытекающей воде, нужно понимать, что на практике сеть имеет дело с дискретными величинами. Реализация алгоритма маркерного ведра подразумевает наличие переменной, считающей маркеры. Счетчик увеличивается на $R/\Delta T$. В нашем примере счетчик будет увеличиваться на 200 кбит за 1 мс. Каждый раз при отправке трафика счетчик уменьшается; когда его значение доходит до нуля, передача пакетов останавливается.

Если все пакеты имеют одинаковый объем, уровень ведра можно измерять в них (например, 200 кбит — это 20 пакетов по 1250 байт). Однако чаще всего используются пакеты разного размера. Тогда уровень ведра может исчисляться в байтах. Если байтов в ведре недостаточно для отправки пакета, он вынужден ждать, пока ситуация не изменится (что может произойти не сразу, если скорость заполнения ведра небольшая).

Рассчитывать длительность максимального всплеска (когда ведро пустеет) немного проблематично. Необходимо учитывать, что пока ведро опустошается, появляются новые маркеры. (В силу этого в нашем случае всплеск длится дольше, чем результат деления 9600 Кбайт на 125 Мбайт/с.) При длительности пачки $S$ с, емкости маркерного ведра $B$ байт, скорости появления маркеров $R$ байт/с и максимальной выходной скорости $M$ байт/с очевидно, что максимальное количество переданных байтов в пачке будет равно $B + RS$ байт. Также известно, что количество байтов, переданных в пачке с максимальной скоростью, равно $MS$. Таким образом,

$$B + RS = MS.$$

Решив это уравнение, получим: $S = B/(M - R)$. При наших параметрах $B$ = = 9600 Кбайт, $M$ = 125 Мбайт/с и $R$ = 25 Мбайт/с длительность пачки равна приблизительно 94 мс.

Недостатком алгоритма маркерного ведра является то, что скорость передачи крупных пачек снижается до постоянного значения *R*. Часто бывает необходимо уменьшить пиковую скорость, не возвращаясь при этом к постоянному значению скорости (но и не увеличивая его, чтобы не пропустить в сеть дополнительный трафик). Один из способов получения более гладкого трафика состоит во внедрении еще одного маркерного ведра после первого. Скорость второго ведра должна быть гораздо выше. По сути, первое ведро определяет характеристики трафика и фиксирует его скорость, иногда позволяя отправлять крупные объемы данных. Второе ведро уменьшает максимальную скорость, с которой могут передаваться такие пачки. Например, если скорость второго ведра равна 500 Мбит/с, а емкость — 0, первая пачка поступит в сеть с максимальной скоростью 500 Мбит/с. Это меньше, чем предыдущее значение 1000 Мбит/с.

Управление такими схемами может оказаться непростым. При использовании маркерных ведер для формирования трафика на хостах пакеты вынуждены ждать в очереди, пока ведро их пропустит. Когда они применяются на маршрутизаторах сети для определения трафика, сеть имитирует алгоритм и гарантирует, что пакетов и байтов посылается не больше, чем разрешено. Тем не менее эти методы позволяют формировать сетевой трафик, приводя его к более управляемому виду и обеспечивая тем самым выполнение требований QoS.

### *Активное управление очередью*

В интернете и многих других компьютерных сетях отправители передают столько трафика, сколько сеть в состоянии успешно доставить. В таком режиме сеть работает до тех пор, пока она не перегружена. Если перегрузка неизбежна, она просит отправителей снизить скорость передачи данных. Подобная обратная связь не исключительная, а вполне обычная ситуация, являющаяся частью работы системы. Этот режим работы называется **предотвращением перегрузки (congestion avoidance)** — в противопоставление ситуации, в которой сеть уже перегружена.

Рассмотрим несколько подходов к замедлению трафика, применяемых в дейтаграммных сетях и сетях виртуальных каналов. Каждый подход должен решать две проблемы. Во-первых, необходимо, чтобы маршрутизаторы узнавали о перегрузке до того, как она произойдет. Для этого каждый из них должен непрерывно отслеживать ресурсы, которые он задействует, — использование выходных линий, буферизацию очереди пакетов данного маршрутизатора и число пакетов, утерянных вследствие неправильной буферизации. Наиболее эффективным является второй вариант. Средние показатели использования линий не отражают реальной картины при прерывистом трафике. Так, значение 50 % — это мало при сплошном трафике и очень много при переменном. Число утерянных пакетов становится известно слишком поздно: пакеты начинают теряться уже после возникновения перегрузки.

Время ожидания в очереди маршрутизатора точно отражает, как перегрузка сказывается на пакетах. Будучи низким в большинстве случаев, этот показатель должен резко возрастать при скачке трафика, когда увеличивается число непереданных пакетов. Такую оценку времени ожидания в очереди $d$ можно получить

с помощью несложных вычислений, периодически замеряя мгновенную длину очереди $s$ и рассчитывая новое значение переменной $d$ по формуле

$$d_{\text{new}} = \alpha d_{\text{old}} + (1-\alpha)s,$$

где константа $\alpha$ определяет, насколько быстро маршрутизатор забывает свою недавнюю историю. Это **экспоненциально взвешенное скользящее среднее (Exponentialy Weighted Moving Average, EWMA)**. Оно сглаживает различные флуктуации и работает как фильтр низких частот. Как только значение $d$ выходит за пороговый уровень, маршрутизатор узнает о начале перегрузки.

Вторая проблема состоит в том, что маршрутизаторы должны вовремя доставлять сообщения обратной связи тем источникам, чей трафик вызывает перегрузку. Хотя перегрузка происходит внутри сети, ее устранение требует участия отправителей, пользующихся сетью. Маршрутизатор должен определить их, а затем аккуратно передать им уведомления, не отправляя лишних пакетов в уже перегруженную сеть. Разные алгоритмы используют различные механизмы обратной связи. О них мы и поговорим далее.

### *Произвольное раннее обнаружение перегрузки*

С перегрузкой гораздо проще бороться в тот момент, когда она только началась, чем дать ей развиться до критических размеров и пытаться справиться с этой ситуацией. Это соображение приводит к интересной модификации идеи сброса нагрузки, при которой отбрасывание пакетов происходит еще до того, как все буферное пространство будет заполнено скопившимися необработанными данными.

Причина развития этой идеи в том, что большинство интернет-хостов все еще узнают о перегрузке косвенно (вместо явных уведомлений). Единственным достоверным сигналом служит утеря пакетов. В конце концов, трудно представить себе маршрутизатор, который не удаляет пакеты в такой ситуации. Реакция транспортных протоколов (например, TCP) на утерю пакетов при перегрузке — ответное снижение трафика от источника. Это происходит, поскольку TCP предназначен для проводных сетей, которые по своей сути очень надежны, и потеря пакетов в них чаще всего сигнализирует о переполнении буфера, а не об ошибках передачи. Для эффективной работы TCP беспроводные линии связи должны справляться с ошибками передачи на канальном уровне (так, чтобы на сетевом они были не видны).

Этой ситуацией можно воспользоваться для уменьшения перегрузок. Если заставить маршрутизаторы отбрасывать пакеты еще до того, как ситуация станет безнадежной, то останется время на то, чтобы источник мог предпринять какие-то действия. Популярный алгоритм, реализующий данную идею, называется **произвольным ранним обнаружением перегрузки (Random Early Detection, RED)** (Флойд и Джейкобсон; Floyd and Jacobson, 1993). Чтобы определить, когда следует удалять пакеты, маршрутизаторы постоянно высчитывают скользящее среднее длин своих очередей. Если средняя длина очереди на каком-либо соединении превышает пороговое значение, эта линия объявляется перегруженной и небольшая часть пакетов удаляется случайным образом. Именно случайный

выбор увеличивает вероятность того, что самые быстрые отправители обнаружат утерю пакета. Это наилучший вариант, поскольку маршрутизатор не знает, какой именно источник является самым проблемным в дейтаграммной сети. Отправитель заметит утерю пакета без всяких уведомлений, после чего транспортный протокол замедлит работу. Таким образом, утерянный пакет несет ту же информацию, что и пакет уведомления, но неявно, без отправки маршрутизатором прямого сигнала.

RED-маршрутизаторы эффективнее тех, которые удаляют пакеты только при заполнении буфера, хотя и требуют правильной настройки. Например, оптимальное число пакетов на удаление зависит от числа отправителей, которых нужно оповестить о перегрузке. Однако по возможности лучше использовать прямые уведомления. Они работают так же, но передают сообщения в явном виде, а не косвенно через утерю пакета; RED используется в тех случаях, когда хосты не принимают такие уведомления.

### *Сдерживающие пакеты*

Самый простой способ уведомить отправителя о перегрузке — сообщить об этом прямо. При таком подходе маршрутизатор выбирает перегружающий пакет и отправляет источнику **сдерживающий пакет (choke packet)**. Данные об отправителе имеются в исходном пакете. Он помечается (специальным битом в его заголовке), чтобы больше не порождать сдерживающих пакетов на пути следования, и передается далее по своему обычному маршруту. Для предотвращения роста нагрузки на сеть при перегрузке маршрутизатор отправляет сдерживающие пакеты только на низкой скорости.

Когда хост-отправитель получает сдерживающий пакет, он должен уменьшить трафик к указанному получателю, к примеру, на 50 %. В дейтаграммной сети случайный выбор, скорее всего, приведет к тому, что сдерживающие пакеты дойдут до самых быстрых отправителей, поскольку именно их пакеты составляют большую часть очереди. Такая неявная обратная связь позволяет предотвратить перегрузку, не мешая тем источникам, действия которых не вызывают проблем. По той же причине есть вероятность, что многочисленные сдерживающие пакеты будут отправлены на нужный хост и адрес. В течение фиксированного интервала времени (пока уменьшение трафика не подействует) хост будет игнорировать дополнительные пакеты. Затем новые сдерживающие пакеты сообщат ему, что сеть все еще перегружена.

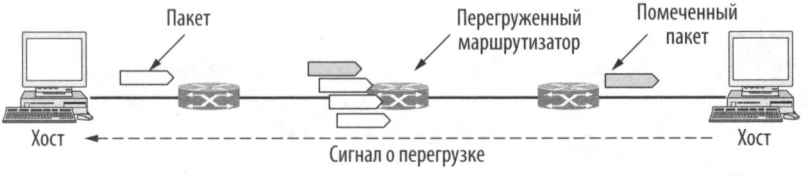
На ранних этапах развития интернета в качестве сдерживающего пакета использовалось сообщение SOURCE QUENCH (Постел; Postel, 1981). Оно не прижилось отчасти потому, что не были четко определены условия и результаты его рассылки. В современном интернете применяется другая схема уведомлений, о которой мы поговорим позже.

### *Явное уведомление о перегрузке*

Вместо создания дополнительных пакетов маршрутизатор может добавить специальную метку (например, 1 или 0 в заголовке) в любой передаваемый пакет, тем

самым сообщая о перегрузке. Когда пакет будет доставлен, получатель поймет, что сеть перегружена, и добавит эту информацию в ответный пакет. После этого отправитель сможет, как и раньше, регулировать свой трафик.

Этот метод называется **явным уведомлением о перегрузке (Explicit Congestion Notification, ECN)** и используется в интернете (Рамакришнан; Ramakrishnan, 1988). Это усовершенствованный вариант ранних протоколов подобного рода, в частности бинарной схемы обратной связи (Рамакришнан и Джейн; Ramakrishnan and Jain, 1988), применявшейся в архитектуре DECnet. На информацию о перегрузке в заголовке пакета отводится два бита. В момент отправки пакет не имеет метки (илл. 5.25). При проходе через перегруженный маршрутизатор пакет получает отметку о перегрузке. Затем получатель передает эти сведения отправителю, добавив явное уведомление в следующий ответный пакет. На рисунке этот процесс обозначен пунктирной линией, чтобы показать, что он происходит над IP-уровнем (например, на уровне TCP). Далее, как и в случае со сдерживающими пакетами, отправитель должен начать регулировать трафик.

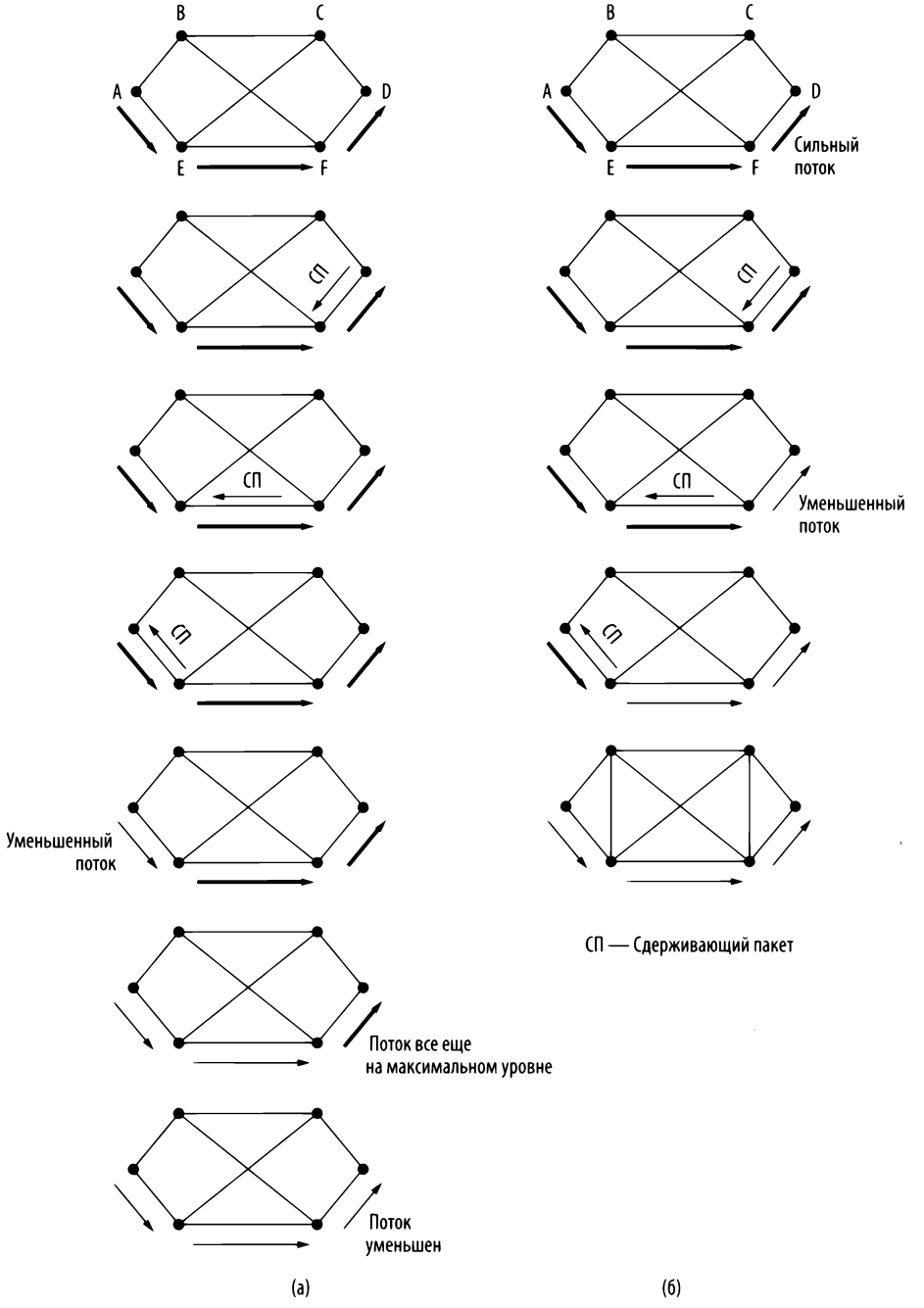



**Илл. 5.25.** Явное уведомление о перегрузке

### *Обратное давление на ретрансляционных участках*

При высоких скоростях или больших расстояниях до хостов множество новых пакетов может быть передано даже после отправки уведомлений о перегрузке, поскольку реакция на них занимает некоторое время. Рассмотрим, к примеру, хост в Сан-Франциско (маршрутизатор *A* на илл. 5.26), посылающий поток данных на хост, расположенный в Нью-Йорке (маршрутизатор *D* на илл. 5.26), со скоростью OC-3 155 Мбит/с. Если у нью-йоркского хоста станет заканчиваться буферная память, сдерживающему пакету потребуется около 40 мс на то, чтобы вернуться обратно в Сан-Франциско и сообщить, что необходимо снизить объем трафика. Явное уведомление о перегрузке займет еще больше времени, поскольку оно доставляется через получателя. На илл. 5.26 (а) распространение сдерживающего пакета схематично показано на второй, третьей и четвертой диаграммах. За те 40 мс, пока пакет движется по сети, в сторону нью-йоркского маршрутизатора передается еще 6,2 Мбит данных, с которыми надо как-то справиться. Только к седьмой диаграмме (илл. 5.26 (а)) маршрутизатор заметит замедление потока.

Однако есть альтернативный метод, позволяющий бороться с этой проблемой. Он заключается в том, что сдерживающий пакет влияет на трафик каждого маршрутизатора, через который он проходит. Это показано на последовательности диаграмм на илл. 5.26 (б). Как только сдерживающий пакет достигает




**Илл. 5.26.** (а) Сдерживающий пакет влияет только на источник. (б) Сдерживающий пакет влияет на все промежуточные участки

точки *F*, поток данных от *F* в сторону *D* должен снизиться. Таким образом, *F* резервирует для соединения большее количество буферной памяти: источник все еще продолжает заваливать это направление своими данными. Нагрузка на *D* мгновенно снижается, как головная боль — от таблетки в телевизионной рекламе. На следующем шаге сдерживающий пакет, продолжая свой путь, достигает *E* и приказывает уменьшить поток в сторону *F*. В результате точке *E* приходится выдерживать повышенную нагрузку, но зато *F* немедленно становится легче. Наконец, победный марш сдерживающего пакета приводит его к источнику всех бед — точке *A*, и теперь поток снижается по-настоящему.

Результат применения этой схемы — максимально быстрое устранение перегрузки на самом проблемном участке за счет использования большего объема буферной памяти промежуточных маршрутизаторов. Таким образом, проблема решается без потери пакетов. Эта идея обсуждается более подробно в работе Мишры и др. (Mishra et al., 1996).

## 5.4. QOS И QOE ПРИЛОЖЕНИЙ

Методы, представленные в предыдущих разделах, направлены на устранение перегрузок и повышение производительности сети. Однако некоторым приложениям (и клиентам) нужны более строгие гарантии качества, чем просто обязательство предоставить «лучшее из возможного в данных обстоятельствах» (иногда это называется подходом **best effort**). Кроме того, многие приложения требуют определенной минимальной пропускной способности и плохо функционируют, если задержка превышает некоторое пороговое значение. В этом разделе мы продолжим разговор о производительности сети с акцентом на методах обеспечения качества обслуживания, отвечающего требованиям конкретных приложений. Уже долгое время в данной области интернета происходят значительные изменения. В последнее время важную роль приобретает **QoE (Quality of Experience — качество пользовательского опыта)**, поскольку в конечном итоге важен лишь полученный пользователем опыт взаимодействия, а у разных приложений пороговые значения и требования к производительности сети могут сильно различаться. Все большее внимание уделяется способам оценки QoE, когда для наблюдения доступен только зашифрованный сетевой трафик.

### 5.4.1. Требования приложений к QoS

Последовательность пакетов, передающихся от источника к получателю, называется **потоком (flow)** (Кларк; Clark, 1988). При этом в сетях, ориентированных на установление соединения, его могут составлять все пакеты канала, а в сетях без установления соединения — все пакеты, отправленные от одного процесса к другому. Каждый поток требует определенных условий, которые можно охарактеризовать четырьмя основными параметрами: пропускная способность, задержка, джиттер и потери. Все вместе они формируют необходимый потоку **QoS (Quality of Service — качество обслуживания)**.

Некоторые часто используемые приложения и их требования к сети приведены в таблице на илл. 5.27. Следует отметить, что требования приложений более строгие, чем требования к сети, так как в некоторых случаях приложение может само улучшить уровень обслуживания, предоставленный сетью. В частности, для надежной передачи файлов сеть не обязана работать без потерь, а время задержки не должно быть одинаковым при передаче пакетов аудио и видео. Потери можно компенсировать за счет повторной передачи, а для сглаживания джиттера можно сохранять пакеты в буфере получателя. Но при слишком низкой пропускной способности или слишком большой задержке приложения бессильны.

| Приложение | Пропускная способность (bandwidth) | Задержка (delay) | Джиттер (jitter) | Потери (loss) |
|---|---|---|---|---|
| Электронная почта | Низкая | Низкая | Низкая | Средняя |
| Передача файлов | Высокая | Низкая | Низкая | Средняя |
| Веб-доступ | Средняя | Средняя | Низкая | Средняя |
| Удаленный доступ | Низкая | Средняя | Средняя | Средняя |
| Аудио по запросу | Низкая | Низкая | Высокая | Низкая |
| Видео по запросу | Высокая | Низкая | Высокая | Низкая |
| Телефония | Низкая | Высокая | Высокая | Низкая |
| Видеоконференции | Высокая | Высокая | Высокая | Низкая |

**Илл. 5.27.** Строгость требований приложений к QoS

У приложений могут быть разные требования к пропускной способности: для электронной почты, аудио в различных форматах и удаленного доступа они незначительны, тогда как передача файлов и видео в любых формах требует больших ресурсов.

С задержкой дело обстоит иначе. Приложения передачи файлов, включая электронную почту и видео, нечувствительны к задержкам. Даже если все пакеты будут доставляться с задержкой в несколько секунд, ничего страшного не произойдет. Интерактивные приложения — например, веб-поиска или удаленного доступа — более критичны к задержкам. Что касается приложений, работающих в реальном времени, их требования очень строги. Если при телефонном разговоре все слова собеседников будут приходить с большой задержкой, пользователи сочтут такую связь неприемлемой. С другой стороны, проигрывание видео- или аудиофайлов, хранящихся на сервере, допускает наличие некоторой задержки.

Колебание (то есть стандартное отклонение) времени задержки или времени прибытия пакета называется **джиттером (jitter)**. Первые три приложения на илл. 5.27 спокойно отнесутся к неравномерной задержке доставки пакетов. При организации удаленного доступа этот фактор имеет большее значение, поскольку при сильном джиттере обновления экрана будут происходить скачками.

Видео- и особенно аудиоданные крайне чувствительны к джиттеру. Если пользователь смотрит видео по сети и все фреймы приходят с задержкой ровно 2,000 с, проблем не возникнет. Но если время передачи колеблется от 1 до 2 с и приложению не удается скрыть джиттер, результат будет просто ужасен. При прослушивании звукозаписей заметен джиттер даже в несколько миллисекунд.

Первые четыре приложения на илл. 5.27 предъявляют высокие требования к потерям, так как для них (в отличие от аудио и видео) важен каждый бит информации. Обычно потерянные пакеты передаются транспортным уровнем повторно. Однако это неэффективно; было бы лучше, если бы сеть отвергала те пакеты, которые, скорее всего, будут утеряны. Для аудио- и видеоприложений потеря пакета не всегда требует повторной передачи: короткие паузы и пропущенные фреймы могут остаться незамеченными.

Чтобы удовлетворить требования различных приложений, сеть может предлагать разные категории QoS. Яркий пример — сети ATM, которые когда-то считались перспективными, но впоследствии стали нишевой технологией. Они поддерживают:

1. Постоянную битовую скорость (например, телефония).
2. Переменную битовую скорость в реальном времени (например, передача сжатых видеоданных во время видеоконференций).
3. Переменную битовую скорость не в реальном времени (например, просмотр фильмов по запросу).
4. Доступную битовую скорость (например, передача файлов).

Такое разбиение по категориям может пригодиться для иных целей и в других сетях. Постоянная битовая скорость — это попытка имитации проводной сети путем предоставления фиксированной пропускной способности и равномерной задержки. Битовая скорость может быть переменной, к примеру, при передаче сжатого видео, когда одни фреймы сжимаются в большей степени, чем другие. Для передачи видеокадра, содержащего множество деталей, может потребоваться много битов, тогда как видеокадр, запечатлевший белую стену, сожмется очень хорошо. Фильмы по запросу на самом деле проигрываются не в реальном времени: часто несколько секунд видеозаписи буферизуется на стороне получателя. Поэтому джиттер всего лишь приводит к колебаниям полученного объема видео, но не мешает его просмотру. Доступная битовая скорость подходит для электронной почты и других подобных приложений; они нечувствительны к задержкам и джиттеру и используют столько пропускной способности, сколько возможно.

## 5.4.2. Избыточное обеспечение

Существует простой способ предоставления QoS высокого уровня — создание сети с пропускной способностью, позволяющей передавать любой трафик. Этот метод называется **избыточным обеспечением (overprovisioning)**. Такая сеть сможет передавать трафик приложений без особых потерь, а при хорошей схеме маршрутизации пакеты будут доставляться с низкой задержкой. Этим

ограничиваются преимущества такого алгоритма с точки зрения производительности. В какой-то мере телефонная сеть является системой с избыточным обеспечением. Ситуация, когда абонент поднимает трубку и не слышит гудка, маловероятна. Дело в том, что в систему заложена настолько большая пропускная способность, что превысить ее тяжело.

Единственная проблема — такой способ обходится очень дорого. Конечно, ее можно решить путем крупных финансовых вливаний. Но с тем же успехом сеть с более низкой пропускной способностью может удовлетворить требования приложений при меньших затратах благодаря механизмам QoS. Более того, избыточное обеспечение основывается на данных об ожидаемом трафике. Ситуация кардинально меняется, если схема трафика слишком сильно отличается от предполагаемой. С помощью QoS сеть может выполнить свои обязательства даже при резком увеличении объема трафика за счет отклонения некоторых запросов.

Чтобы обеспечить QoS, необходимо ответить на следующие вопросы.

1. Что приложениям нужно от сети?
2. Как регулировать трафик, поступающий в сеть?
3. Как зарезервировать ресурсы на маршрутизаторах, необходимые для обеспечения производительности?
4. Может ли сеть принять больший объем трафика?

Ни один подход не в состоянии эффективно решить все эти проблемы. Поэтому для сетевого (и транспортного) уровня разработано множество различных методов. На практике для обеспечения QoS используются комбинации методов. Мы рассмотрим два варианта QoS для интернета: комплексное и дифференцированное обслуживание.

## 5.4.3. Планирование пакетов

Возможность регулировать форму предлагаемого трафика — это хорошее начало. Но чтобы предоставлять клиенту гарантии производительности, необходимо резервировать достаточное количество ресурсов. Будем считать, что все пакеты в потоке следуют по одному и тому же пути. При случайном распределении пакетов между маршрутизаторами сложно что-либо гарантировать. Следовательно, между источником и получателем должно быть установлено нечто вроде виртуального канала, и все пакеты данного потока должны следовать по указанному маршруту.

Механизмы, распределяющие ресурсы маршрутизаторов между пакетами одного потока или между конкурирующими потоками, называются **алгоритмами планирования пакетов (packet scheduling algorithm)**. Для разных потоков могут резервироваться три типа ресурсов:

1. Пропускная способность.
2. Буферное пространство.
3. Время центрального процессора (CPU).

Наиболее очевидным является резервирование пропускной способности. Если потоку необходима скорость в 1 Мбит/с, а исходящая линия может работать со скоростью 2 Мбит/с, то направить по ней три потока с такими параметрами не удастся. Таким образом, резервирование пропускной способности предотвращает предоставление канала слишком большему числу абонентов.

Второй дефицитный ресурс — буферное пространство. Когда приходит пакет, он хранится в буфере маршрутизатора, пока не будет передан по выбранной исходящей линии. В буфере временно содержатся небольшие пачки трафика, пока потоки конкурируют друг с другом. Если буферное пространство недоступно, входящий пакет приходится игнорировать, поскольку его просто негде хранить. Чтобы обеспечить хороший уровень QoS, можно резервировать некоторую часть буферной памяти под конкретный поток, чтобы ему не пришлось бороться за нее с другими потоками. В этом случае ему всегда будет предоставляться выделенная часть буфера (до определенного максимального значения).

Наконец, время CPU также может быть очень ценным ресурсом. Маршрутизатор тратит его на обработку пакета, а значит, скорость этого процесса ограничена. Современные маршрутизаторы в основном быстро справляются с этой задачей, но некоторые типы пакетов (в частности, ICMP, о которых мы поговорим в разделе 5.7.4) все же требуют более длительной работы CPU. Уверенность в том, что процессор не перегружен, — залог своевременной обработки пакетов.

### *Планирование пакетов по принципу FIFO*

Алгоритмы планирования пакетов распределяют пропускную способность и другие ресурсы маршрутизатора, решая, какой пакет из буфера отправить по исходящей линии следующим. Простейший вариант планировщика мы уже обсуждали, когда говорили о принципе работы маршрутизатора. Каждый маршрутизатор помещает пакеты в очередь (отдельную для каждой исходящей линии), где они ждут отправки. Дальнейшая передача пакетов происходит в том же порядке, в каком они пришли. Этот принцип называется **FIFO (First-In First-Out, первым пришел — первым ушел)** или **FCFS (First-Come First-Serve, первым пришел — первым обслуживается)**.

Маршрутизаторы, работающие по принципу FIFO, при переполнении очереди обычно удаляют пакеты, которые пришли последними. Новые пакеты помещаются в конец очереди, поэтому такой принцип работы называется **«обрубанием хвоста» (tail drop)**. Трудно представить альтернативу настолько простому и привычному методу. Однако алгоритм RED (см. раздел 5.3.2) при росте очереди отбрасывает прибывающие пакеты случайным образом. Другие алгоритмы планирования, которые мы обсудим, применяют самые разные принципы выбора пакетов для удаления.

### *Справедливое обслуживание*

Планирование по принципу FIFO легко реализовать, но оно не обеспечивает высокий уровень QoS: если потоков несколько, один из них может влиять на производительность других. Если поток ведет себя агрессивно и отправляет

большие объемы трафика, его пакеты попадают в очередь. При обработке в порядке поступления этот отправитель захватывает большую часть мощности маршрутизаторов, отбирая ее у других потоков и тем самым уменьшая их уровень QoS. Ситуация усугубляется тем, что пакеты других потоков, прошедшие через маршрутизатор, скорее всего, придут с опозданием, поскольку они задержались в очереди, ожидая отправки пакетов агрессивного потока.

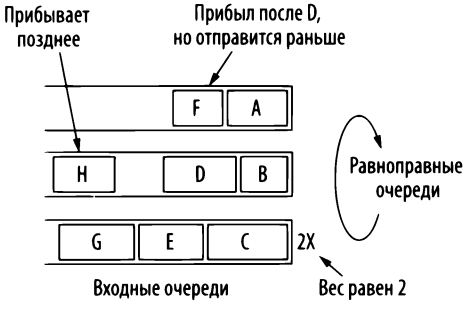
Чтобы обеспечить изоляцию потоков и предотвратить их конфликты, было разработано множество различных алгоритмов планирования пакетов (Бхатти и Кроукрофт; Bhatti and Crowcroft, 2000). Одним из первых был алгоритм **равноправных очередей (fair queueing)** (Нейгл; Nagle, 1987). Маршрутизаторы организуют отдельные очереди для каждой исходящей линии, по одной для каждого потока. Как только линия освобождается, маршрутизатор циклически сканирует очереди (илл. 5.28) и выбирает первый пакет следующей очереди. Таким образом, если за данную исходящую линию конкурируют $n$ хостов, то каждый из них отправляет один пакет из каждых $n$ пакетов. Получается, что все потоки передают пакеты с одинаковой скоростью и агрессивному хосту не поможет отправка большего числа пакетов.

**Илл. 5.28.** Циклическая обработка равноправных очередей

Однако у этого метода есть один недостаток. Предоставляемая им пропускная способность напрямую зависит от размера пакетов, отправляемых хостом: чем они крупнее, тем больше ресурса ему выделяется. В книге Демерса и др. (Demers et al., 1990) предлагается улучшенная версия алгоритма, в которой циклический опрос производится с целью выхватывания байта, а не пакета. Идея заключается в вычислении виртуального времени, то есть номера цикла, на котором отправка пакета завершится. Каждый цикл выхватывает один байт из всех очередей, содержащих пакеты для передачи. Затем пакеты сортируются согласно времени завершения отправки и передаются в этой последовательности.

На илл. 5.29 показана работа алгоритма и примеры значений времени окончания отправки пакетов для трех потоков. Если длина пакета равна $L$, его отправка завершится через $L$ циклов. Передача пакета начинается либо сразу после отправки предыдущего, либо в момент прибытия этого пакета, если очередь пуста.

Таблица на илл. 5.29 (б) показывает, что первые два пакета в двух верхних очередях прибывают в порядке $A$, $B$, $D$, $F$. Пакет $A$ приходит на нулевом цикле, а его длина равна 8 байт, поэтому отправка завершается на 8-м цикле. Точно так же отправка пакета $B$ завершается на цикле 11. Пакет $D$ прибывает в момент отправки $B$. Его передача заканчивается через 9 циклов после окончания




| Пакет | Время прибытия | Длина | Время окончания отправки | Порядок отправки |
|---|---|---|---|---|
| A | 0 | 8 | 8 | 1 |
| B | 5 | 6 | 11 | 3 |
| C | 5 | 10 | 10 | 2 |
| D | 8 | 9 | 20 | 7 |
| E | 8 | 8 | 14 | 4 |
| F | 10 | 6 | 16 | 5 |
| G | 11 | 10 | 19 | 6 |
| H | 20 | 8 | 28 | 8 |

(б)

**Илл. 5.29.** (а) WFQ. (б) Время окончания отправки для пакетов

отправки *B*; время равно 20. Аналогичным образом время завершения отправки *F* равно 16. Если новых пакетов нет, порядок окончания отправки пакетов будет таким: *A*, *B*, *F*, *D* (хотя *F* прибывает после *D*). Может случиться так, что в верхний поток поступит небольшой пакет, который будет передан раньше *D*. Но он обойдет *D* только в том случае, если отправка *D* еще не началась. При использовании равноправных очередей передача пакета не прерывается. Алгоритм передает пакеты целиком и потому является лишь приближением идеальной схемы побайтовой передачи. Но это достаточно хорошее приближение: в каждый момент времени передается ровно один пакет.

### *Взвешенные равноправные очереди*

Проблема данного алгоритма в том, что он дает всем хостам одинаковые приоритеты. Как правило, видеосерверам лучше предоставлять большую пропускную способность, чем обычным файл-серверам, чтобы они могли передавать два или несколько байтов за цикл. Такая модификация алгоритма называется **взвешенными равноправными очередями (Weighted Fair Queueing, WFQ)**. Если количество байтов, передаваемое за цикл, считать весом потока *W*, то формула времени окончания передачи выглядит так:

$$F_i = \max(A_i, F_{i-1}) + L_i/W,$$

где $A_i$ — время прибытия, $F_i$ — время окончания отправки, а $L_i$ — длина $i$-го пакета. Нижняя очередь (илл. 5.29 (а)) имеет вес 2, поэтому ее пакеты передаются быстрее. Это хорошо видно, если посмотреть на значения времени окончания отправки на илл. 5.29 (б).

Еще один важный фактор — сложность реализации. WFQ помещает пакеты в очередь, сортируя их по времени окончания отправки. Для *N* потоков это требует по меньшей мере $O(\log N)$ операций для каждого пакета, а это трудновыполнимо при большом количестве потоков на высокоскоростных маршрутизаторах. Упрощенная схема **дефицитного циклического обслуживания (Deficit Round Robin, DRR)** работает гораздо эффективнее: всего за $O(1)$ операций для каждого пакета (Шридхар и Варгезе; Shreedhar and Varghese, 1995). Она широко применяется для алгоритма WFQ.

Существуют другие алгоритмы планирования пакетов, например приоритетный, при котором пакеты обладают каким-либо приоритетом. Высокоприоритетные пакеты передаются раньше, чем низкоприоритетные; последние помещаются в буфер. Пакеты с одинаковым приоритетом отправляются по принципу FIFO. Важным недостатком этого алгоритма является то, что высокоприоритетные пакеты препятствуют отправке низкоприоритетных, которые могут ждать своей очереди бесконечно долго. С этой точки зрения более удачным вариантом является WFQ. Если присвоить высокоприоритетной очереди большой вес (скажем, 3), пакеты в ней будут в основном проходить по быстрой линии (поскольку их сравнительно немного). При этом часть низкоприоритетных пакетов тоже будет передаваться. Фактически бинарная приоритетная система представляет собой две очереди, одна из которых имеет бесконечный вес.

Наконец, существует алгоритм планирования, при котором у каждого пакета есть временной штамп, определяющий порядок отправки. В реализации, которую предложили Кларк и др. (Clark et al., 1992), временной штамп регистрирует информацию о том, насколько пакет, проходя через маршрутизаторы сети, отстает от графика или опережает его. Как правило, пакеты, ожидающие отправки, отстают, а передаваемые в первую очередь — опережают график. Использование временных штампов — эффективный способ ускорить отправку медленных пакетов и замедлить передачу быстрых. При таком подходе все пакеты доставляются с приблизительно равной задержкой, что является очевидным преимуществом.

### *Собираем все воедино*

Теперь, когда мы познакомились с основными компонентами QoS, самое время объединить их, чтобы реально его обеспечить. Гарантии QoS предоставляются через управление допуском. Мы рассматривали его как средство борьбы с перегрузкой, что является гарантией производительности, пусть даже не очень эффективной. Здесь мы предъявляем к гарантиям более серьезные требования, но модель остается той же. Пользователь передает в сеть поток, предъявляя определенные требования QoS. Сеть принимает или отвергает этот поток в зависимости от своих возможностей и обязательств перед другими клиентами. Если поток принимается, сеть должна заранее зарезервировать ресурсы, чтобы при передаче по нему трафика клиент получил необходимый уровень QoS.

Необходимо зарезервировать ресурсы на всех маршрутизаторах, расположенных в узлах пути, по которому следуют пакеты. Иначе может возникнуть коллапс, и гарантии QoS не будут выполнены. Многие алгоритмы маршрутизации выбирают наилучший путь от отправителя до получателя и направляют по нему весь трафик. Однако некоторые потоки могут быть отвергнуты из-за недостатка ресурсов в узлах наилучшего маршрута. Тогда, чтобы выполнить свои обязательства, сеть выберет другой путь для отправки крупного потока. Этот подход называется **QoS-маршрутизацией (QoS routing)**; см. работу Чэня и Нарштедт (Chen and Nahrstedt, 1998). Также можно разделить трафик для одного адресата по нескольким маршрутам, чтобы было проще найти дополнительные ресурсы.

Маршрутизаторы могут выбирать пути с одинаковой стоимостью и использовать маршрутизацию, пропорциональную или эквивалентную емкостям исходящих связей. Однако существуют и более сложные алгоритмы (Нелакудити и Чжан; Nelakuditi и Zhang, 2002).

С учетом выбранного пути решение о принятии или отклонении потока не сводится к сравнению запрашиваемых ресурсов (пропускной способности, буферной памяти, времени CPU) с возможностями маршрутизатора. Во-первых, несмотря на то что многие приложения знают свои требования по пропускной способности, два других параметра им неизвестны. Следовательно, как минимум необходим иной способ описания потоков и определения ресурсов, выделяемых маршрутизатором. Вскоре мы это обсудим.

Также отметим, что приложения значительно различаются по толерантности в отношении пропущенного предельного срока обработки. Поэтому приложение должно выбрать один из типов гарантий, предлагаемых сетью: от строгих до максимально лояльных. При прочих равных условиях наиболее популярными были бы строгие гарантии. Но проблема в том, что они затратные, так как ограничивают поведение сети в наихудшем случае. Для приложения обычно достаточно тех же гарантий, что и для большинства пакетов; кроме того, они позволяют добавить дополнительный поток при фиксированной емкости.

Наконец, некоторые приложения могут поторговаться за параметры пакетов, а некоторые нет. Скажем, проигрыватель видео, обычно предоставляющий 30 фреймов/с, может работать на 25 фреймах/с, если для 30 не хватает пропускной способности. Аналогично можно настраивать количество пикселей на фрейм, полосу пропускания для аудиоданных и другие свойства потоков различных приложений.

Поскольку в споре о том, что делать с потоком, участвует много сторон (отправитель, получатель и все маршрутизаторы на пути между ними), поток необходимо описывать крайне аккуратно с помощью параметров, о которых можно договориться. Набор таких параметров называется **спецификацией потока (flow specification)**. Как правило, отправитель (например, сервер видеоданных) создает спецификацию, указывая нужные ему параметры. По мере движения спецификации по пути потока маршрутизаторы анализируют ее и меняют параметры, если нужно. Эти изменения могут быть направлены только на уменьшение потока (например, скорость передачи данных может быть понижена, но не повышена). Когда спецификация доходит до получателя, устанавливаются окончательные параметры.

В качестве содержимого спецификации потока рассмотрим пример на илл. 5.30. Он основан на RFC 2210 и RFC 2211 для комплексного обслуживания — технологии QoS, о которой мы поговорим в следующем разделе. В спецификации содержится пять параметров. Первые два, *скорость маркерного ведра* и *размер маркерного ведра*, позволяют вычислить максимальную скорость, которую источник может поддерживать в течение долгого времени, усредненную по большому временному отрезку, а также максимальный объем пачки, передаваемой за короткий промежуток времени.

| Параметр | Единицы измерения |
|---|---|
| Скорость маркерного ведра | байт/с |
| Размер маркерного ведра | байт |
| Пиковая скорость передачи данных | байт/с |
| Минимальный размер пакета | байт |
| Максимальный размер пакета | байт |

**Илл. 5.30.** Пример спецификации потока

Третий параметр, *пиковая скорость передачи данных*, — это максимальная допустимая скорость (даже для коротких временных интервалов). Отправитель ни в коем случае не должен превышать это значение.

Наконец, последние два параметра определяют минимальный и максимальный размеры пакетов, включая заголовки транспортного и сетевого уровней (например, TCP и IP). Минимальный размер удобен, поскольку обработка каждого пакета занимает фиксированное время, пусть даже небольшое. Возможно, маршрутизатор готов принимать 10 000 килобайтных пакетов в секунду, но не готов обрабатывать 100 000 50-байтных пакетов в секунду, хотя во втором случае скорость передачи меньше. Максимальный размер пакета не менее важен, но уже по другой причине. Дело в том, что существуют определенные внутрисетевые ограничения, которые нельзя превышать. Например, если путь потока лежит через Ethernet, то максимальный размер пакета будет ограничен 1500 байтами, независимо от того, пакеты какого размера поддерживаются другими участками сети.

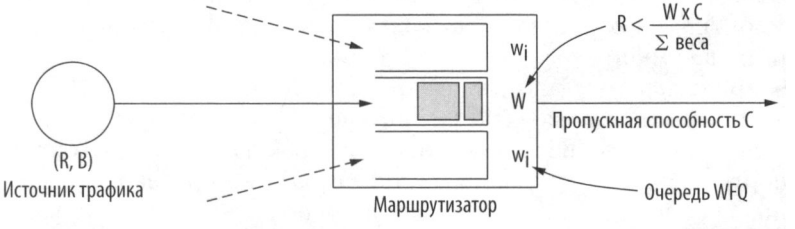
Каким образом маршрутизатор преобразует спецификацию потока в набор определенных резервируемых ресурсов? На первый взгляд может показаться, что если один из каналов маршрутизатора работает со скоростью 1 Гбит/с, а средний размер пакета равен 1000 бит, он может обрабатывать 1 млн пакетов в секунду. Но это не так, поскольку из-за статистического джиттера нагрузки передача будет периодически останавливаться на некоторое время. Если для выполнения всей работы канал должен использовать всю емкость, перерыв длиной в несколько секунд станет причиной завала, который невозможно разгрести.

Однако даже если нагрузка несколько меньше теоретической емкости, все равно могут образовываться очереди и возникать задержки. Рассмотрим ситуацию, в которой пакеты приходят нерегулярно со средней скоростью $\lambda$ пакетов/с. Они имеют случайную длину и могут передаваться по каналу со средней скоростью обслуживания $\mu$ пакетов/с. Предположим, что обе скорости имеют пуассоновское распределение (такие системы называются системами массового обслуживания M/M/1, где «M» означает «марковский процесс», который в данном случае является еще и пуассоновским). Тогда, используя теорию массового обслуживания, можно доказать, что средняя задержка $T$, присущая пакету, равна

$$T = \frac{1}{\mu} \times \frac{1}{1 - \lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1 - \rho},$$

где $\rho = \lambda/\mu$ — коэффициент использования CPU. Первый сомножитель $1/\mu$ — это задержка при отсутствии конкуренции. Второй — дополнительная задержка, возникающая из-за конкуренции с другими потоками. Например, если $\lambda = 950\,000$ пакетов/с, а $\mu = 1\,000\,000$ пакетов/с, тогда $\rho = 0{,}95$ и средняя задержка каждого пакета составляет 20 мкс вместо 1 мкс. Эти подсчеты учитывают и задержку доставки, и задержку обработки: при малом трафике отношение $\lambda/\mu \approx 0$. Если на пути потока стоят, скажем, 30 маршрутизаторов, то одна только задержка обслуживания составит 600 мкс.

Один из методов соотнесения характеристик потока и ресурсов маршрутизатора, необходимых для выполнения гарантий пропускной способности и времени задержки, был предложен Парехом и Галлагером (Parekh and Gallager, 1993; 1994). Отправитель формирует трафик с помощью маркерных ведер ($R$, $B$), а маршрутизаторы используют WFQ. Каждому потоку присваивается вес $W$, достаточный для того, чтобы опустошить маркерное ведро со скоростью $R$ (илл. 5.31). Если, например, скорость потока составляет 1 Мбит/с, а мощности маршрутизатора и исходящей связи равны 1 Гбит/с, вес потока должен превышать одну тысячную от общей суммы весов всех потоков этого маршрутизатора для исходящей связи. Это обеспечит потоку минимальную пропускную способность. Если поток не может получить необходимую ему скорость, он не будет допущен в сеть.




**Илл. 5.31.** Гарантии пропускной способности и задержки с использованием маркерных ведер и WFQ

Самая большая задержка в очереди для данного потока — это функция максимальной емкости маркерного ведра. Рассмотрим два крайних случая. При равномерном трафике пакеты проходят через маршрутизатор с той же скоростью, с какой прибывают. При этом не происходит никаких задержек (не считая эффектов пакетирования). С другой стороны, если трафик передается пачками, то пачка максимального размера $B$ может прийти на маршрутизатор полностью. Тогда максимальная задержка $D$ будет равна времени прохождения пакета через маршрутизатор при фиксированной пропускной способности, или $B/R$ (опять же, не считая эффектов пакетирования). Если этот показатель слишком высокий, поток может запросить бóльшую пропускную способность.

Это достаточно строгие гарантии: маркерные ведра ограничивают неравномерность трафика, а справедливое обслуживание изолирует пропускную способность, выделяемую для разных потоков. Это значит, что гарантии пропускной способности и задержки для потока будут выполнены, даже если другие потоки будут копить трафик и отправлять его одновременно.

Более того, результат не зависит от количества маршрутизаторов в узлах пути и от топологии сети. Каждому потоку предоставляется минимальная пропускная способность благодаря тому, что она зарезервирована на каждом маршрутизаторе. Причина, по которой каждый поток получает максимальную задержку, менее явная. В наихудшем случае, если крупный объем трафика поступит на первый маршрутизатор и будет соревноваться с трафиком других потоков, максимальная задержка будет равна *D*. Однако после этого трафик станет более равномерным, и поэтому на следующих маршрутизаторах такой задержки уже не будет. В результате общая задержка в очереди не будет превышать *D*.

## 5.4.4. Комплексное обслуживание

В 1995–1997 годах IETF приложила множество усилий по продвижению архитектуры потокового мультимедиа. В результате появилось две дюжины документов RFC: от RFC 2205 до RFC 2212. Общее название этих трудов — **комплексное обслуживание (integrated services)**. Эта технология предназначена как для одноадресных, так и для многоадресных приложений. В первом случае это может быть просмотр потокового видео на новостном сайте одним пользователем; во втором — набор станций цифрового телевидения, транслирующих свои программы в виде потоков IP-пакетов. Данной услугой может пользоваться большое число абонентов в разных географических точках. Далее мы подробнее рассмотрим многоадресную рассылку, поскольку одноадресная передача — это лишь частный случай многоадресной.

Во многих приложениях с многоадресной маршрутизацией группы пользователей могут динамически меняться. Например, люди участвуют в видеоконференции, потом им становится скучно, и они переключаются на мыльную оперу или спортивный канал. В данном случае стратегия предварительного резервирования пропускной способности не совсем подходит, потому что каждому источнику пришлось бы запоминать все изменения в составе аудитории. В системах, предназначенных для передачи телевизионного сигнала миллионам абонентов, этот метод вообще не сработает.

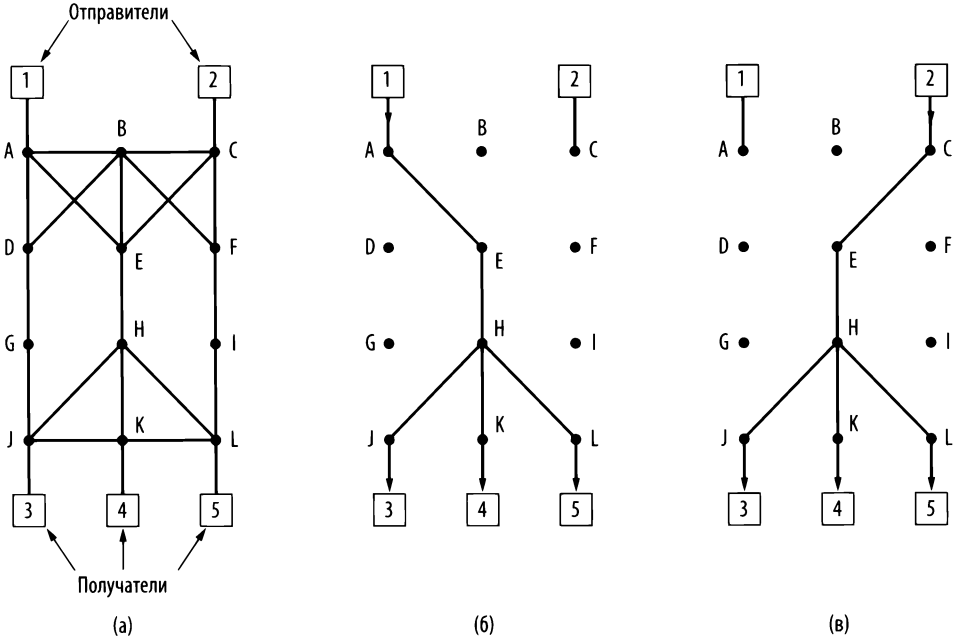
### *RSVP — протокол резервирования ресурсов*

Главный компонент архитектуры комплексного обслуживания, открытый для пользователей сети, — **протокол резервирования ресурсов (Resource reSerVation Protocol, RSVP)**. Он описывается в стандартах RFC 2205–RFC 2210. Как следует из названия, протокол предназначен для резервирования ресурсов; другие протоколы применяются для описания передачи данных. RSVP позволяет нескольким отправителям отправлять данные нескольким группам абонентов, разрешает отдельным получателям переключать каналы и оптимизирует использование пропускной способности, в то же время устраняя перегрузки.

Самый простой вариант этого протокола использует упомянутую ранее многоадресную маршрутизацию с применением связующих деревьев. Каждая группа получает адрес. Чтобы передать ей данные, отправитель помещает этот адрес в заголовки пакетов. Затем стандартный алгоритм многоадресной

маршрутизации строит связующее дерево, охватывающее всех членов группы. Этот алгоритм не является частью протокола RSVP. Единственное отличие от обычной многоадресной маршрутизации состоит в том, что группе периодически рассылается дополнительная информация, с помощью которой маршрутизаторы обновляют в памяти определенные структуры данных.

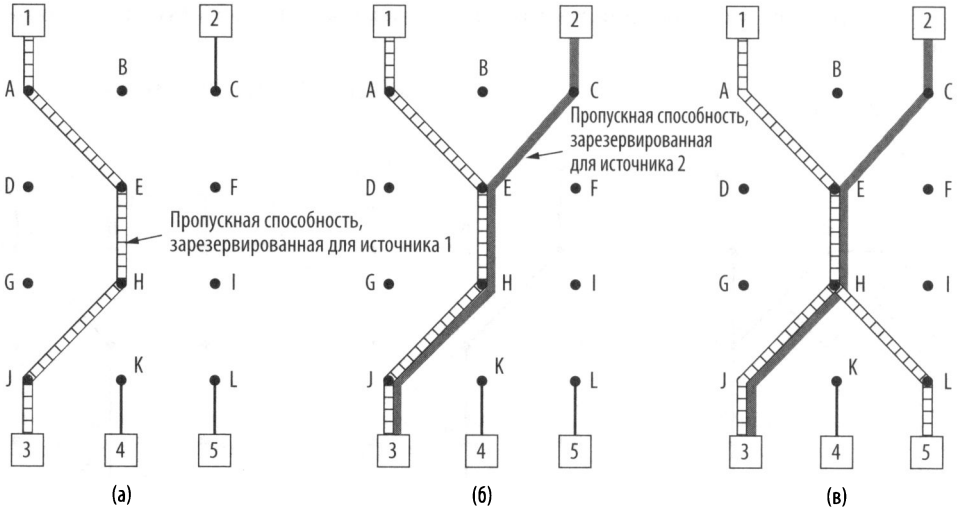
Рассмотрим сеть на илл. 5.32 (а). Хосты 1 и 2 — многоадресные отправители, а хосты 3, 4 и 5 — многоадресные получатели. В данном примере они разделены, однако в большинстве случаев эти два множества могут перекрываться. Деревья многоадресной рассылки для хостов 1 и 2 показаны на илл. 5.32 (б) и (в) соответственно.




**Илл. 5.32.** Протокол резервирования ресурсов (а) Сеть. (б) Связующее дерево многоадресной рассылки для хоста 1. (в) Связующее дерево многоадресной рассылки для хоста 2

Для улучшения качества приема и устранения перегрузки каждый получатель в группе может отправить источнику (вверх по дереву) запрос на резервирование. Это сообщение продвигается с использованием переадресации в обратном направлении, которую мы обсуждали ранее. На каждом транзитном участке маршрутизатор замечает запрос и резервирует необходимую пропускную способность. В предыдущем разделе мы видели, как это делает планировщик WFQ. Если пропускной способности недостаточно, маршрутизатор информирует об ошибке. К тому моменту, как запрос приходит обратно к источнику, пропускная способность зарезервирована на пути от отправителя к получателю по связующему дереву.

Пример резервирования показан на илл. 5.33 (а). Здесь хост 3 запросил канал к хосту 1. После создания канала поток пакетов от хоста 1 к хосту 3 может идти без перегрузок. Рассмотрим, что произойдет, если теперь хост 3 зарезервирует канал к другому отправителю, хосту 2, чтобы пользователь мог одновременно смотреть две телевизионные программы. Резервирование второго канала показано на илл. 5.33 (б). Обратите внимание: между хостом 3 и маршрутизатором *E* должно быть два отдельных канала, так как передаются два независимых потока.




**Илл. 5.33.** Пример резервирования. (а) Хост 3 запрашивает канал к хосту 1. (б) Затем хост 3 запрашивает второй канал к хосту 2. (в) Хост 5 запрашивает канал к хосту 1

Наконец, на илл. 5.33 (в) хост 5 решает посмотреть программу, передаваемую хостом 1, и также резервирует себе канал. Сначала требуемая пропускная способность резервируется до маршрутизатора *H*. Затем он замечает, что у него уже есть канал от хоста 1, поэтому дополнительное резервирование выше по дереву не требуется. Обратите внимание, что хосты 3 и 5 могут запросить разную пропускную способность (например, у хоста 3 маленький экран, поэтому ему не нужно высокое разрешение). Следовательно, *H* должен зарезервировать пропускную способность, достаточную для получателя с самым большим запросом.

При подаче запроса получатель может (по желанию) указать один или несколько источников, от которых он хотел бы получать сигнал. Он также может сообщить, является ли выбор источников фиксированным в течение времени резервирования или он оставляет за собой право сменить их. Эти сведения используются маршрутизаторами для оптимизации планирования пропускной способности. К примеру, двум получателям выделяется общий путь, только если они соглашаются не менять впоследствии свой источник.

В основе такой динамической стратегии лежит независимость зарезервированной пропускной способности от выбора источника. Зарезервировав полосу,

получатель может переключаться с одного источника на другой, сохраняя часть существующего пути, которая подходит для нового источника. Например, если хост 2 передает несколько видеопотоков в реальном времени (например, при многоканальном телевещании), хост 3 может переключаться между ними по желанию, не меняя своих параметров резервирования: маршрутизаторам все равно, какую программу смотрит получатель.

## 5.4.5. Дифференцированное обслуживание

Потоковые алгоритмы способны обеспечивать хороший уровень QoS одного или нескольких потоков за счет резервирования любых необходимых ресурсов на протяжении всего маршрута. Однако есть у них и недостаток. Они требуют предварительной настройки при установке каждого потока, что не подходит для систем с тысячами или миллионами потоков. Кроме того, потоковые алгоритмы используют внутреннюю информацию о потоках, которая хранится в маршрутизаторах. Это делает их уязвимыми к выходу маршрутизаторов из строя. Наконец, они требуют значительных программных изменений в маршрутизаторах, связанных со сложными процессами обмена между ними при установке потоков. В результате с развитием комплексного обслуживания подобные алгоритмы используются крайне редко.

По этим причинам IETF разработал упрощенный подход к QoS. Его можно реализовать локально в каждом маршрутизаторе без предварительной настройки и без участия остальных устройств маршрута. Подход известен как **основанное на классах** (в отличие от основанного на потоках) **QoS**. IETF стандартизировал специальную архитектуру под названием **дифференцированное обслуживание (differentiated services)** и описал в документах RFC 2474, RFC 2475 и во многих других.

Дифференцированное обслуживание может предоставляться набором маршрутизаторов, которые образуют административный домен (например, интернет-провайдер или телефонную компанию). Администрация определяет набор классов обслуживания и соответствующие правила маршрутизации. Пакеты от абонента, пользующегося дифференцированным обслуживанием, получают метку с информацией о классе. Эти сведения записываются в поле `Differentiated services` пакетов IPv4 и IPv6 (см. раздел 5.7.1). Классы определяются как **поведение при переходах (per hop behaviors)**, так как они отвечают за то, что будет происходить с пакетом на маршрутизаторе, а не во всей сети. Таким пакетам предоставляется улучшенное обслуживание (например, премиум-обслуживание) по сравнению с остальными пакетами (обычное обслуживание). Может потребоваться, чтобы трафик внутри класса соответствовал определенной форме (например, дырявому ведру с заданной скоростью «вытекания» данных). Оператор с хорошим бизнес-чутьем может брать дополнительную плату за передачу каждого премиум-пакета либо установить абонентскую плату за передачу $N$ таких пакетов в месяц. Обратите внимание: здесь не требуется никакой предварительной настройки, резервирования ресурсов и трудоемких согласований параметров для каждого потока, как при

комплексном обслуживании. Это делает дифференцированное обслуживание относительно простым в реализации.

Система классов обслуживания встречается и в других сферах. Например, службы доставки посылок могут предлагать несколько уровней обслуживания на выбор: доставка на следующий день, через день или через два дня. Авиакомпании предлагают первый класс, бизнес-класс и эконом. То же самое касается поездов дальнего следования. В парижском метро одно время даже использовались два класса обслуживания при одинаковом качестве сидячих мест. Что касается нашей темы, то классы пакетов могут отличаться друг от друга задержкой, джиттером, вероятностью отклонения в случае коллизии, а также другими параметрами (которых, впрочем, не больше, чем у фреймов Ethernet).

Чтобы разница между QoS на основе классов и QoS на основе потоков стала яснее, рассмотрим пример: интернет-телефонию. В потоковой схеме каждому телефонному соединению предоставляются собственные ресурсы и гарантии, в классовой — все соединения совместно получают ресурсы, зарезервированные для данного класса. С одной стороны, эти ресурсы не может отнять никто извне (потоки систем веб-просмотра и прочие соединения других классов), с другой стороны, ни одно телефонное соединение не может получить частные ресурсы, зарезервированные только для него.

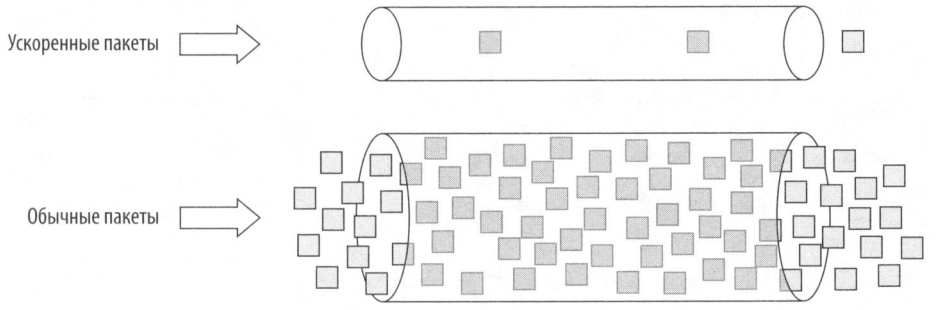
### *Беспрепятственная пересылка*

Выбор класса обслуживания зависит от оператора, но поскольку пакеты зачастую необходимо пересылать между сетями разных операторов, IETF установил классы, не зависящие от сети. Простейший из них — класс **беспрепятственной пересылки (expedited forwarding)**, описанный в стандарте RFC 3246. С него и начнем.

Идея, на которой основана беспрепятственная пересылка, очень проста. Существует два класса обслуживания: обычный и ускоренный. Ожидается, что подавляющая часть трафика будет использовать обычный класс. Но есть ограниченная доля пакетов, которые необходимо передавать в ускоренном режиме. Их нужно пересылать так, будто кроме них в сети больше нет никаких пакетов. Тогда они получат обслуживание с низкими потерями, низкой задержкой и низким джиттером — как раз то, что нужно для IP-телефонии. Графическое представление такой двухканальной системы показано на илл. 5.34. Имейте в виду, что физическая линия здесь только одна. Два логических пути на рисунке обозначают резервирование пропускной способности для разных классов (а вовсе не два физических провода).

Данную стратегию можно реализовать следующим образом. Пакеты разделяются на обычные и ускоренные, после чего они получают соответствующие отметки. Это может выполнять хост-источник или входной (первый) маршрутизатор. Преимущество первого варианта в том, что источник располагает большей информацией о распределении пакетов по потокам. Классификация пакетов может производиться сетевым ПО или операционной системой, что позволяет избежать изменений в существующих приложениях. Например, сейчас VoIP-пакеты все чаще помечаются хостами как ускоренные. Если они передаются по

корпоративной сети или через провайдера, поддерживающего беспрепятственную пересылку, им будет предоставлено приоритетное обслуживание. Иначе отметка не будет иметь никаких негативных последствий. Таким образом, имеет смысл поставить ее на всякий случай.




**Илл. 5.34.** Ускоренные пакеты движутся по свободной от трафика сети

Разумеется, если пакет получает метку на хосте, входной маршрутизатор, скорее всего, проверит, не выходит ли объем срочного трафика за установленные пределы. В сети маршрутизаторы могут использовать две очереди для каждой исходящей линии — для обычных и для ускоренных пакетов. Прибывший пакет ставится в очередь, соответствующую его классу обслуживания. Беспрепятственная очередь получает более высокий приоритет (к примеру, с помощью планировщика приоритетов). Таким образом, для ускоренного трафика сеть кажется свободной, хотя на самом деле она может быть сильно загружена обычным трафиком.

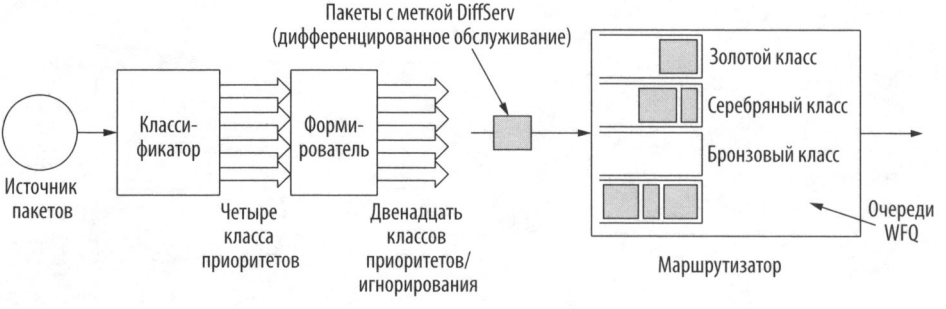
### *Гарантированная пересылка*

Более совершенная схема управления классами обслуживания — **гарантированная пересылка (assured forwarding)**, описанная в документе RFC 2597. Она подразумевает наличие четырех классов приоритетов, каждый из которых обладает своими ресурсами. Первые три класса можно назвать золотым, серебряным и бронзовым. Кроме того, определены три класса игнорирования пакетов при перегрузке (низкий, средний и высокий). В итоге получается 12 сочетаний, то есть 12 классов обслуживания.

На илл. 5.35 показан один из способов обработки пакетов при гарантированной пересылке. На первом этапе пакеты разбиваются на четыре класса приоритетов. Эта процедура также может выполняться как на хосте-источнике (как показано на рисунке), так и на первом маршрутизаторе. Скорость высокоприоритетных пакетов может быть ограничена оператором в рамках соглашения о предоставлении услуг.

Следующий шаг — определение классов игнорирования пакетов. Для этого пакеты каждого класса приоритетов проходят проверку с помощью маркерного ведра или похожей схемы. Небольшим пакетам присваивается низкий класс

игнорирования, средним — средний класс, а большим — высокий. Информация о классах приоритетов и игнорирования кодируется в каждом пакете.




**Илл. 5.35.** Возможная реализация гарантированной пересылки

Наконец, пакеты проходят обработку на маршрутизаторах сети, где планировщик определяет их классы. Чаще всего для четырех классов приоритетов используется WFQ: чем выше класс, тем выше вес. В результате высокоприоритетные пакеты получают большую часть пропускной способности, при этом передача низкоприоритетных пакетов не останавливается. К примеру, вес каждого класса приоритетов может быть вдвое больше, чем вес более низкого класса. В пределах одного класса приоритетов пакеты с высоким классом игнорирования можно удалять в первую очередь, например, с помощью алгоритма RED. Он начнет исключать пакеты еще до того, как в буфере маршрутизатора закончится место. Пакеты с низким классом игнорирования все еще будут приниматься, а с высоким — отвергаться.

## 5.5. МЕЖСЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ

До сих пор мы подразумевали, что существует единая однородная сеть, в которой каждое устройство использует одни и те же протоколы на каждом уровне. К сожалению, это предположение слишком оптимистично. Существует множество различных сетей, включая PAN, LAN, MAN и WAN. Мы уже говорили о сети Ethernet, кабельном интернете, стационарных и мобильных телефонных сетях, сетях стандарта 802.11 и т. д. На каждом уровне этих сетей широко применяются многочисленные протоколы.

### 5.5.1. Интерсети: общие сведения

В следующих разделах особое внимание будет уделено вопросам, возникающим при объединении двух или более сетей, формирующих **объединенную сеть (internetwork)**, или, проще, **интерсеть (internet)**.

Если бы все использовали одну сетевую технологию, объединять сети было бы намного проще. В большинстве случаев существует доминирующая сеть (например, Ethernet). Некоторые ученые считают разнообразие сетевых технологий

временным явлением, которое исчезнет, как только все наконец поймут, как замечательна [вставьте свою любимую сеть]. Однако на это рассчитывать не стоит: история показывает, что такие рассуждения — всего лишь принятие желаемого за действительное. Различные сети решают разные задачи, поэтому, к примеру, Ethernet и спутниковые сети всегда будут отличаться. Создание сетей передачи данных на основе уже существующих систем (кабельной, телефонной или ЛЭП) порождает дополнительные ограничения, которые приводят к расхождению их характеристик. Поэтому нам всегда придется иметь дело с неоднородностью сетей.

Было бы намного проще, если бы нам вообще не пришлось объединять разнообразные сети. Но это также крайне маловероятно. Боб Меткалф выдвинул такой принцип: ценность сети, состоящей из $N$ узлов, пропорциональна числу соединений между узлами, или $N^2$ (Гилдер; Gilder, 1993). Это означает, что чем крупнее сеть, тем выше ее ценность, поскольку она обеспечивает гораздо больше соединений. Поэтому объединение небольших сетей всегда будет иметь смысл.

Главный пример такого объединения — интернет. Цель объединения всех этих сетей — предоставить пользователям из разных сетей возможность общаться. Стоимость услуг провайдера часто зависит от доступной пропускной способности, но на самом деле оплачивается возможность обмена пакетами с другими хостами, также подключенными к интернету. Интернет не приобрел бы такую популярность, если бы пакеты можно было отправлять только хостам, находящимся в том же городе.

Поскольку сети зачастую очень разные, передача пакетов из одной в другую — задача не из легких. Помимо проблем с неоднородностью, нам придется решать вопросы масштабирования, возникающие из-за роста интерсети. Чтобы понять, что нас ждет, мы прежде всего обсудим различия между сетями. Затем мы изучим подход, успешно применяющийся в IP (протоколе сетевого уровня интернета), включая туннелирование, маршрутизацию в интерсетях и фрагментацию пакетов.

## 5.5.2. Различия сетей

Сети могут значительно отличаться друг от друга по разным параметрам. Некоторые параметры, например методы модуляции или форматы фреймов, мы оставим в стороне, поскольку они относятся к физическому и канальному уровням. На илл. 5.36 перечислены некоторые различия, которые проявляются на сетевом уровне. Именно смягчение этих расхождений усложняет межсетевое взаимодействие по сравнению с обеспечением работы одной сети.

Если пакеты должны пройти через одну или несколько сетей, прежде чем достичь сети назначения, может возникнуть множество проблем на интерфейсах между ними. Во-первых, источник должен иметь возможность адресовать пакет получателю. Что делать, если отправитель находится в сети Ethernet, а получатель — в мобильной телефонной сети? Даже если удастся задать адрес назначения, пакеты еще нужно как-то переправить из сети, не требующей соединения, в сеть, ориентированную на установление соединения. Это может потребовать создания нового соединения в кратчайшие сроки, что приведет к задержке и неэффективному использованию ресурсов, так как этот канал не будет активно использоваться.

| Показатель | Возможные варианты |
|---|---|
| Предлагаемая служба | Ориентированные на установление соединения или не требующие соединения |
| Адресация | Разного размера, плоская или иерархическая |
| Широковещание | Присутствует/отсутствует (то же относится к многоадресной рассылке) |
| Размер пакета | У каждой сети есть свой максимум |
| Порядок доставки | Упорядоченная и неупорядоченная доставка |
| QoS | Присутствует/отсутствует; много разновидностей |
| Надежность | Различные уровни потерь |
| Безопасность | Правила секретности, шифрование и т. д. |
| Параметры | Различные тайм-ауты, спецификация потока и др. |
| Тарификация | По времени соединения, за пакет, побайтно или никак |

**Илл. 5.36.** Некоторые показатели, по которым сети могут отличаться

Существует еще много различий, к которым необходимо приспособиться. К примеру, как передать пакет группе, некоторые участники которой находятся в сети, не поддерживающей многоадресную рассылку? Различные ограничения по размеру пакета также могут стать серьезной проблемой. Как передать 8000-байтный пакет по сети с максимальным размером пакета 1500 байт? Когда пакеты из ориентированной на соединение сети пересекают сеть, не требующую соединений, их порядок может нарушиться. Для отправителя это может оказаться неприятной неожиданностью — впрочем, как и для получателя.

При определенных усилиях с такими различиями можно справиться. Например, шлюз на стыке двух сетей может имитировать многоадресную рассылку, генерируя отдельные пакеты для каждого адреса назначения. Крупные пакеты могут разбиваться на части, пересылаться по частям, а затем снова объединяться. Принимающие устройства могут помещать пакеты в буфер, а затем доставлять их в правильном порядке.

Однако сети могут отличаться и в более сложных вопросах. Самый яркий пример — QoS. Если одна сеть предоставляет хороший уровень обслуживания, а другая — best effort, нельзя гарантировать определенную пропускную способность и задержку для непрерывного трафика в реальном времени. Точнее, это можно сделать, только если вторая сеть работает с низкой загрузкой или почти не используется, что маловероятно. Проблему представляют и механизмы безопасности, но по крайней мере шифрование в целях конфиденциальности и целостности данных можно обеспечить там, где они не поддерживаются. И наконец, разная тарификация может стать причиной неожиданно высоких счетов за обычные операции — ситуация, в которой часто оказываются пользователи мобильных телефонов в зоне роуминга.

### 5.5.3. Объединение гетерогенных сетей

Существует два основных способа объединения различных сетей. Можно создать специальные устройства, транслирующие или конвертирующие пакеты между сетями разного типа. Или, как это часто делают специалисты в области компьютерных наук, можно добавить уровень косвенной адресации и создать общий уровень над сетями. В любом случае на границе между сетями размещаются устройства, которые изначально назывались **шлюзами (gateways)**.

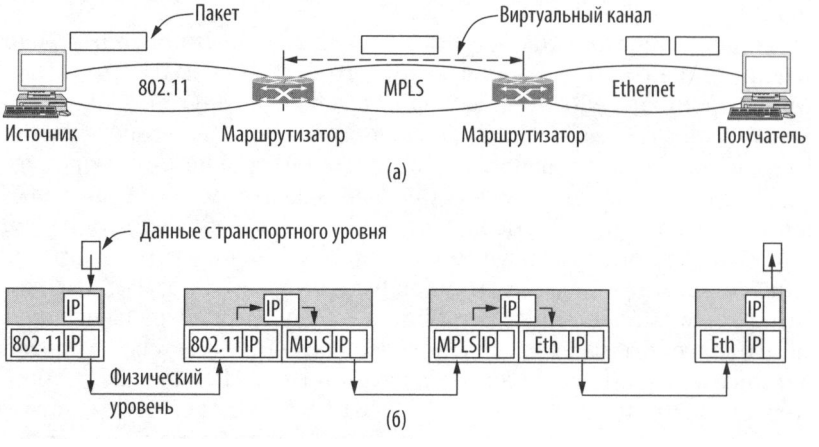
Идея создания общего уровня для устранения различий между сетями была предложена Серфом и Каном (Cerf and Kahn) в 1974 году. Этот подход имел невероятный успех и нашел применение в протоколах IP и TCP. Почти 40 лет спустя IP стал основой современного интернета. За это достижение в 2004 году Серф и Кан были удостоены премии Тьюринга, которая неофициально считается Нобелевской премией в области информатики. В IP используется универсальный формат пакетов, который распознается всеми маршрутизаторами и может быть передан почти по любой сети. Действие IP распространяется также и на телефонные сети. Кроме того, сегодня этот протокол работает в сенсорных сетях и на малых устройствах, хотя раньше считалось, что это невозможно из-за ограничений на ресурсы.

Мы уже говорили о нескольких устройствах для соединения сетей. Среди них повторители, концентраторы, мосты, коммутаторы и шлюзы. Повторители и концентраторы просто переносят биты с одного кабеля на другой. Чаще всего это аналоговые устройства, не имеющие отношения к протоколам вышележащих уровней. Мосты и коммутаторы работают на канальном уровне. Они могут использоваться для построения сетей, осуществляя по ходу дела минимальные преобразования протоколов, например, между сетями Ethernet со скоростями 10, 100 и 1000 Мбит/с. В этом разделе мы изучим устройства сетевого взаимодействия, работающие на сетевом уровне, то есть маршрутизаторы. Шлюзы — высокоуровневые интеркоммуникационные устройства — будут рассмотрены позднее.

Прежде всего выясним, как взаимодействие через общий сетевой уровень можно использовать для объединения сетей разного типа. На илл. 5.37 (а) изображена интерсеть, состоящая из сетей 802.11, MPLS и Ethernet. Пусть источник в 802.11 хочет отправить пакет получателю в Ethernet. Так как технологии отправки в этих сетях различны и при этом пакету придется пройти через сеть другого типа (MPLS), на границах сетей пакеты должны быть дополнительно обработаны.

Поскольку обычно разные сети используют различные способы адресации, пакет содержит адрес сетевого уровня, который может идентифицировать любой хост трех сетей. Сначала пакет приходит на границу между 802.11 и MPLS. Сеть 802.11 не требует соединения, а MPLS, наоборот, ориентирована на его установление. Это значит, что необходимо создать виртуальный канал. Когда пакет пройдет по этому каналу, он достигнет границы сети Ethernet. На данном этапе пакет может оказаться слишком большим, так как 802.11 работает с более крупными фреймами, чем Ethernet. В таком случае он разделяется на фрагменты, каждый из которых отправляется по отдельности. Фрагменты снова объединяются по прибытии на адрес назначения. Так заканчивается путешествие пакета.

Выполнение протоколов для этого путешествия показано на илл. 5.37 (б). Отправитель получает данные от транспортного уровня и создает пакет с заголовком общего сетевого уровня — в данном случае IP. Адрес получателя в заголовке указывает, что пакет должен быть отправлен через первый маршрутизатор. Пакет вставляется во фрейм 802.11 с адресом этого маршрутизатора и передается. Маршрутизатор извлекает пакет из поля данных и отбрасывает заголовок фрейма. Далее он анализирует содержащийся в пакете IP-адрес. Его нужно найти в таблице маршрутизации. В соответствии с ним принимается решение об отправке пакета на второй маршрутизатор. Для этого нужно установить виртуальный канал MPLS, ведущий ко второму маршрутизатору, а пакет поместить во фрейм с заголовками MPLS. На противоположном конце заголовок MPLS удаляется, и на основании адреса определяется следующий транзитный участок сетевого уровня. Этот участок ведет к получателю. Так как пакет слишком длинный для Ethernet, он делится на две части, каждая из которых помещается в поле данных фрейма Ethernet и передается на адрес назначения. Там заголовки фреймов считываются, и содержимое исходного пакета восстанавливается. Пакет достиг адресата.




**Илл. 5.37.** (а) Пакет проходит через разные сети. (б) Выполнение протоколов на сетевом и канальном уровнях

Следует отметить, что между коммутацией (установкой моста) и маршрутизацией есть существенная разница. При применении маршрутизатора пакет извлекается из фрейма, и для принятия решения используется адрес, содержащийся в пакете. Коммутатор (мост) пересылает весь пакет, обосновывая свое решение значением MAC-адреса. Коммутаторы (в отличие от маршрутизаторов) не обязаны вникать в подробности устройства протокола сетевого уровня, с помощью которого производится коммутация.

К сожалению, межсетевое взаимодействие не такая простая задача, как может показаться. С появлением мостов предполагалось, что они будут соединять разные типы сетей или, по крайней мере, локальных, преобразовывая

фреймы одной LAN во фреймы другой. Однако это не сработало, поскольку трудно преодолеть различия между свойствами LAN: максимальным размером пакета и наличием/отсутствием классов приоритетов. Поэтому сегодня мосты в основном соединяют сети одного типа на канальном уровне; для объединения разных сетей на сетевом уровне используются маршрутизаторы.

Межсетевое взаимодействие успешно применяется в построении крупных сетей, но это возможно только при наличии общего сетевого уровня. Со временем появилось множество различных протоколов. Сложно заставить всех использовать один формат, учитывая, что каждая компания видит коммерческую выгоду в том, чтобы иметь собственный протокол. Помимо IP, который на данный момент является практически универсальным сетевым протоколом, существуют IPX, SNA и AppleTalk. Ни один из них не используется повсеместно, а новые протоколы будут появляться всегда. Наиболее яркий пример —IPv4 и IPv6. И хотя оба они являются версиями IP, они несовместимы (иначе не было бы необходимости в разработке IPv6).

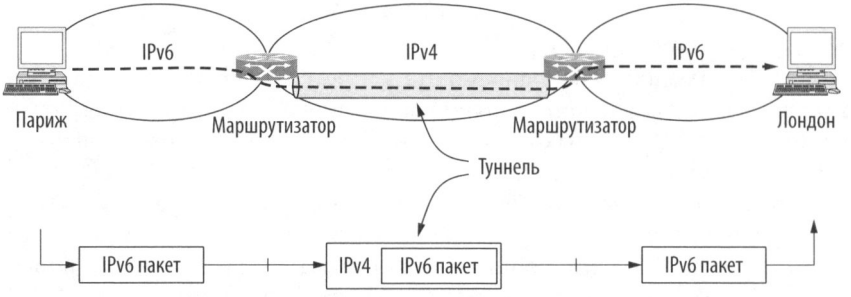
Маршрутизатор, поддерживающий несколько сетевых протоколов, называется **многопротокольным маршрутизатором (multiprotocol router)**. Он должен либо преобразовывать их, либо обеспечивать соединение на вышележащем уровне протокола. Ни один из двух вариантов не отвечает всем требованиям сети. Соединение на более высоком уровне, скажем, с применением TCP, предполагает, что TCP реализован во всех сетях (а это не всегда так). При этом в сетях могут функционировать только приложения, использующие TCP (к ним не относятся многие программы, работающие в реальном времени).

Другой вариант — конвертация пакетов между разными сетями. Однако если пакеты не близки по формату и не имеют одинаковых информационных полей, такое преобразование всегда будет неполным и часто обречено на ошибку. К примеру, длина IPv6-адресов составляет 128 бит. И как бы ни старался маршрутизатор, такой адрес ни за что не поместится в 32-битное поле адреса IPv4. Проблема использования IPv4 и IPv6 в одной сети оказалась серьезным препятствием к внедрению IPv6. (И честно говоря, по этой же причине потребителей так и не удалось убедить в том, что они должны его использовать.) Но еще более серьезные проблемы возникают, если нужно выполнять конвертацию между принципиально разными протоколами — например, между не требующим соединения и ориентированным на его установление. Такие преобразования практически никогда не выполняются. Вероятно, IP работает так хорошо только потому, что служит чем-то вроде наименьшего общего знаменателя. Он не многого требует от сетей, однако предоставляет только уровень обслуживания best effort.

## 5.5.4. Соединение конечных точек в гетерогенных сетях

Объединение сетей в общем случае является исключительно сложной задачей. Однако существует частный случай, реализация которого вполне осуществима даже для разных сетевых протоколов. Это ситуация, при которой хост-источник и хост-приемник расположены в одинаковых сетях, но между ними находится сеть другого типа. Например, представьте себе международный банк, у которого

имеется сеть IPv6 в Париже и такая же — в Лондоне, а между ними находится IPv4, как показано на илл. 5.38.




**Илл. 5.38.** Туннелирование пакета из Парижа в Лондон

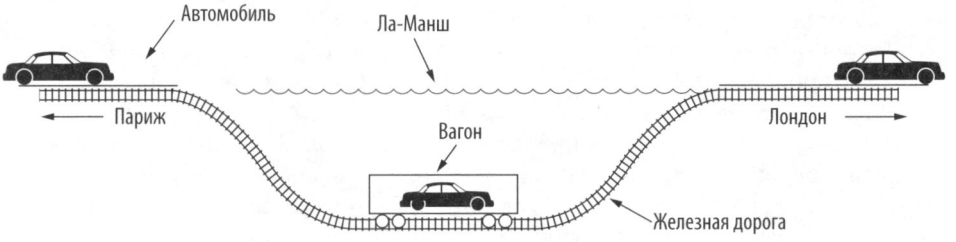
Для решения этой проблемы используется **туннелирование (tunneling)**. Чтобы передать IP-пакет хосту в Лондоне, хост в Париже формирует пакет, содержащий лондонский IPv6-адрес, и отправляет его на многопротокольный маршрутизатор, соединяющий парижскую сеть IPv6 и сеть IPv4. Получив пакет IPv6, маршрутизатор помещает его в другой пакет с IPv4-адресом маршрутизатора, соединяющего сеть IPv4 и лондонскую сеть IPv6. Когда пакет попадает на этот адрес, лондонский многопротокольный маршрутизатор извлекает исходный IPv6-пакет и передает его дальше на хост назначения.

Путь через сеть IPv4 можно рассматривать как большой туннель, идущий от одного многопротокольного маршрутизатора до другого. IPv6-пакет просто перемещается от одного конца туннеля до другого в удобной «коробке». Ему не нужно беспокоиться о взаимодействии с сетью IPv4. Это также касается хостов Парижа и Лондона. Переупаковкой пакета и переадресацией занимаются многопротокольные маршрутизаторы. Для этого им нужно уметь разбираться в IPv6- и IPv4-пакетах. В результате весь путь от одного многопротокольного маршрутизатора до другого работает как один транзитный участок.

Чтобы сделать этот пример еще проще и понятнее, рассмотрим аналогию. Представьте водителя автомобиля, направляющегося из Парижа в Лондон. По дорогам Франции автомобиль едет самостоятельно. Но достигнув Ла-Манша, он загружается в высокоскоростной поезд и транспортируется под проливом по туннелю (автомобилям запрещено ездить по этому туннелю). Фактически автомобиль перевозится как груз (илл. 5.39). Прибыв в Великобританию, он спускается с железнодорожной платформы на английское шоссе и снова продолжает путь своим ходом. Точно такой же принцип применяется при туннелировании пакетов, проходящих через чужеродную сеть.

Туннелирование широко используется для соединения изолированных хостов и сетей посредством других сетей. В результате появляется новая сеть, которая, по сути, накладывается на старую. Она называется **оверлейной сетью (overlay)**. Использование сетевого протокола с новым свойством (как в нашем примере, где сети IPv6 соединяются через IPv4) — достаточно распространенная

причина применения этого метода. Недостатком туннелирования является то, что пакет невозможно доставить на хосты сети-посредника. Однако это становится преимуществом в сетях VPN. Это обычная оверлейная сеть, использующаяся в качестве меры безопасности. Более подробно о VPN мы поговорим в главе 8.




**Илл. 5.39.** Туннелирование автомобиля из Парижа в Лондон

## 5.5.5. Межсетевая маршрутизация

Маршрутизация в интерсетях сопровождается теми же основными проблемами, что и в одной сети, но с некоторыми дополнительными сложностями. Рассмотрим две сети с разными алгоритмами маршрутизации: с учетом состояния линий и по вектору расстояний. В первом случае алгоритму требуется информация о топологии сети, во втором — нет, поэтому неясно, как вычислять кратчайшие пути в такой интерсети.

Более серьезные проблемы возникают, если работу сети обеспечивают разные операторы. Прежде всего, они могут по-разному представлять себе, что такое хороший маршрут: одни обращают больше внимания на время задержки, другие — на затраты. В результате стоимость пути указывается в разных единицах: в миллисекундах задержки или денежных единицах. Из-за невозможности сопоставления весовых коэффициентов вычисление кратчайших путей становится затруднительным.

Более того, провайдер может не предоставлять другим операторам доступ к данным о весах и маршрутах в своей сети. В них может быть информация, составляющая коммерческую тайну (например, стоимость).

Наконец, интерсеть может быть гораздо больше, чем любая из ее составляющих. В такой ситуации требуются алгоритмы маршрутизации, масштабируемые с учетом иерархии, даже если в отдельных сетях такой необходимости нет.

Следовательно, нам необходим двухуровневый алгоритм маршрутизации. В пределах каждой сети для маршрутизации используется **внутридоменный (intradomain)**, или внутренний, шлюзовый протокол (термин «шлюз» ранее обозначал «маршрутизатор»). Это может быть обычный протокол, учитывающий состояние линий. Между сетями применяется **междоменный (interdomain)**, или внешний, шлюзовый протокол. Сети могут использовать разные внутридоменные протоколы, но междоменный протокол должен быть общим. В интернете используется междоменный протокол пограничной маршрутизации (Border Gateway Protocol, BGP). Мы подробно обсудим его в разделе 5.7.7.

Здесь следует рассказать еще об одном важном понятии. Так как все сети управляются независимо, их часто называют автономными системами (АС). Хорошая умозрительная модель АС — сеть интернет-провайдера. На практике она может состоять из нескольких АС, если они управляются независимо. Но разница не слишком существенна.

Два уровня маршрутизации не являются в строгом смысле иерархическими. Если крупную международную сеть объединить с небольшой региональной сетью, пути могут быть близкими к оптимальным. Но для вычисления маршрутов в интерсети отдельные сети предоставляют сравнительно мало информации о своих путях. Это позволяет преодолеть сложности. В результате улучшается масштабирование сети, а операторы свободны в выборе протоколов маршрутизации внутри своих сетей. Кроме того, не требуется сравнивать веса различных сетей и раскрывать секретную внутреннюю информацию.

До сих пор мы практически ничего не рассказали о том, как вычисляются маршруты в интерсети. В интернете ключевым фактором являются коммерческие договоренности между провайдерами. Каждый из них может взимать с других провайдеров плату за передачу трафика. Еще одна особенность — если при межсетевой маршрутизации пересекаются границы государств, в игру могут вступить их законы. Так, в Швеции персональные данные граждан находятся под строжайшей защитой. Все эти внешние предпосылки объединяются в понятие **политики маршрутизации (routing policy)**, в соответствии с которой автономные сети выбирают пути. К этому понятию мы еще вернемся, когда будем говорить о BGP.

## 5.5.6. Поддержка различных размеров пакета: фрагментация пакета

Все сети и каналы накладывают на размер своих пакетов ограничения, обусловленные разными факторами:

1. Аппаратное обеспечение (например, размер фрейма Ethernet).
2. Операционная система (например, все буферы имеют размер 512 байт).
3. Протоколы (например, количество битов в поле длины пакета).
4. Соответствие какому-либо международному или национальному стандарту.
5. Желание снизить количество пакетов, отправляемых повторно из-за ошибок передачи.
6. Желание предотвратить ситуацию, когда один пакет слишком долгое время занимает канал.

С учетом всех этих факторов разработчики не могут выбирать максимальный размер пакета по своему усмотрению. Максимальный размер поля полезной нагрузки составляет 1500 байт для сети Ethernet и 2272 байта для 802.11. Протокол IP более щедр: размер пакета может достигать 65 515 байт.

Обычно хосты стараются отправлять крупные пакеты, так как это уменьшает издержки — например, позволяет сэкономить на заголовках. Очевидно, возникает

проблема, когда большой пакет должен пройти по сети с недостаточным максимальным размером пакетов. Эта проблема была актуальна долгое время, а ее решения разрабатывались во многом на основании опыта использования интернета.

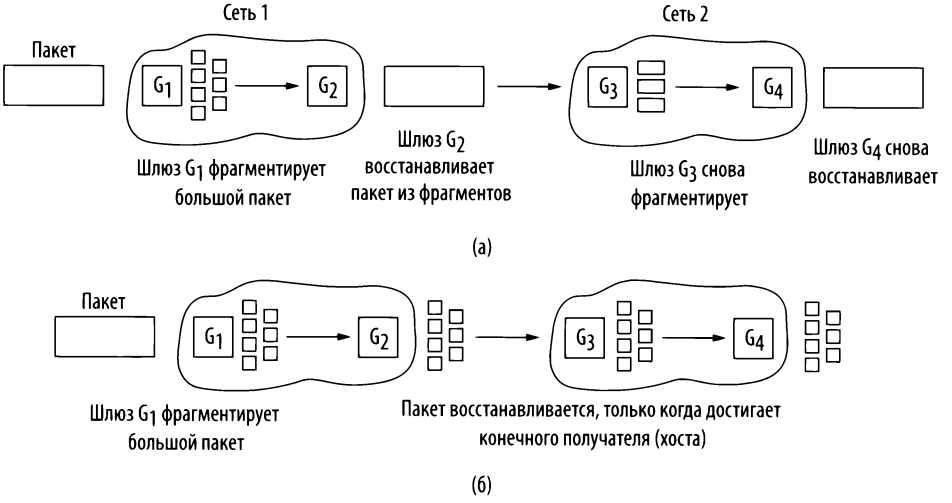
Одно из решений проблемы состоит в ее предотвращении. Но сделать это непросто. Обычно отправитель не знает, по какому пути будут передаваться данные, а значит, ему неизвестно, какого размера должен быть пакет, чтобы добраться до места назначения. Такой размер пакета называется **путевым значением MTU (Path Maximum Transmission Unit** — максимальный размер пакета для выбранного пути). Неважно, знает ли отправитель путевое значение MTU. В сети, не требующей соединения (например, в интернете), маршруты в любом случае выбираются независимо. Это означает, что путь может неожиданно измениться, а тогда изменится и путевое значение MTU.

Альтернативное решение — позволить шлюзам разбивать пакеты на **фрагменты (fragments)** и отправлять каждый из них в виде отдельного пакета сетевого уровня. Однако, как известно любому родителю маленького ребенка, разобрать большой объект на мелкие части существенно проще, чем соединить их обратно. (Физики даже дали этому эффекту специальное название: второй закон термодинамики.) В сетях с коммутацией пакетов также существует проблема объединения фрагментов.

Для восстановления исходных пакетов применяются две противоположные стратегии. Согласно первой, порожденная сетью с малым размером пакетов фрагментация должна быть прозрачной для всех сетей, через которые пакет проходит на пути к адресату. Этот вариант показан на илл. 5.40 (а). Когда на $G_1$ приходит слишком большой пакет, он разбивается на фрагменты. Все они адресуются одному и тому же выходному маршрутизатору $G_2$, который восстанавливает из них исходный пакет. Таким образом, прохождение данных через сеть с небольшим размером пакетов оказывается прозрачным. Соседние сети даже не знают, что произошла фрагментация.

Прозрачная фрагментация проста, но тем не менее, создает некоторые проблемы. Во-первых, выходной маршрутизатор должен узнать, что он получил пакет полностью, поэтому фрагменты должны содержать либо поле счетчика, либо бит, обозначающий конец пакета. Кроме того, тот факт, что для последующего восстановления пакета все фрагменты должны выходить через один и тот же маршрутизатор, ограничивает возможности маршрутизации. Фрагменты не могут идти к конечному получателю по разным маршрутам, и в результате может потеряться часть производительности. Еще более важный вопрос касается действий маршрутизатора. Возможно, ему потребуется поместить в буфер все полученные данные и определить, в какой момент их можно удалить, если пришли не все фрагменты. Наконец, фрагментация и последующая сборка пакетов при прохождении каждой сети с малым размером пакетов приводят к дополнительным накладным расходам.

Другая стратегия фрагментации заключается в отказе от восстановления пакета на промежуточных маршрутизаторах. С каждым фрагментом обращаются как с отдельным пакетом. Все фрагменты проходят через маршрутизаторы, как показано на илл. 5.40 (б). Задача восстановления оригинального пакета возложена на получающий хост.



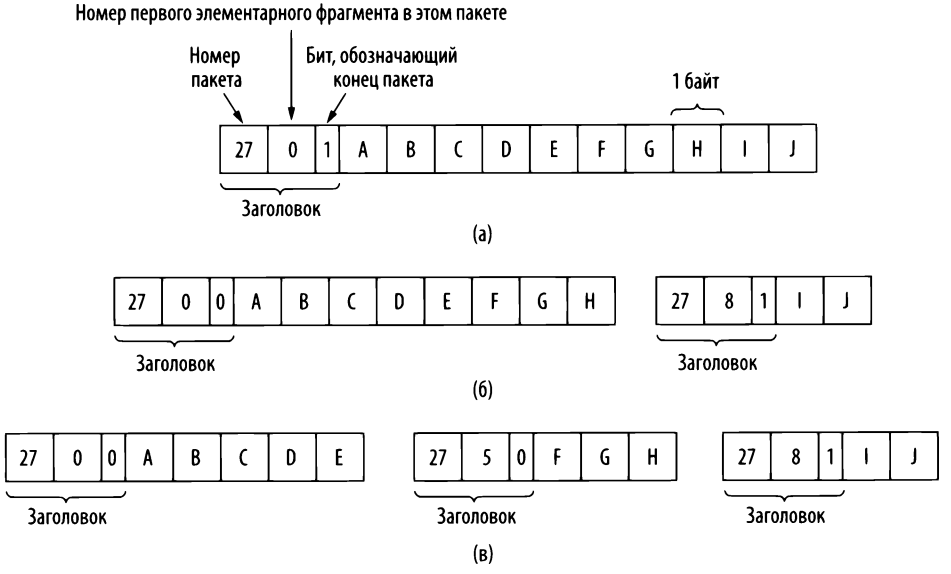

**Илл. 5.40.** Фрагментация. (а) Прозрачная. (б) Непрозрачная

Основное преимущество непрозрачной фрагментации состоит в том, что маршрутизаторы выполняют меньше работы. Так работает IP. При этом фрагменты пакета должны нумероваться таким образом, чтобы можно было восстановить исходный поток данных. В случае IP каждому фрагменту сообщается номер, который есть у каждого пакета, абсолютное байтовое смещение внутри пакета и метка, показывающая, является ли этот фрагмент последним в пакете. Пример такой фрагментации представлен на илл. 5.41. Схема очень проста, но обладает рядом важных преимуществ. По прибытии в место назначения фрагменты можно помещать в буфер в любом порядке. Кроме того, при пересечении сети с меньшим значением MTU фрагменты могут быть разбиты на более мелкие части (см. илл. 5.41 (в)). При повторной передаче (если не все фрагменты дошли до адресата) пакет может быть разделен по-другому. И наконец, размер фрагментов может быть произвольным, вплоть до одного байта (плюс заголовок). В любом случае пакет будет восстановлен: номер пакета и смещение помогут расположить данные в правильном прядке, а метка укажет на конец пакета.

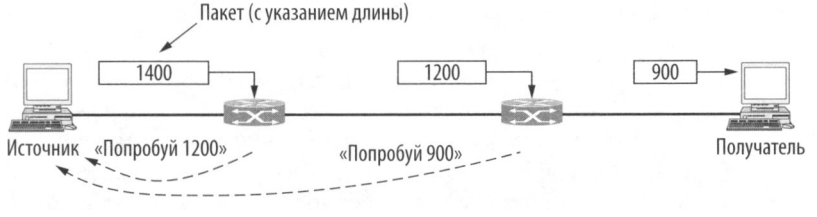
К сожалению, у этой схемы есть недостатки. Во-первых, она может быть более затратной, чем прозрачная фрагментация, так как заголовки фрагментов иногда передаются по каналам, где без них можно обойтись. Но настоящая проблема заключается в самом существовании фрагментов. Кент и Могул (Kent and Mogul, 1987) считают, что фрагментация работает в ущерб производительности, так как при утере одного фрагмента теряется весь пакет (не говоря уже о затратах на заголовки). Для хостов фрагментация представляет большую нагрузку, чем предполагалось вначале.

Это возвращает нас к первоначальной идее полного избавления от фрагментации в сети. Стратегия, использующаяся в современном интернете, называется **поиском путевого значения MTU (Path MTU discovery)** (Могул и Диринг;

Mogul and Deering, 1990). При отправке IP-пакета в его заголовке указывается, что фрагментация запрещена. Если маршрутизатор получает слишком большой пакет, он удаляет его и отправляет источнику сообщение об ошибке (илл. 5.42). Используя эту информацию, отправитель перераспределяет данные так, чтобы пакеты смогли пройти через маршрутизатор. Если такая проблема возникнет на одном из следующих маршрутизаторов, процесс повторится.




**Илл. 5.41.** Фрагментация при элементарном размере 1 байт. (а) Исходный пакет, содержащий 10 байт данных. (б) Фрагменты после прохождения через сеть с максимальным размером пакета 8 байт плюс заголовок. (в) Фрагменты после прохождения через шлюз размера 5




**Илл. 5.42.** Поиск путевого значения MTU

Преимущество поиска путевого значения MTU состоит в том, что теперь источник знает необходимый размер пакета. При изменении маршрута отправитель узнает новое значение MTU из новых сообщений об ошибке. Однако фрагментация между отправителем и получателем все равно необходима, если только значение MTU не вычислено на более высоком уровне и не передано IP. Чтобы

обеспечить передачу такой информации, TCP и IP обычно используются вместе (в виде TCP/IP). И хотя для некоторых протоколов это пока не реализовано, фрагментация все же была перенесена из сети на хосты.

Недостатком этого метода является то, что отправка пакета может вызвать дополнительную задержку. Эта задержка может увеличиться в несколько раз в зависимости от того, сколько раз отправителю придется повторять отправку (меняя размер пакета), прежде чем какие-либо данные достигнут адресата. Возникает вопрос: существуют ли более эффективные схемы? Вероятно, да. Рассмотрим, к примеру, вариант, при котором слишком большие пакеты просто обрезаются. В таком случае адресат узнает путевое значение MTU максимально быстро (по размеру доставленного пакета), а также получает некоторую часть данных.

## 5.6. ПРОГРАММНО-КОНФИГУРИРУЕМЫЕ СЕТИ

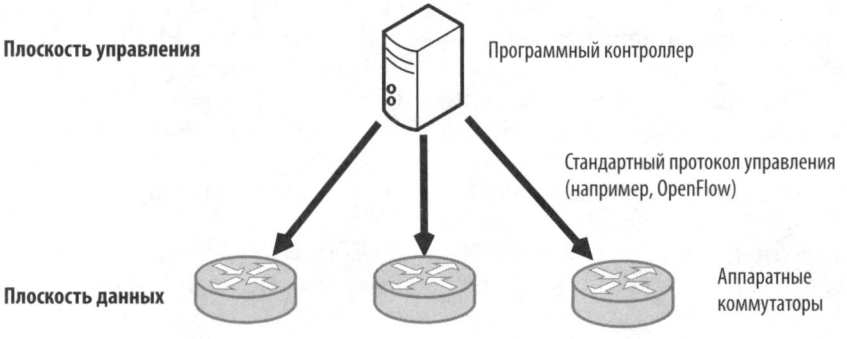
Процесс управления трафиком всегда был очень непростым: он требует от операторов сетей настройки параметров конфигурации протоколов маршрутизации, которые затем производят перерасчет путей. Трафик идет по новым маршрутам, что ведет к перераспределению нагрузки. К сожалению, механизмы управления трафиком носят косвенный характер: изменение конфигурации протоколов меняет маршрутизацию и в отдельных сетях, и между ними. При этом протоколы часто ведут себя непредсказуемо. Эти проблемы во многом решаются с помощью **программно-конфигурируемых сетей (Software-Defined Networking, SDN)**, которые мы обсудим далее.

### 5.6.1. Общие сведения

По сути, компьютерные сети всегда были «программно-конфигурируемыми», поскольку в маршрутизаторах используется конфигурируемое программное обеспечение, извлекающее информацию из пакетов и принимающее решения об их передаче. В то же время та часть ПО, которая запускает алгоритмы маршрутизации и реализует остальную логику передачи пакетов, всегда была вертикально интегрированной в сетевое оборудование. Купив маршрутизатор Cisco или Juniper, оператор сети был вынужден работать с программными технологиями, которые внедрил поставщик оборудования. Например, изменить какие-либо параметры протокола OSPF или BGP было просто невозможно. Главной причиной разработки SDN стало понимание, что **плоскость управления (control plane)** (логика выбора маршрутов и принятия решений о передаче) работает на программном уровне и может выполняться абсолютно независимо от **плоскости данных (data plane)** (аппаратных технологий, которые непосредственно извлекают информацию из пакетов и решают, что с ними делать). Эти плоскости показаны на илл. 5.43.

Если структурно разделить плоскость управления и плоскость данных, то логично предположить, что плоскость управления совсем необязательно запускать на сетевом оборудовании. На самом деле одна из популярных спецификаций SDN подразумевает наличие логически централизованной программы, зачастую

написанной на языке высокого уровня (например, Python, Java, Golang, C). Эта программа принимает логические решения о пересылке и уведомляет о них все участвующие устройства. В качестве канала связи между высокоуровневой программой и нижележащим аппаратным обеспечением можно использовать любой механизм, поддерживаемый сетевым устройством. Один из первых SDN-контроллеров использовал в качестве плоскости управления протокол BGP (Фимстер и др.; Feamster et al., 2003). Позднее появились технологии OpenFlow, NETCONF и YANG, которые предложили более гибкие способы передачи информации плоскости управления сетевым устройствам. В некотором смысле технология SDN стала реинкарнацией давно известной архитектурной концепции централизованного управления, появлению которой способствовало наличие уже устоявшихся вспомогательных технологий (свободно распространяемых API для чипсетов, программного управления распределенными системами).




**Илл. 5.43.** Разделение плоскости управления и плоскости данных в программно-конфигурируемых сетях

Хотя технология SDN активно развивается в течение многих лет, основной принцип разделения плоскости данных и плоскости управления остается неизменным. Вы можете подробно ознакомиться с историей появления этой набирающей популярность технологии в работе Фимстера и др. (Feamster et al., 2013). Далее мы рассмотрим несколько основных направлений развития SDN. Это контроль переадресации и маршрутизации (технологии плоскости управления); программируемое оборудование и настраиваемая переадресация (технологии, делающие плоскость данных более программируемой); и программируемая сетевая телеметрия (приложение для сетевого администрирования, которое соединяет эти два компонента и играет ключевую роль для SDN).

## 5.6.2. Плоскость управления в SDN: логически централизованное программное управление

Одна из главных технических идей в основе SDN — наличие плоскости управления, работающей независимо от маршрутизаторов, часто в виде единственной логически централизованной программы. В некотором смысле технология SDN

существовала всегда: поскольку маршрутизаторы можно настраивать, крупные сети часто автоматически генерировали их конфигурации на основе централизованной базы данных, обеспечивали контроль версий и выполняли их на маршрутизаторах с помощью скриптов. Такой способ настройки можно назвать SDN, но с технической точки зрения он дает операторам очень ограниченные возможности контроля передачи трафика по сети. Чаще всего управляющая программа SDN (контроллер) в большей степени отвечает за логику управления. К примеру, она вместо маршрутизаторов вычисляет пути и затем просто обновляет таблицы переадресации.

Первые исследования в области SDN были направлены на то, чтобы упростить для сетевых операторов задачу по регулированию трафика. Вместо настройки конфигурации сети предлагалось напрямую контролировать, какой путь выбирает каждый маршрутизатор. Первые реализации технологии SDN должны были работать в рамках существующих интернет-протоколов маршрутизации для непосредственного управления выбором путей. Одним из таких решений стала **платформа управления маршрутизацией (Routing Control Platform, RCP)** (Фимстер и др.; Feamster et al., 2003), впоследствии развернутая в магистральных сетях для балансировки нагрузки и защиты от DoS-атак. Позднее появился ряд других систем, в частности Ethane (Касадо и др.; Casado et al., 2007), обеспечивающая централизованный программный контроль аутентификации хостов в сети. Однако для ее реализации требовались нестандартные коммутаторы, что не способствовало ее распространению.

Когда преимущества технологии SDN для сетевого администрирования стали очевидны, к ней начали проявлять интерес операторы сетей и поставщики оборудования. Кроме того, была обнаружена удобная «лазейка» для еще более успешного контроля коммутаторов. Во многих из них использовался чипсет Broadcom с интерфейсом, позволяющим производить прямую запись в память коммутатора. Группа исследователей провела совместную работу с поставщиками коммутаторов, чтобы сделать этот интерфейс доступным для ПО. В результате был создан протокол **OpenFlow** (Маккиоун и др.; McKeown et al., 2008). Его поддержку обеспечили многие поставщики, которые пытались конкурировать с доминирующим игроком на этом рынке, компанией Cisco. Изначально OpenFlow поддерживал очень простой интерфейс: данные записывались в ассоциативную память в виде обычной **таблицы сопоставления действий (match-action table)**. Эта таблица позволяла коммутатору идентифицировать пакеты по одному или нескольким полям заголовка (например, по полю MAC-адреса, IP-адреса и т. д.) и выполнить какое-либо действие, включая переадресацию пакета на определенный порт, его удаление или отправку в находящийся вне маршрута программный контроллер.

Было создано несколько версий протокола OpenFlow. Версия OpenFlow 1.0 предусматривала наличие *одной* таблицы сопоставления действий. С помощью этой таблицы можно было найти точные совпадения с указанной комбинацией полей заголовка пакета (полей MAC-адреса, IP-адреса и т. д.) или произвести поиск по шаблону (например, по префиксу IP-адреса или MAC-адреса). В последующих версиях (наиболее заметная из них – OpenFlow 1.3) была добавлена поддержка более сложных операций, включая работу с *цепочками* таблиц, но лишь немногие поставщики реализовали эти возможности. Применять логические

операции «И» и «ИЛИ» к таким сопоставлениям оказалось непросто, особенно для программистов. Это привело к появлению ряда технологий, которые упрощали задачу выражения более сложных комбинаций условий (Фостер и др.; Foster et al., 2011). Также они позволяли учитывать время и другие показатели при принятии решения о переадресации (Ким и др.; Kim et al., 2015). В итоге некоторые из этих технологий получили весьма ограниченное распространение: OpenFlow стал использоваться в крупных центрах обработки данных, где сетевые операторы обладали полным контролем над сетью. В глобальных и корпоративных сетях они использовались еще реже, поскольку в таблице переадресации можно было выбрать лишь очень ограниченный набор действий. Кроме того, многие поставщики коммутаторов так и не предоставили полную реализацию последних версий данного протокола. Это затрудняло практическое внедрение решений, использующих эти возможности. Однако в конечном счете протокол OpenFlow оставил после себя в качестве наследия пару важных идей: контроль над сетью с помощью одной централизованной программы, координирующей сетевые устройства и элементы пересылки, и выражение этого контроля с помощью одного высокоуровневого языка программирования (например, Python или Java).

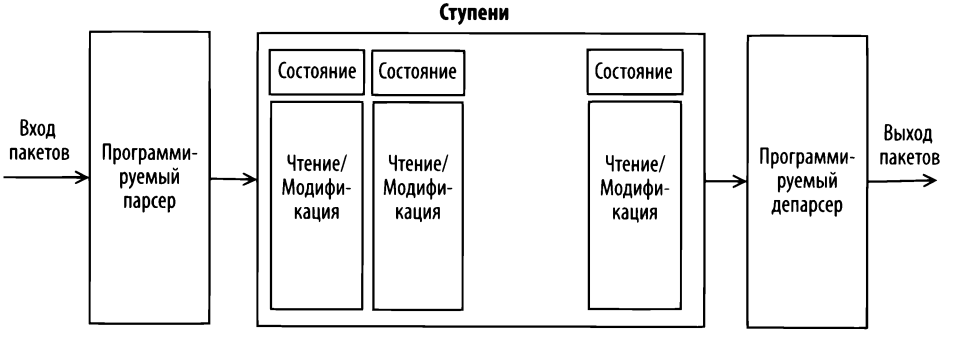
По сути, OpenFlow является очень ограниченным интерфейсом. Он не был рассчитан на гибкое управление сетью. Скорее он стал удобным сиюминутным решением, разработка которого была обусловлена рынком. Сетевые устройства уже имели в своих коммутаторах таблицы сопоставления на базе троичной ассоциативной памяти. Прежде всего OpenFlow был инициативой по созданию интерфейса для этих таблиц, чтобы сторонние программы могли производить в них запись. Вскоре исследователи сетевых технологий начали задумываться о проектировании аппаратных средств для обеспечения более гибкого управления плоскостью данных. В следующем разделе мы расскажем, как прогресс в области программируемого оборудования привел к повышению степени программируемости самих коммутаторов.

Наряду с этим программный контроль на уровне ПО, изначально рассчитанный в основном на транзитные сети и сети центров обработки данных, постепенно находит применение и в мобильных сетях. Например, проект переоборудования узла связи в дата-центр (Central Office Re-Architected as a Datacenter, CORD) направлен на разработку сети 5G на основе стандартных аппаратных средств и программных компонентов с открытым исходным кодом (Петерсон и др.; Peterson et al., 2019).

## 5.6.3. Плоскость данных в SDN: программируемое оборудование

В силу очевидных ограничений чипсета OpenFlow цель последующей работы с SDN заключалась в повышении степени программируемости самого оборудования. Множество разработок в этой области, касающихся сетевых карт и коммутаторов, обеспечили настройку практически всех параметров, от формата пакета до режима передачи.

Для этой архитектуры используется общее название — **протоколонезависимая архитектура коммутатора (protocol-independent switch architecture)**. Она предполагает наличие фиксированного набора конвейеров обработки. Каждый из них обладает памятью для таблиц сопоставления действий и определенным объемом регистровой памяти, а также поддерживает ряд простых операций, например добавление (Босхарт и др.; Bosshart et al., 2013). Соответствующая модель переадресации называется **реконфигурируемыми таблицами сопоставления (Reconfigurable Match Tables, RMT)**; ее конвейерная архитектура была вдохновлена RISC-архитектурами. Каждая ступень конвейера обработки может считывать информацию из заголовков пакетов, модифицировать значения заголовка с помощью простых арифметических операций и записывать эти значения обратно в пакеты (илл. 5.44). Архитектура чипа содержит программируемый парсер, набор ступеней сопоставления в определенном состоянии. Они выполняют арифметические вычисления над пакетами и принимают простейшие решения по их передаче или удалению. Другой компонент чипа, депарсер, записывает полученные значения обратно в пакеты. Каждая ступень чтения/модификации может менять как собственное состояние, так и любые метаданные пакетов (например, информацию о том, какую глубину очереди видит отдельный пакет).



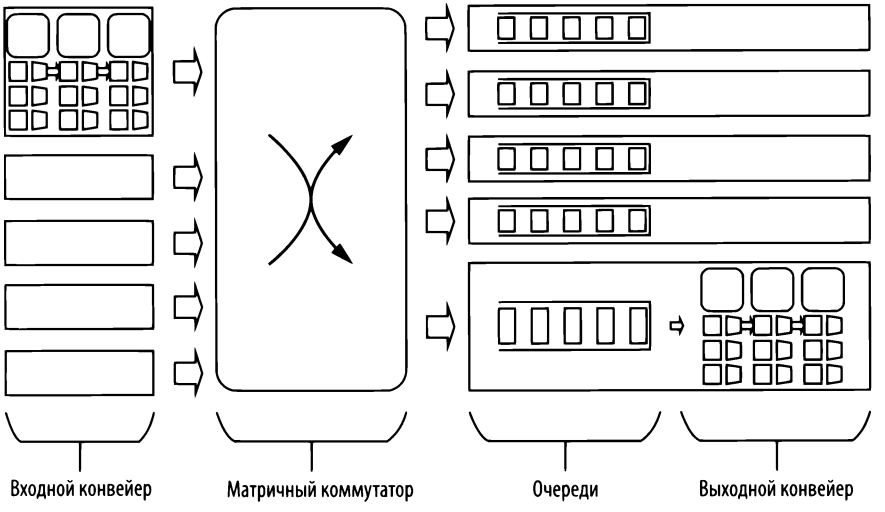

**Илл. 5.44.** Реконфигурируемый конвейер сопоставления действий для обеспечения программируемой плоскости данных

Модель RMT также позволяет использовать нестандартные форматы заголовка, а значит, в каждом пакете можно хранить дополнительные данные помимо информации, предусмотренной в стандартных протоколах. При использовании RMT программист может модифицировать аппаратную плоскость данных без внесения изменений в само оборудование, а также задать несколько таблиц сопоставления произвольного размера с учетом общего ограничения ресурсов. Оператор также получает достаточную свободу модификации полей заголовка по своему усмотрению.

Современные чипсеты, такие как Barefoot Tofino, позволяют выполнять протоколонезависимую обработку нестандартных пакетов на входе и на выходе (илл. 5.45). Благодаря этому можно анализировать временные параметры очередей (например, время нахождения в очереди отдельных пакетов), а также

производить нестандартную инкапсуляцию и декапсуляцию. Кроме того, можно применять метод активного управления очередью (к примеру, RED) к выходным очередям, используя метаданные, получаемые из входных очередей. В настоящее время исследуются способы использования этой архитектуры для управления трафиком и перегрузками, в частности, путем мелкоструктурной оценки параметров очередей (Чэнь и др.; Chen et al., 2019).

Такая степень программируемости оказалась наиболее полезной в сетях датацентров. Благодаря своей архитектуре они способны извлечь выгоду из широких возможностей настройки. С другой стороны, данная модель обеспечивает общее повышение эффективности и расширение функциональности. Она позволяет включать в пакеты информацию о состоянии самой сети с помощью **внутриполосной сетевой телеметрии (In-band Network Telemetry, INT)** (например, величину задержки на каждом транзитном участке пути).




**Илл. 5.45.** Реконфигурируемые конвейеры сопоставления действий на этапах входа и выхода

На сегодняшний день существуют программируемые сетевые карты, комплект разработчика плоскости данных (Data Plane Development Kit, DPDK) от Intel и другие библиотеки, а также гибкие конвейеры обработки (например, чипсет Barefoot Tofino, программируемый на языке P4 (Босхарт и др.; Bosshart et al., 2014)). Благодаря этому операторы сетей разрабатывают нестандартные протоколы и производят расширенную обработку пакетов в самом коммутационном оборудовании. P4 — высокоуровневый язык для программирования протоколонезависимых обработчиков пакетов (например, RMT-чипов). Также были разработаны программируемые плоскости данных для виртуальных коммутаторов (точнее, это произошло задолго до появления программируемых аппаратных коммутаторов). Еще одним важным шагом в области программируемого управления коммутаторами стала разработка открытой программной

реализации Open vSwitch (OVS), которая позволяет обрабатывать пакеты на нескольких уровнях и работает в виде модуля ядра Linux. OVS предлагает целый ряд возможностей, от VLAN до IPv6. Операторы сетей получили возможность настраивать переадресацию в дата-центрах, в частности, запуская OVS в качестве коммутатора в гипервизоре серверов.

## 5.6.4. Программируемая сетевая телеметрия

Одним из наиболее важных преимуществ технологии SDN является то, что она позволяет обеспечить программируемую оценку параметров сети. Многие годы сетевое оборудование предоставляло весьма ограниченные сведения о сетевом трафике. Это могла быть общая статистика потоков трафика с точки зрения сетевого коммутатора (например, по стандарту IPFIX). В то же время возможность перехвата каждого сетевого пакета ведет к большим издержкам, учитывая необходимый размер пропускной способности и памяти, а также объем обработки для последующего анализа этих данных. Для многих приложений нужно найти баланс между степенью детализации трассировки пакетов и масштабируемостью агрегатных данных IPFIX. Этот баланс необходим и для таких задач сетевого администрирования, как оценка производительности приложений, и для упоминавшихся ранее задач управления перегрузками.

Программируемое коммутационное оборудование, представленное в предыдущем разделе, обеспечивает более гибкий сбор телеметрии. Одна из тенденций, к примеру, заключается в предоставлении операторам сетей возможности запрашивать данные о сетевом трафике на высокоуровневом языке с использованием таких фреймворков, как MapReduce (Дин и Гемават; Dean and Ghemawat, 2008). Этот подход изначально рассчитан на обработку данных в крупных кластерах. Поэтому с его помощью можно запрашивать информацию о сетевом трафике (например, количество байтов или пакетов, отправленных на определенный адрес или порт в заданном временном интервале). К сожалению, программируемое коммутационное оборудование еще (пока) не настолько совершенно, чтобы поддерживать сложные запросы. Иногда их нужно разделять между потоковым процессором и сетевым коммутатором. Для этого существует несколько технологий (Гупта и др.; Gupta et al., 2019). Вопрос об эффективном способе отображения высокоуровневых абстракций и конструкций запросов на коммутационное аппаратное и программное оборудование более низкого уровня остается открытым.

Наконец, крупной проблемой в сфере программируемой сетевой телеметрии в ближайшие годы будет все большее распространение в интернете зашифрованного трафика. С одной стороны, это обеспечивает конфиденциальность, затрудняя несанкционированный доступ к пользовательским данным. Но с другой стороны, невозможность увидеть содержимое трафика усложняет задачу сетевого администрирования для операторов. К примеру, рассмотрим отслеживание качества видеопотоков. Если трафик не зашифрован, можно узнать битрейт и разрешение видео. В противном случае эти сведения приходится логически выводить на основе тех свойств сетевого трафика, которые доступны для непосредственного наблюдения (например, интервалы получения пакетов и объем

трафика в байтах). В недавних исследованиях были представлены способы автоматического логического вывода высокоуровневых свойств трафика сетевых приложений на основе низкоуровневой статистики (Бронзино и др.; Bronzino et al., 2020). Рано или поздно операторам потребуются более эффективные модели, позволяющие логически оценить влияние перегрузки и других факторов на производительность приложений.

## 5.7. СЕТЕВОЙ УРОВЕНЬ ИНТЕРНЕТА

Теперь самое время подробно поговорить о сетевом уровне интернета. Но прежде чем перейти к деталям, имеет смысл познакомиться с принципами, которые легли в его основу при разработке и обеспечили его дальнейший успех. В наше время ими все чаще пренебрегают. Между тем эти принципы пронумерованы и включены в документ RFC 1958, который стоит изучить (а разработчики просто обязаны его прочитать и сдать по нему экзамен). Этот документ использует идеи, изложенные в работах Кларка (Clark, 1988) и Зальцера и др. (Saltzer et al., 1984). Ниже мы приведем 10 основных принципов, начиная с самых важных.

1. **Убедитесь в работоспособности.** Нельзя считать разработку (или стандарт) законченной, пока не осуществлен ряд успешных соединений между прототипами. Очень часто разработчики сначала пишут тысячестраничное описание стандарта, утверждают его, а потом обнаруживается, что он еще очень сырой или вообще неработоспособен. Тогда пишется версия 1.1. Так быть не должно.
2. **Упрощайте.** Если есть сомнения, выбирайте самое простое решение. Уильям Оккам (William Occam) декларировал этот принцип еще в XIV веке (так называемая «Бритва Оккама»). Его можно кратко выразить так: «Борись с излишествами». Если какое-то свойство не является абсолютно необходимым, забудьте о нем, особенно если того же эффекта можно добиться комбинированием уже имеющихся свойств.
3. **Всегда делайте четкий выбор.** Если есть несколько способов реализации одного и того же, выбирайте только один из них. Увеличение количества способов приведет к проблемам. В стандартах часто можно встретить несколько опций, режимов или параметров лишь потому, что при разработке несколько авторитетных сторон настаивали на своем. Разработчики должны решительно сопротивляться подобным тенденциям. Надо просто уметь говорить «нет».
4. **Используйте модульный принцип.** Это правило напрямую ведет к идее стеков протоколов, в которых каждый уровень работает независимо от остальных. Таким образом, если обстоятельства требуют изменения одного модуля или уровня, то это не затрагивает другие части системы.
5. **Учитывайте разнородность.** Любая крупная сеть может содержать различные типы оборудования, средства передачи данных и приложения. Сетевая технология должна быть достаточно гибкой, простой и обобщенной, чтобы работать в таких условиях.

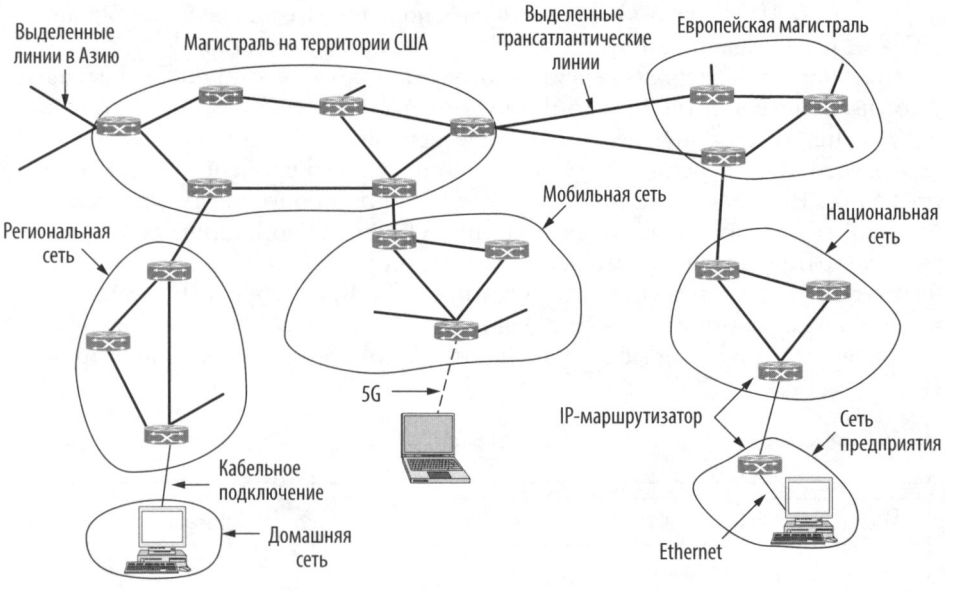
6. **Избегайте статичности свойств и параметров.** Если есть какие-то обязательные параметры (например, максимальный размер пакета), то лучше заставить отправителя и получателя договариваться об их конкретных значениях, чем жестко закреплять их.
7. **Лучшее — враг хорошего.** Очень часто разработчики создают хорошие проекты, но не могут предусмотреть какие-нибудь необычные частные случаи. Не стоит портить то, что в большинстве ситуаций работает нормально. Лучше переложить бремя ответственности за «улучшения» проекта на тех, кто предъявляет свои странные требования.
8. **Будьте строги при отправке, но снисходительны при получении.** Другими словами, передавайте только те пакеты, которые полностью соответствуют всем требованиям стандартов. При этом имейте в виду, что входящие пакеты не всегда идеальны и нужно постараться их обработать.
9. **Продумайте масштабируемость.** Если в сети работают миллионы хостов и миллиарды пользователей, о централизации можно забыть. Нагрузка должна быть распределена максимально равномерно между имеющимися ресурсами.
10. **Помните о производительности и цене.** Никто не будет использовать низкопроизводительную или дорогостоящую сеть.

Перейдем от общих принципов к деталям построения сетевого уровня интернета. Здесь интернет можно рассматривать как набор соединенных друг с другом сетей или автономных систем (АС). Структуры как таковой он не имеет, но все же существует несколько магистралей. Они состоят из высокопроизводительных линий и быстрых маршрутизаторов.

Самые крупные магистрали (к которым необходимо подключиться, чтобы получить доступ к остальной части интернета) называются **сетями Tier 1 (Tier 1 networks)**. К ним присоединены провайдеры, обеспечивающие доступ к интернету для домашних пользователей и предприятий, дата-центров и станций колокации с большим числом серверов, а также для региональных сетей (сетей среднего уровня). Центры обработки данных обслуживают большую часть интернет-трафика. К региональным сетям присоединяются другие интернет-провайдеры, LAN многочисленных университетов и компаний, а также прочие периферийные сети. Эта квазииерархическая структура схематично показана на илл. 5.46.

Вся эта конструкция держится благодаря протоколу **IP (Internet Protocol)**. В отличие от большинства ранних протоколов сетевого уровня, IP с самого начала разрабатывался для межсетевого обмена. Его задача — предоставить уровень обслуживания best effort (то есть без гарантий) при передаче пакетов от отправителя к получателю, независимо от того, находятся они в одной сети или нет.

Обмен данными в интернете происходит следующим образом. Транспортный уровень разбивает потоки данных так, чтобы их можно было отправить в виде IP-пакетов. В теории каждый пакет может достигать 64 Кбайт, но на практике он обычно не превышает 1500 байт (укладывается в один фрейм Ethernet). IP-маршрутизаторы передают пакеты по сети, от одного маршрутизатора к другому,




**Илл. 5.46.** Интернет представляет собой набор соединенных друг с другом сетей

пока они не достигнут места назначения. Там сетевой уровень отдает данные транспортному, а тот помещает их во входной поток принимающего процесса. Когда фрагменты приходят на устройство адресата, сетевой уровень собирает их в исходную дейтаграмму, которая затем передается транспортному уровню.

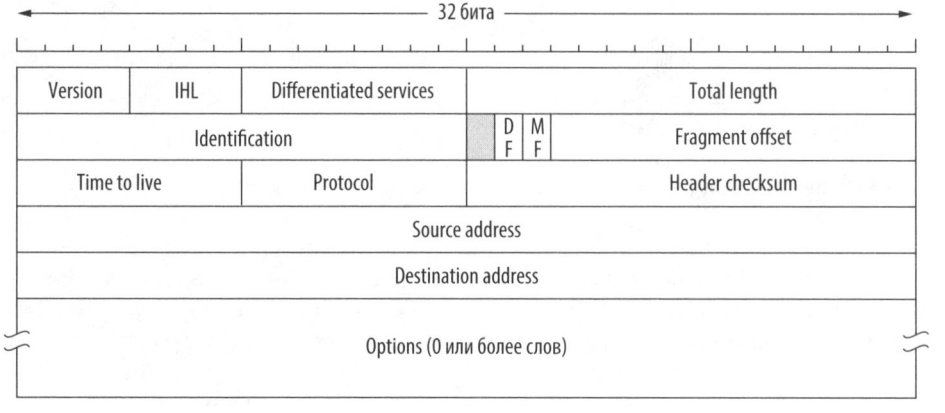
В примере на илл. 5.46 пакет, отправленный хостом домашней сети, пройдет через четыре сети и множество IP-маршрутизаторов, прежде чем доберется до сети предприятия, где расположен хост-получатель. Это обычная практика, а маршруты бывают значительно длиннее. С точки зрения связности интернет является избыточным: магистрали и провайдеры часто соединяются в нескольких точках. Отсюда и множественные пути между хостами. Задача IP — выбрать наилучший маршрут.

## 5.7.1. Протокол IP версии 4

Мы начнем изучение сетевого уровня интернета с формата IPv4-дейтаграмм. Такая дейтаграмма состоит из заголовка и основной части (пользовательских данных). Заголовок содержит фиксированную 20-байтную часть, а также необязательную часть, длина которой варьируется. Формат заголовка показан на илл. 5.47. Биты передаются слева направо и сверху вниз, то есть старший бит поля `Version` (Версия) идет первым. (Такой порядок байтов называется «от старшего к младшему» («big-endian»). На компьютерах с порядком «от младшего к старшему» («little-endian»), например Intel x86, требуется программное преобразование, как при передаче, так и при приеме.) Сегодня уже совершенно

ясно, что для IP лучше было использовать порядок «от младшего к старшему», но на момент создания протокола это было не столь очевидно.

Поле `Version` содержит версию протокола, к которому принадлежит дейтаграмма. Сейчас в интернете доминирует версия 4, поэтому с нее мы и начали обсуждение IP. Указание версии в начале каждой дейтаграммы позволяет переходить от одной версии к другой в течение долгого времени. На самом деле протокол IPv6 (следующая версия IP) разработан более десяти лет назад, но применять его начинают только сегодня. О нем мы поговорим позже в этом разделе. Широкое распространение протокол IPv6 получит, когда у каждого из почти $2^{31}$ жителей Китая будет настольный ПК, ноутбук и IP-телефон. Что касается нумерации, то ничего странного в ней нет: в свое время существовал малоизвестный экспериментальный потоковый протокол реального времени IPv5.




**Илл. 5.47.** Заголовок IPv4

Поскольку длина заголовка является переменной величиной, она указывается в поле `IHL` и выражается в 32-разрядных словах. Минимальное значение длины — при отсутствии поля `Options` (Опции) — равно 5. Максимальное значение этого 4-битного поля равно 15, что соответствует заголовку длиной 60 байт; таким образом, максимальный размер поля `Options` равен 40 байтам. Для некоторых функций, например для записи маршрута, пройденного пакетом, 40 байт недостаточно, и дополнительное поле оказывается бесполезным.

Поле `Differentiated services` (Дифференцированное обслуживание) — одно из немногих полей, смысл которых с годами слегка изменился. Изначально оно называлось `Type of service` (Тип службы). Его задача (была и остается) — определять различные классы обслуживания. Возможны разные комбинации надежности и скорости. Для цифрового голосового сигнала скорость доставки важнее точности. При передаче файла, наоборот, отсутствие ошибок важнее быстрой доставки. В поле `Type of service` 3 бита обозначали приоритет, еще 3 бита показывали, что беспокоит хост больше всего: задержка, пропускная способность или надежность. Никто не знал, что делать со всеми этими битами на

маршрутизаторах, поэтому они не использовались в течение многих лет. Когда появилось дифференцированное обслуживание, IETF сдался и нашел этому полю другое применение. Теперь первые 6 бит задают класс обслуживания; о срочном и гарантированном обслуживании мы уже говорили ранее в этой главе. В последние два бита помещаются явные уведомления о перегрузке, которые обсуждались в разделе 5.3.

Поле `Total length` (Полная длина) содержит длину всей дейтаграммы: заголовок и данные. Максимальная длина дейтаграммы – 65 535 байт. В настоящий момент этого достаточно, однако для будущих сетей могут понадобиться дейтаграммы большего размера.

Поле `Identification` (Идентификация) позволяет хосту-получателю определить, какому пакету принадлежат принятые им фрагменты. Все фрагменты одного пакета содержат одно и то же значение идентификатора.

Далее идет неиспользуемый бит, что достаточно странно, так как место в IP-заголовке крайне ограниченно. В качестве первоапрельской шутки Белловин (Bellovin, 2003) предложил использовать его для обнаружения вредоносного трафика. Это значительно упростило бы защиту сети: пакеты с таким битом можно было бы просто удалять, зная, что они отправлены злоумышленниками. К сожалению, сетевую безопасность невозможно обеспечить столь простым способом, как бы заманчиво идея ни выглядела.

Затем следуют два однобитных поля, относящиеся к фрагментации. Бит `DF` означает «Don't Fragment» («Не фрагментировать»); он запрещает маршрутизатору фрагментировать пакет. Изначально предполагалось, что это поле будет помогать хостам, которые не могут восстановить пакет из фрагментов. Сейчас оно используется при определении путевого значения MTU, которое равно максимальному размеру пакета, передаваемого по пути без фрагментации. Пометив дейтаграмму битом `DF`, отправитель гарантирует, что либо дейтаграмма дойдет единым блоком, либо отправитель получит сообщение об ошибке.

Бит `MF` означает «More Fragments» («Дополнительные фрагменты»). Он устанавливается во всех фрагментах, кроме последнего. По этому биту получатель узнает о прибытии последнего фрагмента дейтаграммы.

Поле `Fragment offset` (Смещение фрагмента) указывает положение фрагмента в исходном пакете. Длина всех фрагментов в байтах, кроме длины последнего фрагмента, должна быть кратной 8. Так как на это поле выделено 13 бит, максимальное количество фрагментов в дейтаграмме равно 8192, что дает максимальную длину пакета вплоть до пределов поля `Total length`. Поля `Identification`, `MF` и `Fragment offset` используются при реализации схемы фрагментации, описанной в разделе 5.5.6.

Поле `TtL`, `Time to live` (Время жизни) представляет собой счетчик, ограничивающий время существования пакета. Изначально предполагалось, что он будет отсчитывать время в секундах. Максимальное значение равнялось 255 с. На каждом маршрутизаторе оно должно было уменьшаться как минимум на единицу плюс время стояния в очереди. Однако на практике отсчитывается количество переходов через маршрутизаторы. Когда значение этого поля достигает нуля, пакет отвергается, а отправителю отсылается пакет с предупреждением. Таким образом удается избежать вечного странствования пакетов, что

может произойти, если таблицы маршрутизаторов по какой-либо причине будут повреждены.

Собрав пакет из фрагментов, сетевой уровень должен решить, что с ним делать. Поле `Protocol` (Протокол) сообщит ему, какому процессу транспортного уровня нужно передать пакет: TCP, UDP или какому-либо другому. Нумерация процессов глобально стандартизирована по всему интернету. Номера протоколов (и некоторые другие) были перечислены в RFC 1700, но теперь они собраны в базе данных по адресу www.iana.org.

Поскольку заголовок содержит важную информацию (в частности, адрес), для ее защиты существует отдельное поле `Header checksum` (Контрольная сумма заголовка). Алгоритм вычисления суммы просто складывает все 16-разрядные полуслова заголовка по мере их поступления с помощью арифметики дополнительных кодов, а затем использует дополнение результата. Алгоритм предполагает, что контрольная сумма заголовка по прибытии должна быть нулевой. Этот метод полезен для обнаружения ошибок, возникающих во время прохождения пакета по сети. Обратите внимание, что значение поля `Header checksum` должно подсчитываться заново на каждом транзитном участке, поскольку хотя бы одно поле постоянно меняется (поле `Time to live`). Для ускорения расчетов применяются некоторые хитрости.

Поля `Source address` (Адрес отправителя) и `Destination address` (Адрес получателя) указывают IP-адреса сетевых интерфейсов отправителя и получателя. Интернет-адреса будут обсуждаться в следующем разделе.

Поле `Options` было разработано, чтобы с появлением новых вариантов протокола не пришлось вносить в заголовок поля, отсутствующие в нынешнем формате. Оно может служить пространством для различного рода экспериментов и испытания новых идей. Кроме того, оно позволяет не включать в стандартный заголовок редко используемую информацию. Размер поля `Options` варьируется. В начале поля всегда находится однобайтный идентификатор. Иногда за ним может располагаться также однобайтное поле длины, а затем один или несколько информационных байтов. В любом случае размер поля должен быть кратен 4 байтам. Изначально было определено пять разновидностей этого поля, перечисленных на илл. 5.48.

| Тип | Описание |
|---|---|
| Security | Указывает уровень секретности дейтаграммы |
| Strict source routing | Задает полный путь следования дейтаграммы |
| Loose source routing | Задает список маршрутизаторов, которые нельзя миновать |
| Record route | Требует, чтобы все маршрутизаторы добавляли свой IP-адрес |
| Timestamp | Требует, чтобы все маршрутизаторы добавляли свой IP-адрес и временную метку |

**Илл. 5.48.** Некоторые типы поля опций IP-дейтаграммы

Опция `Security` (Безопасность) указывает уровень секретности дейтаграммы. Теоретически военный маршрутизатор может использовать это поле, чтобы запретить маршрутизацию пакета по территории определенных государств. На практике все маршрутизаторы игнорируют эту опцию, и ее единственное применение состоит в помощи шпионам при поиске ценной информации.

Опция `Strict source routing` (Строгая маршрутизация от источника) задает полный путь следования дейтаграммы от отправителя до получателя в виде последовательности IP-адресов. Дейтаграмма обязана следовать именно по этому маршруту. Эта опция наиболее полезна потому, что с ее помощью системный администратор может отправить экстренные пакеты в случае повреждения таблиц маршрутизатора или измерить параметры времени и производительности.

Опция `Loose source routing` (Свободная маршрутизация от источника) требует, чтобы пакет прошел через указанный список маршрутизаторов в заданном порядке, но при этом он может проходить через любые другие маршрутизаторы. Обычно указывается лишь небольшое число маршрутизаторов. Например, чтобы заставить пакет, отправляемый из Лондона в Сидней, двигаться не на восток, а на запад, можно указать IP-адреса маршрутизаторов в Нью-Йорке, Лос-Анджелесе и Гонолулу. Эта опция нужна, когда по политическим или экономическим соображениям пакет должен пересечь определенные страны или, напротив, обойти их.

Опция `Record route` (Запись маршрута) требует от всех маршрутизаторов на пути следования пакета добавлять свой IP-адрес к полю `Options`. Это позволяет системным администраторам выявлять ошибки в алгоритмах маршрутизации (к примеру, когда все пакеты, отправляемые из Хьюстона в Даллас, сначала попадают в Токио). Когда появилась сеть ARPANET, пакеты проходили максимум через 9 маршрутизаторов, поэтому 40 байт для этого параметра вполне хватало. Как уже говорилось, сегодня этого недостаточно.

Наконец, опция `Timestamp` (Временная метка) аналогична `Record route`, но кроме 32-разрядного IP-адреса каждый маршрутизатор также записывает 32-разрядную запись о текущем времени. `Timestamp` также применяется в основном для измерения параметров сети.

В последнее время опции IP теряют свою значимость. Маршрутизаторы либо игнорируют их, либо обрабатывают неэффективно, отодвигая в сторону как нечто нетипичное. В общем, они поддерживаются лишь частично и используются редко.

## 5.7.2. IP-адреса

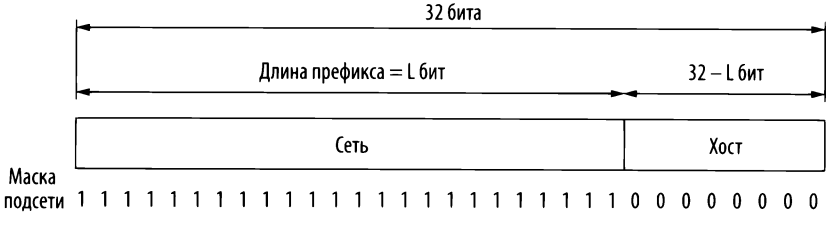
Определяющим признаком IPv4 является его 32-битный адрес. У каждого хоста и маршрутизатора в интернете есть IP-адрес, который может использоваться в полях `Source address` и `Destination address` IP-пакетов. Важно отметить, что IP-адрес на самом деле не имеет отношения к хосту. Он относится к сетевому интерфейсу, поэтому хост, соединенный с двумя сетями, должен иметь два IP-адреса. Однако на практике большинство хостов подключены к одной сети, следовательно, имеют один адрес. Маршрутизаторы, наоборот, обычно имеют несколько интерфейсов, а значит, и несколько IP-адресов.

### *Префиксы*

В отличие от Ethernet-адресов IP-адреса имеют иерархическую организацию. Первая часть 32-битного адреса имеет переменную длину и задает сеть, а последняя указывает на хост. Сетевая часть совпадает для всех хостов одной сети (например, LAN Ethernet). Таким образом, сеть соответствует непрерывному блоку пространства IP-адресов, который называется **префиксом (prefix)**.

IP-адреса обычно записываются в виде четырех десятичных чисел (которые соответствуют отдельным байтам), разделенных точками (**dotted decimal notation**). Например, шестнадцатеричный адрес 80D00297 записывается как 128.208.2.151. Префикс задается наименьшим IP-адресом в блоке и размером блока. Размер определяется числом битов в сетевой части; оставшиеся биты в части хоста могут варьироваться. Таким образом, размер является степенью двойки. По традиции он пишется после префикса IP-адреса в виде слеша и длины сетевой части в битах. В нашем примере префикс содержит $2^8$ адресов и поэтому для сетевой части отводится 24 бита. Пишется это так: 128.208.2.0/24.

Поскольку длина префикса не выводится из IP-адреса, протоколы маршрутизации вынуждены передавать префиксы на маршрутизаторы. Иногда они задаются с помощью указания длины (например, «/16»). Длина префикса соответствует двоичной маске, в которой единицы указывают на сетевую часть. Такая маска называется **маской подсети (subnet mask)**. Выполнение операции AND между маской и IP-адресом позволяет выделить сетевую часть. В нашем примере (илл. 5.49) маска подсети выглядит так: 255.255.255.0.




**Илл. 5.49.** Префикс IP-адреса и маска подсети

У иерархических адресов есть достоинства и недостатки. Важное преимущество префиксов состоит в том, что маршрутизаторы могут направлять пакеты, используя только сетевую часть адреса, поскольку каждой сети соответствует свой уникальный адресный блок. Маршрутизатору не нужно учитывать часть адреса, задающую хост, так как пакеты для всех хостов одной сети передаются в одном направлении. Пакеты передаются на хосты только после того, как попадают в нужную сеть. В результате таблицы маршрутизации значительно сокращаются. Учитывая, что число хостов в интернете превышает миллиард, это очень существенное преимущество, так как иначе таблицы маршрутизации были бы невероятно большими. Но благодаря иерархии им приходится хранить пути лишь для 300 000 префиксов.

Хотя иерархия упрощает маршрутизацию в интернете, у нее есть два недостатка. Во-первых, IP-адрес хоста зависит от его местоположения в сети. Адреса Ethernet можно использовать в любой точке мира, а IP-адрес принадлежит конкретной сети, и поэтому маршрутизаторы могут доставить пакет, предназначенный для данного адреса, только в эту сеть. Чтобы хосты могли перемещаться из одной сети в другую, сохраняя свой IP-адрес, необходимы новые решения, такие как мобильный IP.

Второй недостаток состоит в том, что неправильная иерархия может привести к неэффективному использованию адресов. Если адреса приписываются сетям слишком крупными блоками, много адресов будет выделено, но не будет использоваться. Этот факт не имел бы большого значения, если бы адресов было достаточно, но уже более десяти лет назад стало ясно, что свободное адресное пространство в интернете заполняется с невероятной скоростью. Протокол IPv6 решил эту проблему, но пока он не распространится повсеместно, эффективное выделение адресов будет вызывать серьезные затруднения.

### Подсети

Во избежание конфликтов номера сетей назначаются некоммерческой организацией — **Корпорацией по управлению доменными номерами и IP-адресами (Internet Corporation for Assigned Names and Numbers, ICANN)**. В свою очередь, ICANN передала полномочия по присвоению некоторых частей адресного пространства региональным органам, занимающимся выделением IP-адресов провайдерам и другим компаниям. Именно так компании получают блоки IP-адресов.

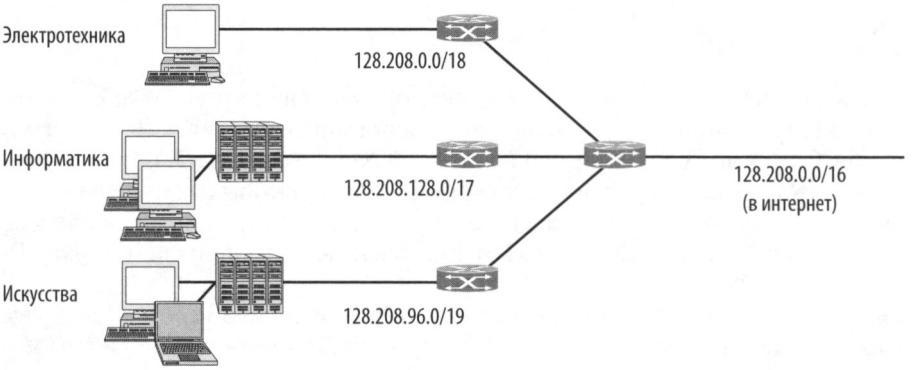
Однако это только начало, так как компаниям по мере роста требуются новые IP-адреса. Как мы уже говорили, маршрутизация по префиксу требует, чтобы у всех хостов сети был один и тот же сетевой номер. Это свойство IP-адресации может вызвать проблемы при увеличении сети. Например, представьте, что университет создал сеть с префиксом /16 из нашего примера. Она используется факультетом информатики в качестве Ethernet. Год спустя подключиться к интернету захотел факультет электротехники, а затем и факультет искусств. Какие IP-адреса будут использовать эти факультеты? Если запросить дополнительные блоки, то получится, что новые сети не будут частью сети университета; к тому же это может быть дорого и неудобно. Более того, префикс /16 позволяет подключить более 60 000 хостов. Возможно, он был выбран с учетом того, что сеть может вырасти. Но пока этого не произошло, выделять университету новые блоки будет неэффективно. Требуется новая архитектура.

Проблема решается предоставлением сети возможности разделения на несколько частей с точки зрения внутренней организации. Это называется **разбиением на подсети (subnetting)**. Сети, полученные в результате разделения крупной сети (например, локальные сети Ethernet), называются **подсетями (subnets)**. Как уже упоминалось в главе 1, новое использование этого термина конфликтует со старым понятием «подсети», обозначающим множество всех маршрутизаторов и линий связи в сети.

На илл. 5.50 показано, как разбиение на подсети может пригодиться в нашем примере. Единая сеть /16 разделена на части. Они не должны быть одинаковыми, но адреса распределяются с учетом длины оставшейся части хоста. В нашем случае половина блока (/17) выделяется факультету информатики, четверть (/18) — факультету электротехники, восьмая часть (/19) — факультету искусств. Оставшаяся восьмая часть не используется. Еще один способ понять, как блок разбивается на части, — посмотреть на префиксы в двоичной нотации.

| | | | | |
|---|---|---|---|---|
| Информатика | 10000000 | 11010000 | 1\|xxxxxxx | xxxxxxxx |
| Электротехника | 10000000 | 11010000 | 00\|xxxxxx | xxxxxxxx |
| Искусства | 10000000 | 11010000 | 011\|xxxxx | xxxxxxxx |

Вертикальная черта (|) обозначает границу между номером подсети и номером хоста.




**Илл. 5.50.** Разделение IP-префикса при разбиении на подсети

Как центральный маршрутизатор узнает, в какую подсеть направить входящий пакет? Именно здесь могут пригодиться специфические свойства префиксов. Конечно, каждый маршрутизатор может иметь таблицу из 65 536 записей с информацией о том, какая исходящая линия ведет к каждому хосту. Но это сведет на нет основное преимущество иерархии, касающееся размеров таблиц маршрутизации. Вместо этого можно сделать так, чтобы маршрутизаторы знали маски этих подсетей.

Когда приходит пакет, маршрутизатор просматривает адрес назначения и определяет, к какой подсети он относится. Для этого он может выполнить операцию AND от этого адреса и маски каждой подсети, сравнивая результат с соответствующим префиксом. Пусть у нас есть пакет с адресом 128.208.2.151. Чтобы проверить, относится ли он к факультету информатики, прибавим к нему (используя логическое AND) маску 255.255.128.0, таким образом отрезав первые 17 бит (то есть 128.208.0.0). Далее сравним полученный результат с префиксом (128.208.128.0). Они не совпадают. Для факультета электротехники аналогичным образом берем первые 18 бит адреса и получаем 128.208.0.0. Это значение

совпадает с префиксом, поэтому пакет передается на интерфейс, ведущий к сети факультета электротехники.

Разбиение на подсети можно впоследствии изменить. Для этого нужно обновить сведения о сетевых масках подсетей на всех маршрутизаторах университета. За пределами сети это разделение незаметно, поэтому нет нужды с появлением каждой подсети обращаться в ICANN или менять какие-либо внешние базы данных.

### *CIDR — бесклассовая междоменная маршрутизация*

Даже при эффективном выделении IP-адресов проблема разрастания таблицы маршрутизации сохраняется.

Если маршрутизатор находится на границе сети какой-либо организации (например, университета), он должен хранить информацию обо всех подсетях, чтобы знать, по какой линии следует передавать пакеты для этой сети. Если адрес назначения находится за пределами данной организации, он может использовать простое правило по умолчанию: отправлять пакеты по линии, соединяющей эту организацию с остальным интернетом. Прочие адреса назначения, очевидно, находятся поблизости.

Маршрутизаторы интернет-провайдеров и магистралей не могут позволить себе такую роскошь. Они должны знать путь к любой сети, поэтому для них не может существовать простого правила по умолчанию. Про магистральные маршрутизаторы говорят, что они находятся в **свободной от умолчаний зоне (default-free zone)** интернета. Никто не знает точно, сколько всего сетей подключено к интернету, но очевидно, что их много — возможно, порядка миллиона. Из них можно составить огромную таблицу. Может, она не слишком большая с точки зрения компьютерных стандартов, но представьте, что маршрутизатор должен просматривать ее при отправке каждого пакета (а за секунду он отправляет миллионы таких пакетов). Для такой скорости обработки требуются специализированные аппаратные средства и быстродействующая память; обычный компьютер для этого не подойдет.

Различные алгоритмы маршрутизации требуют, чтобы маршрутизаторы обменивались информацией о доступных им адресах между собой. Чем больше таблица, тем больше данных необходимо передавать и обрабатывать. С ростом таблицы время обработки увеличивается как минимум линейно. Чем больше данных приходится передавать, тем выше вероятность потери (в лучшем случае временной) части информации, что может привести к нестабильности работы алгоритмов.

Проблему таблиц маршрутизации можно было бы решить, увеличив число уровней иерархии, как это происходит в телефонных сетях. Например, каждый IP-адрес мог бы содержать номер страны, региона, города, сети и хоста. В таком случае маршрутизатор должен знать, как достичь любого другого государства, а также всех регионов своей страны, всех городов в них и каждой сети своего города. К сожалению, этот подход потребует гораздо больше, чем 32 бита, а адресное поле будет использоваться неэффективно (для Лихтенштейна будет выделено столько же разрядов, сколько для США).

К счастью, способ сократить таблицы маршрутизации все же существует. Используем тот же принцип, что и при разбиении на подсети: маршрутизатор может узнавать о расположении IP-адресов по префиксам разной длины. Но вместо разделения сети на подсети мы объединим несколько коротких префиксов в один. Этот процесс называется **агрегацией маршрута (route aggregation)**. Длинный префикс, полученный в результате, называется **суперсетью (supernet)**, в противоположность подсетям с разделением блоков адресов.

При агрегации IP-адреса содержатся в префиксах различной длины. Один и тот же IP-адрес может рассматриваться одним маршрутизатором как часть блока /22 (содержащего $2^{10}$ адресов), а другим — как часть более крупного блока /20 (содержащего $2^{12}$ адресов). Это зависит от информации, которой обладает маршрутизатор. Этот метод работает и для разбиения на подсети и называется **бесклассовой междоменной маршрутизацией (Classless InterDomain Routing, CIDR)**. Последняя на сегодняшний день версия описана в RFC 4632 (Фуллер и Ли; Fuller and Li, 2006). Название иллюстрирует отличие от адресов, кодирующих иерархию с помощью классов, о которой мы в скором времени поговорим.

Чтобы лучше понять алгоритм маршрутизации, рассмотрим пример. Предположим, у нас есть блок из 8192 адресов, начиная с 194.24.0.0. Допустим, что Кембриджскому университету требуется 2048 адресов, и ему выделяются адреса от 194.24.0.0 до 194.24.7.255, а также маска 255.255.248.0. Это будет префикс /21. Затем Оксфордский университет запрашивает 4096 адресов. Так как блок из 4096 адресов должен располагаться на границе, кратной 4096, то ему не могут быть выделены адреса, начинающиеся с 194.24.8.0. Вместо этого он получает адреса от 194.24.16.0 до 194.24.31.255 вместе с маской 255.255.240.0. Наконец, Эдинбургский университет просит выделить ему 1024 адреса и получает адреса от 194.24.8.0 до 194.24.11.255 и маску 255.255.252.0. Все эти присвоенные адреса и маски сведены в таблицу на илл. 5.51.

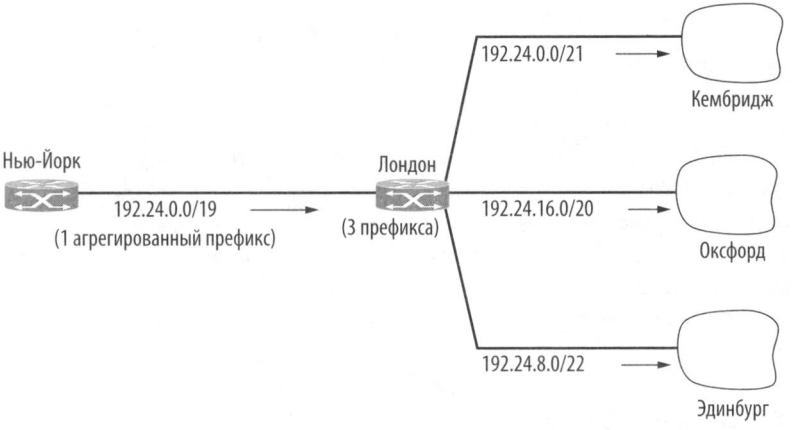
После этого всем маршрутизаторам в свободной от умолчаний зоне сообщаются IP-адреса трех новых сетей. Маршрутизаторы, находящиеся рядом с этими университетами (например, в Лондоне — илл. 5.52), возможно, захотят отправлять пакеты на эти префиксы по разным исходящим линиям. Тогда они запишут эти адреса в свои таблицы маршрутизации.

Теперь посмотрим на эту троицу университетов с точки зрения отдаленного маршрутизатора в Нью-Йорке. Все IP-адреса, относящиеся к этим трем префиксам, должны отправляться из Нью-Йорка (или из США вообще) в Лондон. Процесс маршрутизации в Лондоне узнает об этом, соединяет три префикса в одну агрегированную запись 194.24.0.0/19 и передает ее в Нью-Йорк. Этот префикс содержит 8 Кбайт адресов и включает три университета плюс 1024 свободных адреса. Агрегация позволила объединить три префикса в один, благодаря чему уменьшилось количество префиксов, о которых должен знать маршрутизатор в Нью-Йорке, и число записей в его таблице.

Если агрегация включена, она производится автоматически. Этот процесс зависит от того, где расположены различные префиксы, а не от администратора, который выделяет сетям адреса. В интернете агрегация используется очень активно, снижая размер таблиц маршрутизации примерно до 200 000 префиксов.

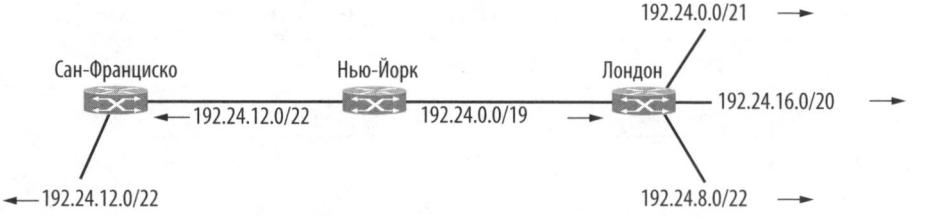
| Университет | Первый адрес | Последний адрес | Количество | Форма записи |
|---|---|---|---|---|
| Кембридж | 194.24.0.0 | 194.24.7.255 | 2048 | 194.24.0.0/21 |
| Эдинбург | 194.24.8.0 | 194.24.11.255 | 1024 | 194.24.8.0/22 |
| (Свободно) | 194.24.12.0 | 194.24.15.255 | 1024 | 194.24.12.0/22 |
| Оксфорд | 194.24.16.0 | 194.24.31.255 | 4096 | 194.24.16.0/20 |

**Илл. 5.51.** Набор присвоенных IP-адресов




**Илл. 5.52.** Агрегация IP-префиксов

Дальше все становится еще интереснее: префиксы могут пересекаться. Согласно правилу, пакеты передаются в направлении самого специализированного блока, или **самого длинного совпадающего префикса (longest matching prefix)**, в котором находится меньше всего IP-адресов. Поведение маршрутизатора в Нью-Йорке (илл. 5.53) показывает, насколько гибким является такой тип маршрутизации. Для отправки пакетов в три университета нью-йоркский маршрутизатор использует один агрегированный префикс. Но что делать, если ранее свободный блок адресов теперь принадлежит сети в Сан-Франциско? Например, нью-йоркский маршрутизатор может хранить четыре префикса: один для Сан-Франциско и три для Лондона. Вместо этого маршрутизация по самому длинному совпадающему префиксу позволяет обойтись двумя префиксами, как показано на рисунке. Один общий префикс используется, чтобы направлять трафик, предназначенный для всего лондонского блока. По второму, более точному, данные передаются в Сан-Франциско. В соответствии с правилом самого длинного совпадающего префикса пакеты, предназначенные для IP-адресов в Сан-Франциско, будут переданы по соответствующей исходящей линии. Пакеты, предназначенные для более крупной сети, будут направлены в Лондон.




**Илл. 5.53.** Маршрутизация по самому длинному совпадающему префиксу на нью-йоркском маршрутизаторе

По сути, CIDR работает так. Когда прибывает пакет, необходимо определить, указан ли нужный адрес в префиксе; для этого просматривается таблица маршрутизации. Может оказаться, что по значению подойдет несколько записей, тогда выбирается самый длинный префикс. То есть если найдено совпадение для маски /20 и /24, то для выбора исходящей линии будет использоваться запись, соответствующая /24. Однако если бы таблица действительно просматривалась запись за записью, этот процесс был бы слишком долгим. Чтобы этого избежать, был разработан сложный алгоритм для ускорения процесса поиска адреса в таблице (Руис-Санчес и др.; Ruiz-Sanchez et al., 2001). В маршрутизаторах для коммерческого использования применяются специальные чипы VLSI, в которые эти алгоритмы встроены на аппаратном уровне.

### *Классовая и специальная адресация*

Чтобы лучше понять преимущества CIDR, рассмотрим метод, который применялся ранее. До 1993 года IP-адреса разделялись на 5 категорий (илл. 5.54). Такое распределение называется **классовой адресацией (classful addressing)**.

**Илл. 5.54.** Форматы IP-адреса

Форматы классов A, B и C позволяют задавать адреса для 128 сетей с 16 млн хостов в каждой, 16 384 сетей с 64 536 хостами или 2 млн сетей (например, LAN) с 256 хостами (хотя некоторые из них могут быть специальными). Предусмотрен класс для многоадресной рассылки (D), при которой дейтаграммы рассылаются одновременно на несколько хостов. Адреса, начинающиеся с 1111, зарезервированы для будущего применения. Неплохо было бы использовать их уже сейчас, когда адресное пространство IPv4 практически закончилось. Но так как в течение долгого времени они были запрещены, многие хосты будут их отвергать — не так просто заставить старый хост изменить свои привычки.

Такая структура является иерархической. Но в отличие от CIDR здесь используются фиксированные размеры блоков адресов. Существует свыше 2 млрд 21-битных адресов, однако организация адресного пространства с использованием классов ведет к потере миллионов адресов. И настоящий виновник этого — класс сетей B. Для большинства организаций 16 млн адресов в классе A — это слишком много, а 256 адресов класса C — слишком мало. Класс B с 65 536 адресами — то, что нужно. В интернет-фольклоре эта дилемма известна как **проблема трех медведей (three bears problem)**. Совсем как в «Сказке про трех медведей»[1] Роберта Саути (Southey, 1848).

На самом деле класс B также слишком велик для большинства организаций. Исследования показали, что более чем в половине случаев сети класса B включают в себя менее 50 хостов. Всем этим компаниям хватило бы и сетей класса C, но почему-то все уверены, что в один прекрасный день маленькое предприятие разрастется настолько, что сеть выйдет за пределы 8-битного адресного пространства хостов. Сейчас, оглядываясь назад, кажется, что лучше было бы использовать в классе C 10-битную адресацию (до 1022 хостов в сети). Возможно, в этом случае многие организации приняли бы разумное решение устанавливать сети класса C, а не B. Таких сетей могло бы быть полмиллиона, а не 16 384, как в случае класса B.

В создавшейся ситуации нельзя обвинять разработчиков интернета за то, что они не увеличили (или не уменьшили) адресное пространство сетей класса B. Когда принималось решение о создании трех классов сетей, интернет был инструментом научно-исследовательских организаций США (и еще нескольких компаний и военных организаций). Никто не предполагал, что интернет станет коммерческой системой коммуникации общего пользования, соперничающей с телефонной сетью. Несомненно, в тот момент кто-то сказал: «В США около 2000 колледжей и университетов. Даже если все они подключатся к интернету и к ним присоединятся университеты из других стран, мы никогда не превысим число 16 000, потому что высших учебных заведений по всему миру не так уж много. Зато мы будем кодировать номер хоста целым числом байтов, что ускорит процесс обработки пакетов» (в то время это выполнялось исключительно программными средствами). Быть может, однажды кто-то воскликнет, обвиняя

[1] Эта сказка (которую многие считают русской народной) на самом деле была написана англичанином Робертом Саути. В ней медведи — крошечный, среднего размера и огромный — были не семейством, а просто друзьями. — *Примеч. ред.*

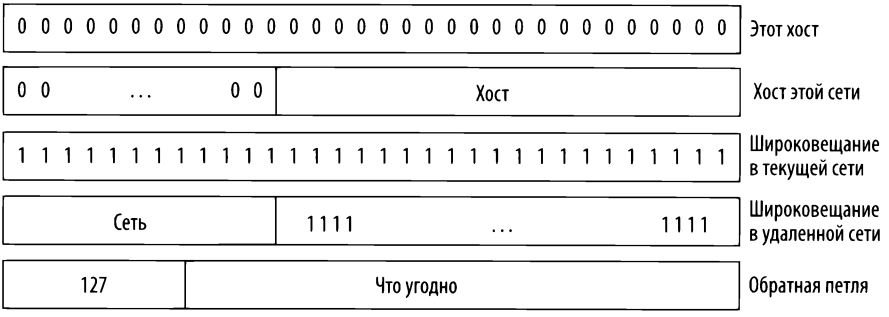
разработчиков телефонной сети: «Вот идиоты! Почему они не включили номер планеты в телефонный номер?» Когда телефонные сети создавались, никто не думал, что это понадобится.

Для решения этих проблем были созданы подсети, обеспечивающие гибкий механизм выделения блоков адресов для отдельных организаций. Позднее в употребление вошла новая схема, позволяющая уменьшить размер таблиц маршрутизации — CIDR. Сейчас биты, указывающие на класс сети (A, B или C), не используются, хотя ссылки на них в литературе все еще встречаются, и довольно часто.

Отказ от классов усложнил процесс маршрутизации. В старой классовой системе маршрутизация происходила следующим образом. По прибытии пакета на маршрутизатор копия IP-адреса, извлеченного из пакета и сдвинутого вправо на 28 бит, давала 4-битный номер класса. Затем с помощью 16-стороннего ветвления пакеты рассортировывались на A, B, C (а также D и E): восемь были зарезервированы для A, четыре для B, два для C. Затем при помощи маскировки по коду каждого класса определялся 8-, 16 или 32-битный сетевой номер, который и записывался с выравниванием по правым разрядам в 32-битное слово. Сетевой номер отыскивался в таблице A, B или C, причем для A и B применялась индексация, а для C — хеш-функция. По найденной записи определялась выходная линия, и по ней пакет отправлялся далее. Этот метод гораздо проще, чем нахождение наиболее длинного совпадающего префикса, при котором простой поиск по таблице невозможен, так как префиксы IP-адресов могут иметь произвольную длину.

Адреса класса D и сейчас применяются в интернете для многоадресной рассылки. Вообще-то правильнее было бы сказать, что они начинают использоваться для этих целей, так как до недавнего времени многоадресная рассылка не была распространена в интернете.

Некоторые адреса имеют особое назначение (илл. 5.55). IP-адрес 0.0.0.0 — наименьший адрес — используется хостом только при включении. Он означает «эта сеть» или «этот хост». IP-адреса с нулевым номером сети обозначают текущую сеть. Такие адреса позволяют устройствам обращаться к хостам собственной сети, не зная ее номера (но они должны знать маску сети, чтобы знать количество используемых нулей). Адрес, состоящий только из единиц, или 255.255.255.255 — наибольший адрес — обозначает все хосты в указанной сети.




**Илл. 5.55.** Специальные IP-адреса

Он обеспечивает широковещание в пределах текущей (обычно локальной) сети. Адреса, в которых указана сеть, но в поле номера хоста одни единицы, обеспечивают широковещание в пределах любой удаленной LAN, соединенной с интернетом. Однако многие сетевые администраторы отключают эту возможность по соображениям безопасности. Наконец, все адреса вида 127.*xx.yy.zz* зарезервированы для тестирования обратной петли. Пакеты, переданные на этот адрес, не попадают на линию, а обрабатываются локально как входные. В результате они могут быть отправленными на хост, даже если отправитель не знает его номера, что полезно для тестирования.

### *NAT — трансляция сетевого адреса*

IP-адреса являются дефицитным ресурсом. У провайдера может быть /16-адрес (бывший класс B), дающий возможность подключить 65 534 хоста. Если клиентов становится больше, возникают проблемы. Так, количество 32-разрядных адресов составляет только $2^{32}$, и все они закончились.

Этот дефицит ресурсов стал причиной появления методов эффективного использования IP-адресов. Согласно одному из них IP-адрес выделяется компьютеру, который в данный момент включен и подключен к сети; когда этот компьютер становится неактивным, его адрес присваивается новому соединению. В этом случае один /16-адрес будет обслуживать до 65 534 активных пользователей.

Такая стратегия хорошо работает в некоторых ситуациях, например, при подключении к сети через телефон, для мобильных и других ПК, где соединение или питание могут временно отсутствовать. Но если заказчиком является организация, эта стратегия, как правило, не подходит. Дело в том, что корпоративные клиенты предпочитают, чтобы компьютеры работали постоянно. Некоторые компьютеры являются рабочими станциями сотрудников, на которых ночью происходит резервное копирование данных, а некоторые служат веб-серверами и поэтому должны незамедлительно отвечать на любой удаленный запрос. Такие организации обладают линией доступа, обеспечивающей постоянное интернет-соединение.

Проблема усугубляется еще и тем, что все больше частных пользователей желают иметь ADSL или кабельное соединение с интернетом, поскольку при этом не нужно (как когда-то) платить за подключение на почасовой основе — взимается только ежемесячная абонентская плата. Многие пользователи имеют дома два и более компьютера (например, по одному на каждого члена семьи) и хотят, чтобы все устройства имели постоянный выход в интернет. Решение таково: необходимо установить (беспроводной) маршрутизатор и объединить все компьютеры в домашнюю LAN. Сам маршрутизатор должен быть подключен к провайдеру. С точки зрения провайдера, в этом случае семья будет выступать в качестве аналога маленькой фирмы с несколькими компьютерами. Добро пожаловать в компанию «Ивановы»! Если использовать обычные схемы, то каждый компьютер будет сохранять свой IP-адрес в течение всего дня. Но для провайдеров с несколькими тысячами клиентов (многие из которых представляют собой целые организации или семьи, также похожие на небольшие организации) такая ситуация недопустима, так как IP-адресов попросту не хватит.

Дефицит IP-адресов не является теоретической проблемой, которая может возникнуть в далеком будущем. Это происходит здесь и сейчас. В долгосрочной перспективе решением будет тотальный перевод всего интернета на протокол IPv6 со 128-битной адресацией. Этот переход действительно постепенно происходит, но процесс идет так медленно, что затянется на годы. А пока необходимо найти какое-нибудь временное решение. Им стал популярный метод **трансляции сетевого адреса (Network Address Translation, NAT)**, описанный в RFC 3022. Суть его мы рассмотрим ниже, а более подробную информацию можно найти в работе Датчера (Dutcher, 2001).

Основная идея NAT состоит в присвоении каждой фирме или семье одного IP-адреса (или, по крайней мере, небольшого числа адресов) для интернет-трафика. *Внутри* абонентской сети каждый компьютер получает уникальный IP-адрес для маршрутизации внутреннего трафика. Однако как только пакет покидает пределы абонентской сети и направляется к провайдеру, выполняется трансляция адреса, при которой уникальный внутренний IP-адрес становится общим публичным IP-адресом. Для реализации этой схемы было создано три диапазона так называемых частных IP-адресов. Они могут использоваться внутри сети по ее усмотрению. Единственное ограничение — пакеты с такими адресами ни в коем случае не должны появляться в самом интернете. Вот эти три зарезервированных диапазона:

| | | |
|---|---|---|
| 10.0.0.0 | – 10.255.255.255/8 | (16 777 216 хостов) |
| 172.16.0.0 | – 172.31.255.255/12 | (1 048 576 хостов) |
| 192.168.0.0 | – 192.168.255.255/16 | (65 536 хостов) |

Итак, первый диапазон может обеспечить адресами 16 777 216 хостов (кроме всех нулей и всех единиц, как обычно), и именно его чаще всего предпочитают абоненты, даже для небольших сетей.

Работа метода NAT показана на илл. 5.56. В пределах территории абонента у каждого компьютера имеется собственный уникальный адрес вида 10.*x.y.z*. Тем не менее перед тем, как пакет выходит за пределы помещения абонента, он проходит через **NAT-блок (NAT box)**, транслирующий внутренний IP-адрес источника (10.0.0.1 на рисунке) в реальный IP-адрес, полученный абонентом от провайдера (198.60.42.12 для нашего примера). NAT-блок обычно представляет собой единое устройство с межсетевым экраном, обеспечивающим безопасность путем строгого отслеживания входящего и исходящего трафика абонентской сети. Межсетевые экраны мы изучим отдельно в главе 8. NAT-блок может быть также интегрирован с маршрутизатором или модемом ADSL.

Мы до сих пор обходили одну маленькую деталь: когда приходит ответ на запрос (например, от веб-сервера), он адресуется 198.60.42.12. Как же NAT-блок узнает, каким внутренним адресом заменить общий адрес компании? В этом и состоит главная проблема применения NAT. Если бы в заголовке IP-пакета было свободное поле, его можно было бы использовать для запоминания адреса того, кто отправил запрос. Но в заголовке остается неиспользованным всего один бит. В принципе, можно было бы создать поле для реального адреса источника, но это потребовало бы изменения IP-кода на всех устройствах интернета. Это не лучший выход, особенно если мы хотим найти быстрое решение проблемы нехватки IP-адресов.

**Илл. 5.56.** Расположение и работа NAT-блока

На самом деле происходит следующее. Разработчики NAT заметили, что большая часть пользовательских данных IP-пакетов — это либо TCP, либо UDP. Когда мы будем рассматривать TCP и UDP в главе 6, мы увидим, что оба формата имеют заголовки, содержащие номера портов источника и получателя. Ниже мы обсудим порты TCP, но с портами UDP происходит то же самое. Номера портов представляют собой 16-разрядные целые числа, показывающие, где начинается и заканчивается TCP-соединение. Место хранения номеров портов используется в качестве поля, необходимого для работы NAT.

Когда процесс хочет установить TCP-соединение с удаленным процессом, он связывается со свободным TCP-портом на своем компьютере. Этот порт становится **портом источника (source port)** и сообщает TCP-коду, куда направлять пакеты данного соединения. Процесс также определяет **порт назначения (destination port)**. Посредством этого порта сообщается, кому отдать пакет на удаленной стороне. Порты с 0-го по 1023-й зарезервированы для хорошо известных служб. Например, 80-й порт используется веб-серверами, соответственно, на них могут ориентироваться удаленные клиенты. Каждое исходящее сообщение TCP содержит информацию о портах источника и назначения. Вместе они служат для идентификации процессов на обоих концах, использующих соединение.

Проведем аналогию, которая прояснит принцип использования портов. Допустим, у компании есть один общий телефонный номер. Когда человек набирает этот номер, он слышит голос оператора, который спрашивает, с кем именно нужно соединиться, и подключает звонящего к соответствующему добавочному номеру. Основной номер является аналогией IP-адреса абонента, а добавочные на обоих концах аналогичны портам. Для адресации портов используется 16-битное поле, которое идентифицирует процесс, получающий входящий пакет.

С помощью поля `Source port` решается проблема отображения адресов. Когда исходящий пакет приходит в NAT-блок, адрес источника вида 10.*x.y.z* заменяется настоящим IP-адресом. Кроме того, поле `Source port` TCP заменяется индексом таблицы перевода NAT-блока, содержащей 65 536 записей. Каждая запись включает исходный IP-адрес и номер исходного порта. Наконец, пересчитываются и вставляются в пакет контрольные суммы заголовков TCP и IP. Поле `Source port` нужно заменять, поскольку устройства с местными адресами 10.0.0.1

и 10.0.0.2 могут случайно пожелать воспользоваться одним и тем же портом (5000-м, например). Так что для однозначной идентификации процесса отправителя единственного поля `Source port` оказывается недостаточно.

Когда пакет прибывает на NAT-блок со стороны провайдера, извлекается значение поля `Destination port` заголовка TCP. Оно используется в качестве индекса таблицы отображения NAT-блока. По найденной в этой таблице записи определяются внутренний IP-адрес и настоящий порт TCP. Эти два значения вставляются в пакет. Затем заново подсчитываются контрольные суммы TCP и IP. Пакет передается на главный маршрутизатор абонента для нормальной доставки с адресом вида 10.*x.y.z*.

Хотя описанная выше схема частично решает проблему нехватки IP-адресов, сетевые пуристы из IP-сообщества рассматривают NAT как некую заразу, распространяющуюся по земле. И их можно понять. Во-первых, сам принцип NAT никак не вписывается в архитектуру IP, которая подразумевает, что каждый IP-адрес уникальным образом идентифицирует только одно устройство в мире. Вся программная структура интернета построена на этом факте. При использовании NAT получается, что тысячи компьютеров могут (и так происходит в действительности) иметь адрес 10.0.0.1.

Во-вторых, NAT нарушает «сквозной» принцип, согласно которому каждый хост должен уметь отправлять пакет любому другому хосту в любой момент времени. Поскольку отображение адресов в NAT-блоке задается исходящими пакетами, входящие пакеты не принимаются до тех пор, пока не отправлены исходящие. На деле это означает, что пользователь домашней сети с NAT может создать TCP/IP-соединение с удаленным сервером, но удаленный пользователь не может подключиться к игровому серверу в домашней сети. Чтобы это сделать, необходимы специальные настройки или методы **обхода NAT (NAT traversal)**.

В-третьих, NAT превращает интернет из сети без установления соединения в нечто, напоминающее сеть, ориентированную на его установление. Проблема в том, что NAT-блок должен поддерживать таблицу отображения для всех соединений, проходящих через него. Запоминать состояние подключений — задача сетей, ориентированных на установление соединения, но никак не сетей без него. Если NAT-блок выходит из строя, а его таблицы отображения теряются, то про все проходящие через него TCP-соединения можно забыть. Без NAT сбой или перезагрузка маршрутизатора не оказывают никакого эффекта на TCP-соединение. Отправляющий процесс просто ждет несколько секунд и отправляет все неподтвержденные пакеты заново. С NAT интернет становится таким же восприимчивым к сбоям, как сеть с коммутацией каналов.

В-четвертых, NAT нарушает одно из фундаментальных правил построения многоуровневых протоколов: уровень $k$ не должен строить никаких предположений относительно того, что именно уровень $k + 1$ поместил в поле `Payload`. Этот принцип обеспечивает независимость уровней друг от друга. Если когда-нибудь на смену TCP придет TCP-2, у которого будет другой формат заголовка (например, 32-битная адресация портов), то NAT потерпит фиаско. Вся идея многоуровневых протоколов состоит в том, чтобы изменения в одном уровне никак не влияли на остальные уровни. NAT разрушает эту независимость.

В-пятых, процессы в интернете вовсе не обязаны использовать только TCP или UDP. Если пользователь устройства *A* придумает новый протокол транспортного уровня для общения с пользователем устройства *B* (например, с целью поддержки какого-либо мультимедийного приложения), то ему придется бороться с тем, что NAT-блок не сможет корректно обработать поле TCP `Source port`.

В-шестых, некоторые приложения предусматривают использование множественных TCP/IP-соединений или UDP-портов. Так, стандартный **протокол передачи файлов (File Transfer Protocol, FTP)** вставляет IP-адрес в тело пакета, чтобы затем получатель извлек его оттуда и обработал. Так как NAT об этом не знает, он не сможет переписать адрес или как-то иначе его обработать. Это означает, что FTP и другие приложения, например протокол интернет-телефонии H.323 (мы рассмотрим его в главе 7), не смогут работать с NAT, если не будут приняты специальные меры. Зачастую NAT можно исправить и заставить его корректно работать в этих случаях, но невозможно дорабатывать его всякий раз, когда появляется новое приложение.

Наконец, поскольку поле TCP `Source port` является 16-разрядным, то одному IP-адресу может соответствовать максимум 65 536 устройств. На самом деле это число несколько меньше: первые 4096 портов зарезервированы для служебных нужд. В общем, если есть несколько IP-адресов, то каждый из них может поддерживать до 61 440 устройств.

Эти и другие проблемы, связанные с NAT, обсуждаются в RFC 2993. Несмотря на все трудности, NAT представляет собой единственный работающий механизм борьбы с дефицитом IP-адресов. Поэтому он широко применяется на практике, особенно в домашних сетях и сетях небольших компаний. Теперь NAT активно использует межсетевые экраны и средства обеспечения конфиденциальности и по умолчанию блокирует все нежелательные входящие пакеты. Поэтому маловероятно, что эта технология исчезнет после внедрения IPv6.

## 5.7.3. Протокол IP версии 6

Протокол IP активно применяется вот уже десятки лет и работает превосходно, что подтверждается экспоненциальным ростом интернета. К сожалению, он стал жертвой собственной популярности: его адресное пространство заканчивается. Даже несмотря на то что CIDR и NAT используют это пространство достаточно экономно, последние адреса IPv4 были выделены 25 ноября 2019 года. Надвигающуюся угрозу заметили почти 20 лет назад, и вопрос о том, что с этим делать, вызвал в интернет-сообществе бурю дискуссий и скандалов.

В этом разделе мы обсудим проблему и несколько предлагаемых решений. Единственной перспективной идеей является переход к более длинным адресам. **IP версии 6 (IP version 6, IPv6)** — новая разработка, призванная заменить IPv4. IPv6 использует 128-битные адреса; в обозримом будущем дополнительного увеличения длины адреса, скорее всего, не потребуется. Но оказалось, что внедрить IPv6 не так просто. Это принципиально новый протокол сетевого уровня, несовместимый с IPv4, несмотря на многочисленные сходства с ним. Кроме того, компании и отдельные пользователи не понимают, почему им надо переходить на

IPv6. На данный момент он охватывает лишь небольшую часть интернета (около 25 %), несмотря на то что признан стандартом еще в 1998 году. В следующие несколько лет нас ждут интересные события. Каждый адрес IPv4 стоит около $19. В 2019 году некий мужчина был осужден за попытку перепродажи на черном рынке 750 000 IP-адресов, общая стоимость которых составляет около $14 млн.

На фоне проблем с адресами вырисовываются и другие вопросы. До недавнего времени интернетом пользовались в основном университеты, высокотехнологичные предприятия и правительственные организации США (особенно Министерство обороны). В середине 90-х годов интерес к интернету резко возрос, и теперь доступ к нему имеет огромное количество людей с принципиально разными требованиями. Во-первых, многочисленные владельцы смартфонов с помощью интернета поддерживают связь с домашними компьютерами. Во-вторых, при неминуемом сближении компьютерной, коммуникационной и развлекательной индустрии очень скоро все телефоны и телевизоры планеты могут стать интернет-узлами. Это приведет к появлению миллиардов устройств, используемых для аудио и видео по запросу. В таких обстоятельствах становится очевидно, что IP должен эволюционировать и стать более гибким.

Предвидя появление этих проблем, в 1990 году IETF начал работу над новой версией IP, которая решила бы вопрос нехватки адресов, а также ряд других задач и в целом была бы более гибкой и эффективной. IETF сформулировал следующие основные цели.

1. Поддерживать миллиарды хостов даже при неэффективном использовании адресного пространства.
2. Сократить таблицы маршрутизации.
3. Упростить протокол для ускорения обработки пакетов маршрутизаторами.
4. Обеспечить более высокий уровень безопасности (аутентификации и конфиденциальности).
5. Уделять больше внимания типу службы, особенно при передаче данных в реальном времени.
6. Упростить работу многоадресных рассылок с помощью указания областей рассылки.
7. Предоставить хосту возможность перемещаться, сохраняя сетевой адрес.
8. Предусмотреть будущее развитие протокола.
9. Обеспечить сосуществование старого и нового протоколов в течение нескольких лет.

Разработка IPv6 дала шанс улучшить характеристики IPv4, исходя из потребностей современного интернета. Чтобы найти протокол, удовлетворяющий всем этим требованиям, IETF опубликовал в RFC 1550 призыв к дискуссиям и предложениям. Был получен двадцать один ответ. В декабре 1992 года были рассмотрены семь серьезных предложений. Их содержание варьировалось от небольших изменений в IP до полного отказа от него и замены совершенно другим протоколом.

В частности, было предложено запустить TCP на базе CLNP, протокола сетевого уровня, разработанного для OSI. Обладая 160-разрядным адресом, CLNP навсегда обеспечил бы достаточное адресное пространство. Он мог бы предоставить адрес каждой молекуле воды в Мировом океане (а это порядка $2^5$ адресов), чтобы они могли создать свою небольшую сеть. Кроме того, это решение объединило бы два основных сетевых протокола. Но тогда пришлось бы признать, что кое-что в модели OSI было сделано правильно, а это утверждение является политически некорректным в интернет-кругах. Протокол CLNP на самом деле очень мало отличается от IP. Окончательный выбор был сделан в пользу протокола, отличающегося от IP значительно сильнее. Еще одним аргументом против CLNP была его слабая поддержка типов служб, которые требовались для эффективной передачи мультимедиа.

Три лучших предложения были опубликованы в журнале *IEEE Network* Дирингом (Deering, 1993), Фрэнсисом (Francis, 1993), а также Кацем и Фордом (Katz and Ford, 1993). После долгих обсуждений, переработок и борьбы за первое место была выбрана модифицированная комбинированная версия Диринга и Фрэнсиса, получившая название **Простого интернет-протокола Плюс (Simple Internet Protocol Plus, SIPP)**. Новому протоколу было дано обозначение **IPv6**.

Протокол IPv6 прекрасно справляется с поставленными задачами. Он обладает достоинствами IP и лишен некоторых его недостатков (либо обладает ими в меньшей степени), к тому же наделен новыми свойствами. Как правило, IPv6 несовместим с IPv4. Зато он сочетается со всеми остальными протоколами интернета, включая TCP, UDP, ICMP, IGMP, OSPF, BGP и DNS. Иногда это требует небольших изменений для работы с более длинными адресами. Основные функции IPv6 представлены далее. Дополнительные сведения о нем можно найти в стандартах RFC 2460–RFC 2466.

Прежде всего, поля адресов у протокола IPv6 больше, чем у IPv4. Их длина составляет 128 бит, что решает основную задачу, поставленную при разработке протокола, — обеспечить практически неограниченный запас интернет-адресов. Мы кратко обсудим адреса чуть позднее.

Второе заметное улучшение IPv6 по сравнению с IPv4 состоит в упрощенном заголовке пакета. Он состоит всего из 7 полей (вместо 13 у IPv4). Таким образом, маршрутизаторы могут быстрее обрабатывать пакеты, что повышает производительность. Краткое описание заголовков будет приведено ниже.

Третье усовершенствование состоит в улучшенной поддержке необязательных параметров. Такое изменение было необходимо, поскольку в новом заголовке требуемые прежде поля стали необязательными (они и так использовались редко). Кроме того, изменился способ представления необязательных параметров. Это упростило для маршрутизаторов пропуск не относящихся к ним параметров и ускорило обработку пакетов.

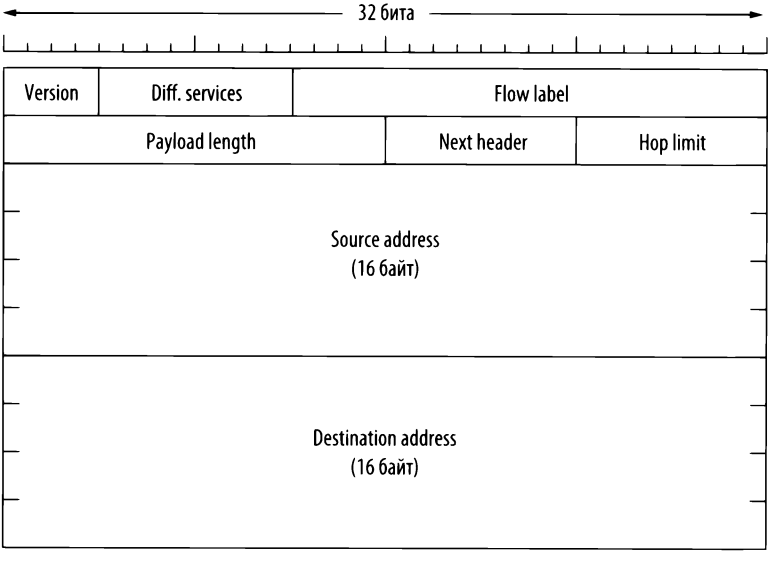
Далее, протокол IPv6 демонстрирует большой шаг вперед в области безопасности. У IETF была полная папка вырезок из газет с сообщениями о том, как 12-летние мальчишки со своего персонального компьютера взломали банк или военную базу. Было ясно, что надо как-то улучшить систему безопасности. Аутентификация и конфиденциальность являются ключевыми чертами нового

IP-протокола. Позже IPv4 был модифицирован, и разница с точки зрения безопасности стала не так уж и велика.

Наконец, в новом протоколе было уделено больше внимания QoS. Различные нерешительные попытки по реализации QoS предпринимались и в прошлом, но рост мультимедийного интернет-трафика требует безотлагательных действий в этой сфере.

### Основной заголовок IPv6

Заголовок IPv6 показан на илл. 5.57. Поле `Version` содержит число 6 для IPv6 (и 4 для IPv4). На период перехода с IPv4 на IPv6, который длится уже более десяти лет, маршрутизаторы по значению этого поля смогут различать пакеты нового и старого стандартов. Заметим, что такая проверка приводит к потере нескольких инструкций в критически важном для производительности тракте. В заголовке с информацией о канале передачи данных обычно указан протокол для демультиплексирования, так что некоторые маршрутизаторы могут пропустить проверку. Например, в Ethernet у поля `Type` (Тип) есть несколько разных значений, определяющих пользовательские данные IPv4- или IPv6-пакета. Бурная дискуссия между сторонниками принципов «делай правильно» и «делай быстро», несомненно, затянется на долгие годы.




**Илл. 5.57.** Фиксированный заголовок IPv6 (обязательные поля)

Поле `Differentiated services` (изначально `Traffic class`, Класс трафика) необходимо, чтобы отличать пакеты с разными требованиями к доставке в реальном времени. Оно используется для обеспечения QoS вместе с архитектурой дифференцированного обслуживания (аналогично одноименному полю IPv4).

Кроме того, так же как и в IPv4, младшие 2 бита отводятся под явные уведомления о перегрузке.

Поле `Flow label` (Метка потока) применяется для того, чтобы отправитель и получатель могли сообщить сети об определенных свойствах пакетов и требованиях к их обработке; при этом между ними устанавливается псевдосоединение. Например, поток пакетов между двумя процессами на разных хостах может иметь строгие требования к задержкам, что потребует резервирования пропускной способности. Поток устанавливается заранее и получает идентификатор. Когда прибывает пакет с ненулевым значением в поле `Flow label`, все маршрутизаторы проверяют свои таблицы, чтобы определить, какой тип особой обработки ему требуется. Таким образом, новый протокол пытается объединить достоинства подсетей различных типов: гибкость дейтаграмм и гарантии виртуальных каналов.

С целью обеспечения QoS каждому потоку присваивается адрес источника, адрес назначения и номер потока. Это означает, что для каждой пары IP-адресов можно создать до $2^{20}$ активных потоков. Кроме того, если два потока приходят с разных хостов, но имеют одинаковую метку, маршрутизатор может отличить их по адресам источника и получателя. Предполагается, что метки потоков выбираются случайно, а не назначаются подряд, начиная с единицы, так что подразумевается, что маршрутизаторы будут их хешировать.

Поле `Payload length` (Длина пользовательских данных) сообщает, сколько байтов следует за 40-байтным заголовком на илл. 5.57. В заголовке IPv4 аналогичное поле называлось `Total length` и определяло весь размер пакета. В новом протоколе 40 байт заголовка учитываются отдельно. Это значит, что теперь пользовательские данные могут занимать 65 535 байт вместо 65 515.

Поле `Next header` (Следующий заголовок) раскрывает секрет упрощения заголовка. Дело в том, что можно использовать дополнительные (необязательные) расширенные заголовки. Это поле сообщает, какой из шести таких заголовков (на текущий момент) следует за основным. В последнем IP-заголовке поле `Next header` информирует, какому обработчику транспортного уровня (то есть TCP или UDP) передать пакет.

Поле `Hop limit` (Максимальное число транзитных участков) не дает пакетам вечно блуждать по сети. Оно имеет практически то же назначение, что и поле `Time to live` в заголовке IPv4. Это поле уменьшается на единицу на каждом транзитном участке. Теоретически в IPv4 это поле должно было содержать секунды существования пакета, однако ни один маршрутизатор не использовал его подобным образом, поэтому имя поля было приведено в соответствие со способом его применения.

Следом идут поля `Source address` и `Destination address`. В изначальном предложении Диринга (протоколе SIPP) использовались 8-байтные адреса, но при рассмотрении проекта было решено, что они закончатся всего через несколько десятилетий, в то время как 16-байтных адресов хватит навсегда. Другие возражали, что 16 байт — это слишком много, третьи настаивали на 20-байтных адресах для совместимости с дейтаграммным протоколом OSI. Еще одна фракция выступала за адреса переменной длины. После продолжительных споров, содержащих непечатную лексику, было решено, что наилучшим компромиссным решением являются 16-байтные адреса фиксированной длины.

Для написания 16-байтных адресов была выработана новая нотация. Адреса в IPv6 записываются в виде восьми групп по четыре шестнадцатеричные цифры, разделенных двоеточиями, например:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Поскольку многие адреса содержат большое количество нулей, разрешены три метода сокращенной записи. Во-первых, могут быть опущены ведущие нули в каждой группе, например 0123 можно записывать как 123. Во-вторых, одна или несколько групп, полностью состоящих из нулей, могут заменяться парой двоеточий. Таким образом, приведенный выше адрес принимает вид:

8000::123:4567:89AB:CDEF

Наконец, адреса IPv4 могут записываться как пара двоеточий, после которой пишется адрес в старом десятичном формате, например:

::192.31.20.46

Возможно, об этом нет необходимости говорить столь подробно, но количество всех возможных 16-байтных адресов очень велико — $2^{128}$, что приблизительно равно $3 \times 10^{38}$, или $7 \times 10^{23}$ IP-адресов на каждый квадратный метр земной поверхности. Те, кто изучал химию, могут заметить, что это число больше числа Авогадро[1]. Хотя в планы разработчиков не входило предоставление IP-адреса каждой молекуле на поверхности земли, они не так далеки от этого.

На практике не все адресное пространство используется эффективно, так же как и комбинации телефонных номеров. Например, номера Манхэттена (код 212) почти полностью заняты, тогда как в штате Вайоминг (код 307) практически не задействованы. В RFC 3194 Дьюранд (Durand) и Уитема (Huitema) приводят свои вычисления. Утверждается, что если ориентироваться на использование телефонных номеров, то даже при самом пессимистическом сценарии все равно получается более 1000 IP-адресов на квадратный метр поверхности планеты (включая сушу и море). При любом вероятном сценарии обеспечиваются триллионы адресов на квадратный метр. Маловероятно, что в обозримом будущем возникнет их нехватка.

Полезно сравнить заголовок IPv4 (см. илл. 5.47) с заголовком IPv6 (илл. 5.57) и посмотреть, что осталось от старого стандарта. Поле `IHL` исчезло, так как заголовок IPv6 имеет фиксированную длину. Поле `Protocol` также было убрано, поскольку поле `Next header` сообщает, что следует за последним IP-заголовком (например, UDP- или TCP-сегмент).

Были удалены все поля, относящиеся к фрагментации, так как в IPv6 используется другой подход к ней. Во-первых, все хосты, поддерживающие IPv6, должны динамически определять нужный размер пакета. Для этого используется метод поиска путевого значения MTU, о котором мы говорили в разделе 5.5.6. Вкратце, когда хост отправляет слишком большой IPv6-пакет вместо того, чтобы его фрагментировать, маршрутизатор, неспособный переслать пакет дальше, возвращает сообщение об ошибке. Получив это сообщение, хост должен прекратить

[1] Число молекул в моле любого вещества или число атомов в моле простого вещества; обозначается как $N_A$, реже $L$. $N_A = 6{,}022\ 140\ 76 \times 1023$ моль$^{-1}$. — *Примеч. ред.*

всю передачу этому адресату. Гораздо правильнее научить все хосты отправлять пакеты требуемого размера, нежели учить маршрутизаторы фрагментировать их на лету. Кроме того, минимальный размер пакета был увеличен с 576 до 1280, чтобы можно было передавать 1024 байт данных и множество заголовков.

Наконец, поле `Checksum` было удалено, так как подсчет контрольной суммы значительно снижает производительность. В настоящее время все шире используются надежные линии связи, а на канальном и транспортном уровнях подсчитываются свои контрольные суммы, так что наличие еще одной проверки не стоит требуемых затрат ресурсов. В результате удаления всех этих функций получился простой, быстрый и в то же время гибкий протокол сетевого уровня с огромным адресным пространством.

### *Заголовки расширений*

Иногда удаленные поля заголовка могут понадобиться, поэтому в протоколе IPv6 была представлена новая концепция необязательного **расширения заголовка (extension header)**. С его помощью можно добавить дополнительную информацию, при этом она эффективно закодирована. На сегодня определены шесть типов таких заголовков (илл. 5.58). Все они необязательны, но если их два и более, то они должны идти сразу за фиксированным заголовком, желательно в указанном порядке.

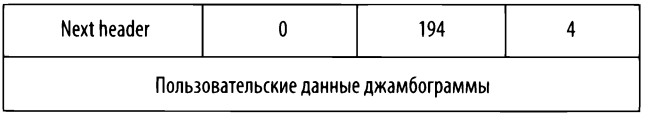
У одних заголовков формат фиксированный, другие могут содержать разное количество полей переменной длины. Для них каждый пункт кодируется в виде набора взаимосвязанных величин (тип, длина, значение). Поле `Type` представляет собой однобайтное поле, обозначающее опцию. Первые два бита указывают маршрутизаторам, которые не знают, как ее обрабатывать. Возможны четыре варианта действий: пропустить опцию, игнорировать пакет, игнорировать пакет и отправить обратно ICMP-пакет, а в случае многоадресной рассылки — игнорировать пакет, но не отвечать ICMP-пакетом (чтобы один неверный пакет не породил миллионы ICMP-донесений).

Поле `Length` (Длина) также имеет размер 1 байт. Оно сообщает, насколько велико значение (от 0 до 255 байт). Поле `Value` (Значение) содержит необходимую информацию размером до 255 байт.

| Заголовок расширения | Описание |
|---|---|
| Hop-by-hop options | Разнообразная информация для маршрутизаторов |
| Destination options | Дополнительная информация для получателя |
| Routing | Частичный список транзитных маршрутизаторов на пути пакета |
| Fragment | Управление фрагментами дейтаграмм |
| Authentication | Проверка подлинности отправителя |
| Encrypted security payload | Информация о зашифрованном содержимом |

**Илл. 5.58.** Заголовки расширения IPv6

Заголовок `Hop-by-hop options` (Опции переходов) содержит информацию, которая должна исследоваться маршрутизаторами на протяжении всего пути следования пакета. На данный момент определен один вариант его использования: поддержка дейтаграмм, превышающих 64 Кбайт. Формат заголовка показан на илл. 5.59. При этом полю `Payload length` в фиксированном заголовке присваивается значение 0.



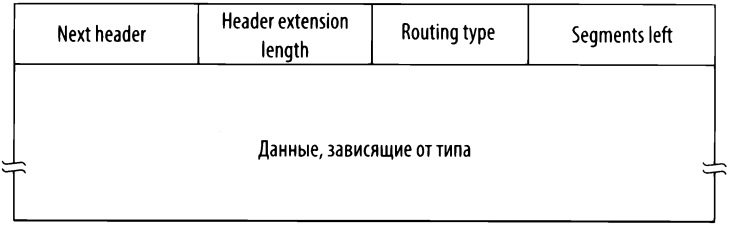

**Илл. 5.59.** Заголовок Hop-by-hop options для крупных дейтаграмм (джамбограмм)

Как и все заголовки расширений, он начинается с байта, означающего тип следующего заголовка. Следующий байт содержит длину заголовка `Hop-by-hop options` в байтах, не считая первых 8 байт, которые являются обязательными. С этого начинаются все расширения.

Следующие 2 байта указывают, что данная опция несет информацию о размере дейтаграммы (код 194) в виде 4-байтного числа. Последние 4 байта содержат размер. Размеры меньше 65 536 не допускаются, так как могут привести к тому, что первый же маршрутизатор проигнорирует данный пакет и отправит ICMP-сообщение. Дейтаграммы с такими заголовками расширений называются **джамбограммами (jumbograms)**. Джамбограммы нужны для суперкомпьютерных приложений, передающих по интернету гигабайты данных.

Заголовок `Destination options` (Опции адресата) предназначен для полей, которые должны интерпретироваться только хостом-получателем. В изначальной версии IPv6 задавались только «нулевые» опции для дополнения этого заголовка до кратности 8 байтам, и сам заголовок не использовался. Это делалось, чтобы новое программное обеспечение маршрутизаторов и хостов могло их обработать, если кто-нибудь подумает об опциях адресата в будущем.

Заголовок `Routing` (Маршрутизация) содержит информацию об одном или нескольких маршрутизаторах, которые следует посетить на пути к получателю. Это весьма напоминает свободную маршрутизацию IPv4, так как перечисленные маршрутизаторы должны быть пройдены строго по порядку, а остальные посещаются попутно. Формат заголовка `Routing` показан на илл. 5.60.




**Илл. 5.60.** Заголовок расширения Routing

Первые четыре байта заголовка Routing содержат четыре однобайтных целых числа. Поля Next header и Header extension length (Длина расширения заголовка) были описаны ранее. В поле Routing type (Тип маршрутизации) описывается формат оставшейся части заголовка. Если он равен 0, это означает, что далее следует зарезервированное 32-разрядное слово, а за ним — некоторое число адресов IPv6. Возможно в будущем по мере необходимости станут разрабатываться новые поля. Наконец, в поле Segments left (Число оставшихся сегментов) указывается, сколько адресов из списка еще нужно посетить. Его значение уменьшается при прохождении каждого адреса. Когда оно достигает нуля, пакет оставляется на произвол судьбы — без дальнейших указаний относительно его пути. Обычно к этому моменту он уже находится близко к получателю, и оптимальный маршрут очевиден.

Заголовок Fragment (Фрагмент) решает проблему фрагментации способом, схожим с протоколом IPv4. Он содержит идентификатор дейтаграммы, номер фрагмента и бит, информирующий о том, является ли этот фрагмент последним. В отличие от IPv4, в протоколе IPv6 фрагментировать пакет может только хост-источник. Маршрутизаторы фрагментировать пересылаемые пакеты не могут. Хотя это изменение можно считать отказом от оригинальной философии IP, оно вполне в духе современного применения IPv4. Вдобавок оно упрощает и ускоряет работу маршрутизаторов. Как уже было сказано, маршрутизатор отвергает слишком большие пакеты, отправляя в ответ ICMP-пакет, указывающий хосту-источнику на необходимость заново передать пакет, разделив его на меньшие фрагменты.

Заголовок Authentication (Аутентификация) предоставляет механизм подтверждения подлинности отправителя пакета. Заголовок Encrypted security payload (Шифрование пользовательских данных) обеспечивает конфиденциальность: прочесть содержимое пакета сможет только тот, для кого он предназначен. Для выполнения этих задач в заголовках используются криптографические методы, которые будут рассмотрены в главе 8.

### Полемика

С учетом открытости, с которой разрабатывался протокол IPv6, а также убежденности большого количества разработчиков в собственной правоте неудивительно, что многие решения принимались в условиях весьма жарких дискуссий. О некоторых из них будет рассказано ниже. Все жуткие подробности вы найдете в соответствующих RFC.

О спорах по поводу длины поля адреса уже упоминалось. В результате было принято компромиссное решение: 16-байтные адреса фиксированной длины.

Другое сражение разгорелось из-за размера поля Hop limit. Некоторые участники дискуссии считали, что ограничение количества транзитных участков числом 255 (которое явно следует из использования 8-битного поля) — большая ошибка. В самом деле, маршруты из 32 транзитных участков уже стали обычным явлением, а через 10 лет в обиход могут войти более длинные пути. Сторонники этого лагеря заявляли, что использование длинных полей адресов было дальновидным решением, в отличие от внедрения крохотных счетчиков

транзитных участков. Самый страшный грех, который, по их мнению, могут совершить специалисты по вычислительной технике, — выделить для чего-нибудь недостаточное количество разрядов.

В ответ им было заявлено, что подобные аргументы можно привести для увеличения любого поля, что приведет к раздутому заголовку. Кроме того, назначение поля Hop `limit` состоит в том, чтобы не допустить слишком долгого странствования пакетов, и 65 535 транзитных участков — это очень много. К тому же по мере роста интернета будет создаваться все большее количество междугородних линий, что позволит передавать пакеты между любыми странами максимум за шесть транзитных пересылок. Если от получателя или отправителя до соответствующего международного шлюза окажется более 125 транзитных участков, то, видимо, что-то не в порядке с магистралями этого государства. В итоге эту битву выиграли сторонники 8-битного счетчика.

Еще одним предметом спора оказался максимальный размер пакета. Обладатели суперкомпьютеров настаивали на размере пакетов, превышающем 64 Кбайт. Когда суперкомпьютер начинает передачу, он занимается серьезным делом и не хочет, чтобы его прерывали через каждые 64 Кбайт. Аргумент против больших пакетов заключается в том, что если пакет размером в 1 Мбайт будет передаваться по линии T1 со скоростью 1,5 Мбит/с, то он займет линию на целых 5 с, что вызовет слишком большую задержку, заметную для интерактивных пользователей. В итоге удалось достичь компромисса: обычные пакеты ограничены размером 64 Кбайт, но с помощью заголовка расширения можно передавать огромные дейтаграммы.

Еще одним спорным вопросом оказалось удаление контрольной суммы IPv4. Кое-кто сравнивал этот ход с удалением тормозов из автомобиля. При этом автомобиль становится легче и может двигаться быстрее, но если на пути что-то встретится, возникнет проблема.

Аргумент против контрольных сумм состоял в следующем. Каждое приложение, которое действительно заботится о целостности своих данных, считает контрольную сумму на транспортном уровне, поэтому еще одна на сетевом является излишней (кроме того, она подсчитывается еще и на уровне передачи данных). К тому же эксперименты показали, что вычисление контрольной суммы составляло основные затраты IPv4. Это сражение было выиграно лагерем противников контрольной суммы, поэтому в IPv6 ее нет.

Вокруг мобильных хостов также разгорелся спор. Если портативный компьютер окажется на другом конце земли, сможет ли он работать с прежним IPv6-адресом или будет вынужден использовать схему с внутренним и внешним агентами? Появилось много желающих создать в протоколе IPv6 явную поддержку мобильных хостов. Эти попытки потерпели поражение, поскольку ни по одному конкретному предложению не удалось достичь консенсуса.

Вероятно, самые жаркие баталии разгорелись вокруг проблемы безопасности. Все были согласны, что это самый главный вопрос. Спорным было то, где и как следует реализовывать безопасность. Аргументом за размещение системы безопасности на сетевом уровне было то, что при этом она становится стандартной службой, которой могут пользоваться все приложения безо всякого

предварительного планирования. Контраргумент заключался в том, что по-настоящему защищенным приложениям подходит лишь сквозное шифрование: оно осуществляется самим источником, а дешифровка — непосредственным получателем. Во всех остальных случаях пользователь зависит от реализации сетевого уровня, которую он не контролирует (а в ней могут содержаться ошибки). Однако приложение может просто отказаться от использования встроенных в IP функций защиты и выполнять всю эту работу самостоятельно. Возражение на этот довод состоит в том, что пользователи, не доверяющие сетям, не хотят платить за неиспользуемую функцию, реализация которой утяжеляет и замедляет работу протокола, даже если сама функция отключена.

Другая сторона вопроса расположения системы безопасности заключается в том, что во многих (хотя, конечно, далеко не во всех) странах приняты строгие экспортные законы, касающиеся шифрования и зашифрованных данных, особенно личной информации. В некоторых странах, в частности во Франции[1] и Ираке, строго запрещено использование шифрования даже внутри страны, чтобы у населения не могло быть секретов от органов власти. В результате любая реализация IP с достаточно мощной криптографической системой не может быть экспортирована за пределы США (и многих других стран). Таким образом, приходится поддерживать два набора программного обеспечения — один для внутреннего использования и другой для экспорта, против чего решительно выступает большинство производителей компьютеров.

Единственный вопрос, по которому не было споров, — никто не рассчитывал, что IPv4-интернет будет отключен в воскресенье, а в понедельник утром ему на смену придет IPv6-интернет. Вместо этого вначале появятся «островки» IPv6, которые будут общаться по туннелям (см. раздел 5.5.4). По мере роста эти островки будут соединяться в более крупные острова. Наконец, все они объединятся, и интернет будет полностью трансформирован.

Так, по крайней мере, выглядел план. Внедрение протокола IPv6 оказалось его ахиллесовой пятой. Он по-прежнему не слишком распространен, хотя уже более 10 лет его полностью поддерживают все основные операционные системы. Обычно операторы сети вводят IPv6, только когда нуждаются в дополнительных IP-адресах. Но надо заметить, что этот протокол постепенно одерживает верх. На него уже приходится большая часть трафика Comcast и примерно четверть трафика Google, так что прогресс налицо.

Чтобы упростить переход к IPv6, разработано множество стратегий. Среди них методы автоматической настройки туннелей, обеспечивающих передачу IPv6-пакетов через сеть IPv4, и технологии, позволяющие хостам автоматически находить конечную точку туннеля. Двухстековые хосты поддерживают как IPv4, так и IPv6 и могут выбирать протокол в зависимости от адреса получателя. Эти стратегии помогут ускорить процесс перехода к IPv6, когда он будет неизбежным. Подробнее об этом — в книге Дэвиса (Davies, 2008).

---

[1] Во Франции шифрование возможно при условии передачи ключа правительственным органам. — *Примеч. науч. ред.*

## 5.7.4. Управляющие протоколы интернета

Помимо IP, используемого для передачи данных, в интернете есть несколько дополнительных управляющих протоколов сетевого уровня, к которым относятся ICMP, ARP и DHCP. В данном разделе мы рассмотрим их по очереди, описывая те версии, которые соответствуют IPv4 (так как именно они распространены сегодня). У ICMP и DHCP существуют аналогичные версии для IPv6; эквивалентом APR является протокол обнаружения соседей (Neighbor Discovery Protocol, NDP).

### *ICMP — протокол управляющих сообщений интернета*

За работой интернета следят маршрутизаторы. Если во время обработки пакета происходит что-то непредвиденное, маршрутизатор сообщает об этом отправителю с помощью **протокола управляющих сообщений интернета (Internet Control Message Protocol, ICMP)**. Он также используется для тестирования интернета. Существует около десятка типов сообщений ICMP. Каждое ICMP-сообщение вкладывается в IP-пакет. Наиболее важные из них перечислены на илл. 5.61.

| Тип сообщения | Описание |
|---|---|
| Destination unreachable | Пакет не может быть доставлен |
| Time exceeded | Время жизни пакета уменьшилось до нуля |
| Parameter problem | Неверное поле заголовка |
| Source quench | Сдерживающий пакет |
| Redirect | Научить маршрутизатор географии |
| Echo and echo reply | Проверить, работает ли машина |
| Timestamp request/reply | То же, что и запрос отклика, но с временной меткой |
| Router advertisement/solicitation | Найти ближайший маршрутизатор |

**Илл. 5.61.** Основные типы ICMP-сообщений

Сообщение DESTINATION UNREACHABLE (адресат недоступен) используется, когда маршрутизатор не может обнаружить пункт назначения или когда пакет с битом *DF* (не фрагментировать) не может быть доставлен, так как путь ему преграждает сеть с ограничением на размер пакетов.

Сообщение TIME EXCEEDED (время истекло) отправляется, когда пакет игнорируется, так как его счетчик `TtL (Time to live)` уменьшился до нуля. Это событие — признак того, что пакеты двигаются по зацикленному пути или что установлено слишком низкое значение таймера.

В 1987 году Ван Джейкобсон (Van Jacobson) предложил хитрый способ использования этого сообщения об ошибке — утилиту **traceroute**. Она находит маршрутизаторы, расположенные в узлах пути от хоста к получателю. При

этом ей не требуется особый уровень поддержки. Метод состоит в отправке на адрес назначения последовательности пакетов с `TtL` 1, 2, 3 и т. д. Счетчики в пакетах достигают нуля по мере прохождения через маршрутизаторы на пути. Каждый маршрутизатор послушно отправляет обратно на хост сообщение TIME EXCEEDED. Так хост определяет их IP-адреса и получает информацию о маршруте. Конечно, TIME EXCEEDED предназначалось не для этого. Но вполне возможно, что это самый полезный инструмент отладки сети за всю историю.

Сообщение PARAMETER PROBLEM (проблема параметра) указывает на то, что обнаружено ошибочное значение в поле заголовка. Это признак наличия ошибки в программном обеспечении хоста, отправившего этот пакет, или промежуточного маршрутизатора.

Сообщение SOURCE QUENCH (подавление источника) ранее использовалось для «усмирения» хостов, которые отправляли слишком много пакетов. Хост, получивший такое сообщение, должен был снизить свою активность. В настоящее время SOURCE QUENCH используется редко, так как при возникновении перегрузки эти пакеты только подливают масла в огонь, еще больше загружая сеть. К тому же неясно, как на них отвечать. Теперь борьба с перегрузкой в интернете осуществляется в основном на транспортном уровне; а сигналом перегрузки является утеря пакетов. В главе 6 мы подробно обсудим, как это происходит.

Сообщение REDIRECT (переадресовать) отсылается хосту-источнику, когда маршрутизатор замечает, что у пакета ошибка в адресе назначения. Таким способом маршрутизатор предлагает хосту обновить путь.

Сообщения ECHO (запрос отклика) и ECHO REPLY (отклик) отправляются, чтобы определить, достижим ли в данный момент конкретный адресат и функционирует ли он. Получив ECHO, хост должен ответить сообщением ECHO REPLY. Эти сообщения используются утилитой **ping**, которая проверяет, работает ли хост и подключен ли он к интернету.

Сообщения TIMESTAMP REQUEST (запрос временной метки) и TIMESTAMP REPLY (отклик с временной меткой) имеют то же назначение, но при этом в ответе проставляется время получения сообщения и время отправления ответа. Этот механизм применяется для измерения производительности сети.

Сообщения ROUTER ADVERTISEMENT (обнаружение маршрутизатора) и ROUTER SOLICITATION (запрос к маршрутизатору) позволяют хостам находить ближайшие маршрутизаторы. Хосту необходимо знать IP-адрес хотя бы одного из них, чтобы он мог передавать пакеты за пределы LAN.

Кроме перечисленных сообщений определены и другие. Их полный список хранится в интернете по адресу www.iana.org/assignments/icmp-parameters.

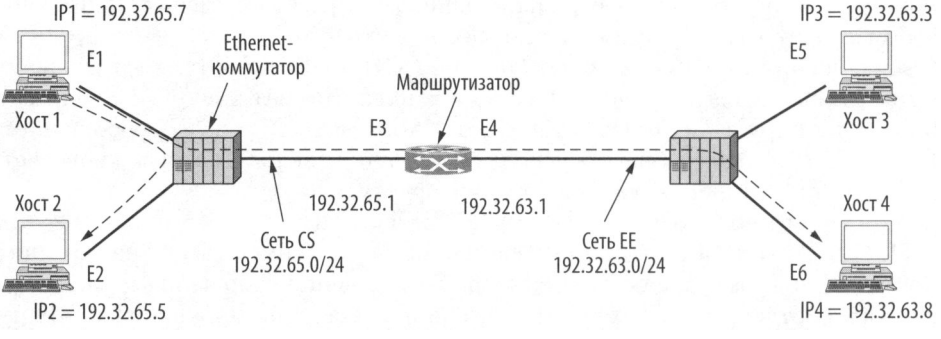
### *ARP — протокол разрешения адресов*

Хотя у каждого устройства в интернете есть один или несколько IP-адресов, их недостаточно для отправки пакетов. Сетевые карты канального уровня, например Ethernet-карты, не понимают интернет-адресов. Каждая когда-либо выпущенная сетевая карта Ethernet имеет 48-разрядный Ethernet-адрес. Производители карт запрашивают у IEEE блок Ethernet-адресов, что гарантирует их

уникальность (это позволяет избежать конфликтов при наличии одинаковых сетевых карт в одной LAN). Сетевые карты отправляют и принимают фреймы, основываясь на 48-разрядных Ethernet-адресах. О 32-разрядных IP-адресах им ничего не известно.

Таким образом, возникает вопрос: как устанавливается соответствие IP-адресов и адресов уровня передачи данных (например, Ethernet-адресов)? Чтобы понять, как это работает, рассмотрим пример на илл. 5.62. Здесь изображен небольшой университет с двумя сетями /24. На рисунке мы видим две коммутируемые сети Ethernet: одна сеть (CS) на факультете кибернетики, с префиксом 192.31.65.0/24, а другая (EE) — на электротехническом факультете, с префиксом 192.31.63.0/24. Они соединены IP-маршрутизатором. У каждого компьютера сети и у каждого интерфейса на маршрутизаторе есть уникальный Ethernet-адрес (на рисунке — от *E1* до *E6*) и уникальный IP-адрес в сети CS или EE.

Рассмотрим, как пользователь хоста 1 передает пакет пользователю хоста 2 в сети CS. Допустим, отправителю известно имя получателя, eagle.cs.uni.edu. Сначала надо найти IP-адрес для хоста 2. Этот поиск осуществляется службой системы имен доменов DNS (Domain Name System), которую мы рассмотрим в главе 7. На данный момент мы просто предположим, что служба DNS возвращает IP-адрес для хоста 2 (192.32.65.5).




| Фрейм | IP-адрес отправителя | Ethernet-адрес отправителя | IP-адрес получателя | Ethernet-адрес получателя |
|---|---|---|---|---|
| От хоста 1 к 2 в сети CS | IP1 | E1 | IP2 | E2 |
| От хоста 1 к 4 в сети CS | IP1 | E1 | IP4 | E3 |
| От хоста 1 к 4 в сети EE | IP1 | E4 | IP4 | E6 |

**Илл. 5.62.** Две коммутируемые LAN Ethernet, соединенные маршрутизатором

Теперь программное обеспечение верхнего уровня хоста 1 создает пакет со значением 192.31.65.5 в поле `Destination address` и передает его IP-программе для пересылки. Программное обеспечение IP может посмотреть на адрес и увидеть, что адресат находится в сети CS (то есть в его собственной сети), но ему нужно как-то определить Ethernet-адрес получателя. Одно из решений состоит в том, чтобы хранить в системе конфигурационный файл, в котором были бы

перечислены соответствия всех локальных IP-адресов Ethernet-адресам. Такое решение, конечно, возможно, но в организациях с тысячами компьютеров обновление этих файлов потребует много времени и подвержено ошибкам.

Более удачное решение заключается в рассылке хостом 1 по сети Ethernet широковещательного пакета с вопросом: «Кому принадлежит IP-адрес 192.32.65.5?» Этот пакет будет получен всеми компьютерами сети CS Ethernet, и каждый из них проверит его IP-адрес. Только хост 2 ответит на вопрос своим Ethernet-адресом *E2*. Таким образом, хост 1 узнает, что IP-адрес 192.32.65.5 принадлежит хосту с Ethernet-адресом *E2*. Протокол, который задает подобный вопрос и получает ответ на него, называется **протоколом разрешения адресов (Address Resolution Protocol, ARP)** и описан в RFC 826. Он работает почти на всех устройствах интернета.

Преимущество ARP над файлами конфигурации заключается в его простоте. Системный администратор должен всего лишь назначить каждому компьютеру IP-адрес и решить вопрос с маской подсети. Все остальное сделает ARP.

Затем программное обеспечение протокола IP хоста 1 создает Ethernet-фрейм для *E2*, помещает в его поле `Payload` IP-пакет, адресованный 192.32.65.5, и передает его по сети Ethernet. IP- и Ethernet-адреса этого пакета приведены на илл. 5.62. Сетевая карта Ethernet хоста 2 обнаруживает фрейм, замечает, что он адресован ей, считывает его и вызывает прерывание. Ethernet-драйвер извлекает IP-пакет из поля `Payload` и передает его IP-программе. Эта программа видит, что пакет адресован правильно, и обрабатывает его.

Существует несколько методов повышения эффективности ARP. Во-первых, компьютер, на котором он работает, может запоминать результат преобразования адреса на случай, если ему придется снова связываться с тем же устройством. В следующий раз он найдет нужный адрес в своем кэше, сэкономив на рассылке широковещательного пакета. Скорее всего, хосту 2 понадобится ответить на пакет, что также потребует от него обращения к ARP для определения адреса отправителя. Этого обращения можно избежать, если отправитель включит в ARP-пакет свои IP- и Ethernet-адреса. Когда широковещательный ARP-пакет прибудет на хост 2, пара (192.32.65.7, *E1*) будет сохранена хостом 2 в ARP-кэше для будущего использования. Более того, эту пару адресов могут сохранить у себя все устройства сети Ethernet.

Чтобы разрешить изменение соответствий адресов, например, если хост использует новый IP-адрес (но Ethernet-адрес остается прежним), записи в ARP-кэше должны устаревать за несколько минут. Существует хороший способ поддержания актуальности информации об адресах в кэше, улучшая производительность. Идея в том, что каждое устройство может рассылать свою пару адресов во время настройки. Обычно эта широковещательная рассылка производится в виде ARP-пакета, запрашивающего свой собственный IP-адрес. Ответа на такой запрос быть не должно, но все устройства могут запомнить эту пару адресов. Это называется **добровольным ARP-сообщением (gratuitous ARP)**. Если ответ все же (неожиданно) придет, это будет означать, что двум компьютерам назначен один и тот же IP-адрес. Они не смогут пользоваться сетью, пока проблема не будет решена системным администратором.

Посмотрим снова на илл. 5.62. Пусть на этот раз хост 1 хочет послать пакет хосту 4 (192.32.63.8) в сети EE. Хост 1 увидит, что IP-адрес получателя не относится к сети CS. Он знает, что такие внешние пакеты нужно передавать на маршрутизатор, который иногда называют **шлюзом по умолчанию (default gateway)**. Принято, что шлюз по умолчанию имеет наименьший адрес сети (198.32.65.1). Но чтобы отправить фрейм на этот маршрутизатор, хост 1 должен знать еще и Ethernet-адрес интерфейса маршрутизатора в сети CS. Поэтому он отправляет широковещательный ARP-пакет для 198.32.65.1 и узнает *E3*. После этого он отправляет фрейм. Аналогичным образом пакеты передаются от одного маршрутизатора к другому на всем пути до места назначения.

Когда сетевая карта Ethernet получает этот фрейм, она передает пакет на обработку программным средствам IP. По сетевым маскам маршрутизатор понимает, что пакет должен быть доставлен на хост 4 в сети EE. Если ему неизвестен Ethernet-адрес хоста 4, он снова использует ARP, чтобы узнать его. В таблице на илл. 5.62 приведен список Ethernet- и IP-адресов из фреймов сетей CS и EE. Обратите внимание на то, что для одного фрейма в разных сетях Ethernet-адреса меняются, а IP-адреса — нет (так как они указывают на конечную точку во всех объединенных сетях).

Существует способ передать пакет от хоста 1 хосту 4 так, чтобы отправитель не знал, что получатель находится в другой сети. Нужно, чтобы маршрутизатор отвечал на ARP-запросы сети CS для хоста 4, передавая при этом свой Ethernet-адрес *E3*. Хост 4 не ответит, так как не увидит широковещательного пакета (маршрутизаторы не переправляют широковещательные пакеты Ethernet-уровня). В результате маршрутизатор получит фреймы для 192.32.63.8 и передаст их в сеть EE. Этот метод называется **ARP-прокси (ARP-proxy)** и используется в особых случаях, когда хосту требуется сымитировать свое присутствие в сети. Например, когда портативный компьютер находится вне домашней сети и хочет, чтобы какой-то другой узел принимал для него пакеты.

### *DHCP — протокол динамической настройки хостов*

ARP (как и другие интернет-протоколы) предполагает, что хосты обладают базовыми сведениями, например, знают свой IP-адрес. Но как хосты получают эту информацию? Можно настраивать их вручную, но это очень трудоемкий процесс, часто ведущий к ошибкам. Есть более удобный способ — **протокол динамической настройки хостов (Dynamic Host Configuration Protocol, DHCP)**.

Каждая сеть должна иметь DHCP-сервер, отвечающий за настройки. При запуске у каждого компьютера есть Ethernet-адрес, встроенный в сетевую карту (или другой адрес канального уровня), но нет IP-адреса. В поисках своего IP-адреса компьютер широковещательным способом рассылает специальный пакет DHCP DISCOVER (Обнаружение DHCP). Он должен прийти на DHCP-сервер. Если сервер не подключен к сети напрямую, пакет будет ретранслирован на DHCP-сервер независимо от того, где он находится.

Когда DHCP-сервер получает пакет, он выделяет свободный IP-адрес и отправляет его обратно с помощью пакета DHCP OFFER (Предложение DHCP)

(который также может ретранслироваться). Даже если у хоста нет IP-адреса, сервер определяет хост по его Ethernet-адресу (который содержится в пакете DHCP DISCOVER).

Возникает вопрос: на какое время можно выдавать в автоматическом режиме IP-адреса из пула? Если хост покинет сеть и не освободит захваченный адрес, этот адрес будет навсегда утерян. С течением времени будет исчезать все больше адресов. Чтобы это предотвратить, IP-адреса выдаются не навсегда, а на определенное время. Это называется **арендой (leasing)**. Перед окончанием срока действия аренды хост отправляет на DHCP-сервер запрос о продлении срока пользования IP-адресом. Если запрос не был сделан или в просьбе было отказано, хост не имеет права продолжать использование выданного ранее адреса.

Протокол DHCP описан в стандартах RFC 2131 и 2132. Он широко применяется в интернете для настройки ряда параметров и приписывания IP-адресов. Помимо сетей предприятий и домашних сетей, DHCP используют провайдеры. С его помощью они настраивают устройства через интернет-соединение, чтобы абонентам не приходилось узнавать эту информацию у своего провайдера по телефону. Чаще всего с помощью DHCP передается маска сети, IP-адрес шлюза по умолчанию, а также IP-адреса DNS и серверов времени. DHCP во многом заменил более ранние протоколы RARP и BOOTP, функциональность которых оставляла желать лучшего.

## 5.7.5. Коммутация меток и MPLS

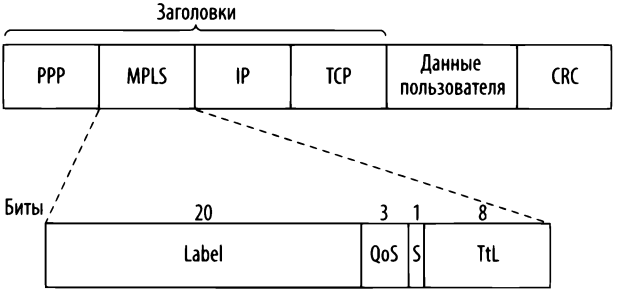
До сих пор, изучая сетевой уровень интернета, мы говорили в основном о пакетах и дейтаграммах, передаваемых IP-маршрутизаторами. Сейчас все чаще используется (особенно провайдерами) еще одна технология, позволяющая передавать интернет-трафик по сети, — **мультипротокольная коммутация меток (MultiProtocol Label Switching, MPLS)**. Она находится в опасной близости к коммутации каналов. Хотя многие участники интернет-сообщества испытывают неприязнь к сетям, ориентированным на установление соединения, похоже, что эта идея снова становится популярной. Как сказал Йоги Берра (Yogi Berra)[1], «и снова это дежавю». Но между созданием маршрутов в интернете и в сетях с установлением соединения есть существенная разница, так что этот метод все же отличается от коммутации каналов.

MPLS присваивает пакету специальную метку, и пересылка производится по ней, а не по адресу. Если добавить метки во внутреннюю таблицу в виде индексов, то чтобы найти нужный выходной канал, достаточно всего лишь ее просмотреть, что существенно ускоряет пересылку. Эта идея легла в основу MPLS. Изначально она разрабатывалась как патентованная технология, известная под разными именами, например, **коммутация меток (tag switching)**. В конечном счете IETF начал стандартизировать эту идею. Она описана в RFC 3031 и многих других RFC. Ее главные преимущества, проверенные временем, — гибкая

[1] Американский бейсболист, известный парадоксальными высказываниями, «йогизмами». — *Примеч. ред.*

маршрутизация и быстрая передача пакетов, позволяющая обеспечить необходимый уровень QoS.

Первая проблема заключается в том, куда поставить метку. IP-пакеты не предназначены для виртуальных каналов, и в их заголовке не предусмотрено место для номеров таких каналов. Значит, нужно добавлять новый заголовок MPLS в начало IP-пакета. На линии между маршрутизаторами используется протокол, включающий заголовки PPP, MPLS, IP и TCP (илл. 5.63).




**Илл. 5.63.** Передача TCP-сегмента с использованием IP, MPLS и PPP

Обычно в заголовок MPLS входит четыре поля, наиболее важное из которых — поле `Label` (Метка), в котором содержится индекс. Поле `QoS` указывает на применяемый класс обслуживания. Поле `S` связано со стеком меток (речь об этом пойдет ниже). Поле `TtL` показывает, сколько еще раз пакет можно переслать. Его значение уменьшается на каждом маршрутизаторе; если оно равно 0, пакет игнорируется. Благодаря этому исключаются бесконечные циклы при сбое маршрутизации.

MPLS располагается между протоколом сетевого уровня IP и протоколом канального уровня PPP. Этот уровень нельзя назвать третьим, так как метки задаются на основе IP-адреса или другого адреса сетевого уровня. Но это и не второй уровень хотя бы потому, что MPLS контролирует передачу пакета на нескольких транзитных участках, а не на одном. Иногда его называют протоколом уровня 2.5. Это яркий пример того, что реальные протоколы не всегда вписываются в идеальную уровневую модель.

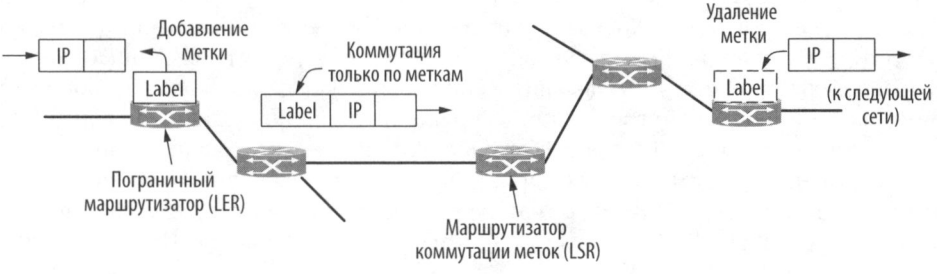
Заголовки MPLS не являются частью пакетов сетевого уровня и не имеют отношения к фреймам канального уровня. Поэтому MPLS — это метод, не зависящий от этих уровней. Помимо прочего, это свойство означает, что можно создать такие коммутаторы MPLS, которые могут по необходимости пересылать как IP-, так и не-IP-пакеты. Именно отсюда следует «мультипротокольность», отраженная в названии MPLS. Также он может передавать IP-пакеты через не-IP-сети.

Когда пакет, расширенный за счет заголовка MPLS, прибывает на **маршрутизатор коммутации меток (Label Switched Router, LSR)**, извлеченная из него метка используется в качестве индекса таблицы. Далее по таблице определяется исходящая линия и значение новой метки. Смена меток используется во всех

сетях с виртуальными каналами. Метки имеют только локальное значение, и два разных маршрутизатора могут снабдить независимые пакеты одной меткой, если их нужно направить на одну и ту же линию третьего маршрутизатора. Поэтому, чтобы метки можно было различить на стороне получателя, их приходится менять при каждом переходе. Мы видели этот механизм в действии — на илл. 5.3. В MPLS используется такой же метод.

Вообще, иногда различают *пересылку (forwarding)* и *коммутацию (switching)*. Под пересылкой при этом понимается поиск адреса, наиболее точно совпадающего с адресом назначения, в таблице маршрутизации, чтобы решить, куда отправлять пакет. Примером IP-пересылки является алгоритм поиска наиболее длинного совпадающего префикса. При коммутации в таблице маршрутизации производится поиск по меткам, извлеченным из пакета. Это проще и быстрее. Но эти определения далеко не универсальны.

Так как большинство хостов и маршрутизаторов не воспринимают MPLS, мы могли бы задаться вопросом, как и когда метки прикрепляются к пакетам. Это происходит в тот момент, когда пакет достигает границы MPLS-сети. **Пограничный маршрутизатор (Label Edge Router, LER)** проверяет IP-адрес назначения и другие поля, определяя, по какому MPLS-пути должен пойти пакет, и присваивает пакету соответствующую метку. По ней пакет пересылается в сети MPLS. На другой границе MPLS-сети метка уже не нужна, и она удаляется, после чего IP-пакет становится открытым для другой сети. Этот процесс показан на илл. 5.64. От традиционных виртуальных каналов он отличается уровнем агрегации. Конечно, можно каждому проходящему через MPLS-сеть потоку дать собственный набор меток. Но более распространенный прием — группировка потоков, заканчивающихся на данном маршрутизаторе или в данной LAN, и использование для них одной метки. Такие потоки принадлежат одному **классу эквивалентности пересылок (Forwarding Equivalence Class, FEC)**. В него входят пакеты, идущие по одному и тому же маршруту и обслуживаемые по одному классу (в терминах дифференцированного обслуживания), поскольку при пересылке все они обрабатываются одинаково.




**Илл. 5.64.** Пересылка IP-пакета через MPLS-сеть

При традиционной маршрутизации с использованием виртуальных каналов невозможно сгруппировать пути с разными адресатами в один виртуальный канал, потому что получатель не сможет их различить. В MPLS пакеты содержат

не только метку, но и адрес назначения, поэтому в конце помеченного пути заголовок с меткой можно удалить, а дальнейшая маршрутизация может быть традиционной — с использованием адреса назначения сетевого уровня.

На самом деле MPLS идет дальше. Этот протокол может работать одновременно на многих уровнях, используя несколько меток. Представьте себе ряд пакетов с различными метками (например, если сеть должна по-разному их обрабатывать). Они должны пройти один и тот же путь до определенного адреса. В таком случае мы можем задать для всех пакетов один путь. Когда пакеты, уже имеющие метку, попадают в стартовую точку пути, в начало пакета записывается новая метка. Это называется стеком меток. Внешняя метка служит проводником пакета по маршруту. В конце пути она удаляется, и оставшиеся метки (если они есть) ведут пакет дальше. Бит S (см. илл. 5.63) позволяет маршрутизатору, удаляющему метку, узнать, есть ли еще метки у пакета. Единичное значение бита сообщает, что метка — последняя в стеке, а нулевое значение говорит об обратном.

Наконец, остается понять, как устроены таблицы передачи по меткам. В этом вопросе MPLS существенно отличается от традиционных схем с виртуальными каналами. В них пользователь, желающий установить соединение, отправляет установочный пакет для создания пути и соответствующей ему записи в таблице. В MPLS этого не происходит; в этом методе вообще отсутствует установочная фаза для каждого соединения (иначе пришлось бы менять слишком большую часть программного обеспечения интернета).

Вместо этого информация, необходимая для передачи, задается специальным управляющий протоколом, который совмещает функции протокола маршрутизации и протокола установления соединения. Он отделен от передачи меток, что позволяет использовать множество разных управляющих протоколов. Один из вариантов этого подхода работает следующим образом. При загрузке маршрутизатора выясняется, для каких путей он является пунктом назначения (например, какие префиксы принадлежат его интерфейсам). Для них создается один или несколько FEC; каждому из них выделяется метка, значение которой сообщается соседям. Соседи заносят эти метки в свои таблицы пересылки и отсылают новые метки своим соседям. Процесс продолжается до тех пор, пока все маршрутизаторы не получат представление о путях. По мере формирования маршрутов могут резервироваться ресурсы, что позволяет обеспечить надлежащий уровень QoS. В остальных вариантах устанавливаются другие пути (например, пути управления трафиком, учитывающие неиспользуемую пропускную способность) или создаются пути «по требованию», предоставляющие нужный уровень QoS.

Основные идеи MPLS просты, однако его детали чрезвычайно запутанны, при этом существует множество вариаций, находящихся в стадии активной разработки. Дополнительную информацию можно найти в книгах Дейви и Фаррела (Davie and Farrel, 2008), а также Дейви и Рехтера (Davie and Rekhter, 2000).

## 5.7.6. Протокол внутреннего шлюза OSPF

Итак, мы изучили процесс передачи пакетов в интернете. Перейдем к новой теме — маршрутизации в интернете. Мы уже упоминали, что интернет состоит из множества независимых сетей или **автономных систем, АС (Autonomous**

**Systems, AS)**, которыми управляют различные организации — компании, университеты, провайдеры. В своей сети организация может использовать собственный алгоритм **внутридоменной** (или **внутренней**) **маршрутизации (intradomain routing)**. Но популярных стандартных протоколов существует совсем немного. В этом разделе мы рассмотрим внутридоменную маршрутизацию и популярный протокол OSPF. Протокол внутридоменной маршрутизации также называют **протоколом внутреннего шлюза (Interior Gateway Protocol, IGP)**. В следующем разделе мы обсудим маршрутизацию между независимыми сетями — **междоменную маршрутизацию (interdomain routing)**. В этом случае все сети должны использовать один и тот же протокол междоменной маршрутизации, или **протокол внешнего шлюза (exterior gateway protocol)**. В интернете применяется протокол пограничной маршрутизации (Border Gateway Protocol, BGP). Мы подробно обсудим его в разделе 5.7.7.

Изначально в качестве протокола внутридоменной маршрутизации использовалась схема маршрутизации по вектору расстояний, основанная на распределенном алгоритме Беллмана — Форда (Bellman — Ford) и унаследованная от ARPANET. В первую очередь это использующийся до сих пор **протокол маршрутной информации (Routing Information Protocol, RIP)**. Он хорошо работал в небольших системах, но по мере увеличения АС стали проявляться его недостатки (к примеру, проблема счета до бесконечности и медленная сходимость), поэтому в мае 1979 года он был заменен протоколом состояния каналов. В 1988 году IETF начал работу над протоколом внутридоменной маршрутизации, учитывающим состояние линий. Он получил название **открытого алгоритма предпочтительного выбора кратчайшего маршрута (Open Shortest Path First, OSPF)** и был признан стандартом в 1990 году. Идея была заимствована из протокола **связи между промежуточными системами (Intermediate System to Intermediate System, IS-IS)**, ставшего стандартом ISO. У OSPF и IS-IS больше сходств, чем различий. Более подробное описание см. в RFC 2328.

Это основные протоколы внутридоменной маршрутизации. Сегодня они поддерживаются многочисленными производителями маршрутизаторов. OSPF чаще используется в корпоративных сетях, IS-IS — в сетях интернет-провайдеров. Ниже дано краткое описание работы протокола OSPF.

Учитывая большой опыт работы с алгоритмами, группа разработчиков согласовывала свои действия с длинным списком требований, которые нужно было соблюсти. Во-первых, алгоритм должен публиковаться в открытых источниках, отсюда буква «O» (Open — открытый) в OSPF. Патентованный алгоритм, принадлежащий одной компании, не подходит. Во-вторых, новый протокол учитывает широкий спектр параметров: физическое расстояние, задержку и т. д. В-третьих, это динамический алгоритм, который автоматически и быстро адаптируется к изменениям топологии.

В-четвертых (это требование впервые было предъявлено именно к OSPF), он должен поддерживать выбор маршрутов, основываясь на типе службы. Новый протокол должен уметь по-разному выбирать путь трафика реального времени и других видов трафика. В то время IP-пакет содержал поле `Type of service`, но ни один из имевшихся протоколов маршрутизации не использовал его. Это поле было и в OSPF, но и здесь оно игнорировалось. Поэтому в результате его

убрали. Возможно, это требование опередило свое время, так как появилось до дифференцированного обслуживания, возродившего классы обслуживания.

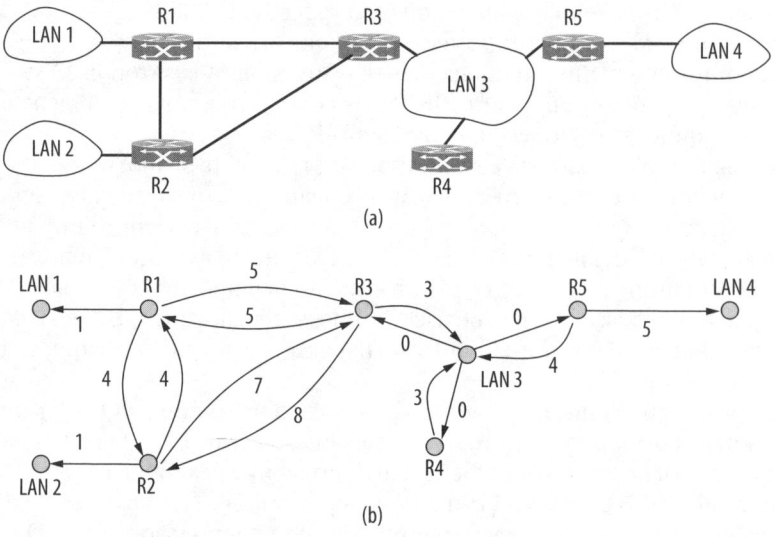
В-пятых, протокол должен уметь распределять нагрузку на линии. Это связано с предыдущим пунктом. Большинство предыдущих протоколов отправляли все пакеты по одному наилучшему маршруту, даже если таких маршрутов два. Следующий по оптимальности маршрут не использовался совсем. Между тем во многих случаях распределение нагрузки по нескольким линиям дает лучшую производительность.

В-шестых, нужна поддержка иерархических систем. К 1988 году интернет вырос настолько, что ни один маршрутизатор не мог вместить сведения о его полной топологии. Таким образом, требовалась разработка нового протокола.

В-седьмых, необходим оптимальный минимум безопасности, который защищает маршрутизаторы от студентов-шутников, присылающих неверную информацию о пути. Наконец, требовалась поддержка для маршрутизаторов, подключенных к интернету по туннелю. Прежние протоколы справлялись с этим плохо.

OSPF поддерживает двухточечные линии (например, SONET) и широковещательные сети (большинство LAN). Он также поддерживает сети с множественными маршрутизаторами, каждый из которых может напрямую соединяться с любым другим (**сети множественного доступа — multi-access networks**), даже если в них невозможно широковещание. Предыдущие протоколы не так хорошо справлялись с этой задачей.

На илл. 5.65 (а) показан пример АС. Здесь не указаны хосты, так как обычно они не играют большой роли в OSPF. Значительно важнее — маршрутизаторы и сети (которые могут содержать хосты). Большинство маршрутизаторов соединены между собой двухточечными линиями, а также с сетями, к хостам




**Илл. 5.65.** Сеть множественного доступа. (а) Автономная система. (б) Представление (а) в виде графа

которых им нужен доступ. Но *R3*, *R4* и *R5* соединены широковещательной LAN, например коммутируемой Ethernet.

В основе работы OSPF лежит обобщенное представление о множестве сетей, маршрутизаторов и каналов в виде направленного графа, каждому ребру которого присвоен весовой коэффициент (расстояние, задержка и т. д.). Двухточечное соединение между двумя маршрутизаторами представляется в виде пары ребер, по одному в каждом направлении. Их веса могут различаться. Широковещательная сеть представляется в виде узла для самой сети, а также в виде узла для каждого маршрутизатора. Ребра, идущие от сетевого узла к узлам маршрутизаторов, обладают нулевым весом. Но они все равно важны, так как без них не будет пути через сеть. У других сетей, состоящих исключительно из хостов, имеются ребра, ведущие только к ним; ребра в обратном направлении отсутствуют. То есть маршруты к хостам возможны, а через них — нет.

Сеть на илл. 5.65 (а) представлена в виде графа на илл. 5.65 (б). По сути, как раз это и делает OSPF. Когда представление в виде графа получено, маршрутизаторы могут вычислить кратчайшие пути до всех остальных узлов с помощью метода, учитывающего состояние линий. Возможно, некоторые пути одинаково короткие. Тогда OSPF запоминает оба пути и использует их для разделения трафика. Этот метод называется **использованием множества равноценных маршрутов (Equal Cost MultiPath, ECMP)**. С его помощью нагрузка распределяется более равномерно.

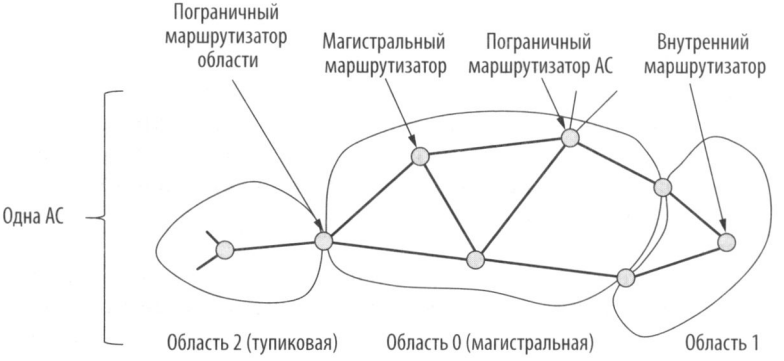
Многие АС в интернете сами по себе довольно крупные, и управлять ими непросто. OSPF позволяет делить их на пронумерованные **области (areas)**, то есть на сети или множества смежных сетей. Области не должны перекрываться, но не обязаны быть исчерпывающими: некоторые маршрутизаторы могут не принадлежать ни к одной из них. Если маршрутизатор полностью принадлежит к какой-то области, он называется **внутренним маршрутизатором (internal router)**. Область — это обобщение отдельной сети. За пределами области видны ее адреса, но не ее топология. Это упрощает масштабирование процесса маршрутизации.

У каждой АС есть **магистральная область (backbone area)**, область 0. Маршрутизаторы, расположенные в ней, называются **магистральными маршрутизаторами (backbone routers)**. Все области соединены с магистралью, например, туннелями, так что по магистрали можно попасть из одной области АС в любую другую. Туннель представляется на графе в виде ребра и обладает стоимостью. Как и в случае других областей, топология магистрали за ее пределами не видна.

Маршрутизатор, подключенный одновременно к двум или нескольким областям, называется **пограничным маршрутизатором области (area border router)**. Он также должен быть частью магистрали. Его задача — собирать сведения об адресах одной области и передавать их другим областям. Эти сведения включают стоимость передачи, но не содержат полной информации о топологии области. Зная стоимость, хосты других областей могут выбрать пограничный маршрутизатор, через который они войдут в эту область. Отсутствие информации о топологии уменьшает трафик и упрощает вычисление кратчайших путей для маршрутизаторов вне данной области. Но если вне области есть только один

пограничный маршрутизатор, эти сведения передавать бессмысленно. Все пути за ее пределы начинаются с указания «Идите на пограничный маршрутизатор». Такая область называется **тупиковой областью (stub area)**.

Последний тип маршрутизаторов — **пограничные маршрутизаторы автономной системы (AS boundary routers)**. Они прокладывают пути на внешние адреса в других АС области. Внешние маршруты становятся адресатами, до которых можно добраться через пограничный маршрутизатор АС с определенными затратами. Внешний путь может проходить один или несколько таких маршрутизаторов. Связь между АС, областями и разными типами маршрутизаторов показана на илл. 5.66. Один маршрутизатор может играть несколько ролей — например, быть и пограничным, и магистральным.




**Илл. 5.66.** Взаимосвязь между АС, магистралями и областями в OSPF

При нормальной работе алгоритма у всех маршрутизаторов, принадлежащих к одной области, имеется одна и та же база данных состояния каналов и один алгоритм выбора кратчайшего пути. Работа маршрутизатора состоит в расчете кратчайшего пути от самого себя до всех остальных маршрутизаторов этой области. Маршрутизатор, соединенный с несколькими областями, должен иметь базы данных для каждой из них. Кратчайший путь для каждой области вычисляется отдельно.

Для отправителя и получателя из одной области выбирается наилучший внутриобластной маршрут (не выходящий за ее пределы). Для отправителя и получателя из разных областей межобластной маршрут проходит от источника к магистрали, затем по магистрали к области назначения, а затем уже к получателю. В результате возникает конфигурация «звезда», в которой магистраль исполняет роль концентратора, а области являются лучами. При выборе пути учитываются издержки, поэтому разные маршрутизаторы могут попадать на магистраль через разные пограничные маршрутизаторы области. Пакеты направляются от отправителя к получателю в исходном виде. Они не помещаются в другие пакеты и не туннелируются (помимо случаев, когда они направляются в области, с которыми магистраль соединена туннелем). Кроме того, пути к внешним адресам сопровождаются внешними издержками

(от пограничного маршрутизатора по внешнему пути) или только внутренними (в пределах АС).

При запуске маршрутизатор отправляет сообщение HELLO (приветствие) по всем своим двухточечным линиям и производит многоадресную рассылку по локальным сетям группе, куда входят все остальные маршрутизаторы. Благодаря ответам маршрутизатор знакомится со своими соседями. Все маршрутизаторы одной LAN являются соседями.

Протокол OSPF работает за счет обмена информацией между смежными маршрутизаторами (а это не то же самое, что соседние). В частности, взаимодействие каждого маршрутизатора со всеми остальными неэффективно. Поэтому один из них становится **назначенным маршрутизатором (designated router, DR)**. Он считается **смежным (adjacent)** со всеми остальными маршрутизаторами данной LAN и обменивается с ними информацией. По сути, он выступает в роли узла, представляющего LAN. Соседние маршрутизаторы, не являющиеся смежными, не обмениваются данными друг с другом. На случай выхода из строя основного назначенного маршрутизатора существует резервный, готовый заменить его в любой момент.

Работая в штатном режиме, каждый маршрутизатор периодически рассылает методом лавинной адресации сообщение LINK STATE UPDATE (Обновление состояния каналов) смежным маршрутизаторам. Сообщение предоставляет сведения о состоянии маршрутизатора и о затратах на топологическую базу данных. В ответ приходят подтверждения LINK STATE ACK, что повышает надежность. Каждое сообщение имеет последовательный номер, чтобы маршрутизатор мог определить, какое из них новее: только что полученное или уже имеющееся. Маршрутизаторы также рассылают эти сообщения, когда включается или выключается канал или изменяется его стоимость.

Сообщение DATABASE DESCRIPTION (Описание базы данных) содержит порядковые номера всех имеющихся у отправителя записей о состоянии линий. Сравнив свои значения с данными источника, получатель определяет, чья информация актуальнее. Эти сообщения передаются при восстановлении линии.

Каждый маршрутизатор может запросить информацию о состоянии линий у своего партнера с помощью сообщения LINK STATE REQUEST (Запрос о состоянии канала). Таким образом каждая пара смежных маршрутизаторов выясняет, чьи сведения более свежие, и по области распространяется самая новая информация. Все сообщения отсылаются в виде IP-пакетов. Пять типов сообщений приведены на илл. 5.67.

Подведем итоги. С помощью алгоритма лавинной адресации каждый маршрутизатор информирует всех остальных внутри области о своих связях с другими маршрутизаторами и сетями, а также о стоимости этих связей. Эта информация позволяет всем маршрутизаторам построить граф своей области и рассчитать кратчайшие пути. В магистральной области происходит то же самое. Кроме того, магистральные маршрутизаторы получают информацию от пограничных маршрутизаторов областей и с ее помощью вычисляют оптимальные пути от каждого магистрального до всех остальных маршрутизаторов.

| Тип сообщения | Описание |
|---|---|
| Hello | Используется для знакомства с соседями |
| Link state update | Сообщает соседям информацию о каналах отправителя |
| Link state ack | Подтверждает обновление состояния каналов |
| Database description | Сообщает о том, насколько свежей информацией располагает отправитель |
| Link state request | Запрашивает информацию у партнера |

**Илл. 5.67.** Пять типов сообщений протокола OSPF

Эта информация рассылается обратно пограничным маршрутизаторам областей, а они распространяют ее в своих областях. Обладая такими данными, внутренний маршрутизатор может найти кратчайший путь до внешнего адресата и оптимальный пограничный маршрутизатор области с выходом к магистрали.

## 5.7.7. Протокол внешнего шлюза BGP

В пределах одной АС наиболее часто используются протоколы OSPF и IS-IS. При выборе маршрута между различными системами используется **пограничный межсетевой протокол (Border Gateway Protocol, BGP)**. Для этого действительно требуется другой протокол, так как цели внутридоменного и междоменного протоколов различны. Внутридоменный протокол должен максимально эффективно передавать пакеты от отправителя к получателю. Политика маршрутизации его не интересует.

Междоменный протокол, напротив, вынужден ее учитывать (Мец; Metz, 2001). Допустим, корпоративная АС хочет принимать и отправлять пакеты на любой интернет-сайт. Но скорее всего, она откажется от роли транзитной сети для пакета, источник и адресат которого находятся за пределами данного государства (даже если через нее проходит кратчайший путь — «Это их проблемы!»). С другой стороны, АС совсем не против передавать трафик других систем (возможно, соседних), которые оплачивают эту услугу. Например, телефонные компании были бы рады предоставлять такой сервис, но только своим клиентам. Протоколы внешнего шлюза вообще и BGP в частности разрабатывались, чтобы обеспечить реализацию разных стратегий выбора маршрута между АС.

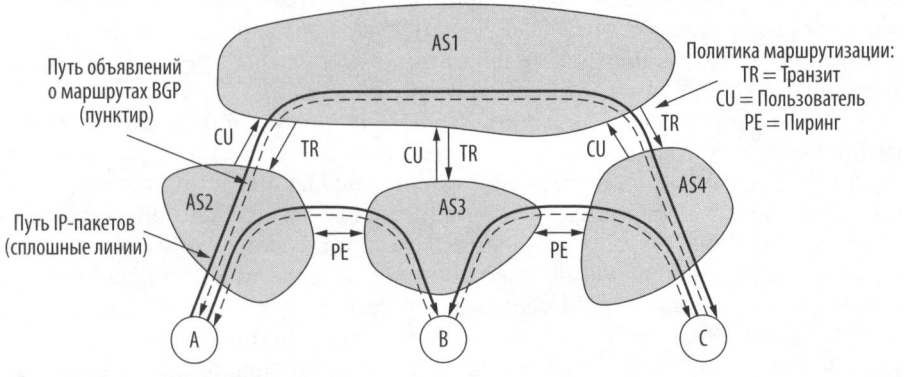
Как правило, стратегии маршрутизации учитывают политический и экономический факторы, а также соображения безопасности. Приведем несколько примеров ограничений при выборе пути.

1. Не передавать коммерческий трафик в образовательные сети.
2. Никогда не передавать трафик Пентагона по маршруту, идущему через Ирак.
3. Использовать TeliaSonera вместо Verizon, потому что это обойдется дешевле.
4. Не пользоваться услугами AT&T в Австралии, так как производительность будет низкой.
5. Трафик компании Apple (или для нее) не должен проходить через Google.

Как вы, наверное, уже поняли, политика маршрутизации может быть очень индивидуальной. Часто она не разглашается, поскольку содержит закрытую корпоративную информацию. Но мы можем описать логику, которая служит отправной точкой при выработке этих правил в компаниях.

Реализация этой политики сводится к принятию решений о том, какой трафик будет передаваться между АС и какие каналы будут использоваться. Один из распространенных вариантов выглядит так: провайдер-клиент платит другому провайдеру за обмен пакетами с любыми получателями в интернете, то есть покупает **транзитные услуги (transit service)**. Эта ситуация аналогична тому, как обычный пользователь домашней сети покупает у провайдера доступ в интернет. Чтобы это сработало, провайдер сообщает абоненту маршруты ко всем интернет-адресам по каналу между ними. Тогда пользователь будет знать путь, по которому можно отправить пакет куда угодно. И наоборот, абонент информирует провайдера о путях ко всем адресам в пределах своей сети. Это позволит провайдеру передавать пакеты только на эти адреса — клиенту не нужен трафик, предназначенный для кого-то еще.

Предоставление транзитных услуг показано на илл. 5.68. Перед нами четыре АС (*AS1*, *AS2*, *AS3*, *AS4*), соединенные между собой. Соединение часто осуществляется через **точку обмена интернет-трафиком (Internet eXchange Point, IXP)**. Это инфраструктура, к которой подключаются многие интернет-провайдеры, чтобы получить доступ к другим провайдерам. *AS2*, *AS3* и *AS4* — клиенты *AS1*. Они покупают у *AS1* транзитные услуги. Таким образом, пакет, отправленный источником *A* на *C*, передается из *AS2* в *AS1*, а затем в *AS4*. Объявления о маршрутах перемещаются в противоположном направлении. Чтобы источник мог передать пакет *C* через *AS1*, *AS4* объявляет *C* в качестве адреса назначения своему провайдеру (*AS1*). Затем *AS1* объявляет путь до *C* остальным своим клиентам, включая *AS2*, чтобы они могли отправлять трафик на *C* через *AS1*.




**Илл. 5.68.** Политика маршрутизации между четырьмя АС

На илл. 5.68 все остальные АС покупают транзитные услуги у *AS1*. Благодаря этому они могут связаться с любым интернет-хостом. Но за эту возможность им приходится платить. Пусть *AS2* и *AS3* обмениваются большим объемом трафика.

Если эти сети соединены, они могут выбрать другую политику — передавать трафик напрямую и бесплатно. Это уменьшит количество трафика, пересылаемого *AS1* от их имени, и сократит их расходы. Такая политика называется **пирингом без взаиморасчетов (settlement-free peering)** или **подключением без взаиморасчетов (settlement-free interconnection)**.

Для реализации пиринга без взаиморасчетов две АС передают друг другу объявления о маршрутах для своих адресов. Это позволяет *AS2* отправлять пакеты *AS3* из *A* в *B* и наоборот. Но следует заметить, что пиринг без взаиморасчетов не транзитивен. На илл. 5.68 сети *AS3* и *AS4* тоже используют политику пиринга, поэтому трафик от *C* к *B* может передаваться напрямую в *AS4*. Но что, если пакет нужно отправить от *C* к *A*? *AS3* объявляет *AS4* маршрут до *B*, но не объявляет маршрута до *A*. Поэтому трафик не сможет пройти из *AS4* в *AS3*, а затем в *AS2*, хотя физический путь существует. Но именно это и выгодно *AS3*. Она хочет обмениваться трафиком с *AS4*, но не хочет, чтобы *AS4* передавала через нее трафик для остального интернета. Вместо этого *AS4* придется пользоваться транзитными услугами *AS1*, которая, кстати, и передаст пакет от *C* к *A*.

Теперь, разобравшись с транзитными услугами и пирингом без взаиморасчетов, мы видим, что у хостов *A*, *B* и *C* есть транзитные соглашения. К примеру, клиент *A* должен покупать интернет-доступ у *AS2*. *A* может быть представлен одним компьютером или сетью организации с многочисленными LAN. Но в любом случае клиент *A* не нуждается в BGP, так как он является **тупиковой сетью (stub network)**, соединенной с остальным интернетом одной линией. Отправлять пакеты за пределы сети можно только по этой единственной линии к *AS2*. Такой маршрут можно задать в качестве пути по умолчанию. Именно поэтому клиенты *A*, *B* и *C* на рисунке не представлены в качестве АС, участвующих в междоменной маршрутизации.

Коммерческие договоренности о транзите и пиринге без взаиморасчетов реализуются за счет объединения двух принципов маршрутизации — приоритизации путей к получателю и фильтрации способов их объявления в соседних сетях. Приоритизация состоит в следующем: в первую очередь обрабатываются маршруты для платных пользователей, затем — для узлов пиринговой сети без взаиморасчетов и, наконец, — для сетей провайдеров. За этим стоит простая логика: АС предпочитает передавать трафик по тем маршрутам, где за него платят, а не по тем, где за него приходится платить. По тем же причинам АС объявляет клиентам все свои маршруты, но не должна при этом повторно объявлять маршруты, полученные от узла пиринговой сети без взаиморасчетов или от транзитного провайдера, остальным узлам пиринговой сети или провайдерам. АС также реализуют и несколько других коммерческих договоренностей, включая **платный пиринг (paid peering)**, при котором одна АС платит другой за доступ к маршрутам, полученным от ее клиентов. Это похоже на пиринг без взаиморасчетов, но с денежными выплатами. Также возможны договоренности о **частичном транзите (partial transit)**, при котором одна АС платит другой за маршруты к некоторому подмножеству получателей в интернете.

Иногда сети компаний подключены сразу к нескольким интернет-провайдерам. Это делается с целью повышения надежности, так как при отказе одного провайдера трафик может быть передан через другого. Этот метод называется

**многолинейным подключением (multihoming)**. В таком случае компания, скорее всего, будет использовать протокол междоменной маршрутизации (например, BGP), чтобы АС знали, через какого провайдера следует передавать трафик.

Существует множество вариантов политики маршрутизации, и это хорошая иллюстрация того, как деловые отношения и контроль над объявлениями маршрутов рождают новые принципы. Далее мы обсудим, как BGP-маршрутизаторы обмениваются объявлениями о маршрутах и выбирают пути для передачи пакетов.

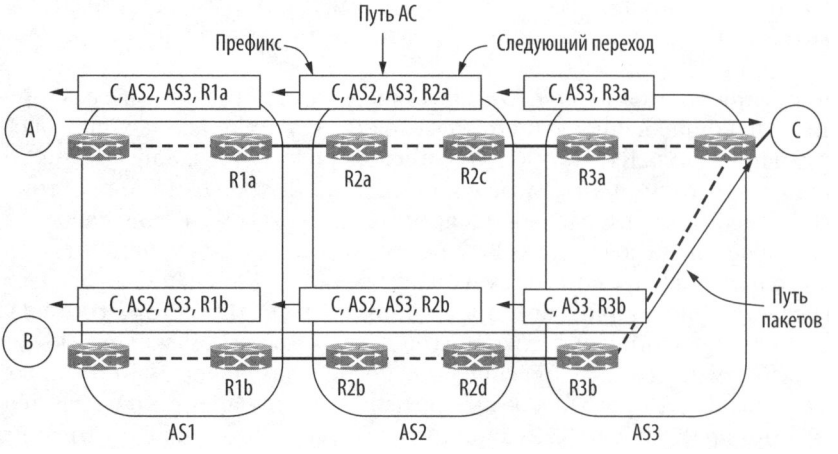
BGP, по сути, является вариантом протокола маршрутизации по вектору расстояний, однако он значительно отличается от других подобных протоколов, например протокола RIP. Как мы убедились, при выборе маршрута вместо минимального расстояния учитывается политика. Другое отличие состоит в следующем. Вместо того чтобы периодически сообщать всем соседям свои расчеты стоимости передачи до каждого возможного адресата, BGP-маршрутизатор передает им точную информацию о тех маршрутах, которые он использует. Этот подход называется **протоколом векторов маршрутов (Path Vector Protocol)**. Путь включает следующий маршрутизатор (он не должен быть смежным и может находиться в другой части сети провайдера) и последовательность автономных систем, или **путь АС (AS path)**, через которые проходит маршрут (АС даются в обратном порядке). Пары BGP-маршрутизаторов взаимодействуют друг с другом, устанавливая TCP-соединения. Так обеспечивается надежная связь и скрываются детали устройства сети, по которой проходит трафик.

На илл. 5.69 показано, как маршрутизаторы обмениваются объявлениями о BGP-путях. Мы видим три АС; средняя предоставляет транзитные услуги провайдерам слева и справа. Объявление о маршруте к префиксу *C* начинается на *AS3*. Когда оно доходит до *R2c* (вверху), то состоит из *AS3* и следующего маршрутизатора *R3a*. Внизу этот маршрут имеет тот же путь АС, но другой следующий маршрутизатор, так как он пришел по другому каналу. Это объявление распространяется дальше и пересекает границу *AS1*. На маршрутизаторе *R1a* (вверху) путь АС состоит из *AS2* и *AS3*, а следующий маршрутизатор при этом — *R2a*.

Указание полного пути при прохождении маршрута упрощает обнаружение и устранение маршрутных петель. Главный принцип состоит в том, что каждый маршрутизатор, передающий маршрут за пределы своей АС, добавляет в его начало номер АС. (Именно поэтому список имеет обратный порядок.) Когда маршрутизатор получает маршрут, он проверяет путь АС на наличие номера его собственной АС. Положительный ответ означает, что в маршруте есть петля; в таком случае объявление отвергается. Однако несмотря на такие меры предосторожности, в конце 1990-х было обнаружено, что BGP все же сталкивается с вариантом проблемы счета до бесконечности (Лейбовиц и др.; Labovitz et al., 2001). Из-за медленной конвергенции маршрутов могут возникать временные петли, пусть и ненадолго.

Задавать путь с помощью списка АС — довольно грубый способ. АС может быть как небольшой компанией, так и международной магистральной сетью. Это нельзя определить по маршруту. BGP даже не пытается этого сделать, так как в различных АС могут использоваться разные внутридоменные протоколы,

из-за чего стоимость передачи невозможно сравнить. Но если бы это было возможно, с большой вероятностью возникла бы другая проблема: АС зачастую скрывают данные о своих внутренних параметрах. Это одно из основных отличий междоменной маршрутизации от внутридоменной.




**Илл. 5.69.** Распространение объявлений о BGP-маршруте

Итак, мы обсудили, как объявление маршрута передается по каналу между двумя интернет-провайдерами. Но кроме этого, нужно уметь отправлять BGP-маршруты в другие части сети провайдера, чтобы оттуда они могли быть переданы другим провайдерам. С такой задачей может справиться внутридоменный протокол. Но поскольку BGP отлично масштабируется для крупных сетей, в этих случаях часто используется его вариант. Он называется **внутренним BGP (internal BGP, iBGP)**, в противоположность обычному, или **внешнему BGP (external BGP, eBGP)**.

Правило распространения маршрутов внутри сети провайдера звучит так: маршрутизатор, расположенный на границе сети, узнает обо всех маршрутах, о которых знают другие пограничные маршрутизаторы (в целях согласованности их действий). Если какому-то пограничному маршрутизатору становится известен префикс к IP 128.208.0.0/16, все другие маршрутизаторы тоже о нем узнают. Тогда до этого префикса можно будет добраться из любой точки сети, при этом не важно, как пакет попал в эту сеть.

Во избежание перегруженности этот процесс не показан на илл. 5.69. Но можно привести такой пример: маршрутизатор *R2b* узнает, что он может достичь *C* либо через *R2c* (вверху), либо через *R2d* (внизу). Данные о следующем маршрутизаторе будут обновляться, пока маршрут будет перемещаться по сети провайдера. Благодаря этому маршрутизаторы на разных концах сети будут знать, через какой маршрутизатор можно выйти за пределы сети с другой стороны. Если посмотреть на крайние левые маршруты, то можно увидеть, что для них следующий маршрутизатор находится в той же сети, а не в соседней.

Теперь можно перейти к обсуждению важного вопроса: как BGP-маршрутизаторы выбирают маршрут до адресата? Каждый BGP-маршрутизатор узнает этот маршрут от соединенного с ним маршрутизатора в соседней сети провайдера, а также от других пограничных маршрутизаторов (которые узнают другие возможные пути от своего соседнего маршрутизатора). Далее каждый маршрутизатор выбирает наилучший маршрут. В конечном итоге получается, что политику выбора маршрута определяет провайдер. Однако это слишком общее и не вполне удовлетворительное объяснение, так что мы опишем несколько возможных стратегий.

Первая стратегия заключается в том, что маршруты через одноранговые сети обладают большим приоритетом, чем транзитные маршруты. Первые бесплатны, вторые — нет. Существует еще один похожий вариант: наибольший приоритет присваивается маршрутам клиента. При этом трафик передается напрямую тому, кто за него платит.

Другая стратегия по умолчанию использует следующее правило: чем короче путь *AC*, тем он лучше. Это спорное утверждение, так как путь через три маленькие *AC* может быть короче, чем путь через одну большую. Но в среднем выбор кратчайших путей дает неплохие результаты, так что решение за вами.

Наконец, существует стратегия, согласно которой выбирается самый дешевый маршрут в сети провайдера. Пример ее реализации показан на илл. 5.69. Пакеты из *A* в *C* покидают *AS1* через верхний маршрутизатор *R1a*. Пакет из *B* выходит через нижний маршрутизатор *R1b*. Это происходит, поскольку *A* и *B* выбирают наименее затратный способ выхода из *AS1*. А поскольку они расположены в разных частях сети провайдера, наилучшие пути для них различаются. То же самое происходит, когда пакеты проходят через *AS2*. На последнем участке *AS3* должна переправить пакет из *B* по своей сети.

Эта стратегия называется **ранним выходом (early exit)**, или **методом «горячей картошки» (hot-potato routing)**. Интересно, что при такой маршрутизации пути обычно несимметричны. Рассмотрим маршрут пакета, переданного из *C* в *B*. Пакет сразу же выйдет из *AS3* через верхний маршрутизатор. При переходе из *AS2* в *AS1* он останется наверху, а уже потом будет долго путешествовать внутри *AS1*. Это зеркальное отражение пути от *B* к *C*.

Из всего этого следует, что каждый BGP-маршрутизатор выбирает свой оптимальный путь из нескольких возможных. Но вопреки первоначальным ожиданиям, нельзя сказать, что BGP отвечает за маршруты между разными *AC*, а OSPF — за маршруты внутри одной *AC*. BGP и протокол внутреннего шлюза связаны более сложным образом. В частности, это значит, что BGP может найти наилучшую точку выхода за пределы сети провайдера. При этом выбор точки будет отличаться для разных уголков сети, как в случае метода «горячей картошки». Также это значит, что BGP-маршрутизаторы из разных частей одной *AC* могут использовать разные пути *AC* к одному и тому же получателю. Провайдер должен позаботиться о том, чтобы при такой свободе выбора маршруты были совместимы, — это вполне осуществимо на практике.

Все эти стратегии реализуются с помощью различных настроек и конфигураций протоколов. Важно понять основной принцип, позволяющий

маршрутизатору выбрать один вариант из нескольких возможных. Выбор маршрута осуществляется следующим образом.

1. Выбрать маршрут с наибольшим значением локального приоритета.
2. Выбрать маршрут с наименьшей длиной пути *AC*.
3. Выбрать маршруты, полученные через внешние подключения (с помощью протокола eBGP), а не через внутренние (с помощью протокола iBGP).
4. Среди маршрутов, полученных от одной и той же соседней AC, выбрать маршрут с наименьшим значением атрибута MED (multiple exit discriminator — дискриминатор множественного выхода).
5. Выбрать маршруты с наименьшей стоимостью пути IGP к IP-адресу следующего транзитного участка в BGP-маршруте (в качестве которого обычно выступает IP-адрес пограничного маршрутизатора).

Эти шаги выполняются последовательно до тех пор, пока не останется один маршрут для каждого IP-префикса. Маршрутизатор делает это для каждого IP-префикса, указанного в его таблице маршрутизации. Хотя этот процесс кажется длинным и запутанным, в действительности он достаточно прост. Значение **локального предпочтения (local preference)** для каждого маршрута устанавливается оператором LAN и остается внутренним для этой *AC*. Оно имеет наивысший приоритет среди правил маршрутизации, что позволяет оператору реализовать упомянутые выше способы приоритизации (например, выбрать маршрут от клиента вместо пути без взаиморасчетов). Остальные правила служат главным образом для выбора коротких маршрутов и реализации «раннего выхода». В последнем случае выбираются маршруты от внешней AC вместо полученных от внутреннего маршрутизатора. Той же цели служит и выбор маршрутов с наименьшей стоимостью пути IGP к пограничному маршрутизатору.

Как это ни удивительно, мы лишь слегка коснулись вопроса использования BGP. Для более подробной информации см. спецификацию BGP версии 4 в RFC 427 и другие RFC. Но помните, что большинство трудностей касаются политики маршрутизации, о которой не говорится в спецификации этого протокола.

### *Регулирование междоменного трафика*

Как уже говорилось в этой главе, операторам сетей часто приходится настраивать параметры и конфигурацию сетевых протоколов для управления загруженностью сети и недопущения перегрузок. Такие методы регулирования трафика широко применяются при использовании BGP. Оператор контролирует выбор маршрутов протоколом, чтобы управлять поступлением трафика в сеть или его выходом из сети. В первом случае процесс называется **регулированием входящего трафика (inbound traffic engineering)**, во втором — **регулированием исходящего трафика (outbound traffic engineering)**.

Наиболее распространенный способ регулирования входящего трафика сводится к тому, чтобы корректировать то, как маршрутизаторы устанавливают атрибут локального приоритета для отдельных маршрутов. Например, можно присвоить более высокое значение приоритета всем маршрутам, полученным от

*АС* конкретного клиента. Таким образом оператор гарантирует, что при наличии возможности всегда будет выбираться один из них, а не, скажем, транзитный маршрут. Регулирование исходящего трафика — более сложный процесс, поскольку протокол BGP не позволяет одной *АС* сообщать другой о том, как выбирать маршруты (именно поэтому они и называются автономными). Но оператор может отправлять маршрутизаторам соседних сетей косвенные сигналы, чтобы контролировать, как они выбирают маршруты. Один из распространенных способов — искусственно увеличить длину пути *АС* путем многократного повторения собственной АС сети в объявлении маршрута, то есть произвести **префиксацию пути АС (AS path prepending)**. Другой подход — использовать самый длинный совпадающий префикс, просто разбив его на несколько мелких (более длинных) префиксов так, чтобы восходящие маршрутизаторы предпочитали маршруты с такими префиксами. Например, маршрут для префикса /20 можно разбить на маршруты для двух префиксов /21, четырех префиксов /22 и т. д. Этот подход, однако, приводит к некоторым затратам, поскольку таблицы маршрутизации разрастаются, а при превышении определенного порога их размера маршрутизаторы начинают фильтровать объявления маршрутов.

## 5.7.8. Многоадресная интернет-рассылка

Обычно IP-связь устанавливается между одним отправителем и одним получателем. Но для некоторых приложений полезна возможность отправки сообщения одновременно большому количеству получателей. Это может быть спортивная трансляция для множества зрителей, рассылка программных обновлений для пула реплицируемых серверов и цифровые телеконференции с участием нескольких собеседников.

IP поддерживает многоадресную рассылку при использовании адресов класса D. Каждый такой адрес соответствует группе хостов. Для обозначения номера группы может быть использовано 28 бит, то есть одновременно могут существовать 250 млн групп. Когда процесс отправляет пакет по адресу класса D, протокол прилагает максимальные усилия по его доставке всем членам группы, но не даст гарантий. Некоторые хосты могут не получить пакет.

Диапазон IP-адресов 224.0.0.0/24 зарезервирован для многоадресной рассылки в локальной сети. В таком случае протокол маршрутизации не требуется. Пакеты передаются всей LAN с помощью широковещания с указанием группового адреса. Все хосты сети получают пакет, но обрабатывают его только члены группы. За пределы LAN пакет не передается. Вот некоторые примеры групповых адресов:

224.0.0.1 — все системы LAN;

224.0.0.2 — все маршрутизаторы LAN;

224.0.0.5 — все OSPF-маршрутизаторы LAN;

224.0.0.251 — все DNS-серверы LAN.

В других случаях члены группы могут располагаться в разных сетях. Тогда протокол маршрутизации необходим. Но для начала маршрутизаторы,

передающие многоадресные сообщения, должны знать, какие хосты входят в группу. Процесс может попросить свой хост присоединиться к какой-либо группе или покинуть ее. Каждый хост следит за тем, в какие группы входят его процессы в текущий момент. Когда последний процесс хоста покидает группу, хост больше не является ее участником. Примерно раз в минуту каждый многоадресный маршрутизатор рассылает пакет с запросом на все хосты своей LAN (естественно, по локальному групповому адресу 224.0.0.1) с просьбой сообщить о группах, к которым они принадлежат в данный момент. Многоадресные маршрутизаторы не обязаны находиться там же, где обычные. Каждый хост отсылает обратно ответы для всех интересующих его адресов класса D. Эти пакеты запросов и ответов используются **протоколом управления группами в интернете (Internet Group Management Protocol, IGMP)**. Он описан в RFC 3376.

Для построения связующего дерева, позволяющего получить пути от отправителей до всех членов группы, используются различные протоколы многоадресной маршрутизации.

О соответствующих алгоритмах мы уже говорили в разделе 5.2.8. Внутри АС чаще всего используется **протокольно-независимая многоадресная рассылка (Protocol Independent Multicast, PIM)**. Существует несколько ее вариантов. При PIM в плотном режиме (Dense Mode PIM) создается усеченное дерево, построенное методом продвижения по встречному пути. Этот вариант подходит, когда члены группы находятся во всех частях сети, например, при рассылке файлов на многочисленные серверы сети дата-центра. При PIM в разреженном режиме (Sparse Mode PIM) создаются связующие деревья, похожие на деревья с основанием в ядре. Такой вид PIM может использоваться, к примеру, когда поставщик контента транслирует ТВ-сигнал абонентам своей IP-сети. Также разработан вариант PIM с указанием конкретного источника (Source-Specific Multicast PIM). Наконец, если члены группы находятся в нескольких АС, для построения многоадресных маршрутов нужны специальные расширения BGP или туннелирование.

## 5.8. ПОЛИТИКА СЕТЕВОГО УРОВНЯ

В последние годы управление трафиком тесно связано с сетевой политикой, поскольку доминирующей составляющей общего трафика стало потоковое видео, а интернет-соединение теперь все чаще представляет собой прямое подключение между контент-провайдерами и сетями доступа. Существует две проблемы сетевого уровня, касающиеся сетевой политики: пиринговые споры и приоритизация трафика (которую иногда связывают с сетевым нейтралитетом). Далее мы подробно рассмотрим оба этих вопроса.

### 5.8.1. Пиринговые споры

Хотя протокол BGP является техническим стандартом, в конечном итоге любое сетевое соединение — это деньги за маршрутизацию. Трафик направляется по тем путям, которые приносят наибольшую выгоду поставщикам услуг и транзитным

сетям; при этом оплата за транзит является крайней мерой. Очевидно, пиринг без взаиморасчетов возможен, только если обе стороны согласны, что это выгодное соединение. Если одна из них считает, что получает недостаточно средств от сделки, она может затребовать оплату от другой сети. Вторая сеть может согласиться или отказаться, но если переговоры заходят в тупик, возникает так называемый **пиринговый спор (peering dispute)**.

Несколько лет назад произошел один такой резонансный спор. В последние годы крупные контент-провайдеры поставляют трафик в объемах, способных перегрузить любое межсетевое соединение. В 2013 году провайдеры видеоданных настолько нагрузили линии между транзитными провайдерами и городскими сетями доступа, что потоковый видеотрафик исчерпал всю их емкость. Такую перегрузку сложно было устранить, не увеличивая пропускную способность сети. Возник вопрос: кто должен за это платить? В результате большинство крупных контент-провайдеров стали платить сетям доступа за прямое подключение. По сути, они заключили соглашение о платном пиринге, о котором мы говорили выше. Многие ошибочно расценили это как несправедливое блокирование видеотрафика или понижение его приоритета. Но на самом деле инцидент возник из-за коммерческих споров о том, какая сеть должна оплачивать предоставление точек подключения. Подробнее о пиринговых спорах и способах их разрешения можно прочитать в книге Нортона «The Internet Peering Playbook» (Norton, 2012).

Пиринговые споры существуют со времен раннего коммерческого интернета. Однако по мере дальнейшего роста доли трафика, приходящейся на частные подключения, характер этих споров, вероятно, будет меняться. Например, сегодня городские сети доступа отправляют значительную часть своего трафика в те же распределенные облачные хранилища, где находится другой контент. Соответственно, не в их интересах сильно нагружать линии подключения к этим платформам. В последнее время некоторые операторы сетей даже предсказывают, что транзитные соединения полностью исчезнут (Хьюстон; Huston, 2018). Дойдет до этого или нет — покажет время, но без сомнения можно сказать, что события в области пиринга, интернет-соединений и транзита трафика будут развиваться очень динамично.

## 5.8.2. Приоритизация трафика

Приоритизация трафика методами, которые упоминались в этой главе, — непростая задача, которая часто переходит в сферу сетевой политики. Один из главных вопросов управления трафиком — приоритизация трафика, чувствительного к задержкам (например, игры и интерактивные видео), чтобы повышение загруженности другим трафиком (например, файлами большого размера) не ухудшало пользовательский опыт в целом. При передаче файлов интерактивность не нужна, а вот интерактивные приложения требуют низких показателей задержки и флуктуации.

Чтобы обеспечить хорошую производительность для смешанного трафика приложений, операторы сетей часто используют различные виды приоритизации, включая описанное выше взвешенное справедливое обслуживание. Кроме

того, как уже упоминалось, последние версии протокола DOCSIS позволяют размещать трафик интерактивных приложений в очередях с низкой задержкой. Такая дифференцированная обработка различных типов трафика может реально повысить качество пользовательского опыта для одних приложений без его ухудшения для других.

Но обеспечить приоритизацию намного сложнее, если приходится иметь дело с денежными расчетами. В сфере политики интернета остро стоит вопрос о **платной приоритизации (paid prioritization)**, когда одна из сторон платит интернет-провайдеру за присвоение ее трафику более высокого приоритета по сравнению с конкурирующим трафиком приложений того же типа. Платная приоритизация часто считается нарушением принципа свободной конкуренции. Приведем еще один пример. Транзитная сеть предлагает пользователям определенный сервис (например, передачу видео или голоса по IP) и присваивает ему более высокий приоритет по сравнению с сервисами конкурентов. Так, однажды компания AT&T была уличена в блокировании видеозвонков по FaceTime. По этой причине приоритизация часто становится важным элементом дискуссий о **сетевом нейтралитете (network neutrality; net neutrality)**. Концепция сетевого нейтралитета поднимает ряд сложных вопросов, связанных с политикой и законодательством, обсуждение которых выходит за рамки технического пособия по компьютерным сетям. Кратко ее можно представить в виде четырех четко определенных правил:

1. Не блокировать трафик.
2. Не замедлять трафик.
3. Не использовать платную приоритизацию.
4. Не скрывать информацию об используемых методах приоритизации.

Любая политика сетевого нейтралитета обычно предусматривает ряд исключений для целесообразных методов сетевого администрирования (использование приоритизации для повышения эффективности сети или блокировка и фильтрация в целях безопасности). Что при этом считать целесообразным, часто решают юристы. Существует еще одна проблема правового характера: нужно определить, кто именно (то есть какой государственный орган) должен устанавливать правила и какое наказание последует за их нарушение. Так, например, в США был поднят вопрос о том, к какой категории компаний отнести интернет-провайдеров: к телефонным операторам (таким, как AT&T) или к провайдерам информации и контента (таким, как Google). От ответа на этот вопрос зависит то, какой государственный орган будет определять правила в отношении любых аспектов сетевого нейтралитета: от приоритизации до конфиденциальности.

## 5.9. РЕЗЮМЕ

Сетевой уровень предоставляет службы транспортному. Он может быть основан либо на виртуальных каналах, либо на дейтаграммах. В обоих случаях его главная задача состоит в выборе маршрута для пакетов от источника до получателя.

В сетях с виртуальными каналами решение о выборе пути осуществляется при установлении виртуального соединения. В дейтаграммных сетях оно принимается для каждого пакета.

В компьютерных сетях применяется большое количество алгоритмов маршрутизации. Лавинная адресация — это простой метод передачи пакетов по всем путям. Большинство алгоритмов определяют кратчайший путь и адаптируются к изменениям топологии сети. Основные алгоритмы — маршрутизация по векторам расстояний и маршрутизация с учетом состояния линий. В большинстве существующих сетей применяется один из них. Другие важные методы — использование иерархии в больших сетях, маршрутизация для мобильных хостов, широковещание, многоадресная и произвольная маршрутизация.

Сети подвержены перегрузкам, приводящим к увеличению задержки и утере пакетов. Для предотвращения перегрузки разработчики проектируют сети с достаточным запасом пропускной способности, а также используют настройку протоколов на преимущественный выбор незагруженных маршрутов, временное прекращение приема трафика, уведомление источника о необходимости снизить скорость и сброс нагрузки.

Решив проблему перегрузки, можно попытаться достичь гарантированного QoS. Некоторые приложения заботятся в первую очередь о пропускной способности, а не о задержке и флуктуациях. Комплексные методы предоставления разных уровней QoS включают формирование трафика, резервирование ресурсов на маршрутизаторах, контроль доступа. Подходы, обеспечивающие высокий уровень QoS, — комплексное обслуживание IETF (включая RSVP) и дифференцированное обслуживание.

Сети могут весьма значительно различаться, поэтому при их объединении возникают определенные сложности. Если у них разные ограничения на максимальный размер пакета, можно применить фрагментацию. Различные сети могут использовать разные внутренние протоколы маршрутизации, но внешний протокол должен быть общим. Иногда проблемы можно решить с помощью туннелирования пакетов сквозь сильно отличающуюся сеть, но если отправитель и получатель находятся в сетях разных типов, этот подход применить не получится.

В интернете существует большое разнообразие протоколов, относящихся к сетевому уровню. В их числе протокол дейтаграмм (IP) и соответствующие управляющие протоколы (ICMP, ARP и DHCP). В некоторых сетях IP-пакеты передает требующий соединения протокол MPLS. Один из основных протоколов маршрутизации внутри сети — OSPF. Для межсетевой маршрутизации используется BGP. Свободные IP-адреса быстро заканчиваются, поэтому была разработана новая версия IP — IPv6, но внедрение происходит чрезвычайно медленно.

Некоторые вопросы организации и управления трафиком относятся к сетевой политике. Наиболее часто обсуждается проблема пиринговых споров, когда сети не могут договориться о коммерческих условиях подключения, а также о приоритизации трафика. Обычно она служит для устранения негативных последствий перегрузок, но может повлиять на сетевой нейтралитет, если при ее применении нарушается принцип свободной конкуренции.

## ВОПРОСЫ И ЗАДАЧИ

1. При каких условиях служба, ориентированная на установление соединения, будет (или по крайней мере должна) доставлять пакеты не по порядку? Обоснуйте свой ответ.
2. Рассмотрим проблему проектирования, которая возникает при реализации сервиса с виртуальными каналами. Если в сети используются виртуальные каналы, то каждый пакет данных должен иметь 3-байтный заголовок, а каждый маршрутизатор должен выделить 8 байт памяти для идентификации канала. Если сеть основана на дейтаграммах, нужны 15-байтные заголовки, но не требуется место в таблице маршрутизатора. Стоимость пропускной способности составляет 1 цент за $10^6$ байт для каждого транзитного участка. Стоимость быстродействующей памяти для маршрутизатора составляет 1 цент за 1 байт; при этом срок ее амортизации равен двум годам при 40-часовой рабочей неделе. Согласно статистике, сеанс в среднем длится 1000 с, и за это время передается 200 пакетов. В среднем пакету требуется четыре транзитных участка. Какая реализация эффективнее и насколько?
3. Докажите, что проблема счета до бесконечности на илл. 5.10 (б) будет решена, если маршрутизаторы добавят в своих векторах расстояний исходящее соединение для каждой пары «получатель — стоимость». Например, узел *C* на илл. 5.10 (а) не только объявит маршрут до узла *A* с расстоянием 2, но и сообщит, что этот путь проходит через узел *B*. Покажите расстояния от каждого маршрутизатора до *A* после каждого обмена векторами расстояний, пока все маршрутизаторы не убедятся в том, что узел *A* больше для них не доступен.
4. Рассмотрим сеть на илл. 5.12 (а). Используется алгоритм дистанционно-векторной маршрутизации. На маршрутизатор *D* только что поступили следующие пакеты состояния линий: от *A*: (*B*: 5, *E*: 4); от *B*: ( *A*: 4, *C*: 1, *F*: 5); от *C*: (*B*: 3, *D*: 4, *E*: 3); от *E*: ( *A*: 2, *C*: 2, *F*: 2); от *F*: (*B*: 1, *D*: 2, *E*: 3). Накладные расходы от *D* до *C* и *F* равны 3 и 4 соответственно. Какой будет новая таблица маршрутизатора *D*? Укажите используемые выходные линии и ожидаемые накладные расходы.
5. Рассмотрим сеть на илл. 5.7, не учитывая при этом указанные на линиях веса. Допустим, что в качестве алгоритма маршрутизации используется лавинная адресация. Перечислите все маршруты пакета, отправленного из *A* в *D*, если максимальное количество транзитных участков равно 3. Сколько пропускной способности затратит этот пакет в пересчете на количество транзитных участков?
6. Предложите простой эвристический метод нахождения двух путей (если они существуют) от конкретного источника к конкретному адресату, гарантирующий сохранение связи при обрыве любой линии. Маршрутизаторы при этом достаточно надежны, поэтому рассматривать возможность их сбоя не нужно.
7. Рассмотрим подсеть на илл. 5.12 (а). Используется алгоритм дистанционно-векторной маршрутизации. На маршрутизатор *C* только что поступили

следующие векторы: от *B*: (5, 0, 8, 12, 6, 2); от *D*: (16, 12, 6, 0, 9, 10); от *E*: (7, 6, 3, 9, 0, 4). Накладные расходы от *C* до *B*, *D* и *E* равны 6, 3 и 5 соответственно. Какой будет новая таблица маршрутизатора *C*? Укажите используемые выходные линии и ожидаемые накладные расходы.

8. Объясните, в чем состоит разница между маршрутизацией, пересылкой и коммутацией.
9. На илл. 5.13 логическое ИЛИ двух наборов *ACF*-битов равно 111 для каждого ряда. Является ли это случайностью или же это значение одинаково во всех подсетях при любых условиях?
10. Какие размеры регионов и кластеров следует выбрать для минимизации таблиц маршрутизации при трехуровневой иерархической маршрутизации, если число маршрутизаторов равно 4800? Начните с гипотезы о том, что решение в виде $k$ кластеров по $k$ регионов из $k$ маршрутизаторов близко к оптимальному. Это значит, что $k$ примерно равно корню кубическому из 4800 (около 16). Методом проб и ошибок подберите все три параметра так, чтобы они были близки к 16.
11. В тексте было отмечено, что когда мобильный хост находится вне дома, пакеты, отправленные на адрес его домашней LAN, перехватываются внутренним агентом этой LAN. Как этот перехват осуществляет внутренний агент в IP-сети на основе локальной сети 802.3?
12. Сколько широковещательных пакетов формируется маршрутизатором *B* на илл. 5.6 с помощью:

    а) пересылки в обратном направлении;

    б) входного дерева?
13. Рассмотрим сеть на рис 5.15 (а). Предположим, что добавляется новая линия между *F* и *G*, но входное дерево на илл. 5.15 (б) не меняется. Какие изменения нужно внести в илл. 5.15 (в)?
14. Два хоста соединены друг с другом через маршрутизатор. Объясните, как может возникнуть перегрузка, даже если оба хоста и маршрутизатор используют управление потоком (но не контроль перегрузок). Также объясните, каким образом может быть перегружен получатель при контроле перегрузок, но без управления потоком.
15. В качестве механизма борьбы с перегрузкой в сети с виртуальными каналами маршрутизатор может воздержаться от подтверждения полученного пакета, пока не узнает, что его последняя передача по виртуальному каналу была успешно доставлена, и пока у него нет свободного буфера. Для простоты предположим, что маршрутизаторы используют протокол с ожиданием и что у каждого виртуального канала есть один буфер, выделенный ему для каждого направления трафика. Передача пакета (данных или подтверждения) занимает $T$ секунд. Путь пакета проходит через $n$ маршрутизаторов. С какой скоростью пакеты доставляются адресату? Предполагается, что ошибки очень редки, а связь между хостом и маршрутизатором почти не отнимает времени.

16. Дейтаграммная сеть позволяет маршрутизаторам при необходимости удалять пакеты. Вероятность того, что маршрутизатор отвергнет пакет, равна $p$. Представим путь, проходящий от хоста к хосту через два маршрутизатора. Если один из маршрутизаторов отвергнет пакет, у хоста-источника рано или поздно истечет интервал ожидания, и он попытается повторить передачу. Если обе линии (хост — маршрутизатор и маршрутизатор — маршрутизатор) считать за транзитные участки, чему равно среднее число:

    а) транзитных участков, преодолеваемых пакетом за одну передачу;

    б) передач для одного пакета;

    в) транзитных участков, необходимых для получения пакета?

17. В чем состоит основная разница между двумя методами предотвращения перегрузки: ECN и RED?

18. Объясните, каким образом передача больших файлов может увеличить задержку, наблюдаемую при пересылке небольших файлов или данных игрового приложения.

19. Одно из возможных решений предыдущей проблемы сводится к тому, чтобы формировать трафик передачи файлов таким образом, чтобы он никогда не превышал определенную скорость. Вы решили формировать трафик так, чтобы его скорость никогда не превышала 20 Мбит/с. Какой алгоритм для этого подойдет — маркерного ведра, дырявого ведра или ни один из них? Какой должна быть скорость протекания ведра?

20. Отправитель передает данные со скоростью 100 Мбит/с. Вы решили автоматически отбрасывать идущий от него трафик через 1 с. Каким должен быть размер ведра в байтах?

21. Компьютер использует маркерное ведро емкостью 500 Мбайт и скоростью 5 Мбайт/с. После того как в ведро поступает 300 Мбайт, скорость возрастает до 15 Мбайт/с. Сколько потребуется времени, чтобы передать 1000 Мбайт?

22. Рассмотрим очереди пакетов на илл. 5.29. Назовите время завершения и порядок отправки пакетов, если вес 2 будет иметь не нижняя, а средняя очередь? Пакеты с одинаковым временем завершения отправки следует расположить в алфавитном порядке.

23. Имеется следующая спецификация потока: максимальный размер пакета — 1000 байт, скорость маркерного ведра — 10 Мбайт/с, его размер — 1 Мбайт, максимальная скорость передачи — 50 Мбайт/с. Сколько может длиться передача непрерывной последовательности данных на максимальной скорости?

24. Сеть на илл. 5.32 использует RSVP в деревьях групповой рассылки для хостов 1 и 2. Допустим, хост 3 запрашивает канал с пропускной способностью 2 Мбайт/с для потока от хоста 1 и еще один канал с пропускной способностью 1 Мбайт/с для потока от хоста 2. Одновременно хост 4 запрашивает канал с пропускной способностью 2 Мбайт/с для потока от хоста 1, а хост 5 запрашивает канал с пропускной способностью 1 Мбайт/с для потока от хоста 2.

Какую суммарную пропускную способность необходимо зарезервировать для удовлетворения перечисленных запросов на маршрутизаторах *A*, *B*, *C*, *E*, *H*, *J*, *K* и *L*?

25. Маршрутизатор может обрабатывать 2 млн пакетов в секунду. Нагрузка составляет в среднем 1,5 млн пакетов в секунду. Сколько времени уйдет на формирование очередей и обслуживание пакетов процессорами, если на пути от источника до получателя имеется 10 маршрутизаторов?
26. Допустим, пользователь получает дифференцированное обслуживание со срочной переадресацией. Есть ли гарантия того, что срочные пакеты будут испытывать меньшую задержку, чем обычные? Ответ поясните.
27. Допустим, хост *A* соединен с маршрутизатором *R1*. Тот, в свою очередь, соединен с маршрутизатором *R2*, а *R2* — с хостом *B*. Сообщение TCP, содержащее 900 байт данных и 20 байт TCP-заголовка, передается IP-программе, установленной на хосте *A*, для доставки его хосту *B*. Каковы значения полей `Total length`, `Identification`, `DF`, `MF` и `Fragment offset` IP-заголовка каждого пакета, передающегося по трем линиям? Предполагается, что на линии *A*–*R1* максимальный размер фрейма равен 1024 байтам, включая 14-байтный заголовок фрейма, на линии *R1*–*R2* он составляет 512 байт, включая 8-байтный заголовок, и на линии *R2*–*B* — 512 байт, включая 12-байтный заголовок.
28. Маршрутизатор обрабатывает IP-пакеты, общая длина которых (включая данные и заголовок) равна 1024 байта. Предполагая, что пакеты существуют в течение 10 с, вычислите максимальную скорость передачи данных, с которой может работать маршрутизатор без риска зацикливания в пространстве идентификационных номеров IP-дейтаграммы.
29. IP-дейтаграмма, использующая опцию `Strict source routing`, должна быть фрагментирована. Копируется ли эта опция в каждый фрагмент или достаточно поместить ее в первый фрагмент? Обоснуйте свой ответ.
30. Представьте, что вместо 16 бит в адресе класса B для обозначения номера сети отводилось бы 20 бит. Сколько было бы тогда сетей класса B?
31. Преобразуйте IP-адрес, шестнадцатеричное представление которого равно C22F1582, в десятичный формат, разделенный точками.
32. Двум устройствам с поддержкой IPv6 необходимо связаться друг с другом в интернете. К сожалению, путь между ними проходит через сеть, в которой еще не была внедрена поддержка протокола IPv6. Каким образом могут связаться эти два устройства?
33. Маска подсети сети интернета равна 255.255.240.0. Чему равно максимальное число хостов в ней?
34. IP-адреса подходят для обозначения сетей, а Ethernet-адреса — нет. Как вы думаете, почему?
35. Существует множество адресов, начинающихся с IP-адреса 198.16.0.0. Допустим, организации *A*, *B*, *C* и *D* запрашивают 4000, 2000, 4000 и 8000 адресов соответственно. Для каждой из них укажите первый и последний выданные адреса, а также маску вида *w.x.y.z/s*.

36. Маршрутизатор только что получил информацию о следующих IP-адресах: 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21 и 57.6.120.0/21. Если для них используется одна и та же исходящая линия, можно ли их агрегировать? Если да, то во что? Если нет, то почему?
37. Набор IP-адресов с 29.18.0.0 по 29.18.127.255 агрегирован в 29.18.0.0/17. При этом, однако, остался пробел из 1024 свободных адресов с 29.18.60.0 по 29.18.63.255, которые впоследствии были присвоены хосту, использующему другую исходящую линию. Нужно ли теперь разделить агрегированный адрес на составляющие, добавить в таблицу новый блок, а потом посмотреть, возможна ли новая агрегация? Если нет, тогда что можно сделать?
38. Имеются три маршрутизатора: *A*, *B* и *C*. Маршрутизатор *A* объявляет маршруты для диапазонов адресов 37.62.5.0/24, 37.62.2.0/23 и 37.62.128.0/17. Маршрутизатор *B* объявляет маршруты для диапазонов адресов 37.61.63.0/24 и 37.62.64.0/18. Затем они агрегируют свои диапазоны и объявляют полученный результат маршрутизатору *C*. К каким ошибкам в маршрутизации это приведет, если таблица маршрутизатора *C* будет содержать только эти два агрегированных диапазона адресов? Что могут сделать маршрутизаторы, чтобы не допустить этого?
39. Многие компании устанавливают два и более маршрутизатора для соединения с провайдером, что гарантирует некоторый запас прочности на случай, если один из маршрутизаторов выйдет из строя. Применима ли такая политика при использовании NAT? Обоснуйте свой ответ.
40. Вы собрались поиграть со своим другом по интернету. Он запустил игровой сервер и сообщил вам номер прослушиваемого сервером порта. Допустим, что и ваша сеть, и сеть вашего друга отделены от интернета NAT-блоком. Что при этом делает NAT с входящими пакетами, которые вы отправляете? Как можно избавиться от этой проблемы, не убирая NAT?
41. Два компьютера, расположенные в одной сети, пытаются использовать один и тот же номер порта для связи с сервером в другой сети. Возможно ли это? Объясните свой ответ. Что изменится, если эти компьютеры будут отделены от других сетей NAT-блоком?
42. Вы рассказали другу про протокол ARP. Когда вы закончили объяснения, он сказал: «Ясно. ARP предоставляет услуги сетевому уровню, а значит, является частью канального». Что вы ему ответите?
43. Вы подключаете телефон к беспроводной сети у себя дома. Эта сеть создана модемом, предоставленным вашим провайдером. Используя протокол DHCP, ваш телефон получает IP-адрес 192.168.0.103. Каким, скорее всего, будет IP-адрес отправителя в сообщении DHCP OFFER?
44. Опишите способ сборки пакета из фрагментов на стороне получателя.
45. В большинстве алгоритмов сборки IP-дейтаграмм из фрагментов используется таймер, чтобы из-за потерянного фрагмента буфер, где производится повторная сборка, не оказался занят остальными фрагментами. Предположим, дейтаграмма делится на четыре фрагмента. Первые три прибывают

к получателю, а четвертый задерживается. У получателя истекает период ожидания, и три фрагмента удаляются из его памяти. Позже приходит последний фрагмент. Как следует с ним поступить?

46. В IP контрольная сумма покрывает только заголовок, но не данные. Почему, как вы полагаете, была выбрана подобная схема?
47. Жительница Бостона едет в Миннеаполис, захватив с собой ноутбук. К ее удивлению, в Миннеаполисе LAN является беспроводной локальной IP-сетью, поэтому ей не нужно подключать свой ноутбук. Нужно ли, тем не менее, проходить процедуру с внутренним и внешним агентом, чтобы электронная почта и другой трафик приходили корректно?
48. Протокол IPv6 использует 16-байтные адреса. На какое время их хватит, если каждую пикосекунду назначать блок в 1 млн адресов?
49. Чтобы решить проблему нехватки адресов IPv4, интернет-провайдеры могут динамически выделять их своим клиентам. После полного развертывания IPv6 адресное пространство станет достаточным для предоставления уникального адреса каждому устройству. При этом для упрощения системы можно было бы выделять каждому устройству постоянный адрес IPv6. Объясните, почему не стоит этого делать?
50. Поле `Protocol`, используемое в заголовке IPv4, отсутствует в фиксированном заголовке IPv6. Почему?
51. Должен ли протокол ARP быть изменен при переходе на IPv6? Если да, то какими должны быть эти изменения — концептуальными или техническими?
52. Напишите программу, моделирующую маршрутизацию методом лавинной адресации. Все пакеты должны содержать счетчик, уменьшающийся на каждом маршрутизаторе. Когда счетчик доходит до нуля, пакет удаляется. Время дискретно, и каждая линия обрабатывает за временной интервал один пакет. Создайте три версии этой программы: с лавинной адресацией по всем линиям; по всем линиям, кроме входной; только по $k$ лучшим линиям (выбираемым статически). Сравните лавину с детерминированной маршрутизацией ($k = 1$) с точки зрения задержки и использования пропускной способности.
53. Напишите программу, моделирующую компьютерную сеть с дискретным временем. Первый пакет в очереди каждого маршрутизатора преодолевает по одному транзитному участку за интервал времени. Число буферов всех маршрутизаторов ограниченно. Прибывший пакет, для которого нет свободного места, игнорируется и повторно не передается. Вместо этого используется сквозной протокол с тайм-аутами и пакетами подтверждения, который в результате вызывает повторную передачу пакета маршрутизатором-источником. Постройте график производительности сети как функции сквозного интервала времени ожидания при разных значениях частоты ошибок.
54. Напишите функцию, осуществляющую пересылку в IP-маршрутизаторе. У процедуры должен быть один параметр — IP-адрес. Также у нее есть доступ к глобальной таблице, представляющей собой массив из троек значений. Каждая тройка содержит следующие целочисленные значения: IP-адрес,

маску подсети и исходящую линию. Функция ищет IP-адрес в таблице, используя CIDR, и возвращает номер исходящей линии.

55. Используя программы *traceroute* (UNIX) или *tracert* (Windows), исследуйте маршрут от вашего компьютера до различных университетов мира. Составьте список трансокеанских линий. Вот некоторые адреса:

    www.berkeley.edu (Калифорния)

    www.mit.edu (Массачусетс)

    www.vu.nl (Амстердам)

    www.ucl.ac.uk (Лондон)

    www.usyd.edu.au (Сидней)

    www.u-tokyo.ac.jp (Токио)

    www.uct.ac.za (Кейптаун)

# ГЛАВА 6
# Транспортный уровень

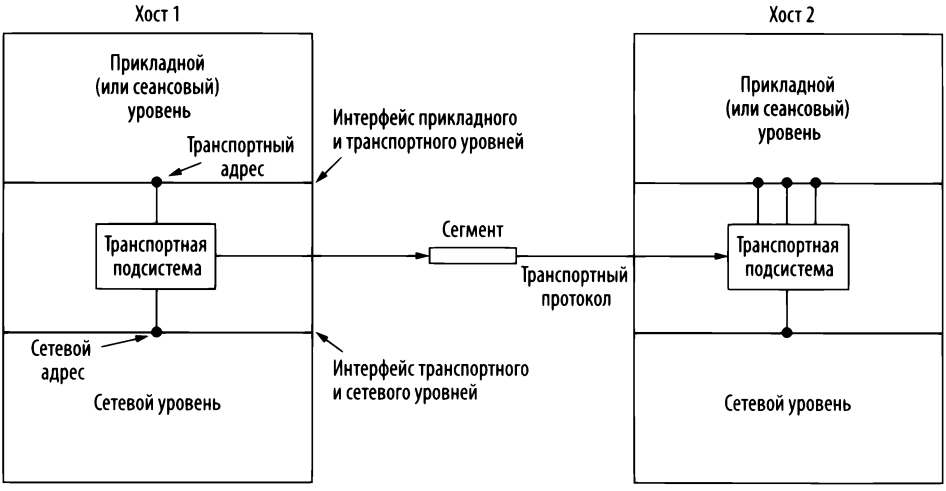
Вместе с сетевым уровнем транспортный уровень составляет сердцевину всей иерархии протоколов. Сетевой уровень обеспечивает сквозную доставку пакетов при помощи дейтаграмм и виртуальных каналов. Транспортный уровень основан на сетевом и отвечает за передачу данных от процесса на устройстве-источнике до процесса на устройстве-адресате, предоставляя необходимый уровень надежности вне зависимости от физических характеристик используемых сетей. Он создает абстракции, с помощью которых приложения могут работать в сети. Без транспортного уровня вся концепция многоуровневых протоколов потеряет смысл. В данной главе мы подробно рассмотрим этот уровень, в том числе его службы и выбор подходящей схемы API, позволяющей решить проблемы надежности, подключения и перегрузок, а также протоколы (такие, как TCP и UDP) и производительность.

## 6.1. ТРАНСПОРТНЫЕ СЛУЖБЫ

В следующих разделах мы познакомимся с транспортными службами. Мы рассмотрим виды служб, предоставляемых прикладному уровню. Чтобы обсуждение было более конкретным, мы разберем два набора базовых операций транспортного уровня. Сначала рассмотрим простой, но условный набор, чтобы показать основные идеи, а затем — реально применяемый в интернете интерфейс.

### 6.1.1. Службы, предоставляемые верхним уровням

Конечная цель транспортного уровня — предоставить эффективную, надежную и экономичную службу передачи данных своим пользователям (обычно это процессы прикладного уровня). Для этого транспортный уровень применяет службы, предоставленные сетевым уровнем. Программа и/или оборудование, выполняющие работу транспортного уровня, называются **транспортной подсистемой**, или **транспортным объектом (transport entity)**. Эта подсистема может находиться в ядре операционной системы, в библиотечном модуле, загруженном сетевом приложении, в отдельном пользовательском процессе или даже в сетевой интерфейсной плате. Первые два варианта чаще всего встречаются в интернете. Логическая взаимосвязь сетевого, транспортного и прикладного уровней проиллюстрирована на илл. 6.1.




**Илл. 6.1.** Сетевой, транспортный и прикладной уровни

На транспортном уровне, как и на сетевом, могут быть службы, ориентированные на установление соединения, и службы без установления соединения. Транспортная служба с установлением соединения во многом похожа на аналогичную сетевую. В обоих случаях соединение проходит три этапа: установление, передача данных и разъединение.

Адресация и управление потоком на этих уровнях также похожи. Более того, даже транспортные службы без установления соединения очень напоминают аналогичные сетевые. Но важно отметить, что реализовать комбинацию транспортной службы без установления соединения и сетевой службы с установлением соединения достаточно трудно, так как для этого приходится разрывать только что созданное соединение сразу же после отправки одного пакета, а это неэффективно.

Возникает закономерный вопрос: если транспортные и сетевые службы так похожи, то зачем нужны два разных уровня? Почему одного недостаточно? Это довольно тонкий, но ключевой вопрос. Программное обеспечение транспортного уровня запускается целиком на пользовательских устройствах, а сетевой уровень — в основном на маршрутизаторах, управляемых оператором связи (по крайней мере, в WAN). Что произойдет, если сетевой уровень будет предоставлять службу с установлением соединения, но она будет ненадежной? Что, если она часто будет терять пакеты? Что случится, если маршрутизаторы будут время от времени выходить из строя?

Все это приведет к большим трудностям. Пользователи не контролируют сетевой уровень, поэтому не смогут решить проблему плохого обслуживания, используя более совершенные маршрутизаторы или совершенствуя обработку ошибок канального уровня (просто потому, что маршрутизаторы им не принадлежат). Единственная возможность — расположить над сетевым уровнем еще один уровень, который будет улучшать QoS. Если в сети без установления

соединения пакеты теряются или передаются с искажениями, транспортная подсистема обнаруживает проблему и выполняет повторную передачу. Если в сети с установлением соединения транспортная подсистема узнает, что ее сетевое соединение было внезапно прервано (без сведений о том, что случилось с передаваемыми в этот момент данными), она может установить новое соединение с удаленной транспортной подсистемой. По нему она может отправить запрос к равноранговому объекту и узнать, какие данные дошли до адресата, а какие нет. Затем транспортная подсистема может продолжить передачу с того момента, где она оборвалась.

По сути, наличие транспортного уровня делает транспортную службу более надежной, чем нижележащая сеть, которая не отличается стабильностью. Более того, транспортные примитивы могут быть реализованы в виде вызовов библиотечных процедур, чтобы не зависеть от сетевых примитивов. Сетевые вызовы варьируются в разных сетях (например, обращения в сети Ethernet без установления соединения могут значительно отличаться от обращений в сети с установлением соединения). Если скрыть сетевую службу за набором транспортных примитивов, то для изменений в сети потребуется простая замена одного набора библиотечных процедур другим. При этом он будет делать то же самое, но с помощью других служб более низкого уровня. Независимость приложений от сетевого уровня несет очевидную пользу.

Благодаря транспортному уровню прикладные программисты могут писать код согласно стандартному набору примитивов и сохранять работоспособность программ в самых разных сетях. Им не приходится учитывать разнообразные сетевые интерфейсы и уровни надежности. Если бы все сети работали идеально, имели одинаковые примитивы служб и никогда не менялись, то транспортный уровень, вероятно, был бы не нужен. Однако в реальности он выполняет ключевую функцию: изолирует верхние уровни от деталей технологии, устройства и несовершенства сети.

Именно по этой причине часто разграничивают уровни с первого по четвертый и уровни выше четвертого. Нижние уровни можно рассматривать как **поставщика транспортных служб (transport service provider)**, а верхние уровни — как **пользователя транспортных служб (transport service user)**. Разделение на поставщика и пользователя серьезно влияет на устройство уровней и делает транспортный уровень ключевым. Он формирует основную границу между поставщиком и пользователем надежной службы передачи данных. Именно этот уровень виден приложениям.

## 6.1.2. Примитивы транспортных служб

Чтобы дать пользователям доступ к транспортной службе, транспортный уровень должен совершить некоторые операции над прикладными программами, то есть предоставить транспортный интерфейс. У каждой службы он свой. В этом разделе мы прежде всего рассмотрим простой (но гипотетический) пример транспортной службы и ее интерфейса, чтобы познакомиться с основными принципами и понятиями. Следующий раздел будет посвящен реальному примеру.

Транспортная служба подобна сетевой, но имеет и некоторые существенные отличия. Главное состоит в том, что сетевая служба предназначена для моделирования служб, предоставляемых реальными сетями со всеми их особенностями. Эти сети могут терять пакеты, поэтому обычно сетевая служба ненадежна.

Транспортная служба, ориентированная на установление соединения, напротив, является стабильной. Конечно, в реальных сетях возникают ошибки, но в этом и заключается задача транспортного уровня — предоставлять надежную службу в ненадежной сети.

В качестве примера рассмотрим два процесса на одном компьютере, соединенные каналом в системе UNIX (или с помощью любого другого средства межпроцессорного взаимодействия). Эти процессы предполагают, что соединение между ними абсолютно идеально. Они не хотят знать о подтверждениях, потерянных пакетах, перегрузках и т. п. Им требуется стопроцентно надежное соединение. Процесс *A* помещает данные на одной стороне канала, а процесс *B* извлекает их на другой. Именно для этого и предназначена транспортная служба, ориентированная на установление соединения, — скрывать несовершенство сетевого обслуживания, чтобы пользовательские процессы считали, что существует безошибочный поток битов, даже если они выполняются на разных устройствах.

Кстати, транспортный уровень также может предоставлять ненадежную (дейтаграммную) службу, но о нем сказать почти нечего (разве что «это дейтаграммы»). Поэтому в данной главе мы сконцентрируемся на службе, ориентированной на установление соединения. Тем не менее есть приложения, например клиент-серверные вычислительные системы и потоковое мультимедиа, основанные на транспортных службах без установления соединения, поэтому мы их еще обсудим.

Второе различие между сетевой и транспортной службой состоит в том, для кого они предназначены. С точки зрения конечных точек сети сетевая служба используется только транспортными подсистемами. Мало кто пишет свои собственные реализации таких подсистем, и поэтому пользователи и программы почти не встречаются с чистой сетевой службой. Напротив, многие программы (а значит, и программисты) видят примитивы транспортной службы. Поэтому транспортная служба должна быть удобной и простой в использовании.

Чтобы понять работу транспортной службы, рассмотрим пять примитивов (илл. 6.2). Это максимально простой пример, но он дает представление о задачах транспортного интерфейса с установлением соединения. Интерфейс позволяет прикладным программам устанавливать, использовать и освобождать соединения, чего вполне достаточно для многих приложений.

Для того чтобы научиться использовать эти примитивы, рассмотрим систему, состоящую из сервера и нескольких удаленных клиентов. Вначале сервер выполняет примитив `LISTEN` — обычно для этого вызывается библиотечная процедура, которая обращается к системе. В результате сервер блокируется, пока клиент не обратится к нему. Когда клиент хочет связаться с сервером, он выполняет примитив `CONNECT`. Транспортная подсистема выполняет этот примитив, блокируя обратившегося к ней клиента и отсылая пакет серверу. Поле

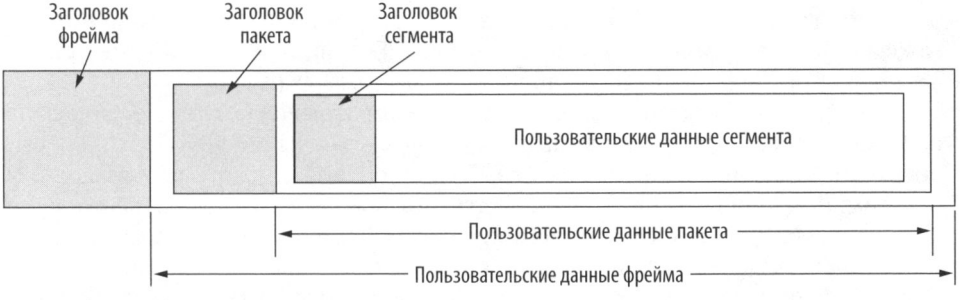
данных пакета содержит сообщение транспортного уровня, адресованное транспортной подсистеме сервера.

| Примитив | Отправленный пакет | Значение |
|---|---|---|
| LISTEN (ОЖИДАТЬ) | (нет) | Блокировать сервер, пока какой-либо процесс не попытается соединиться |
| CONNECT (СОЕДИНИТЬ) | CONNECTION REQUEST (ЗАПРОС СОЕДИНЕНИЯ) | Активно пытаться установить соединение |
| SEND (ОТПРАВИТЬ) | ДАННЫЕ | Отправить информацию |
| RECEIVE (ПОЛУЧИТЬ) | (нет) | Блокировать сервер, пока не поступят данные |
| DISCONNECT (РАЗЪЕДИНИТЬ) | DISCONNECTION REQUEST (ЗАПРОС РАЗЪЕДИНЕНИЯ) | Прервать соединение |

**Илл. 6.2.** Примитивы простой транспортной службы

Следует сказать пару слов о терминологии. За неимением лучшего термина, для сообщений, отправляемых одной транспортной подсистемой другой транспортной подсистеме, нам придется использовать понятие **сегмент (segment)**. Оно используется в TCP, UDP и других интернет-протоколах. В более старых протоколах применялось громоздкое название **модуль данных транспортного протокола (Transport Protocol Data Unit, TPDU)**. Сейчас оно практически не используется, однако вы можете встретить его в старых статьях и книгах.

Итак, сегменты, используемые транспортным уровнем, помещаются в пакеты, которыми обменивается сетевой уровень. Эти пакеты, в свою очередь, содержатся во фреймах, которые передает канальный уровень. Получив фрейм, процесс канального уровня обрабатывает его заголовок, и если адрес назначения совпадает с местом доставки, передает содержимое поля пользовательских данных наверх сетевой подсистеме. Сетевая подсистема похожим образом обрабатывает заголовок пакета и передает содержимое поля пользовательских данных пакета наверх транспортной подсистеме. Эта вложенность проиллюстрирована на илл. 6.3.




**Илл. 6.3.** Вложенность сегментов, пакетов и фреймов

Итак, вернемся к нашему примеру общения клиента и сервера. В результате запроса клиента CONNECT серверу отравляется сегмент, содержащий CONNECTION REQUEST (запрос соединения). Когда он прибывает, транспортная подсистема проверяет, заблокирован ли сервер примитивом LISTEN (то есть готов ли он к обработке запросов). Затем она снимает блокировку сервера и отсылает обратно клиенту сегмент CONNECTION ACCEPTED (соединение принято). Получив этот сегмент, клиент разблокируется, после чего соединение считается установленным.

Теперь клиент и сервер могут обмениваться данными с помощью примитивов SEND и RECEIVE. В простейшем случае каждая из сторон использует блокирующий примитив RECEIVE для перехода в режим ожидания сегмента, который передается другой стороной с помощью SEND. Когда сегмент прибывает, получатель разблокируется. Затем он может обработать полученный сегмент и отправить ответ. Такая схема прекрасно работает, пока обе стороны помнят, чья очередь передавать, а чья — принимать данные.

Обратите внимание, что на транспортном уровне даже простая однонаправленная пересылка данных сложнее, чем на сетевом. Каждый отправленный пакет будет в конце концов подтвержден. Пакеты с управляющими сегментами также подтверждаются, явно или неявно. Эти подтверждения управляются транспортными подсистемами при помощи протокола сетевого уровня и не видны пользователям транспортного уровня. Аналогично транспортные объекты занимаются проблемами таймеров и повторных передач. Все эти механизмы не видны пользователям транспортного уровня, для которых соединение представляется надежным битовым каналом. Биты поступают с одного конца канала и волшебным образом появляются на другом его конце в том же порядке. Эта способность скрывать сложность от пользователей свидетельствует о том, что многоуровневые протоколы являются довольно мощным инструментом.

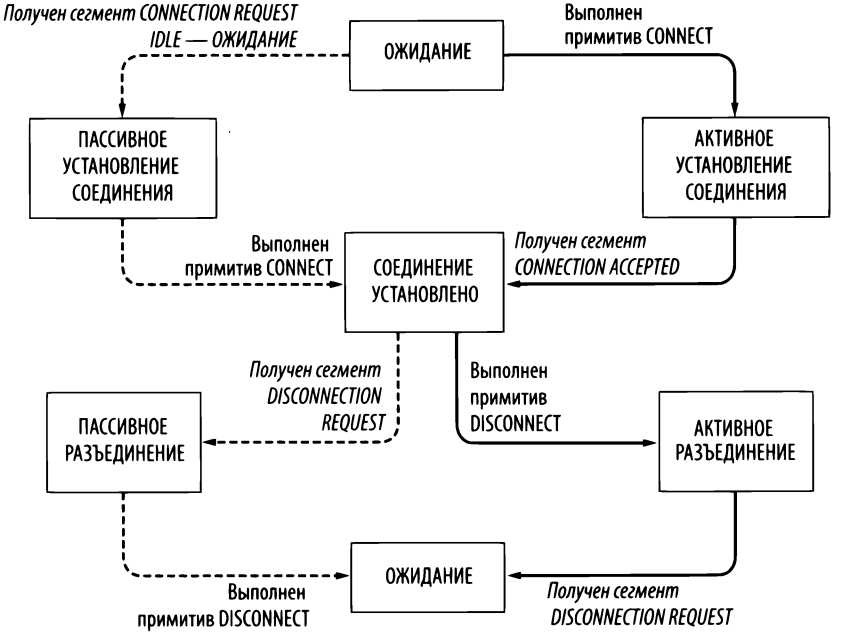
Когда соединение больше не требуется, оно должно быть разорвано, чтобы освободить место в таблицах двух транспортных подсистем. Разъединение бывает симметричным и асимметричным. В асимметричном варианте одна из сторон может вызвать примитив DISCONNECT, в результате чего другая сторона получает управляющий сегмент DISCONNECTION REQUEST (запрос разъединения) и соединение разрывается.

В симметричном варианте каждое направление закрывается отдельно, независимо от другого. Когда одна сторона выполняет операцию DISCONNECT, это означает, что у нее больше нет данных для передачи, но при этом она все еще готова принимать данные от своего партнера. В этой схеме соединение разрывается, когда обе стороны выполняют операцию DISCONNECT.

Диаграмма состояний для установления и разрыва соединения показана на илл. 6.4. Каждый переход вызывается каким-то событием — операцией, выполненной локальным пользователем транспортной службы, или входящим пакетом. Для простоты мы будем считать, что каждый сегмент подтверждается отдельно. Мы также предполагаем, что используется модель симметричного разъединения, в которой клиент делает первый ход. Обратите внимание на простоту этого примера. Позднее, когда мы будем говорить о TCP, мы рассмотрим более реалистичные модели.

## 6.1.3. Сокеты Беркли

Теперь рассмотрим другой набор примитивов транспортного уровня — примитивы сокетов, используемые для протокола TCP. Впервые сокеты стали применяться в 1983 году в операционной системе Berkeley UNIX 4.2BSD. Очень скоро они приобрели популярность и сейчас широко используются для интернет-программирования в большинстве операционных систем, особенно UNIX; кроме того, существует специальный API, предназначенный для программирования сокетов в системе Windows — «winsock».




**Илл. 6.4.** Диаграмма состояний для простой схемы управления соединениями. Переходы, обозначенные курсивом, вызываются поступлением пакетов. Сплошными линиями показана последовательность состояний клиента. Пунктирными линиями показана последовательность состояний сервера

Примитивы сокетов перечислены на илл. 6.5. Модель сокетов во многом похожа на представленную выше, но обладает большей гибкостью и предоставляет больше возможностей. Сегменты, соответствующие этой модели, будут рассматриваться далее в этой главе.

Первые четыре примитива из списка выполняются серверами в указанной последовательности. Примитив `SOCKET` создает новый сокет и выделяет для него место в таблице транспортной подсистемы. Параметры вызова указывают используемый формат адресов, тип требуемой службы (например, надежный поток байтов) и протокол. В случае успеха `SOCKET` возвращает обычный файловый дескриптор, используемый при вызове следующих операций, подобно тому, как процедура `OPEN` работает для файла.

| Примитив | Значение |
|---|---|
| SOCKET (СОКЕТ) | Создать новый сокет (гнездо связи) |
| BIND (СВЯЗАТЬ) | Связать локальный адрес с сокетом |
| LISTEN (ОЖИДАТЬ) | Объявить о желании принять соединение; указать размер очереди |
| ACCEPT (ПРИНЯТЬ) | Пассивно установить входящее соединение |
| CONNECT (СОЕДИНИТЬ) | Активно пытаться установить соединение |
| SEND (ОТПРАВИТЬ) | Отправить данные по соединению |
| RECEIVE (ПОЛУЧИТЬ) | Получить данные у соединения |
| CLOSE (ЗАКРЫТЬ) | Разорвать соединение |

**Илл. 6.5.** Примитивы сокетов для TCP

У только что созданного сокета нет сетевых адресов. Они назначаются с помощью примитива `BIND`. После того как сервер привязывает адрес к сокету, с ним могут связаться удаленные клиенты. Вызов `SOCKET` не создает адрес напрямую, так как некоторые процессы придают своим адресам большое значение (например, они использовали один и тот же адрес годами, и он известен всем).

Далее идет вызов примитива `LISTEN`, который выделяет место для очереди входящих соединений на случай, если несколько клиентов попытаются соединиться одновременно. В отличие от аналогичного примитива в нашем первом примере, в модели сокетов `LISTEN` не является блокирующим вызовом.

Чтобы заблокировать ожидание входящих соединений, сервер выполняет примитив `ACCEPT`. Получив сегмент с запросом соединения, транспортная подсистема создает новый сокет с теми же свойствами, что и у исходного сокета, и возвращает для него файловый дескриптор. При этом сервер может разветвить процесс или поток, чтобы обработать соединение для нового сокета и вернуться к ожиданию следующего соединения для первоначального сокета. `ACCEPT` возвращает файловый дескриптор, который можно использовать для чтения и записи стандартным способом, как это делается в случае файлов.

Теперь посмотрим на этот процесс со стороны клиента. И здесь прежде всего должен быть создан сокет с помощью примитива `SOCKET`, но `BIND` в этом случае не требуется, так как используемый адрес не имеет значения для сервера. `CONNECT` блокирует вызывающего и инициирует активный процесс соединения. Когда этот процесс завершается (то есть когда соответствующий сегмент, отправленный сервером, получен), процесс клиента разблокируется, и соединение считается установленным. После этого обе стороны могут использовать `SEND` и `RECIEVE` для передачи и получения данных по полнодуплексному соединению. Могут также применяться стандартные UNIX-вызовы `READ` и `WRITE`, если нет нужды в использовании специальных свойств `SEND` и `RECIEVE`.

В модели сокетов используется симметричный разрыв соединения. Это происходит, когда обе стороны выполняют примитив `CLOSE`.

Получив широкое распространение, сокеты де-факто стали стандартом абстрагирования транспортных служб для приложений. Часто сокет-API используется вместе с протоколом TCP для предоставления службы, ориентированной на установление соединения, — **надежного потока байтов (reliable byte stream)**; на деле это надежный битовый канал, о котором мы говорили выше. Этот API может сочетаться и с другими протоколами, но в любом случае результат должен быть одинаковым для пользователя.

Преимущество сокет-API состоит в том, что приложение может использовать его и для других транспортных служб. К примеру, с помощью сокетов можно реализовать службу без установления соединения. В этом случае `CONNECT` задает адрес удаленного узла, а `SEND` и `RECEIVE` отправляют и получают дейтаграммы. (Иногда используется расширенный набор вызовов, например примитивы `SENDTO` и `RECEIVEFROM`, позволяющие приложению не ограничиваться одним транспортным узлом.) Иногда сокеты используются с транспортными протоколами, в которых вместо байтового потока применяется поток сообщений и которые могут включать (или не включать) контроль перегрузок. К примеру, **дейтаграммный протокол с контролем перегрузок (Datagram Congestion Control Protocol, DCCP)** является вариантом UDP с управлением перегрузкой (Колер и др.; Kohler et al., 2006). Необходимую службу выбирают сами пользователи.

Тем не менее последнее слово в вопросе транспортных интерфейсов, скорее всего, останется не за сокетами. Довольно часто приложениям приходится работать с группой связанных потоков, например браузер может одновременно запрашивать у сервера несколько объектов. В таком случае применение сокетов обычно означает, что для каждого объекта будет использоваться один поток. В результате управление перегрузкой будет выполняться отдельно для каждого потока (а не для всей группы). Безусловно, это далеко не оптимальный вариант, поскольку управление набором потоков становится задачей приложения. Чтобы более эффективно обрабатывать группы связанных потоков и уменьшить роль приложения в этом процессе, был создан ряд дополнительных протоколов и интерфейсов. В частности, **протокол передачи с управлением потоками (Stream Control Transmission Protocol, SCTP)**, описанный в RFC 4960 (Форд; Ford, 2007), и протокол QUIC (он будет рассмотрен ниже). Эти протоколы слегка изменяют сокет-API для удобства работы с группами потоков, обеспечивая новые возможности, например работу со смешанным трафиком (с установлением соединения и без) и даже поддержку множественных сетевых путей.

### 6.1.4. Пример программирования сокета: файл-сервер для интернета

Чтобы узнать, как выполняются вызовы для сокета на практике, рассмотрим клиентский и серверный код на илл. 6.6. Имеется примитивный файл-сервер, работающий в интернете, и использующий его клиент. У программы много ограничений (о которых еще будет сказано), но теоретически данный код,

описывающий сервер, может быть скомпилирован и запущен на любой UNIX-системе, подключенной к интернету. Код, описывающий клиента, может быть запущен с определенными параметрами. Это позволит ему получить любой файл, к которому у сервера есть доступ. Файл отображается на стандартном устройстве вывода, но, разумеется, может быть перенаправлен на диск или какому-либо процессу.

```
/* На этой странице содержится клиентская программа, запрашивающая файл у серверной программы, расположенной на следующей странице. Сервер в ответ на запрос высылает файл. */

#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 8080                /* По договоренности между клиентом и сервером */
#define BUF_SIZE 4096                   /* Размер передаваемых блоков */
int main(int argc, char **argv)
{
 int c, s, bytes;
 char buf[BUF_SIZE];                    /* буфер для входящего файла */
 struct hostent *h;                     /* информация о сервере */
 struct sockaddr_in channel;            /* содержит IP-адрес */
 if (argc != 3) {printf("Для запуска введите: client имя_сервера имя_файла");
                 exit(-1);}
 h = gethostbyname(argv[1]);            /* поиск IP-адреса хоста */
 if (!h) {printf("gethostbyname не удалось найти %s", argv[1]); exit(-1;}
 s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
 if (s <0) {printf("сбой вызова сокета"); exit(-1);}
 memset(&channel, 0, sizeof(channel));
 channel.sin_family= AF_INET;
 memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
 channel.sin_port= htons(SERVER_PORT);
 c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
 if (c < 0) {printf("сбой соединения"); exit(-1);}
/* Соединение установлено. Отправляется имя файла с нулевым байтом на конце */
 write(s, argv[2], strlen(argv[2])+1);
/* Получить файл, записать на стандартное устройство вывода */
 while (1) {
    bytes = read(s, buf, BUF_SIZE); /* Читать из сокета */
    if (bytes <= 0) exit(0);        /* Проверка конца файла */
    write(1, buf, bytes);           /* Записать на стандартное устройство вывода */
 }
}
```

**Илл. 6.6.** Клиентская программа для использования сокетов. Серверная программа представлена на следующей странице

```
#include <sys/types.h>                    /* Серверная программа */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 8080                  /* По договоренности между клиентом и сервером */
#define BUF_SIZE 4096                     /* Размер передаваемых блоков */
#define QUEUE_SIZE 10
int main(int argc, char *argv[])
{ int s, b, l, fd, sa, bytes, on = 1;
 char buf[BUF_SIZE];                      /* буфер для исходящего файла */
 struct sockaddr_in channel;              /* содержит IP-адрес */
 /* Создать структуру адреса для привязки к сокету */
 memset(&channel, 0, sizeof(channel));    /* Обнуление channel */
 channel.sin_family = AF_INET;
 channel.sin_addr.s_addr = htonl(INADDR_ANY);
 channel.sin_port = htons(SERVER_PORT);
 /* Пассивный режим. Ожидание соединения */
 s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);  /* Создать сокет */
 if (s <0) {printf("сбой вызова сокета"); exit(-1);}
 setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));
 b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
 if (b < 0) {printf("сбой связывания"); exit(-1);}
 l = listen(s, QUEUE_SIZE);               /* Определение размера очереди */
 if (l < 0) {printf("сбой ожидания"); exit(-1);}
 /* Теперь сокет установлен и привязан. Ожидание и обработка соединения. */
 while (1) {
    sa = accept(s, 0, 0);                 /* Блокировка в ожидании запроса соединения */
    if (sa < 0) {printf("сбой доступа"); exit(-1);}
    read(sa, buf, BUF_SIZE);              /* Считать имя файла из сокета */
    /* Получить и возвратить файл. */
    fd = open(buf, O_RDONLY);             /* Открыть файл для обратной отправки */
    if (fd < 0) {printf("сбой открытия файла");
    while (1) {
        bytes = read(fd, buf, BUF_SIZE); /* Читать из файла */
        if (bytes <= 0) break;            /* Проверка конца файла */
        write(sa, buf, bytes);            /* Записать байты в сокет */
    }
    close(fd);                            /* Закрыть файл */
    close(sa);                            /* Разорвать соединение */
 }
}
```

Сначала рассмотрим ту часть программы, которая описывает сервер. Она начинается с включения некоторых стандартных заголовков, последние три из которых содержат основные структуры данных и определения, относящиеся к интернету. Затем *SERVER_PORT* определяется как 8080. Значение выбрано случайным образом. Любое число от 1024 до 65 535 также подойдет, если только

оно не используется другим процессом; порты с номерами 1023 и ниже зарезервированы для привилегированных пользователей.

В последующих двух строках определяются две необходимые серверу константы. Первая из них задает размер блока данных для передачи файлов (в байтах). Вторая определяет максимальное количество незавершенных соединений, после установки которых новые соединения будут отвергаться.

После объявления локальных переменных начинается сама программа сервера. Вначале она инициирует структуру данных, которая будет содержать IP-адрес сервера. Эта структура вскоре будет привязана к серверному сокету. Вызов `memset` полностью обнуляет структуру данных. Последующие три присваивания заполняют три поля этой структуры. Последнее содержит порт сервера. Функции `htonl` и `htons` преобразуют значения в стандартный формат, что позволяет программе нормально выполняться на устройствах с представлением числовых разрядов little-endian (например, Intel x86) и big-endian (например, SPARC).

После этого сервер создает и проверяет сокет на ошибки (определяется по $s < 0$). В окончательной версии программы сообщение об ошибке может быть чуть более понятным. Вызов `setsockopt` нужен для того, чтобы порт мог использоваться несколько раз, а сервер — бесконечно, обрабатывая запрос за запросом. Теперь IP-адрес привязывается к сокету и выполняется проверка успешного завершения вызова `bind`. Конечным этапом инициализации является вызов `listen`. Он свидетельствует о готовности сервера к приему входящих вызовов и сообщает системе о том, что нужно ставить в очередь до *QUEUE_SIZE* вызовов, пока сервер обрабатывает текущий вызов. При заполнении очереди прибытие новых запросов спокойно игнорируется.

В этот момент сервер входит в основной цикл программы и уже из него не выходит. Этот цикл можно остановить только извне. Вызов `accept` блокирует сервер на то время, пока клиент пытается установить соединение. В случае успеха `accept` возвращает дескриптор сокета, который можно использовать для чтения и записи, аналогично тому, как файловые дескрипторы применяются для чтения и записи в каналы. Однако, в отличие от однонаправленных каналов, сокеты двунаправлены, поэтому для чтения (и записи) данных из соединения можно использовать `sa` (принятый сокет). Файловые дескрипторы канала могут применяться для чтения или записи, но не одновременно.

После установления соединения сервер считывает имя файла. Если оно еще недоступно, сервер блокируется, ожидая его. Получив имя файла, сервер открывает файл и входит в цикл, который читает блоки данных из файла и записывает их в сокет. Это продолжается до тех пор, пока не будут скопированы все запрошенные данные. Затем файл закрывается, соединение разрывается и начинается ожидание нового вызова. Данный цикл повторяется бесконечно.

Теперь рассмотрим часть кода, описывающую клиента. Чтобы понять, как работает программа, сначала необходимо разобраться, как она запускается. Если она называется *client*, ее типичный вызов будет выглядеть так:

```
client flits.cs.vu.nl /usr/tom/filename >f
```

Этот вызов сработает, только если сервер расположен по адресу flits.cs.vu.nl, файл /usr/tom/filename существует и у сервера есть доступ по чтению для этого

файла. Если вызов произведен успешно, файл передается по интернету и записывается в *f*, после чего клиентская программа заканчивает свою работу. Поскольку серверная программа продолжает работать, клиент может быть запущен снова с новыми запросами на получение файлов.

Клиентская программа начинается с подключения файлов и объявлений. Прежде всего проверяется корректность числа аргументов (где `argc` = 3 означает, что была вызвана программа с указанием ее имени и двух аргументов). Обратите внимание на то, что `argv[1]` содержит имя сервера (например, flits.cs.vu.nl) и переводится в IP-адрес с помощью функции `gethostbyname`. Для поиска имени эта функция использует DNS. Межсетевые экраны мы изучим отдельно в главе 7.

Затем создается и инициализируется сокет, после чего клиент пытается установить TCP-соединение с сервером посредством `connect`. Если сервер включен, работает на указанном компьютере, соединен с *SERVER_PORT* и либо простаивает, либо имеет достаточно места в очереди `listen` (очереди ожидания), то соединение с клиентом будет рано или поздно установлено. По данному соединению клиент передает имя файла, записывая его в сокет.

Число отправленных байтов превышает количество, необходимое для передачи имени, на единицу, поэтому нужен еще нулевой байт-ограничитель, с помощью которого сервер определяет, где кончается имя файла.

Теперь клиентская программа входит в цикл, читает файл блок за блоком из сокета и копирует на стандартное устройство вывода. По окончании этого процесса она просто завершается.

Процедура `fatal` выводит сообщение об ошибке и завершается. Серверу также требуется эта процедура, и она пропущена в листинге только из соображений экономии места. Поскольку программы клиента и сервера компилируются отдельно и в обычной ситуации выполняются на разных устройствах, они не могут одновременно использовать код процедуры `fatal`.

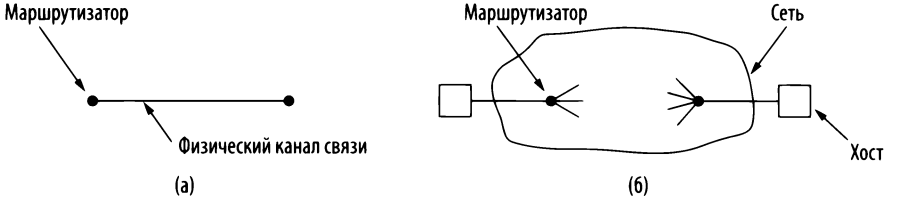
Стоит заметить, что такой сервер построен далеко не по последнему слову техники. Осуществляемая проверка ошибок минимальна, а сообщения об ошибках реализованы весьма посредственно. Система будет обладать низкой производительностью, поскольку все запросы обрабатываются только последовательно (используется один поток запросов). Понятно, что ни о какой защите информации здесь говорить не приходится, а применение аскетичных системных вызовов UNIX не лучшее решение для достижения независимости от платформы. При этом делаются некоторые некорректные с технической точки зрения допущения. Например, что имя файла всегда поместится в буфер и будет передано без ошибок. Несмотря на эти недостатки, с помощью данной программы можно организовать полноценный работающий файл-сервер для интернета. Более подробную информацию вы найдете в работах Донаху и Калверта (Donahoo and Calvert; 2008, 2009), а также Стивенса и др. (Stevens et al., 2004).

## 6.2. ЭЛЕМЕНТЫ ТРАНСПОРТНЫХ ПРОТОКОЛОВ

Транспортные службы реализуются **транспортным протоколом**, который используется между двумя транспортными подсистемами. Транспортные протоколы

несколько напоминают протоколы канального уровня, которые мы подробно рассмотрели в главе 3. И те и другие протоколы, помимо прочего, занимаются обработкой ошибок, управлением очередями и потоками.

Однако у этих протоколов также имеется множество существенных различий, из-за разных условий, в которых они работают (илл. 6.7). На канальном уровне два маршрутизатора общаются напрямую по физическому каналу (проводному или беспроводному), тогда как на транспортном уровне физический канал заменен целой сетью. Это различие сильно влияет на протоколы.




**Илл. 6.7.** Окружение: (а) канального уровня; (б) транспортного уровня

Во-первых, при двухточечном соединении по проводу или оптоволоконной линии маршрутизатор обычно не должен указывать, с каким маршрутизатором он хочет обменяться данными, — каждая выходная линия ведет к конкретному маршрутизатору. На транспортном уровне требуется явно указывать адрес получателя.

Во-вторых, процесс установки соединения по проводу (см. илл. 6.7 (а)) прост: противоположная сторона всегда присутствует (если только она не вышла из строя). В любом случае работы не очень много. Даже в случае беспроводного соединения ситуация не слишком отличается. Простой отправки сообщения достаточно, чтобы оно дошло до всех адресатов. Если подтверждение о получении не приходит (вследствие ошибок), сообщение может быть отправлено повторно. На транспортном уровне начальная установка соединения, как будет показано ниже, происходит достаточно сложно.

Еще одно весьма досадное различие между канальным и транспортным уровнями состоит в том, что сеть потенциально обладает возможностями хранения информации. Когда маршрутизатор отправляет пакет по каналу связи, пакет может прийти или потеряться, но он не может побродить где-то какое-то время, спрятаться на краю света, а затем внезапно появиться и прийти в место назначения после пакетов, отправленных гораздо позже него. Если же сеть использует дейтаграммы, которые маршрутизируются в ней независимо, то всегда есть ненулевая вероятность, что пакет пройдет по какому-то странному пути и придет к получателю позже, чем нужно, и в неправильном порядке; возможно также, что при этом будут получены копии пакета. Последствия способности сети задерживать и копировать пакеты порой катастрофичны и требуют применения специальных протоколов, обеспечивающих правильную передачу данных.

Последнее различие между канальным и транспортным уровнями является скорее количественным, чем качественным. Буферизация и управление потоком

необходимы в обоих случаях. Однако наличие на транспортном уровне большого и непостоянного числа соединений с пропускной способностью, колеблющейся из-за их конкуренции, может потребовать принципиально другого подхода. Некоторые из рассмотренных в главе 3 протоколов выделяют фиксированное количество буферов для каждой линии, поэтому для входящего фрейма всегда имеется свободный буфер. На транспортном уровне из-за множества управляемых соединений и колебаний пропускной способности выделение нескольких буферов каждому соединению не слишком хорошая идея. В следующих разделах мы изучим эти и другие важные вопросы.

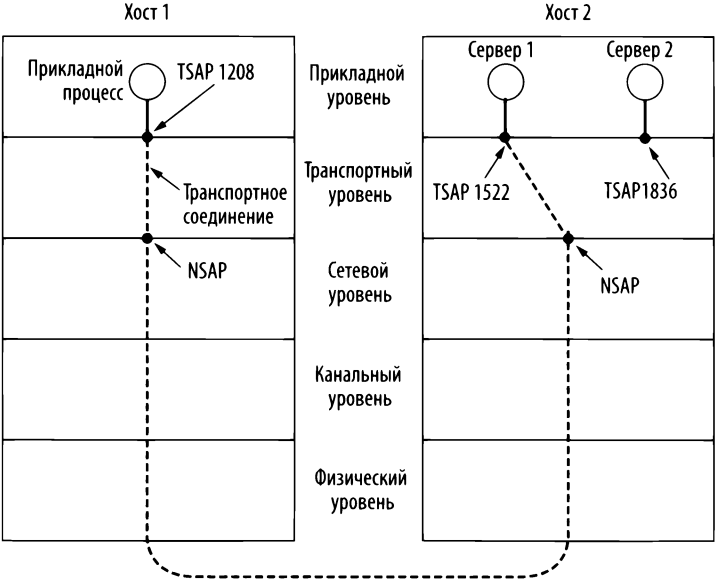
### 6.2.1. Адресация

Когда один прикладной процесс желает установить соединение с другим прикладным процессом, он должен его конкретно указать. Обычно для этого задаются транспортные адреса, куда процессы отправляют запросы на установление соединения. В интернете такие конечные точки на транспортном уровне называются **портами**. Для их обозначения мы будем пользоваться нейтральным термином **точка доступа к службам транспортного уровня (Transport Service Access Point, TSAP)**. Аналогичные конечные точки сетевого уровня (то есть адреса сетевого уровня) называются **точками доступа к сетевым службам (Network Service Access Point, NSAP)**. Примерами NSAP являются IP-адреса.

На илл. 6.8 показаны взаимоотношения между NSAP, TSAP и транспортным соединением. Прикладные процессы клиента и сервера могут связываться с локальной TSAP для установления соединения с удаленной TSAP. Такие соединения проходят через NSAP на каждом хосте, как показано на рисунке. TSAP нужны для того, чтобы различать конечные точки, совместно использующие NSAP, в сетях, где у каждого компьютера есть своя NSAP.

Возможный сценарий для транспортного соединения выглядит следующим образом.

1. Процесс почтового сервера подсоединяется к точке доступа TSAP 1522 на хосте 2 и ожидает входящего вызова. Вопрос о том, как процесс соединяется с TSAP, лежит за пределами сетевой модели и полностью зависит от локальной операционной системы. Например, может вызываться примитив типа `LISTEN`.
2. Прикладной процесс хоста 1 хочет отправить почтовое сообщение, поэтому он подключается к TSAP 1208 и обращается к сети с запросом `CONNECT`, указывая TSAP 1208 на хосте 1 в качестве адреса отправителя и TSAP 1522 на хосте 2 в качестве адреса получателя. Это действие в результате приводит к установке транспортного соединения между прикладным процессом и сервером.
3. Прикладной процесс отправляет почтовое сообщение.
4. Почтовый сервер отвечает, что сообщение будет доставлено.
5. Транспортное соединение разрывается.




**Илл. 6.8.** Точки доступа к службам транспортного и сетевого уровня и транспортные соединения

Обратите внимание, что на хосте 2 могут располагаться и другие серверы, соединенные со своими TSAP и ожидающие входящих запросов на соединение, приходящих с той же NSAP.

Это хорошая схема, только мы обошли стороной один маленький вопрос: как пользовательский процесс хоста 1 узнает, что почтовый сервер соединен с TSAP 1522? Возможно, почтовый сервер подключается к TSAP 1522 уже долгие годы, и постепенно об этом узнали все пользователи сети. В этом случае службы имеют постоянные TSAP-адреса, хранящиеся в файлах, которые расположены в известных местах. Так, например, в /etc/services UNIX-систем перечисляются серверы, за которыми закреплены конкретные порты, в частности, там указано, что почтовый сервер использует TCP порт 25.

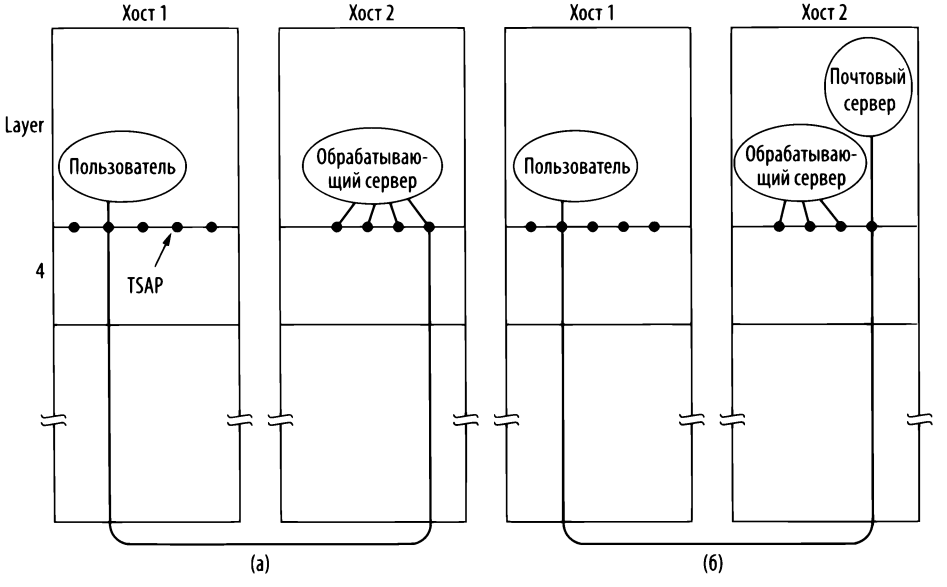
Хотя постоянные TSAP-адреса хорошо подходят для небольшого количества никогда не меняющихся ключевых служб (например, для веб-сервера), зачастую пользовательские процессы хотят обменяться данными с другими процессами, TSAP-адреса которых заранее неизвестны или существуют только в течение короткого времени.

Чтобы справиться с этой ситуацией, можно использовать другую схему. В этой модели задействован специальный процесс: **сопоставитель портов (portmapper)**. Чтобы найти TSAP-адрес, соответствующий заданному имени службы, например «BitTorrent», пользователь устанавливает соединение с сопоставителем портов (TSAP-адрес которого всем известен). Затем он отправляет сообщение с названием нужной ему службы, и сопоставитель портов сообщает ее TSAP-адрес. После этого пользователь разрывает соединение с сопоставителем портов и устанавливает соединение с этой службой.

В этой модели при создании новой службы она регистрируется на сопоставителе портов, сообщая ему свое имя (обычно строка ASCII) и TSAP-адрес. Сопоставитель сохраняет полученную информацию в своей базе данных, чтобы иметь возможность отвечать на будущие запросы.

Функция сопоставителя портов аналогична работе оператора телефонной справочной службы — он преобразует имена в номера. Здесь, как и в телефонной системе, важно, чтобы TSAP-адрес сопоставителя (или обрабатывающего сервера в протоколе начального соединения) был действительно хорошо известен. Если вы не знаете номера телефонной справочной службы, вы не сможете позвонить оператору. Если вам кажется, что номер справочной очевиден, попытайтесь угадать его, находясь в другой стране.

Многие существующие на компьютере серверные процессы используются редко. Поддерживать каждый из них в активном состоянии с постоянным TSAP-адресом слишком расточительно. Альтернативный вариант в упрощенном виде показан на илл. 6.9. Он называется **протоколом начального соединения (initial connection protocol)**. Вместо того чтобы назначать всем возможным серверам хорошо известные TSAP-адреса, каждое устройство, желающее предоставлять службы удаленным пользователям, обзаводится специальным **обрабатывающим сервером (process server)**, действующим как прокси (посредник) для менее активно используемых серверов. В UNIX-системах такой сервер называется *inetd*. Он прослушивает одновременно несколько портов, ожидая запроса на соединение. Потенциальные пользователи начинают с того, что отправляют запрос `CONNECT`, указывая TSAP-адрес нужной им службы. Если ни один сервер их не ждет, они получают соединение с обрабатывающим сервером (илл. 6.9 (а)).




**Илл. 6.9.** Пользовательский процесс хоста 1 устанавливает соединение с почтовым сервером хоста 2 через обрабатывающий сервер

Получив запрос, обрабатывающий сервер порождает подпроцесс запрошенного сервера, передавая ему существующее соединение с пользователем. Новый сервер выполняет нужную задачу, в то время как обрабатывающий сервер возвращается к ожиданию новых запросов (см. илл. 6.9 (б)). Этот метод работает только в тех случаях, когда серверы могут создаваться по требованию.

## 6.2.2. Установление соединения

Установление соединения звучит просто, но неожиданно оказывается весьма трудным делом. На первый взгляд достаточно, чтобы одна транспортная подсистема отправила адресату сегмент `CONNECTION REQUEST` и получила в ответ `CONNECTION ACCEPTED`. Неприятность заключается в том, что сеть может потерять, задержать, повредить или дублировать пакеты. Это может сильно осложнить ситуацию.

### *Проблема: задержка и дублирование пакетов*

Представьте себе настолько перегруженную сеть, что подтверждения практически никогда не доходят вовремя, все пакеты опаздывают и пересылаются повторно по два-три или более раза. Предположим, что сеть основана на дейтаграммах и что каждый пакет следует по своему маршруту. Некоторые пакеты могут застрять в «пробке» и прийти с большим опозданием, когда отправитель уже решит, что они утеряны.

Самый кошмарный сценарий выглядит следующим образом. Пользователь устанавливает соединение с банком и отправляет сообщение с запросом о переводе крупной суммы на счет ненадежного человека. К несчастью, пакеты решают прогуляться по замысловатому маршруту, посетив самые отдаленные уголки сети. Тем временем отправитель вынужден выполнить повторную передачу. На этот раз пакеты идут по кратчайшему пути и доставляются быстро, в результате чего отправитель разрывает соединение.

Происходит очередная неудача: первая порция пакетов выходит из укрытия и добирается до адресата в нужном порядке, предлагая банку установить новое соединение и снова выполнить перевод. У банка нет способа определить, что это дубликаты. Он решает, что это вторая независимая транзакция, и еще раз переводит деньги.

Такой сценарий может показаться маловероятным или даже неправдоподобным, но идея в том, что протоколы всегда должны работать корректно. Самые распространенные случаи должны быть реализованы с максимальной эффективностью, требующейся для высокой производительности сети, однако протокол должен уметь справляться и с редкими сценариями. В противном случае сеть будет ненадежной и может неожиданно дать сбой, даже не сообщив об ошибке.

В оставшейся части этого раздела мы будем изучать проблему задержавшихся дубликатов, уделяя особое внимание алгоритмам, устанавливающим соединение надежным способом. Основная проблема заключается в том, что задержавшиеся дубликаты распознаются как новые пакеты. Избежать копирования и задержки

пакетов мы не можем. Но если это происходит, копии должны отвергаться и не обрабатываться как новые.

Эту проблему можно попытаться решить несколькими способами, но на самом деле ни один из них не является оптимальным. Так, например, можно использовать одноразовые транспортные адреса. При таком подходе каждый раз, когда требуется транспортный адрес, генерируется новый адрес. Когда соединение разрывается, он уничтожается. В этом случае задержавшиеся дубликаты не смогут найти транспортный адрес и, следовательно, перестанут представлять угрозу. Однако при этом будет сложнее установить соединение с процессом.

Другой вариант — каждому соединению присваивается уникальный идентификатор (последовательный номер, возрастающий на единицу для каждого установленного соединения). Он выбирается инициатором соединения и помещается в каждый сегмент, включая тот, который содержит `CONNECTION REQUEST`. После разрыва соединения транспортная подсистема может обновить таблицу, в которой хранятся прежние соединения в виде пар (одноранговая транспортная подсистема, идентификатор). При каждом новом запросе таблица проверяется на наличие соответствующего идентификатора (он мог остаться там от разорванного ранее соединения).

К сожалению, у этой схемы есть существенный изъян: требуется, чтобы каждая транспортная подсистема (как отправитель, так и получатель) практически бесконечно хранила определенное количество информации об истории соединений. Иначе, если устройство выйдет из строя и потеряет данные, оно не сможет определить, какие соединения уже использовались, а какие нет.

Вместо этого можно применить другой подход, который существенно упростит задачу. Нужно разработать механизм, уничтожающий устаревшие заблудившиеся пакеты и не позволяющий им существовать в сети бесконечно долго. При таком ограничении проблема станет более управляемой.

Время жизни пакета может быть ограничено до известного максимума с помощью одного из следующих методов:

1. Проектирование сети с ограничениями.
2. Внедрение в каждый пакет счетчика транзитных участков.
3. Внедрение в каждый пакет временной метки.

К первому способу относятся все методы, предотвращающие зацикливание пакетов в комбинации с ограничением задержки, включая перегрузки по самому длинному возможному пути. Осуществить эту идею достаточно трудно, учитывая, что интерсеть может охватывать как один город, так и весь мир. Второй способ заключается в изначальной установке счетчика на определенное значение и уменьшении его на единицу на каждом маршрутизаторе. Сетевой протокол передачи данных просто игнорирует все пакеты, у которых значение счетчика достигло нуля. Третий способ состоит в том, что в каждом пакете указывается время его создания, а маршрутизаторы договариваются игнорировать все пакеты старше определенного времени. Для этого требуется синхронизация тактовых генераторов маршрутизаторов, что само по себе является нетривиальной задачей.

На самом деле возраст пакета можно довольно точно вычислять с помощью счетчика транзитных участков.

На практике нужно гарантировать не только то, что пакета больше нет, но и что все его подтверждения также исчезли. Поэтому вводится период $T$, который в несколько раз превышает максимальное время жизни пакета (для каждой сети оно является константой и выбирается с запасом; для интернета оно составляет 120 с). На какое число умножается максимальное время жизни пакета, зависит от протокола, и это влияет только на длительность $T$. Если подождать в течение $T$ с момента отправки пакета, то можно быть уверенными, что все следы этого пакета уничтожены и что он не возникнет вдруг как гром среди ясного неба.

При ограниченном времени жизни пакетов можно разработать надежный и практичный способ отвергать задержавшиеся дублированные сегменты. Автор описанного ниже метода — Томлинсон (Tomlinson, 1975); позднее он был улучшен Саншайном и Далалом (Sunshine and Dalal, 1978). Варианты этого метода широко используются на практике, и одним из примеров применения является TCP.

Основная идея метода заключается в том, что отправитель присваивает сегментам последовательные номера, которые не будут повторно использоваться в течение последующих $T$ секунд. Размер такого номера складывается из периода $T$ и скорости пакетов в секунду. Таким образом, в любой момент времени может отправляться только один пакет с данным номером. Копии этого пакета все равно могут появиться, и получатель должен их удалить.

Однако теперь невозможна ситуация, при которой задержавшийся дубликат принимается получателем вместо нового пакета с тем же порядковым номером.

Чтобы обойти проблему потери устройством сведений о предыдущих состояниях (при сбое), можно сделать так, чтобы транспортная подсистема оставалась неактивной в течение первых $T$ секунд после восстановления. В таком случае все старые сегменты исчезнут, и отправитель сможет начать процесс заново, используя любой последовательный номер. Недостаток этой идеи в том, что в крупных интерсетях период времени $T$ может быть достаточно большим.

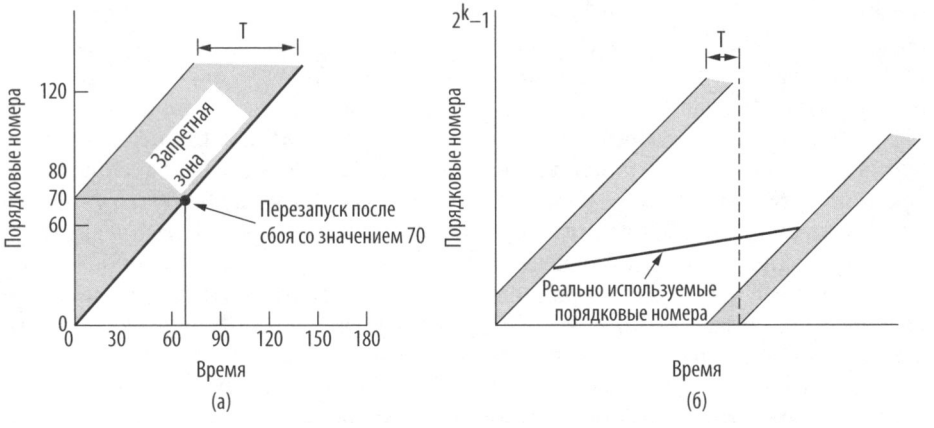
Вместо этого Томлинсон предложил снабдить каждый хост генератором импульсов истинного времени. Генераторы разных хостов синхронизировать не обязательно. Предполагается, что они представляют собой двоичные счетчики, которые увеличиваются через равные интервалы времени. Кроме того, число разрядов счетчика должно равняться числу битов в последовательных номерах (или превосходить его). Последнее и самое важное предположение состоит в том, что таймер продолжает работать, даже если хост зависает.

При установке соединения младшие $k$-биты генераторов импульсов используются в качестве $k$-битного начального порядкового номера. Таким образом, в отличие от протоколов, описанных в главе 3, каждое соединение начинает нумерацию своих сегментов с разных чисел. Диапазон этих номеров должен быть достаточно большим, чтобы к тому моменту, когда порядковые номера сделают полный круг, старые сегменты с такими же номерами уже давно исчезли. Линейная зависимость порядковых номеров от времени представлена на илл. 6.10 (а). Запретная зона показывает, в какой момент времени

определенные порядковые номера сегментов являются недействительными. При отправке сегмента с порядковым номером из этой зоны он может задержаться и сыграть роль другого пакета с таким же номером, который будет отправлен позже. К примеру, если хост выходит из строя и возобновляет работу в момент времени 70 с, он будет использовать начальные порядковые номера на основе генератора импульсов; хост не начинает отсчет с наименьшего порядкового номера в запретной зоне.

Как только обе транспортные подсистемы договариваются о начальном порядковом номере, для управления потоком данных может применяться любой протокол раздвижного окна. Такой протокол безошибочно найдет и удалит дубликаты пакетов, когда они уже будут приняты. В действительности график порядковых номеров (показанный жирной линией) не прямой, а ступенчатый, так как тактовые импульсы генерируются дискретно. Впрочем, для простоты мы проигнорируем эту деталь.

Чтобы порядковые номера пакетов не попадали в запретную зону, необходимо учесть следующее. Проблемы могут возникнуть по двум причинам. Если хост отправляет слишком быстро и слишком много данных, кривая используемых в действительности порядковых номеров может оказаться круче линии зависимости начальных номеров от времени. В результате порядковый номер попадет в запретную зону. Чтобы этого не произошло, скорость передачи данных в каждом открытом соединении должна быть ограничена одним сегментом за единицу времени. Кроме того, после восстановления транспортная подсистема не должна открывать новое соединение до появления нового тактового импульса, чтобы один и тот же номер не использовался дважды. Следовательно, интервал импульсов должен быть коротким (1 мкс или меньше). При этом генератор не должен работать слишком быстро (относительно порядковых номеров). Если частота импульсов равна $C$, а размер пространства порядковых номеров равен $S$, условие $S/C > T$ является обязательным, чтобы номера не сделали полный круг слишком быстро.




**Илл. 6.10.** (а) Сегменты не могут заходить в запретную зону. (б) Проблема ресинхронизации

В запретную зону можно попасть не только снизу, передавая данные слишком быстро. Как видно из илл. 6.10 (б), при любой скорости передачи ниже скорости генератора импульсов кривая фактически используемых порядковых номеров попадет в запретную зону слева, когда порядковые номера пройдут по кругу. Чем круче наклон этой кривой, тем дольше придется ждать этого события. Чтобы избежать такой ситуации, можно ограничить скорость продвижения порядковых номеров (или время жизни соединения).

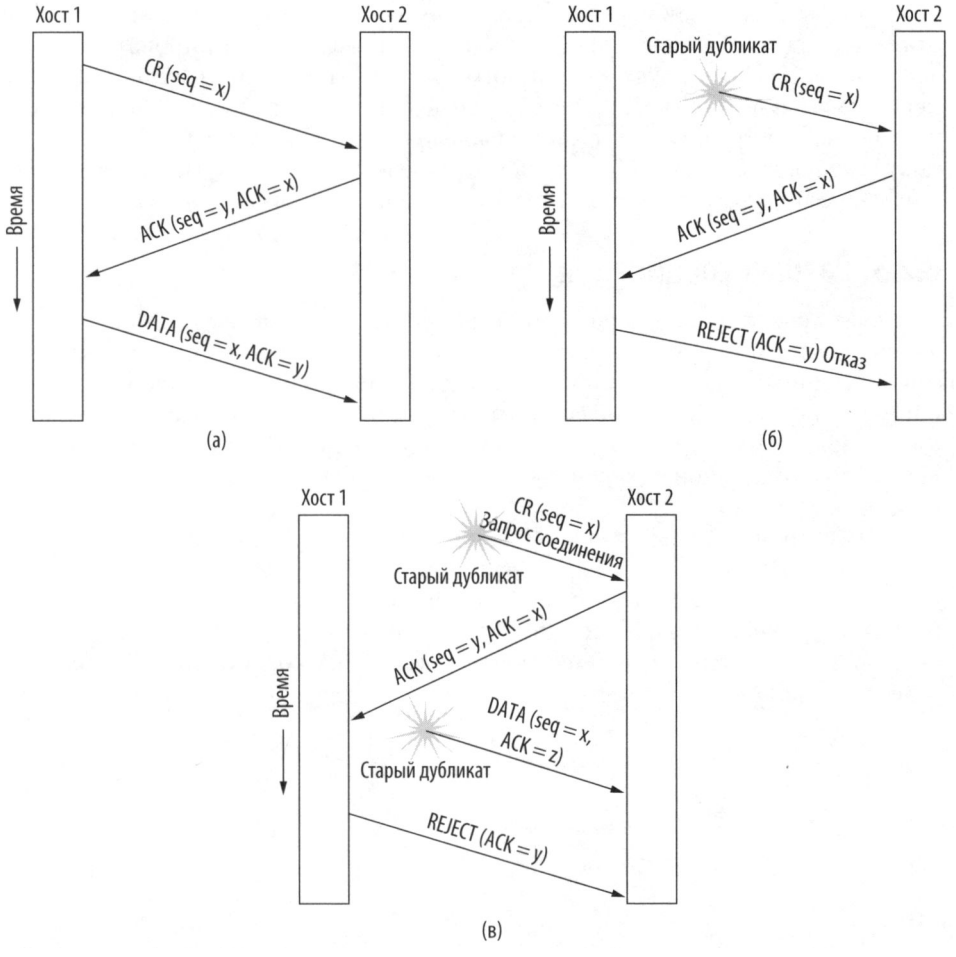
Метод, основанный на генераторе импульсов истинного времени, решает проблему распознавания опаздывающих дубликатов сегментов и новых сегментов. Однако использовать его для установления соединения может быть проблематично. Поскольку обычно получатель не помнит порядковые номера разных соединений, он не может узнать, является ли сегмент `CONNECTION REQUEST`, содержащий какой-либо начальный порядковый номер, дубликатом одного из предыдущих соединений. Во время соединения такой проблемы не возникает, поскольку протокол раздвижного окна знает текущий порядковый номер.

### *Решение: «тройное рукопожатие»*

Для разрешения этой специфической проблемы Томлинсон (1975) предложил **«тройное рукопожатие» (three-way handshake)**. Этот протокол установления соединения предполагает, что одна из сторон проверяет, является ли соединение все еще действующим. Нормальная процедура установления соединения показана на илл. 6.11 (а). Хост 1 выбирает порядковый номер $x$ и отправляет сегмент `CONNECTION REQUEST`, содержащий этот номер, хосту 2. Хост 2 отвечает сегментом `ACK`, подтверждая $x$ и объявляя свой начальный порядковый номер $y$. Наконец, хост 1 подтверждает выбранный хостом 2 номер в первом отправленном им информационном сегменте.

Теперь рассмотрим работу «тройного рукопожатия» при задержке дубликата управляющего сегмента. На илл. 6.11 (б) первый сегмент представляет собой задержавшийся дубликат `CONNECTION REQUEST` от старого соединения. Этот сегмент приходит на хост 2; хост 1 об этом не знает. Хост 2 реагирует на это отправкой хосту 1 сегмента `ACK`, таким образом запрашивая подтверждение того, что хост 1 действительно пытался установить новое соединение. Когда хост 1 отказывается это сделать (`REJECT`), хост 2 понимает, что он был обманут задержавшимся дубликатом, и прерывает соединение. В результате дубликат не причиняет вреда.

При наихудшем сценарии оба сегмента — `CONNECTION REQUEST` и `ACK` — блуждают по подсети. Этот случай показан на илл. 6.11 (в). Как и в предыдущем примере, хост 2 получает задержавшийся сегмент `CONNECTION REQUEST` и отвечает на него. Здесь нужно учитывать, что хост 2 предложил использовать $y$ в качестве начального порядкового номера для трафика от хоста 2 к хосту 1, хорошо зная, что сегментов, содержащих порядковый номер $y$, или их подтверждений в данный момент в сети нет. Когда хост 2 получает второй задержавшийся сегмент, он понимает, что это дубликат, так как в этом модуле подтверждается не $y$, а $z$. Важно понимать, что не существует такой комбинации сегментов, которая заставила бы протокол ошибиться и случайно установить соединение, когда оно никому не нужно.




**Илл. 6.11.** Три сценария установления соединения с помощью «тройного рукопожатия» (CR означает CONNECTION REQUEST). (а) Нормальная работа. (б) Появление старого дубликата CR. (в) Дубликат CR и дубликат ACK

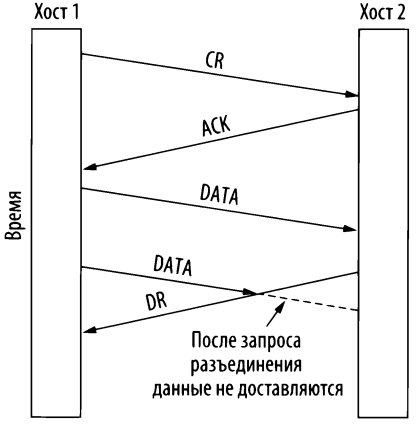
TCP всегда использует «тройное рукопожатие» для установления соединения. Внутри соединения к 32-битному порядковому номеру добавляется метка времени, чтобы он не мог использоваться повторно в течение максимального времени жизни пакета, даже если скорость соединения составляет несколько гигабитов в секунду. Этот механизм был добавлен в TCP для решения проблем, возникающих при использовании быстрых линий. Он описан в RFC 1323 и называется **защитой от повторного использования порядковых номеров (Protection Against Wrapped Sequence numbers, PAWS)**. До появления PAWS при наличии нескольких соединений с начальными порядковыми номерами TCP применял метод генератора импульсов (см. выше). Однако этот метод оказался

неэффективным с точки зрения безопасности. Злоумышленники могли легко угадать следующий начальный порядковый номер и инициировать ложное соединение, обманув схему «тройного рукопожатия». Поэтому на практике используются псевдослучайные начальные порядковые номера. Однако необходимо, чтобы эти номера, со стороны кажущиеся абсолютно случайными, не повторялись в течение определенного промежутка времени. Иначе задержавшиеся дубликаты могут вызвать серьезные неполадки в сети.

## 6.2.3. Разрыв соединения

Разорвать соединение проще, чем установить. Но здесь также имеются подводные камни. Как уже было сказано, существует два стиля разрыва соединения: асимметричный и симметричный. Асимметричный разрыв связи соответствует принципу работы телефонной системы: когда одна из сторон вешает трубку, связь прерывается. При симметричном разрыве соединение рассматривается в виде двух отдельных однонаправленных связей, и требуется отдельное завершение каждого соединения.

Асимметричный разрыв связи является внезапным и может привести к потере данных. Рассмотрим сценарий на илл. 6.12. После установления соединения хост 1 отправляет сегмент, который успешно достигает хоста 2. Затем хост 1 передает другой сегмент. К несчастью, хост 2 отправляет `DISCONNECTION REQUEST` прежде, чем приходит второй сегмент. В результате соединение разрывается, а данные теряются.




**Илл. 6.12.** Внезапное разъединение с потерей данных

Очевидно, требуется более сложный протокол, позволяющий избежать потери данных. Один из способов состоит в использовании симметричного варианта, при котором каждое направление разъединяется независимо. В этом случае хост может продолжать получать данные даже после того, как сам отправил запрос на разъединение.

Симметричное разъединение подходит для тех случаев, когда у каждой стороны есть фиксированное количество данных для передачи и каждая из них

точно знает, когда эти данные заканчиваются. В других случаях определить, что работа окончена и соединение может быть прервано, не так просто. Можно представить себе протокол, в котором хост 1 говорит: «Я закончил передачу. А вы?». Если хост 2 отвечает: «Я тоже. До свидания», соединение можно безо всякого риска разъединять.

К сожалению, этот протокол работает не всегда. Существует знаменитая **проблема «двух армий» (two-army problem)**. Представьте, что армия белых находится в долине (илл. 6.13). На возвышенностях по обеим сторонам долины расположились две армии синих. Белая армия больше, чем любая из синих, но вместе синие превосходят белых. Если одна из армий синих атакует белых в одиночку, она потерпит поражение, но если синие сумеют атаковать белых одновременно, они могут победить.

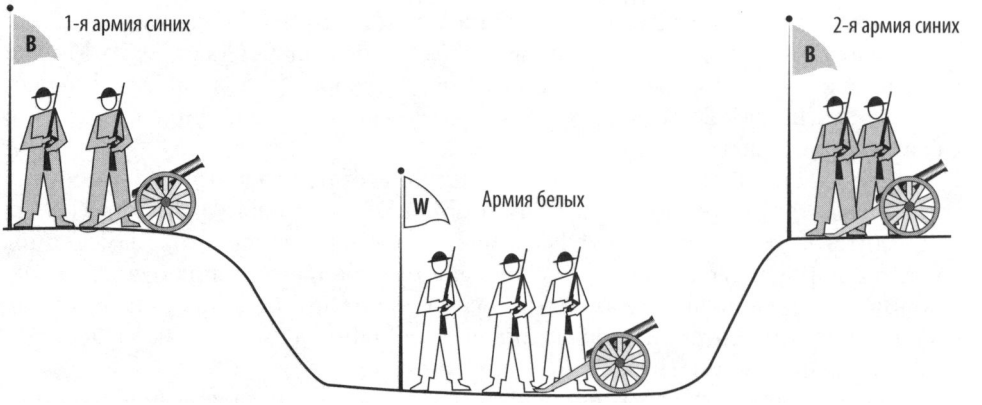
Армии синих хотели бы синхронизировать свое выступление. Однако единственный способ связи состоит в отправке связного пешком по долине, где он может быть схвачен, а донесение потеряно (то есть приходится пользоваться ненадежным каналом). Спрашивается: существует ли протокол, позволяющий армиям синих победить?

Предположим, командир 1-й армии синих отправляет следующее сообщение: «Я предлагаю атаковать 29 марта, на рассвете. Сообщите ваше мнение». Теперь предположим, что сообщение успешно доставляется и что командир 2-й армии синих соглашается, а его ответ успешно доставляется обратно в 1-ю армию. Состоится ли атака? Вероятно, нет, так как командир 2-й армии не уверен, что его ответ получен. Если нет, то 1-я армия синих не будет атаковать, и было бы глупо с его стороны в одиночку ввязываться в сражение.

Теперь улучшим протокол с помощью «тройного рукопожатия». Инициатор оригинального предложения должен подтвердить ответ. Если исходить из предположения, что сообщения не теряются, то 2-я армия синих получит подтверждение, но теперь будет сомневаться командир 1-й армии синих. Он не знает, было ли доставлено его подтверждение, а ведь если оно не доставлено, то 2-я армия не будет атаковать. Протокол четырехкратного «рукопожатия» здесь также не поможет.

В действительности можно доказать, что протокола, решающего данную проблему, не существует. Предположим, что такой протокол все же есть. В этом случае последнее сообщение протокола либо является важным, либо нет. Если оно не является важным, удалим его (а также все остальные несущественные сообщения), пока не останется протокол, в котором все сообщения являются существенными. Что произойдет, если последнее сообщение не дойдет до адресата? Мы только что решили, что сообщение является важным, поэтому если оно потеряется, атака не состоится. Поскольку отправитель последнего сообщения никогда не сможет быть уверенным в его получении, он не станет рисковать. Другая синяя армия это знает и также воздержится от атаки.

Чтобы увидеть, какое отношение проблема «двух армий» имеет к разрыву соединения, просто замените слово «атаковать» на «разъединить». Если ни одна сторона не может разорвать соединение до тех пор, пока она не уверена, что другая сторона также готова к этому, то разъединения не произойдет никогда.




**Илл. 6.13.** Проблема «двух армий»

На практике, чтобы справиться с этой ситуацией, нужно отказаться от идеи договоренности и переложить ответственность на пользователей, чтобы каждая сторона принимала независимое решение о том, когда будет выполнена операция. Такую проблему решить проще. На илл. 6.14 показаны четыре сценария разъединения, использующих «тройное рукопожатие». Этот протокол не является безошибочным, но обычно он работает успешно.

На илл. 6.14 (а) показан нормальный случай, при котором пользователь отправляет DR (DISCONNECTION REQUEST), чтобы инициировать разрыв соединения. Когда запрос доставляется, получатель также отвечает сегментом DR и включает таймер на случай его потери. Когда запрос прибывает, первый пользователь отправляет в ответ на него сегмент ACK и разрывает соединение. Наконец, когда ACK приходит, получатель также разрывает соединение. Разъединение означает, что транспортная подсистема удаляет информацию об этой связи из своей таблицы открытых соединений и сигнализирует о разрыве владельцу соединения (потребителю транспортной службы). Эта процедура отличается от применения пользователем примитива DISCONNECT.

Если последний сегмент ACK теряется (илл. 6.14 (б)), ситуацию спасает таймер. Когда время истекает, соединение разрывается в любом случае.

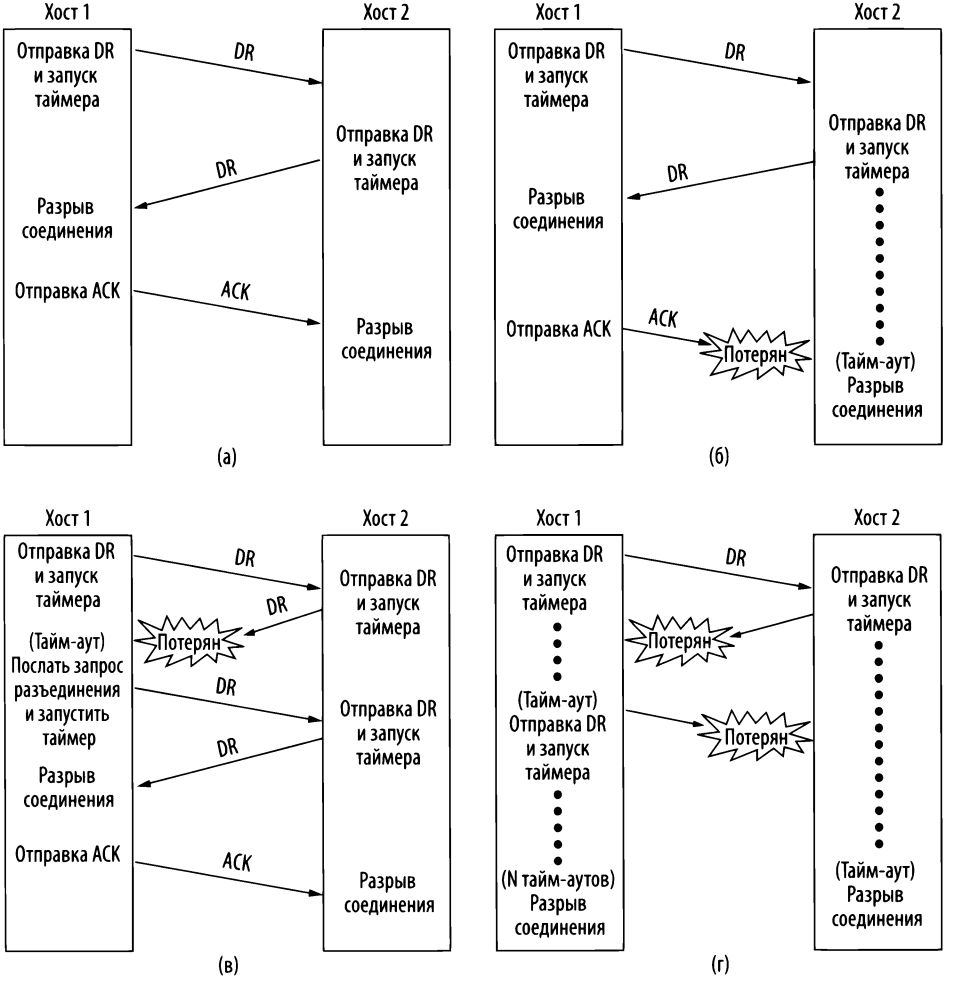
Теперь рассмотрим случай потери второго DR. Пользователь, инициировавший разъединение, не получит ожидаемого ответа, у него истечет время ожидания, и он начнет все сначала. На илл. 6.14 (в) показано, как это происходит, если все последующие запросы и подтверждения успешно доходят до адресатов.

Последний сценарий (илл. 6.14 (г)) аналогичен предыдущему с одной лишь разницей: в этом случае предполагается, что все повторные попытки передать DR также терпят неудачу, поскольку все сегменты теряются. После $N$ попыток отправитель, наконец, сдается и разрывает соединение. Тем временем у получателя также истекает время, и он тоже отсоединяется.

Хотя такого протокола обычно бывает вполне достаточно, теоретически он может ошибиться, если потеряются изначальный DR и все $N$ повторных передач. Отправитель сдается и разрывает соединение, тогда как другая сторона ничего

не знает о попытках разорвать связь и сохраняет активность. В результате получается полуоткрытое соединение, что недопустимо.

Этой ситуации можно избежать, если не позволить отправителю сдаваться после $N$ повторных передач, а заставить его продолжать попытки, пока не будет получен ответ. Однако если другой стороне будет разрешено разрывать связь по таймеру, тогда отправитель действительно будет вечно повторять попытки, так как ответа он не получит никогда. Если же получателю также не разрешать разрывать соединение по таймеру, то протокол зависнет в ситуации, изображенной на илл. 6.14 (г).




**Илл. 6.14.** Четыре сценария разрыва соединения. (а) Нормальный случай «тройного рукопожатия». (б) Потеряно последнее подтверждение. (в) Потерян ответ. (г) Потерян ответ и последующие запросы разъединения

Чтобы избавиться от полуоткрытых соединений, используется правило, гласящее, что если по соединению в течение определенного времени не приходит ни одного сегмента, оно автоматически разрывается. Таким образом, если одна сторона разорвет соединение, другая обнаружит отсутствие активности и также отсоединится. Правило работает и в тех случаях, когда соединение разрывается не по инициативе одной из сторон, а по причине невозможности передачи пакетов между этими хостами в сети.

Для реализации этого правила каждая сторона должна иметь таймер, который перезапускается после отправки каждого сегмента. Если таймер срабатывает, передается пустой сегмент — чтобы другая сторона не отсоединилась. Но если применяется правило автоматического разъединения и передается слишком много пустых сегментов подряд, только чтобы линия не простаивала, то соединение автоматически разрывается сначала одной стороной, а затем и другой.

На этом мы заканчиваем обсуждение данного вопроса, но теперь должно быть ясно, что разорвать соединение без потери данных не так просто, как это кажется на первый взгляд. Из всего сказанного можно сделать вывод, что пользователь службы должен принимать участие в решении вопроса о разъединении — транспортная подсистема не может с этим справиться самостоятельно. Обратите внимание, что хотя TCP обычно использует симметричный разрыв связи (при этом каждая сторона независимо прерывает свое соединение, отправляя пакет `FIN` после окончания передачи данных), веб-серверы часто передают клиентам специальный пакет `RST`, сообщающий о мгновенном разрыве соединения, что больше похоже на асимметричный разрыв. Это возможно только потому, что веб-сервер знаком с процедурой обмена данными. Сначала он получает запрос от клиента (единственное, что отправляет клиент), а затем отсылает ему ответ.

После отправки ответа обмен данными завершен: каждая сторона передала другой то, что требовалось. Поэтому сервер может резко разорвать соединение, отправив клиенту предупреждение. Получив его, клиент сразу же разорвет соединение. Если предупреждение не дойдет до клиента, он через какое-то время поймет, что сервер с ним больше не общается, и отсоединится. В любом случае данные будут успешно переданы.

### 6.2.4. Контроль ошибок и управление потоком данных

Изучив процессы установления и разрыва соединения, рассмотрим, как происходит управление соединением во время его использования. Ключевыми проблемами являются контроль ошибок и управление потоком данных. Контроль ошибок отвечает за передачу данных с необходимой степенью надежности: как правило, это означает, что данные должны доставляться без ошибок. Управление потоком состоит в согласовании скорости источника и получателя.

Оба этих вопроса мы уже обсуждали, когда говорили о канальном уровне в главе 3. На транспортном уровне используются те же механизмы. Вкратце они выглядят так.

1. Фрейм содержит код с обнаружением ошибок (например, CRC-код или контрольную сумму), с помощью которого проверяется, правильно ли была доставлена информация.

2. Фрейм содержит идентифицирующий порядковый номер и передается отправителем до тех пор, пока не придет подтверждение об успешной доставке. Это называется **автоматическим запросом повторной передачи (Automatic Repeat reQuest, ARQ)**.
3. Число фреймов, передаваемых отправителем в любой момент времени, обычно ограниченно: передача приостанавливается, если подтверждения приходят недостаточно быстро. Если этот максимум равен одному пакету, протокол называется **протоколом с остановкой и ожиданием подтверждения (stop-and-wait).** Окна большего размера позволяют использовать конвейерную обработку, а также улучшить производительность при работе с длинными и быстрыми линиями.
4. Протокол **раздвижного окна (sliding window)** сочетает в себе все эти возможности, а также поддерживает двунаправленную передачу данных.

Если эти механизмы применяются к фреймам на канальном уровне, то возникает логичный вопрос: как это можно перенести на сегменты транспортного уровня? На практике выясняется, что эти уровни во многом копируют друг друга. Но хотя к ним и применимы одинаковые механизмы, существуют различия в их функционировании и качестве.

Чтобы проиллюстрировать функциональное различие, обратимся к обнаружению ошибок. Контрольная сумма на канальном уровне защищает фрейм, пока он передается по каналу. На транспортном уровне она защищает сегмент на всем протяжении пути. Это сквозная проверка, которая отличается от проверки на каждом канале. Существуют примеры повреждения пакетов даже внутри маршрутизаторов (Зальцер и др.; Saltzer et al., 1984). Контрольные суммы канального уровня обеспечивали защиту пакетов, пока они передвигались по каналу, но не тогда, когда они были внутри маршрутизатора. В результате мы имеем дело с некорректной доставкой, хотя проверка на каждом канале не выявила ошибок.

Этот и другие примеры позволили Зальцеру и др. сформулировать **«сквозной» принцип (end-to-end argument)**. Согласно этому принципу, сквозная проверка на транспортном уровне необходима для корректной передачи данных; проверка на канальном уровне не является необходимой, но позволяет существенно улучшить производительность (так как иначе поврежденный пакет будет все равно проходить весь путь, что приведет к лишней нагрузке на сеть).

Чтобы продемонстрировать разницу в качестве, рассмотрим повторную передачу данных и протокол раздвижного окна. Большинство беспроводных каналов, в отличие от спутниковых, позволяют отправлять только один фрейм за единицу времени. Это значит, что произведение пропускной способности и времени задержки для канала достаточно мало и внутри канала едва ли может поместиться целый фрейм. Для лучшей производительности следует использовать окно маленького размера. К примеру, стандарт 802.11 применяет протокол с остановкой и ожиданием подтверждения. Он выполняет передачу и повторные передачи одного фрейма до тех пор, пока не придет подтверждение о его получении, и только после этого переходит к другому фрейму. Если бы размер окна был больше одного фрейма, это не повысило бы производительность, а только

усложнило процесс передачи. В проводных и оптоволоконных линиях связи, таких как Ethernet (коммутируемый) и магистрали интернет-провайдеров, частота появления ошибок невелика. Это позволяет отказаться от повторных передач на канальном уровне, так как небольшое количество потерянных фреймов может быть восстановлено с помощью повторных сквозных передач.

С другой стороны, для многих TCP-соединений произведение пропускной способности и времени задержки составляет гораздо больше, чем один сегмент. Рассмотрим соединение, которое передает данные по всей территории США со скоростью 1 Мбит/с, а время, за которое пакет доходит до получателя и обратно, составляет 200 мс. Даже при таком медленном соединении 200 Кбит информации будут храниться на стороне получателе столько, сколько требуется для отправки сегмента и получения подтверждения. В таких случаях следует использовать раздвижное окно большого размера. Протокол с остановкой и ожиданием подтверждения серьезно навредит производительности. В нашем примере он ограничил бы передачу одним сегментом в 200 мс (это 5 сегментов/с) независимо от реальной скорости сети.

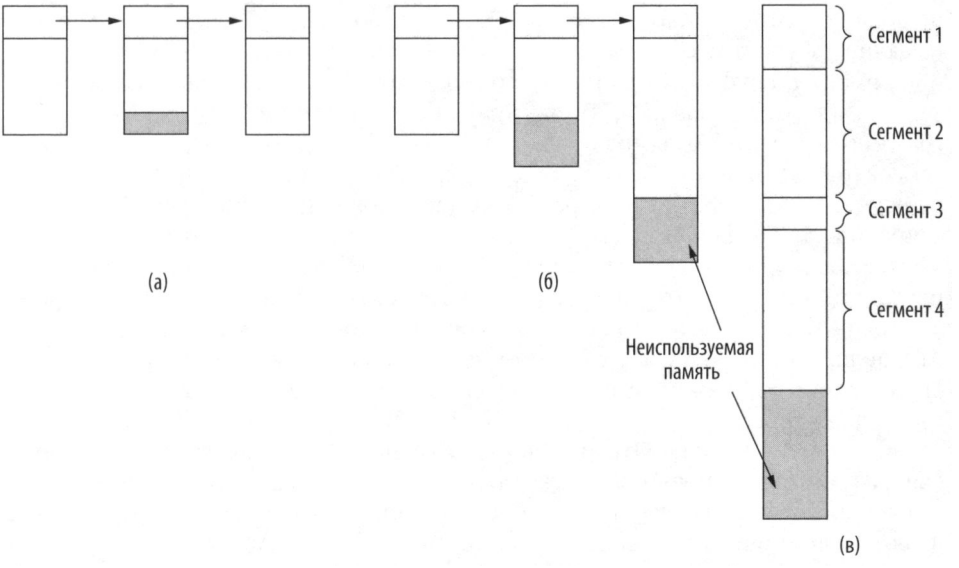
Поскольку транспортные протоколы обычно используют окно большего размера, мы обсудим буферизацию более подробно. Так как у хоста может быть несколько соединений, каждое из которых обрабатывается отдельно, для раздвижных окон ему может потребоваться большое буферное пространство. Буферы должны использоваться как отправителем, так и получателем. Отправителю буфер нужен для хранения всех переданных сегментов, для которых еще не пришло подтверждение о доставке, на случай, если эти сегменты потеряются и их придется отправить повторно.

Зная, что отправитель выполняет буферизацию, получатель может использовать свои буферы как посчитает нужным. Например, он может содержать единый буферный накопитель для всех соединений. Когда приходит сегмент, предпринимается попытка динамически выделить ему новый буфер. Если это удается, то сегмент принимается, в противном случае он отвергается. Отправитель готов к тому, чтобы передавать потерянные сегменты повторно, и это игнорирование сегментов не наносит большого вреда, хотя расходует некоторые ресурсы. Отправитель просто повторяет попытки до тех пор, пока не получит подтверждение.

Выбор компромиссного решения между буферизацией на сторонах отправителя и получателя зависит от типа трафика. В случае неинтенсивного и нестабильного трафика, например, когда пользователь производит клавиатурный ввод на удаленном компьютере, лучше не выделять никаких буферов, а получать их динамически на обоих концах, полагаясь на буферизацию со стороны отправителя (на случай, если сегменты будут случайно удалены). С другой стороны, в случае передачи файла будет лучше, если получатель выделит целое окно буферов, чтобы данные могли передаваться с максимальной скоростью. Такую стратегию использует TCP.

Тем не менее открытым остается вопрос об организации набора буферов. Когда большинство сегментов примерно одинакового размера, логично организовать буферы в виде массива буферов равной величины, каждый из которых может вместить один сегмент (илл. 6.15 (а)). Но если сегменты сильно отличаются

по размеру, от коротких запросов на загрузку веб-страниц до крупных пакетов при одноранговой передаче файлов, массив из буферов фиксированного размера окажется неудобным. Если буфер будет равен наибольшему возможному сегменту, то при хранении небольших сегментов память будет расходоваться неэффективно. Если же он будет меньше, тогда для хранения большого сегмента потребуется несколько буферов с сопутствующими сложностями.




**Илл. 6.15.** Организация набора буферов. (а) Цепочка буферов фиксированного размера. (б) Цепочка буферов переменного размера. (в) Один большой циклический буфер для одного соединения

Другой способ решения проблемы состоит в использовании буферов переменного размера (илл. 6.15 (б)). Преимущество этого метода заключается в оптимальном использовании памяти, но платой за это является более сложное управление буферами. Третий вариант состоит в выделении соединению единого большого циклического буфера (илл. 6.15 (в)). Эта простая и изящная схема работает независимо от размера сегментов, однако она эффективно использует память, только если все соединения сильно нагружены.

При открытии и закрытии соединений и при изменении формы трафика отправитель и получатель должны динамически изменять выделенные буферы. Следовательно, транспортный протокол должен позволять отправителю отправлять запросы на выделение буфера на другой стороне. Буферы могут выделяться для каждого соединения или коллективно на все соединения между двумя хостами. В качестве альтернативы запросам получатель, зная состояние своих буферов, но не зная, какой трафик ему будет предложен, может сообщить отправителю, что он зарезервировал для него $X$ буферов. При увеличении числа открытых соединений может потребоваться уменьшить количество или размеры выделенных буферов. Протокол должен предоставлять такую возможность.

В отличие от протокола раздвижного окна, описанного в главе 3, для реализации динамического выделения буферов следует отделить буферизацию от подтверждений. Динамическое выделение буферов на деле означает использование окна переменного размера. Сначала отправитель запрашивает определенное число буферов согласно своим потребностям. Получатель выделяет столько, сколько может. При отправлении каждого сегмента отправитель должен уменьшать число буферов на единицу, а когда оно достигнет нуля, он должен остановиться. Получатель отправляет обратно на попутных сегментах отдельно подтверждения и сведения об имеющихся у него свободных буферах. Эта схема используется в TCP; при этом информация о буферах хранится в поле заголовка `Window size` (Размер окна).

На илл. 6.16 показан пример управления динамическим окном в дейтаграммной сети с 4-битными порядковыми номерами. Данные передаются в виде сегментов от хоста *A* к хосту *B*, а подтверждения и запросы на предоставление буферов идут в обратном направлении (также в виде сегментов). Вначале *A* запрашивает 8 буферов, но ему выделяется только 4. Затем он отправляет 3 сегмента; последний теряется. На шаге 6 *A* получает подтверждение получения переданных им сегментов 0 и 1, в результате чего он может освободить буферы и отправить еще 3 сегмента (2, 3 и 4). Хост *A* знает, что сегмент номер 2 он уже отправлял, поэтому думает, что может передать сегменты 3 и 4 (что он и делает). На этом шаге он блокируется, так как его счетчик буферов достиг нуля и ждет предоставления новых буферов. На шаге 9 наступает тайм-аут хоста *A*, так как он до сих пор не получил подтверждения для сегмента 2. Этот сегмент отсылается еще раз. В строке 10 хост *B* подтверждает получение всех сегментов, включая 4-й, но отказывается предоставлять буферы хосту *A*. Такая ситуация невозможна в протоколах с фиксированным размером окна, описанных в главе 3. Следующий сегмент, отправленный хостом *B*, разрешает хосту *A* передать еще один сегмент. Это происходит, когда у *B* появляется свободное буферное пространство, — скорее всего, потому, что пользователь службы принял больше данных.

Проблемы при такой схеме выделения буферов в дейтаграммных сетях могут возникнуть, если потеряется управляющий сегмент (а это действительно может произойти). Взгляните на строку 16. Хост *B* выделил хосту *A* дополнительные буферы, но сообщение об этом потерялось. Вот так неожиданность! Поскольку получение управляющих сегментов не подтверждается и они не отправляются повторно по тайм-ауту, хост *A* блокируется всерьез и надолго. Для предотвращения такой тупиковой ситуации каждый хост должен периодически отправлять управляющий сегмент с подтверждением и сведениями о состоянии буферов для каждого соединения. Это позволит в конце концов выбраться из тупика.

До сих пор мы предполагали, что единственное ограничение, накладываемое на скорость передачи данных, состоит в объеме свободного буферного пространства у получателя. Однако часто это не так. По мере колоссального снижения цен (некогда очень высоких) на микросхемы памяти и жесткие диски становится возможным оборудовать хосты таким количеством памяти, что проблема нехватки буферов возникает очень редко, даже если соединение охватывает крупные территории. Конечно, выбранный буфер должен быть достаточно большим; в случае TCP это требование не всегда выполнялось (Чжан и др.; Zhang et al., 2002).

| | A | Сообщение | B | Комментарии |
|---|---|---|---|---|
| 1 | → | < request 8 buffers> | → | A хочет 8 буферов |
| 2 | ← | <ack = 15, buf = 4> | ← | B позволяет переслать только сообщения 0–3 |
| 3 | → | <seq = 0, data = m0> | → | У A теперь осталось 3 буфера |
| 4 | → | <seq = 1, data = m1> | → | У A теперь осталось 2 буфера |
| 5 | → | <seq = 2, data = m2> | • • • | Сообщение потеряно, но A думает, что у него остался 1 буфер |
| 6 | ← | <ack = 1, buf = 3> | ← | B подтверждает получение сегментов 0 и 1, разрешает передать сегменты 2–4 |
| 7 | | | | |
| 8 | → | <seq = 3, data = m3> | → | У A остался 1 буфер |
| 9 | → | <seq = 4, data = m4> | → | У A осталось 0 буферов, и он должен остановиться |
| 10 | → | <seq = 2, data = m2> | → | У A истекло время ожидания, и он передает еще раз |
| 11 | ← | <ack = 4, buf = 0> | ← | Все сегменты подтверждены, но A все еще блокирован |
| 12 | ← | <ack = 4, buf = 1> | ← | Теперь A может отправить сегмент 5 |
| 13 | ← | <ack = 4, buf = 2> | ← | B где-то нашел новый буфер |
| 14 | → | <seq = 5, data = m5> | → | У A остался 1 буфер |
| 15 | → | <seq = 6, data = m6> | → | A снова блокирован |
| 16 | ← | <ack = 6, buf = 0> | ← | A все еще блокирован |
| | • • • | <ack = 6, buf = 4> | ← | Потенциальный тупик |

**Илл. 6.16.** Динамическое выделение буферов. Стрелками показано направление передачи. Многоточие (...) означает потерянный сегмент

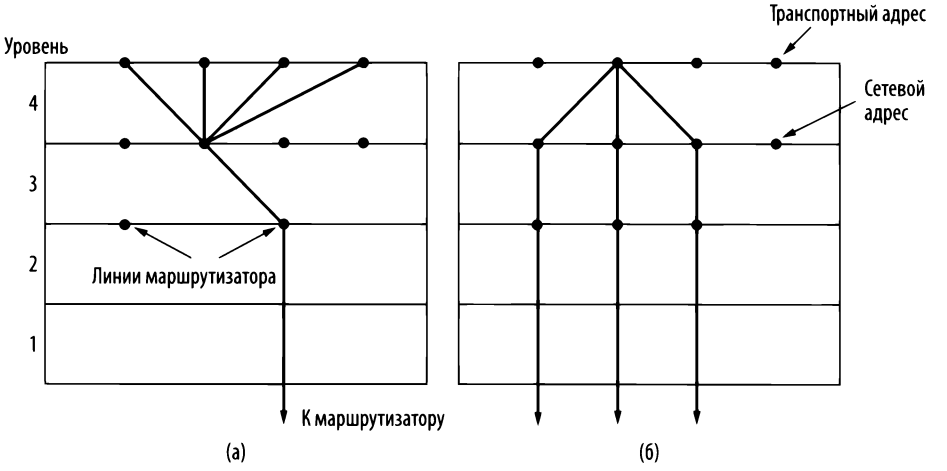
Если размер буферов перестанет ограничивать максимальный поток, возникнет другое узкое место: пропускная способность сети. Допустим, максимальная скорость обмена фреймами между соседними маршрутизаторами составляет $x$ фреймов в секунду, а между двумя хостами имеется $k$ непересекающихся путей. Сколько бы ни было буферов у обоих хостов, они не смогут пересылать друг другу больше чем $kx$ сегментов в секунду. И если отправитель будет передавать быстрее, то сеть будет перегружена.

Требуется механизм, ограничивающий передачу данных со стороны отправителя, который основан не столько на емкости буферов получателя, сколько на пропускной способности сети. В 1975 году Белснес (Belsnes) предложил использовать для управления потоком данных схему раздвижного окна, в которой отправитель динамически приводит размер окна в соответствие с пропускной способностью сети.

Таким образом, раздвижное окно позволяет одновременно реализовать и управление потоком, и контроль перегрузки. Если сеть может обработать $c$ сегментов в секунду, а время цикла (включая передачу, распространение, ожидание в очередях, обработку получателем и возврат подтверждения) равно $r$, тогда размер окна отправителя должен быть равен $cr$. При таком окне отправитель максимально использует канал. Любое снижение производительности сети приведет к его блокировке. Так как пропускная способность меняется с течением времени, размер окна должен настраиваться довольно часто, чтобы отслеживать изменения пропускной способности. Как будет показано ниже, в TCP используется похожая схема.

## 6.2.5. Мультиплексирование

**Мультиплексирование (multiplexing)**, или объединение нескольких сеансов связи в одном соединении, виртуальном канале и одной физической линии, играет важную роль на нескольких уровнях сетевой архитектуры. На транспортном уровне такая потребность возникает в нескольких случаях. Например, если у хоста имеется только один сетевой адрес, он используется всеми соединениями транспортного уровня. Необходим способ, с помощью которого можно различать, какому процессу нужно передать входящий сегмент. Пример мультиплексирования показан на илл. 6.17 (а): четыре различных соединения транспортного уровня используют одно сетевое соединение (например, один IP-адрес) с удаленным хостом.




**Илл. 6.17.** Мультиплексирование: (а) прямое; (б) обратное

Мультиплексирование может применяться на транспортном уровне и по другой причине. Предположим, что хост может использовать несколько различных сетевых путей. Если пользователю требуется больше пропускной способности или более высокая надежность, чем может предоставить один сетевой путь, можно создать соединение, распределяющее трафик между путями, используя их поочередно (илл. 6.17 (б)). Такой метод называется **обратным мультиплексированием (inverse multiplexing)**. При открытии $k$ сетевых соединений эффективная пропускная способность может увеличиться в $k$ раз. Примером обратного мультиплексирования является протокол SCTP, позволяющий устанавливать соединение с множественными сетевыми интерфейсами. TCP, наоборот, использует отдельный сокет. Обратное мультиплексирование применяется и на канальном уровне; при этом несколько медленных каналов связи объединяются в один, работающий гораздо быстрее.

### 6.2.6. Восстановление после сбоев

Если хосты и маршрутизаторы подвержены сбоям или соединения длятся достаточно долго (например, при загрузке мультимедиа или программного обеспечения), вопрос восстановления после сбоев особенно актуален. Если транспортная подсистема полностью размещается в хостах, восстановление после отказов сети и маршрутизаторов не вызывает затруднений. Транспортные подсистемы постоянно ожидают потери сегментов и знают, как с этим бороться: для этого выполняются повторные передачи.

Более серьезную проблему представляет восстановление после сбоя хоста. В частности, клиентам может потребоваться возможность продолжать работу после отказа и быстрой перезагрузки сервера. Чтобы пояснить, в чем тут сложность, предположим, что один хост (клиент) отправляет длинный файл другому хосту (файловому серверу) с помощью простого протокола с остановкой и ожиданием подтверждения. Транспортный уровень сервера просто передает приходящие сегменты один за другим пользователю транспортного уровня. Получив половину файла, сервер сбрасывается и перезагружается, после чего все его таблицы заново инициализируются, поэтому он не знает, на чем он остановился.

Пытаясь восстановить предыдущее состояние, сервер может разослать широковещательный сегмент всем хостам, объявляя, что он только что перезагрузился, и обращаясь с просьбой к своим клиентам сообщить ему о состоянии всех открытых соединений. Каждый клиент находится в одном из двух состояний: один неподтвержденный сегмент (*S1*) или ни одного неподтвержденного сегмента (*S0*). Этой информации клиенту должно быть достаточно, чтобы решить, передавать ему повторно последний сегмент или нет.

На первый взгляд все очевидно: узнав о перезапуске сервера, клиент должен передать повторно последний неподтвержденный сегмент. То есть повторная передача требуется, только если клиент находится в состоянии *S1*. Однако при более детальном рассмотрении выясняется, что все не так просто. Например, рассмотрим ситуацию, в которой транспортная подсистема сервера сначала отправляет подтверждение и лишь затем передает сегмент прикладному процессу. Запись сегмента в выходной поток и отправка подтверждения — это два отдельных события, которые нельзя выполнить одновременно. Если сбой произойдет после отправки подтверждения, но до того, как выполнена запись, клиент получит подтверждение, а при перезапуске сервера окажется в состоянии *S0*. В результате он не отправит сегмент повторно, так как будет считать, что сегмент уже получен, что приведет к потере сегмента.

Должно быть, вы подумали: «А что, если поменять местами последовательность действий транспортной подсистемы сервера, чтобы сначала осуществлялась запись, а потом высылалось подтверждение?» Представим, что запись сделана, но сбой произошел до отправки подтверждения. Тогда клиент окажется в состоянии *S1* и передаст сегмент повторно, а мы получим дубликат сегмента в выходном потоке.

Таким образом, независимо от того, как запрограммированы клиент и сервер, всегда может возникнуть ситуация, в которой протокол не сможет правильно

восстановиться. Сервер можно запрограммировать двумя способами: так, чтобы он в первую очередь высылал подтверждение, или так, чтобы он сначала записывал сегмент. Клиент может быть запрограммирован одним из четырех способов: всегда повторно передавать последний сегмент; никогда не передавать повторно последний сегмент; передавать сегмент повторно только в состоянии *S0*; передавать сегмент повторно только в состоянии *S1*. Таким образом, получаем восемь комбинаций, но как будет показано далее, для каждой комбинации имеется набор событий, ведущий к ошибке протокола.

На сервере могут происходить три события: отправка подтверждения (*A*), запись сегмента в выходной процесс (*W*) и сбой (*C*). Они могут произойти в виде шести возможных последовательностей: *AC(W)*, *AWC*, *C(AW)*, *C(WA)*, *WAC* и *WC(A)*, где скобки означают, что после *C* события *A* или *W* уже не произойдут (ведь уже случился сбой). На илл. 6.18 показаны все восемь комбинаций стратегий сервера и клиента, каждая со своими последовательностями событий. Обратите внимание, что для каждой комбинации существует последовательность, приводящая к ошибке протокола. Например, если клиент всегда передает повторно неподтвержденный сегмент, событие *AWC* приведет к появлению неопознанного дубликата, хотя при двух других последовательностях протокол будет работать правильно.

Стратегия хоста-получателя

| Стратегия хоста-источника | Сначала ACK, потом запись | | | Сначала запись, потом ACK | | |
|---|---|---|---|---|---|---|
| | AC(W) | AWC | C(AW) | C(WA) | W AC | WC(A) |
| Всегда повторять передачу | OK | DUP | OK | OK | DUP | DUP |
| Никогда не повторять передачу | LOST | OK | LOST | LOST | OK | OK |
| Повторять передачу в S0 | OK | DUP | LOST | LOST | DUP | OK |
| Повторять передачу в S1 | LOST | OK | OK | OK | OK | DUP |

OK = Протокол работает корректно
DUP = Протокол формирует дубликат сообщения
LOST = Протокол теряет сообщение

**Илл. 6.18.** Различные комбинации стратегий сервера и клиента

Усложнение протокола не поможет. Даже если клиент и сервер обменяются несколькими сегментами, прежде чем сервер попытается осуществить запись, и клиент будет точно знать, что происходит, у него нет возможности определить, когда произошел сбой: до или после записи. Отсюда следует неизбежный вывод: при жестком условии отсутствия одновременных событий (то есть если отдельные события происходят друг за другом, а не одновременно) более высокие уровни не могут отследить сбой и восстановление хоста.

Все это можно обобщить следующим образом: восстановление после сбоя уровня *N* может быть осуществлено только уровнем $N + 1$ и только при условии,

что на более высоком уровне сохраняется информация о процессе, достаточная для возвращения в прежнее состояние. Это согласуется с утверждением о том, что транспортный уровень может обеспечить восстановление от сбоя на сетевом уровне, если каждая сторона соединения отслеживает свое текущее состояние.

Данная проблема подводит нас к вопросу о значении так называемого сквозного подтверждения. По сути, транспортный протокол является сквозным, а не последовательным, как более низкие уровни. Рассмотрим случай обращения пользователя к удаленной базе данных. Допустим, удаленная транспортная подсистема запрограммирована сначала передавать сегмент вышестоящему уровню, а затем отправлять подтверждение.

Даже в этом случае получение подтверждения устройством пользователя не означает, что удаленный хост успел обновить базу данных. Похоже, что настоящее сквозное подтверждение, получение которого означает, что работа сделана (а отсутствие означает обратное), невозможно. Более подробно этот вопрос обсуждается у Зальцера и др. (Saltzer et al., 1984).

## 6.3. КОНТРОЛЬ ПЕРЕГРУЗКИ

Если транспортные подсистемы нескольких компьютеров будут слишком быстро отправлять чересчур много пакетов, сеть будет перегружена и ее производительность резко снизится, что приведет к задержке и потере пакетов. Контроль перегрузки, направленный на борьбу с такими ситуациями, требует совместной работы сетевого и транспортного уровней. Так как перегрузки происходят на маршрутизаторах, их обнаружением занимается сетевой уровень. Но изначальной причиной этой проблемы является трафик, переданный транспортным уровнем в сеть. Поэтому единственный эффективный способ контроля перегрузки состоит в более медленной передаче пакетов транспортными протоколами.

В главе 5 мы говорили о механизмах контроля перегрузки на сетевом уровне. В этом разделе мы рассмотрим механизмы, действующие на транспортном уровне. После обсуждения основных задач контроля перегрузки мы изучим методы, позволяющие хостам регулировать скорость передачи пакетов в сеть. Контроль перегрузки в интернете во многом опирается на транспортный уровень; для этого в TCP и другие протоколы встроены специальные алгоритмы.

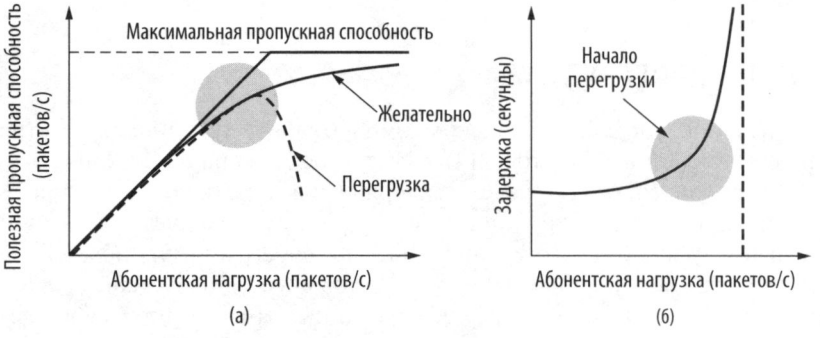
### 6.3.1. Выделение требуемой пропускной способности

Перед тем как перейти к описанию методов регулирования трафика, необходимо понять, чего мы хотим от алгоритма контроля перегрузки. Мы должны определить, какое состояние сети он должен поддерживать. В задачи этого алгоритма входит не только предотвращение перегрузок: он должен правильно распределять пропускную способность между транспортными подсистемами, работающими в сети. Правильное распределение пропускной способности должно обеспечивать хорошую производительность (сеть должна работать с использованием всей доступной мощности без перегрузок), соответствовать принципу равноправия конкурирующих транспортных подсистем и быстро отслеживать изменения

в запросах на выделение ресурсов. Мы рассмотрим каждый из этих критериев отдельно.

### *Эффективность и мощность*

При эффективном распределении пропускной способности между транспортными подсистемами должны использоваться все возможности сети. Однако это не значит, что в канале со скоростью 100 Мбит/с каждая из пяти подсистем получит по 20 Мбит/с. Для хорошей производительности им необходимо выделить меньшую мощность. Причина в том, что трафик часто бывает неравномерным. В разделе 5.3 мы определили **полезную пропускную способность (goodput)**, скорость, с которой полезные пакеты приходят к получателю, как функцию абонентской нагрузки на сеть. Эта кривая и соответствующая ей кривая задержки (как функция нагрузки) приведены на илл. 6.19.




**Илл. 6.19.** (а) Полезная пропускная способность. (б) Задержка как функция нагрузки

С повышением нагрузки на сеть полезная пропускная способность увеличивается с постоянной скоростью (илл. 6.19 (а)), но по мере того как их значения сближаются, рост полезной пропускной способности замедляется. Этот спад необходим, чтобы предотвратить переполнение буферов сети и потерю данных в случае всплесков трафика. Если транспортный протокол повторно передает задерживающиеся, но не потерянные пакеты, может произойти отказ сети из-за перегрузки. В таком случае отправители продолжают передавать все больше и больше пакетов, а пользы от этого все меньше и меньше.

График задержки пакетов приведен на илл. 6.19 (б). Сначала она постоянна и соответствует задержке распространения в сети. Когда значение нагрузки приближается к значению пропускной способности, задержка возрастает, сначала медленно, а затем все быстрее. Это также происходит из-за всплесков трафика, которые возрастают при высокой нагрузке. Задержка не уйдет в бесконечность (разве что в модели, где у маршрутизаторов бесконечный буфер). Вместо этого пакеты будут теряться при переполнении буферов.

Что касается полезной пропускной способности и задержки, производительность начинает снижаться в момент возникновения перегрузки. Логично,

что мы получим наилучшую производительность сети, если будем выделять пропускную способность, пока задержка не начнет быстро расти. Эта точка находится ниже пропускной способности сети. Чтобы ее определить, Клейнрок (Kleinrock) в 1979 году предложил ввести метрику **мощность (power)**, которая вычисляется по формуле:

$$\text{Мощность} = \frac{\text{Нагрузка}}{\text{Задержка}}.$$

Пока задержка достаточно мала и практически постоянна, мощность растет с ростом нагрузки на сеть; при резком повышении задержки она достигает максимума и резко снижается. Нагрузка, при которой мощность становится максимальной, и является наиболее эффективной нагрузкой, которую транспортная подсистема может передавать в сеть. Сеть должна стремиться к этому уровню.

### *Равнодоступность по максиминному критерию*

Мы еще не говорили о том, как распределять пропускную способность между несколькими отправителями. Кажется, что ответ очень прост: можно разделить пропускную способность между ними поровну. Но на самом деле все намного сложнее.

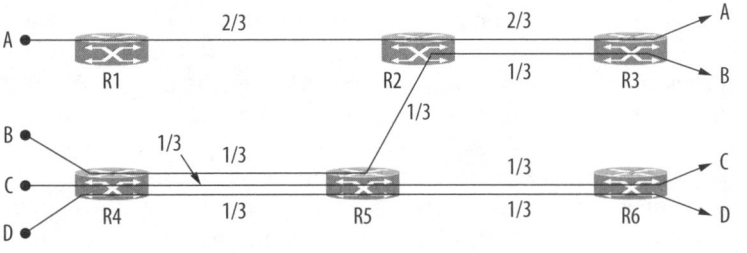
Во-первых, какое отношение эта проблема имеет к контролю нагрузки? Ведь если сеть предоставляет отправителю некоторое количество пропускной способности, он должен просто использовать то, что у него есть. Но на практике сети часто не резервируют строго ограниченное количество пропускной способности для потока или соединения. Конечно, они могут это делать для некоторых потоков (если есть поддержка QoS), но как правило, соединения стремятся использовать максимум доступной пропускной способности; также возможен вариант, когда сеть выделяет ресурсы группе соединений для совместного использования. К примеру, дифференцированное обслуживание IETF делит трафик на два класса, и соединения конкурируют друг с другом в пределах каждого из этих классов. Часто все соединения одного IP-маршрутизатора используют общую пропускную способность. В таких случаях распределение ресурсов между конкурирующими соединениями должно осуществляться за счет механизмов контроля нагрузки.

Во-вторых, не совсем ясно, что означает равнодоступность для потоков в сети. Если $N$ потоков используют один канал, то решение довольно простое: каждому выделяется $1/N$ пропускной способности (в случае импульсного трафика эта величина немного меньше по соображениям эффективности). Но что, если потоки используют разные, но пересекающиеся пути? К примеру, если один поток проходит по трем каналам, а остальные — по одному? Очевидно, что поток, идущий по трем каналам, потребляет больше сетевых ресурсов. Поэтому в некотором смысле справедливее было бы выделить ему меньше пропускной способности, чем остальным. Кроме того, неплохо было бы добавить новые одноканальные потоки за счет уменьшения пропускной способности трехканального. На этом примере видно, что между эффективностью и равнодоступностью всегда существуют противоречия.

Мы будем использовать такое определение равнодоступности, при котором она не зависит от длины сетевого пути. Даже в такой простой модели предоставление соединениям равных долей пропускной способности — непростая задача, так как разные соединения будут использовать различные пути, каждый со своей пропускной способностью. В этом случае один из потоков может замедлиться на входящем канале и получить меньшую долю исходящего канала по сравнению с другими потоками. Уменьшение пропускной способности для этих потоков лишь замедлит их работу, но не исправит ситуацию с «застрявшим» потоком.

Часто в сетях удобнее использовать **равнодоступность по максиминному критерию (max-min fairness)**. Это значит, что увеличение пропускной способности одного потока невозможно без ее уменьшения для какого-либо другого потока с меньшей или равной пропускной способностью. Иными словами, от увеличения пропускной способности потока страдают менее «обеспеченные» потоки.

Рассмотрим пример. На илл. 6.20 показано распределение пропускной способности по максиминному критерию в сети с четырьмя потоками: *A*, *B*, *C* и *D*. Все каналы между маршрутизаторами имеют одинаковую пропускную способность, принятую за 1 (хотя на практике она обычно разная). На пропускную способность канала *R4–R5*, расположенного слева внизу, претендуют три потока. Поэтому каждый из них получает по 1/3. Оставшийся поток *A* конкурирует с *B* на участке *R2–R3*. Поскольку для *B* уже выделена 1/3 емкости канала, *A* получает оставшееся количество, 2/3. Как видно из рисунка, на остальных каналах часть ресурсов не используется. Но их нельзя отдать какому-либо потоку, не снизив пропускную способность другого, более медленного. К примеру, если на участке *R2–R3* выделить больше емкости потоку *B*, то *A* достанется меньше. Это логично, поскольку на данный момент у *A* пропускной способности больше. А чтобы обеспечить большую емкость канала для *B*, ее придется снизить для *C* или *D* (или для обоих), и тогда она станет меньше пропускной способности *B*. Данное распределение соответствует максиминному критерию.




**Илл. 6.20.** Распределение пропускной способности по максиминному критерию для четырех потоков

Такое распределение пропускной способности можно вычислить на основе данных обо всей сети. Чтобы представить это наглядно, предположим, что скорость всех потоков медленно возрастает, начиная от 0. Как только один из потоков

достигает узкого места, его скорость перестает расти. Скорость остальных потоков продолжает увеличиваться (при этом доступная пропускная способность делится поровну), пока каждый из них не дойдет до своего узкого места.

Третий вопрос, возникающий при обсуждении равнодоступности, — на каком уровне ее рассматривать? Сеть может быть справедливой на уровне соединений, соединений между парой хостов или всех соединений одного хоста. Этой проблемы мы касались, когда говорили о взвешенном справедливом обслуживании (см. раздел 5.4): выяснилось, что с каждым из этих вариантов связаны определенные трудности. К примеру, если считать равноправными все хосты, активно работающий сервер будет получать не больше ресурсов, чем обычный мобильный телефон; если же таковыми считать все соединения, хостам выгодно открывать дополнительные линии. На этот вопрос нет однозначного ответа, и равнодоступность часто реализуется на уровне соединений, однако она совсем не обязана быть идеальной. В первую очередь важно, чтобы ни одно соединение не осталось без пропускной способности. На самом деле протокол TCP позволяет создавать множественные соединения, что вызывает еще более жесткую конкуренцию. Эта тактика подходит для особенно ресурсоемких приложений. Например, ее использует BitTorrent для однорангового совместного использования файлов.

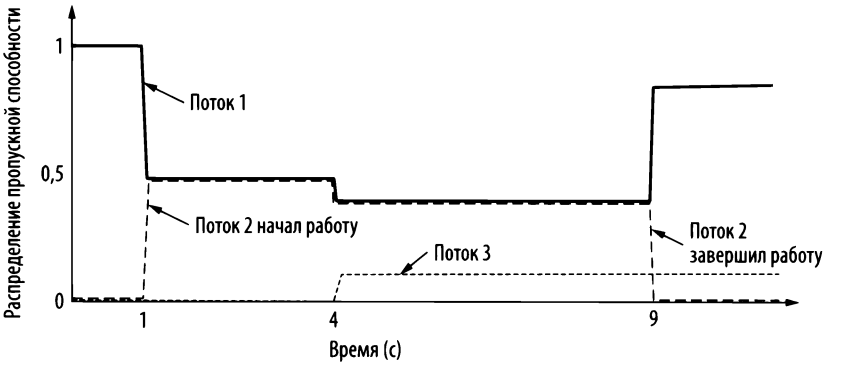
### *Конвергенция*

И наконец, алгоритм контроля перегрузки должен быстро конвергировать, то есть достигать результата — справедливого и эффективного распределения пропускной способности. При обсуждении выбора рабочего режима мы исходили из того, что сетевая среда является статической. Однако на практике соединения то появляются, то исчезают, а пропускная способность, необходимая каждому отдельному соединению, может измениться, например, если пользователь долго просматривает веб-страницу, а затем вдруг начинает загрузку большого видеофайла.

Из-за непостоянства требований к сети идеальный рабочий режим все время меняется. Хороший алгоритм контроля перегрузки должен быстро приводить к идеальному рабочему режиму и следить за его изменением с течением времени. Если алгоритм отрабатывает слишком медленно, он не успевает за сменой рабочего режима. Если он нестабилен, он может не привести к нужной точке или будет долго колебаться вокруг нее.

На илл. 6.21 показан пример распределения пропускной способности, которая меняется со временем и быстро сходится. Изначально поток 1 получает всю емкость канала. Через секунду начинает работать поток 2. Ему тоже нужна пропускная способность. Поэтому распределение быстро меняется таким образом, чтобы оба потока получали по 1/2. Еще через три секунды появляется третий поток. Но он использует только 20 % пропускной способности — меньше, чем то, что ему полагается исходя из принципа равнодоступности (1/3). Потоки 1 и 2 быстро реагируют на изменение ситуации, и каждый из них получает 40 %. На 9-й секунде второй поток отключается, а третий продолжает работать без изменений. Поток 1 быстро занимает 80 % ресурса. В каждый момент времени

суммарная пропускная способность приблизительно равна 100 %, то есть возможности сети используются полностью; при этом все конкурирующие потоки находятся в равных условиях (но не используют больше пропускной способности, чем им нужно).




**Илл. 6.21.** Изменение распределения пропускной способности с течением времени

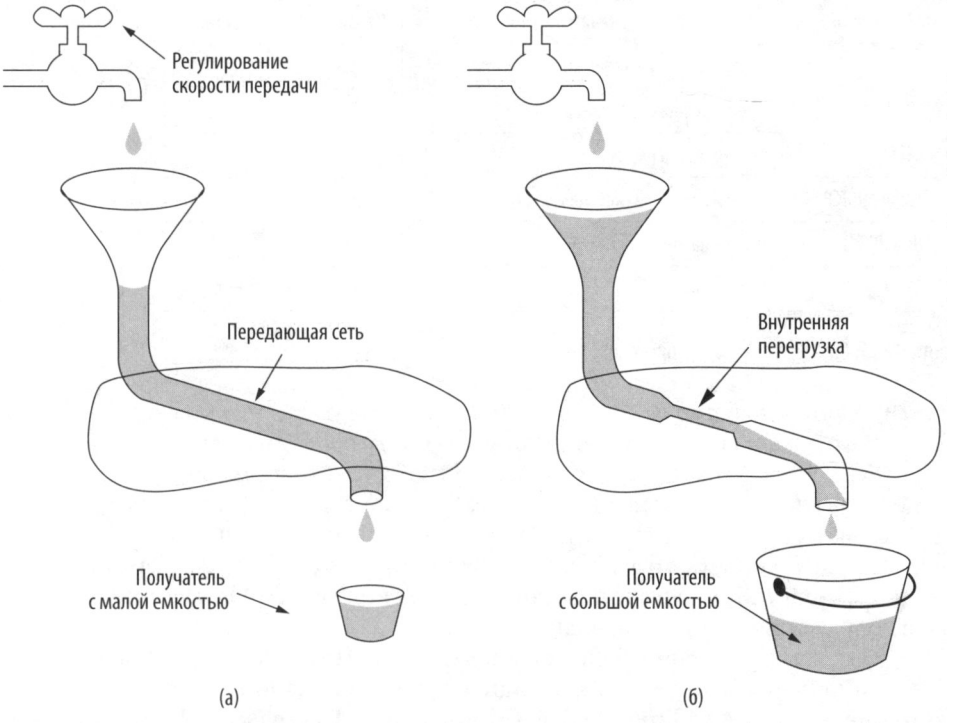
## 6.3.2. Регулирование скорости отправки

Теперь самое время перейти к основному вопросу: как регулировать скорость отправки, чтобы получить необходимую пропускную способность? Скорость отправки может зависеть от двух факторов. Первый фактор — управление потоком данных, которое требуется, если буфер получателя обладает недостаточной емкостью. Второй — перегрузка, которую нужно учитывать при недостаточной пропускной способности сети. На илл. 6.22 эта проблема представлена на примере водопровода.

На илл. 6.22 (а) мы видим толстую трубу, ведущую к получателю с небольшой емкостью. Это тот случай, когда ограничительным фактором является управление потоком. Пока отправитель не передает больше воды, чем может поместиться в ведро, вода не будет проливаться. На илл. 6.22 (б) ограничительным фактором является не емкость ведра, а пропускная способность сети. Если из крана в воронку вода будет литься слишком быстро, то уровень воды в воронке начнет подниматься, и в конце концов часть воды может перелиться через край.

Отправителю эти примеры могут показаться похожими, так как в обоих случаях в результате слишком быстрой передачи данных теряются пакеты. Но эти ситуации происходят по разным причинам и требуют разных решений. Об одном из них — управлении потоком с помощью окна переменного размера — мы уже говорили. Теперь рассмотрим другой подход, связанный с контролем перегрузки. На практике может возникнуть любая из этих проблем, и транспортный протокол должен включать реализацию обоих решений.

То, как транспортный протокол должен регулировать скорость отправки, зависит от формы обратной связи в сети. Различные сетевые уровни могут использовать обратную связь разных типов: явную и неявную, точную и неточную.




**Илл. 6.22.** (а) Быстрая сеть и получатель с малой емкостью. (б) Медленная сеть и получатель с большой емкостью

Пример явной точной обратной связи — ситуация, когда маршрутизаторы сообщают источникам свою допустимую скорость отправки. Так работает, к примеру, **явный протокол перегрузки (eXplicit Congestion Protocol, XCP)** (Катаби и др.; Katabi et al., 2002). Явная и неточная схема — использование **явного уведомления о перегрузке (Explicit Congestion Notification, ECN)** с TCP. В этом случае маршрутизаторы специальным образом маркируют пакеты, страдающие от перегрузки, сообщая отправителю, что ему следует уменьшить скорость передачи данных, но не указывая насколько.

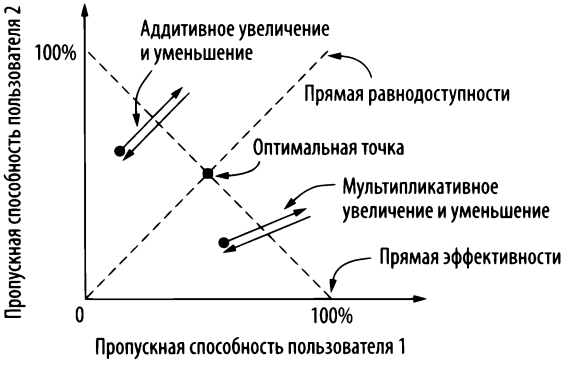
В других схемах явный сигнал отсутствует. FAST TCP измеряет задержку в пути туда и обратно и использует ее в качестве параметра для контроля перегрузки (Вэй и др.; Wei et al., 2006). Наиболее распространенный в современном интернете метод контроля перегрузки использует TCP и маршрутизаторы с отбрасыванием конца очереди или RED-маршрутизаторы. Показателем перегрузки в этом случае является число потерянных пакетов. Существует множество вариантов такого TCP, в частности, CUBIC TCP, используемый в системе Linux (Ха и др; Ha et al., 2008). Возможны комбинации нескольких методов. Например, Windows содержит Compound TCP (составной TCP), где в качестве сигналов обратной связи используется и задержка, и число потерянных пакетов (Тань и др.; Tan et al., 2006). Все эти варианты представлены в таблице на илл. 6.23.

| Протокол | Сигнал | Явный? | Точный? |
|---|---|---|---|
| XCP | Скорость, которую следует использовать | Да | Да |
| TCP с ECN | Предупреждение о перегрузке | Да | Нет |
| FAST TCP | Сквозная задержка | Нет | Да |
| Compound TCP | Потеря пакетов и сквозная задержка | Нет | Да |
| CUBIC TCP | Потеря пакетов | Нет | Нет |
| TCP | Потеря пакетов | Нет | Нет |

**Илл. 6.23.** Сигналы некоторых протоколов контроля перегрузки

Получив явный и точный сигнал, транспортная подсистема может установить скорость отправки пакетов в соответствии с новым рабочим режимом. К примеру, если XCP сообщает отправителям рекомендуемую скорость, они могут просто использовать ее. Но в остальных случаях транспортные подсистемы вынуждены действовать наугад. В отсутствие сигнала о перегрузке отправители должны увеличивать скорость отправки, а при наличии — уменьшать. Рост и снижение скорости происходит согласно **закону управления (control law)**. Он оказывает решающее влияние на производительность.

Рассматривая бинарный сигнал о перегрузке, Чиу и Джейн (Chiu and Jain, 1989) пришли к выводу, что закон **аддитивного увеличения и мультипликативного уменьшения (Additive Increase Multiplicative Decrease, AIMD)** позволяет эффективно вычислять рабочий режим с учетом идеи равнодоступности. Чтобы это показать, они привели простой наглядный пример, в котором два соединения соперничают друг с другом за ресурс общего канала. На илл. 6.24 пропускная способность для пользователя 1 располагается на оси $X$, а для пользователя 2 — на оси $Y$. При справедливом распределении оба пользователя получают одинаковое количество ресурса. Оно обозначено с помощью прямой равнодоступности (пунктирная линия). Когда суммарная пропускная способность, выделяемая всем пользователям, составляет 100 % (то есть равна емкости канала), распределение эффективно. Оно располагается на прямой эффективности. Когда суммарная



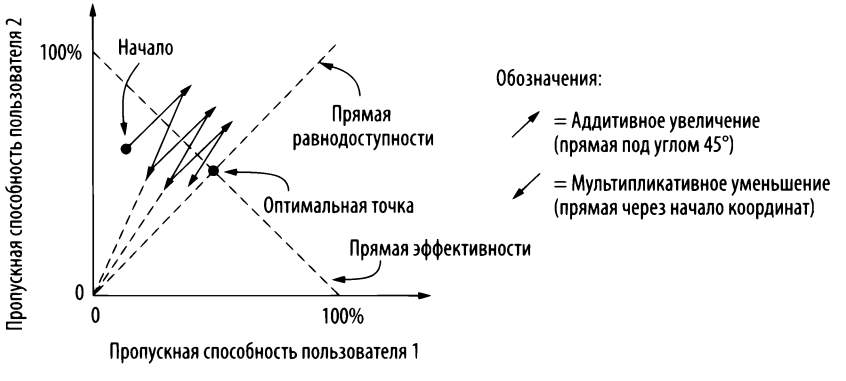

**Илл. 6.24.** Аддитивное и мультипликативное регулирование пропускной способности

пропускная способность пересекает эту прямую, оба пользователя получают сигнал о перегрузке. Точка пересечения этих прямых соответствует оптимальному рабочему режиму, при котором пользователи получают одинаковое количество ресурса и используется вся пропускная способность сети.

Пусть изначально пользователям 1 и 2 выделено некоторое количество пропускной способности. Посмотрим, что произойдет, если они оба начнут аддитивно ее увеличивать. К примеру, они могут повышать скорость отправки пакетов на 1 Мбит/с каждую секунду. В результате рабочий режим пересечет прямую эффективности и оба пользователя получат от сети сигнал о перегрузке. В этот момент им придется снизить свою пропускную способность. Однако аддитивное уменьшение просто приведет к колебаниям значений вдоль аддитивной прямой (см. илл. 6.24). Рабочий режим останется близким к эффективному, однако не обязательно будет справедливым.

Теперь рассмотрим ситуацию, в которой оба пользователя увеличивают свою пропускную способность мультипликативно до тех пор, пока не поступит сигнал о перегрузке. Например, они могут каждую секунду повышать скорость отправки пакетов на 10 %. Если после получения сигнала о перегрузке пользователи начнут мультипликативно уменьшать пропускную способность, рабочий режим будет колебаться вдоль мультипликативной прямой (см. илл. 6.24). Наклон мультипликативной прямой отличается от наклона аддитивной прямой. (Мультипликативная прямая проходит через начало координат, а аддитивная имеет наклон в 45°.) Однако это ничего нам не дает. Ни в том ни в другом случае оптимальная скорость отправки достигнута не будет.

Разберем другой сценарий. Предположим, пользователи аддитивно увеличивают пропускную способность, а после получения сигнала о перегрузке начинают мультипликативно ее уменьшать. Такой метод называется **законом управления (AIMD control law)** (илл. 6.25). Оказывается, что в результате таких преобразований рабочий режим сходится к оптимальной точке, в которой распределение пропускной способности является одновременно справедливым и эффективным, причем это происходит независимо от начальной точки. Поэтому AIMD широко применяется на практике. При обратном варианте –




**Илл. 6.25.** Закон управления AIMD

мультипликативном увеличении и аддитивном уменьшении — рабочий режим будет отклоняться от оптимальной точки.

TCP использует AIMD не только по этой причине, но и по соображениям стабильности (поскольку перейти в состояние перегрузки гораздо проще, чем из него выйти, политика увеличения должна быть мягкой, а уменьшения — агрессивной). Но в результате распределение пропускной способности является не совсем справедливым. Дело в том, что размер окна задается исходя из **времени в пути туда и обратно (round-trip time, RTT)**, индивидуального для каждого соединения. Поэтому соединения с ближними хостами получают бóльшую пропускную способность, чем соединения с удаленными хостами (при прочих равных условиях).

В разделе 6.5 мы подробно поговорим о том, как с помощью AIMD в TCP осуществляется регулировка скорости отправки пакетов и контроль перегрузки. Это гораздо сложнее, чем может показаться. Проблемы возникают из-за того, что значения скорости измеряются через определенный интервал времени, а трафик, как правило, неравномерный. Часто вместо прямого регулирования скорости задается размер раздвижного окна. Такая стратегия используется и в TCP. Если размер окна равен $W$, а время в пути туда и обратно — $RTT$, то эквивалентная скорость равна $W/RTT$. Это удобно, поскольку управление потоком данных также основано на регулировке размера окна. Кроме того, такой метод обладает важным преимуществом: если подтверждения о доставке пакетов перестанут приходить, через время $RTT$ скорость отправки уменьшится.

Наконец, иногда в одной сети используются разные транспортные протоколы. Что произойдет, если они будут одновременно пытаться контролировать перегрузку с помощью разных законов управления? Ответ очевиден: распределение пропускной способности не будет справедливым. TCP — это наиболее распространенная форма контроля перегрузки в интернете. Поэтому новые транспортные протоколы настоятельно рекомендуется разрабатывать так, чтобы при совместной работе с TCP выполнялось условие справедливого распределения ресурсов. Ранние протоколы потокового вещания не соответствовали этому требованию, существенно снижая пропускную способность TCP. В итоге возникла идея **дружественного к TCP** контроля перегрузки, при котором транспортные протоколы TCP и не-TCP могут работать вместе, не мешая друг другу (Флойд и др.; Floyd et al., 2000).

## 6.3.3. Проблемы беспроводного соединения

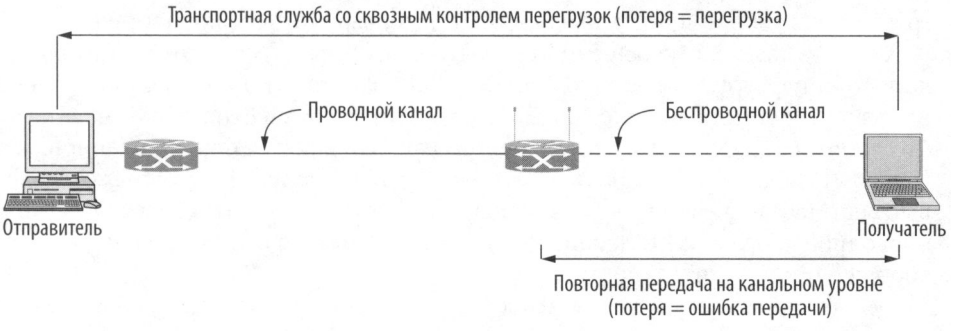
Транспортные протоколы, включающие контроль перегрузки (такие, как TCP), не должны зависеть от используемой сети и устройства канального уровня. В теории это звучит хорошо, но на практике в случае беспроводных сетей возникают определенные проблемы. Главная заключается в том, что во многих протоколах (включая TCP) сигналом перегрузки является потеря пакетов. Но в беспроводных сетях из-за ошибок передачи потеря пакетов происходит постоянно. Просто они не столь надежны, как проводные сети.

Если протокол использует закон управления AIMD, высокая производительность требует минимальной потери пакетов. Исследования Падхая и др.

(Padhye et al., 1998) показали, что пропускная способность растет обратно пропорционально квадратному корню из скорости потери пакетов. Фактически это означает, что скорость потери пакетов для быстрых TCP-соединений очень мала; в среднем этот показатель составляет 1 %, а при значении 10 % соединение перестает работать. Однако для беспроводных сетей, таких как LAN 802.11, вполне обычными являются значения скорости потери фреймов порядка 10 % и выше. Если этого не учитывать, схемы контроля перегрузки, использующие в качестве сигнала потерю пакетов, попросту «задушат» беспроводные соединения, крайне ограничив их скорость.

Для корректной работы алгоритм контроля перегрузки должен учитывать только те потери пакетов, которые произошли из-за недостатка пропускной способности, но не из-за ошибок передачи. Одно из возможных решений — скрыть потерю данных с помощью их повторной передачи по беспроводным каналам. К примеру, в 802.11 для доставки каждого фрейма используется протокол с остановкой и ожиданием подтверждения: передача фрейма повторяется несколько раз, и только после этого информация о потере пакета поступает на более высокий уровень. В большинстве случаев пакеты доставляются получателю, несмотря на ошибки передачи (о которых вышележащие уровни ничего не знают).

На илл. 6.26 показан путь по проводному и беспроводному каналу, для которого используется эта стратегия. Обратим внимание на два момента. Во-первых, источник не знает, что путь содержит беспроводную часть. Он видит только проводной канал, к которому он непосредственно подключен. В интернете пути гетерогенны, однако не существует универсального способа, позволяющего отправителю определить, из каких каналов состоит конкретный путь. Это осложняет контроль перегрузки, поскольку использовать разные протоколы для проводных и беспроводных каналов — очень непростая задача.




**Илл. 6.26.** Контроль перегрузок для пути, содержащего беспроводной канал

Второй важный момент — своего рода загадка. Рисунок иллюстрирует два механизма, основанных на данных о потере пакетов: повторную передачу фрейма на канальном уровне и контроль перегрузки на транспортном уровне. Загадка в том, как эти два механизма могут сосуществовать. Ведь потеря пакетов должна приводить в действие только один из механизмов: это либо ошибка передачи

данных, либо сигнал о перегрузке — но не и то и другое одновременно. Если же начинают работать оба механизма (то есть происходит и повторная передача фрейма, и уменьшение скорости передачи), мы возвращаемся к нашей первоначальной проблеме: схема работает слишком медленно для беспроводных каналов. Попробуйте немного поразмышлять и разгадать эту загадку.

Решение загадки заключается в том, что эти механизмы работают в разных временных масштабах. В беспроводных сетях, таких как 802.11, повторные передачи канального уровня происходят через промежутки времени порядка нескольких микросекунд или миллисекунд. Таймеры транспортных протоколов, следящие за потерей пакетов, срабатывают раз в несколько миллисекунд или секунд. Благодаря разнице в три порядка беспроводные каналы успевают обнаружить потерю фреймов и решить проблему путем их повторной передачи задолго до того, как об этом узнает транспортная подсистема.

Этот подход позволяет транспортным протоколам корректно работать с беспроводными каналами в большинстве случаев. Но существуют сценарии, при которых это решение не подходит. Некоторые беспроводные каналы (например, спутниковые) обладают большой задержкой в пути туда и обратно. В таких ситуациях требуются другие методы, позволяющие скрыть потерю пакетов, например упреждающая коррекция ошибок (Forward Error Correction, FEC), или же транспортный протокол должен использовать для контроля перегрузки сигналы об отсутствии потерь.

Еще одна проблема, связанная с контролем перегрузки в беспроводных каналах, — переменная пропускная способность. Пропускная способность беспроводного канала меняется со временем, причем иногда скачкообразно, при перемещении узлов и изменении соотношения сигнал/шум; в проводных каналах она постоянна. Транспортный протокол вынужден адаптироваться к изменениям пропускной способности беспроводного канала. В противном случае либо произойдет перегрузка сети, либо мощность канала будет использоваться не полностью.

Одно из возможных решений — просто не задумываться об этом. Алгоритмы контроля перегрузки и так должны хорошо работать при добавлении новых пользователей или изменении скорости отправки пакетов. Несмотря на то что пропускная способность проводного канала является фиксированной, для каждого отдельного пользователя меняющееся поведение других пользователей выглядит как колебание доступной полосы. Поэтому для пути, содержащего беспроводной канал 802.11, можно просто использовать протокол TCP и добиться хорошей производительности.

Однако при высокой нестабильности беспроводного канала транспортные протоколы, разработанные для проводных соединений, могут не успевать отслеживать изменения, в результате чего производительность существенно снижается. В таком случае требуется специальный транспортный протокол, созданный для беспроводных каналов. Особый интерес представляет разработка такого протокола для беспроводной ячеистой сети, где пересекается множество беспроводных каналов, маршруты постоянно меняются из-за мобильности и теряется много пакетов. Исследования в этой области продолжаются. Пример транспортного протокола для беспроводных каналов можно найти в работе Ли и др. (Li et al., 2009).

# 6.4. ТРАНСПОРТНЫЕ ПРОТОКОЛЫ ИНТЕРНЕТА: UDP

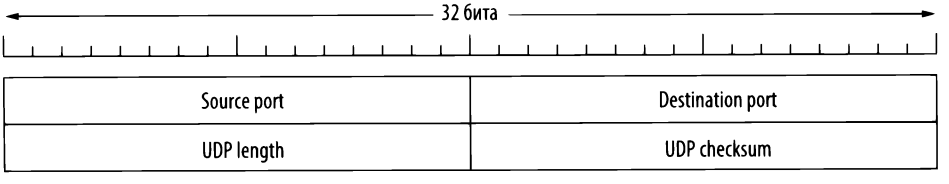
В интернете применяются два основных протокола транспортного уровня, дополняющих друг друга; один из них требует установления соединения, другой — нет. Протокол без установления соединения, UDP, просто передает пакеты между приложениями, позволяя им надстраивать свои собственные протоколы. TCP — протокол, ориентированный на установление соединения. В его задачи входит практически все. Он устанавливает соединения и обеспечивает надежность сети, выполняя повторную передачу данных, а также осуществляет управление потоком данных и контроль перегрузки — и все это в интересах приложений, которые его используют.

В следующих разделах мы изучим UDP и TCP. Мы начнем с UDP, так как он проще, и рассмотрим два варианта применения этого протокола. Поскольку сам UDP обычно работает в операционной системе, а использующие его протоколы — в пользовательской области, эти варианты вполне можно считать приложениями. Однако методы, которые в них используются, полезны для многих приложений и их лучше рассматривать как часть транспортной службы. Поэтому их мы тоже обсудим.

## 6.4.1. Основы UDP

В наборе интернет-протоколов есть **протокол передачи пользовательских дейтаграмм (User Datagram Protocol, UDP)**. Это транспортный протокол без установления соединения, описанный в RFC 768. UDP позволяет приложениям отправлять инкапсулированные IP-дейтаграммы без установления соединений.

С помощью UDP передаются **сегменты**, состоящие из 8-байтного заголовка, за которым следует поле `Payload`. Заголовок показан на илл. 6.27. Два **порта** служат для идентификации сокетов на отправляющем и принимающем устройствах. Когда приходит пакет UDP, содержимое `Payload` передается процессу, связанному с портом назначения. Это связывание происходит при выполнении `BIND` (или похожего примитива). Это показано на илл. 6.6 применительно к TCP (в UDP процесс связывания происходит так же). Представьте, что порты — это почтовые ящики, арендуемые приложениями, чтобы получать пакеты. Мы поговорим о них подробнее при обсуждении TCP, который тоже использует порты. В сущности, весь смысл применения UDP вместо обычного IP заключается как раз в указании портов отправителя и получателя. Без этих двух полей




**Илл. 6.27.** Заголовок UDP

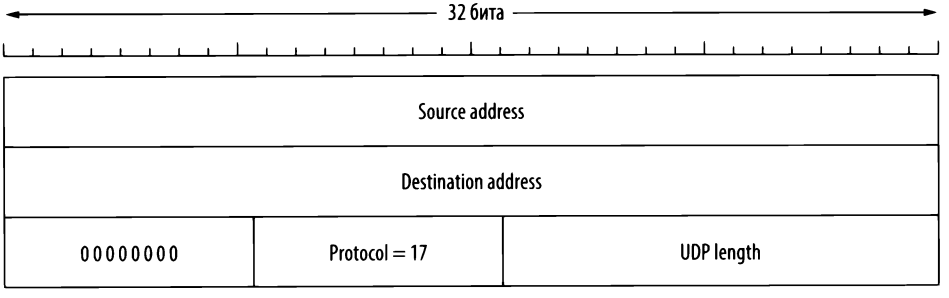
на транспортном уровне невозможно было бы определить, что делать с входящим пакетом. В соответствии с полями портов вложенные сегменты доставляются соответствующим приложениям.

Информация о порте источника требуется прежде всего при создании ответа, пересылаемого отправителю. Копируя значения поля `Source port` (Порт отправителя) входящего сегмента в поле `Destination port` (Порт получателя) исходящего сегмента, процесс, отправляющий ответ, указывает, какому именно процессу на противоположной стороне этот ответ предназначен.

Поле `UDP length` состоит из заголовка и данных. Минимальная длина равна длине заголовка, то есть 8 байт. Максимальная длина составляет 65 515 байт — меньше, чем максимальное число, которое может поместиться в 16 бит, из-за ограничений на размер IP-пакета.

Необязательное поле `Checksum` служит для повышения надежности. Оно содержит контрольную сумму заголовка, данных и псевдозаголовка. При выполнении вычислений в поле `Checksum` устанавливается нулевое значение, а поле данных дополняется нулевым байтом, если его длина представляет собой нечетное число. Алгоритм вычисления контрольной суммы просто складывает все 16-разрядные слова в дополнительном коде, а затем вычисляет дополнение для всей суммы. В результате, когда получатель считает контрольную сумму всего сегмента, включая поле `Checksum`, результат должен быть равен 0. Если она не подсчитывается, ее значение равно 0 (настоящая нулевая контрольная сумма кодируется всеми единицами). Однако отключать функцию подсчета контрольной суммы глупо, за исключением одного случая — когда нужна высокая производительность (например, при передаче оцифрованной речи).

В случае IPv4 псевдозаголовок будет выглядеть так, как показано на илл. 6.28. Он содержит 32-разрядные IP-адреса отправителя и получателя, номер протокола для UDP (17) и счетчик байтов для UDP-сегмента (включая заголовок). В случае IPv6 он выглядит аналогично, хотя и с некоторыми отличиями. Включение псевдозаголовка в контрольную сумму UDP помогает обнаружить неверно доставленные пакеты, хотя это нарушает иерархию протоколов, так как IP-адреса в нем принадлежат IP-уровню, а не UDP-уровню. В TCP для контрольной суммы используется такой же псевдозаголовок.




**Илл. 6.28.** Псевдозаголовок, включаемый в контрольную сумму UDP

Наверное, стоит прямо сказать, чего UDP *не делает*. Итак, UDP не занимается управлением потоком данных, контролем перегрузки, повторной передачей после приема испорченного сегмента. Все это перекладывается на пользовательские процессы. Что же он делает? UDP предоставляет интерфейс для IP путем демультиплексирования нескольких процессов с использованием портов и необязательного сквозного обнаружения ошибок. На этом все.

Для процессов, которые должны управлять потоком, контролировать ошибки и временные интервалы, протокол UDP — это как раз то, что доктор прописал. Особенно это полезно в клиент-серверных ситуациях. Зачастую клиент отправляет короткий запрос серверу и надеется получить короткий ответ. Если запрос или ответ теряется, клиент по прошествии определенного временного интервала может попытаться еще раз. Это позволяет не только упростить код, но и уменьшить требуемое количество сообщений по сравнению с протоколами, которым требуется начальная настройка (например, TCP).

Служба имен доменов (Domain Name System, DNS) — это приложение, которое использует UDP именно так, как описано выше. Мы изучим его в главе 7. В двух словах, если программе нужно найти IP-адрес по имени хоста, например www.cs.berkeley.edu, она может отослать UDP-пакет с этим именем на сервер DNS. Сервер в ответ на запрос отправляет UDP-пакет с IP-адресом хоста. Не требуется никакой предварительной настройки, как и разрыва соединения после завершения задачи. По сети просто передаются два сообщения.

## 6.4.2. Вызов удаленной процедуры

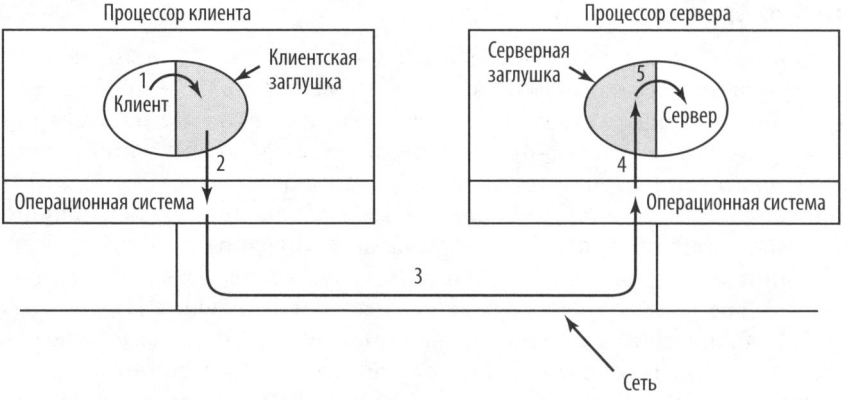
В определенном смысле процессы отправки сообщения на удаленный хост и получения ответа очень похожи на вызов функции в языке программирования. В обоих случаях вы передаете один или несколько параметров и получаете результат. Это соображение навело разработчиков на мысль о том, что можно попробовать организовать запросно-ответное взаимодействие по сети, выполняемое в форме вызовов процедур. Такое решение позволяет упростить и сделать более привычной разработку сетевых приложений. Например, представьте себе процедуру запроса IP-адреса хоста, `get_IP_address(host name)`, работающую посредством отправки UDP-пакетов на сервер DNS, ожидания ответа и отправки повторного запроса в случае наступления тайм-аута (если одна из сторон работает недостаточно быстро). Таким образом, все детали, связанные с особенностями сетевых технологий, скрыты от программиста.

Ключевая работа в этой области написана в 1984 году Бирреллом и Нельсоном (Birrell and Nelson). По сути, было предложено разрешить программам вызывать процедуры, расположенные на удаленных хостах. Когда процесс на устройстве 1 вызывает процедуру на устройстве 2, вызывающий процесс устройства 1 блокируется, и на устройстве 2 выполняется вызванная процедура. Информация от вызывающего процесса может передаваться в виде параметров и приходить обратно в виде результата процедуры. Передача сообщений по сети скрыта от программиста приложения. Такая технология известна под названием

**удаленного вызова процедур (Remote Procedure Call, RPC)**. Она легла в основу многих сетевых приложений. Традиционно вызывающая процедура считается клиентом, а вызываемая — сервером. Мы будем называть их так же.

Идея RPC состоит в том, чтобы сделать вызов удаленной процедуры максимально похожим на локальный вызов. В простейшем случае для вызова удаленной процедуры клиентская программа должна быть связана с маленькой библиотечной процедурой, **клиентской заглушкой (client stub)**, которая представляет серверную процедуру в пространстве клиентских адресов. Аналогично сервер должен быть связан с процедурой под названием **серверная заглушка (server stub)**. Эти процедуры скрывают тот факт, что вызов клиентом серверной процедуры не является локальным.

Реальные шаги, выполняемые при удаленном вызове процедуры, показаны на илл. 6.29. Шаг 1 заключается в вызове клиентом клиентской заглушки. Это локальный вызов процедуры, при котором параметры самым обычным образом помещаются в стек. Шаг 2 состоит в упаковке параметров клиентской заглушки в сообщение и в осуществлении системного вызова для его отправки. Упаковка параметров называется **маршалингом (marshaling)**. На шаге 3 операционная система передает сообщение с клиентского устройства на сервер. Шаг 4 заключается в том, что операционная система передает входящий пакет серверной заглушке. На шаге 5 серверная заглушка вызывает серверную процедуру с распакованными параметрами. При ответе выполняются те же самые шаги, но передача происходит в обратном направлении.




**Илл. 6.29.** Этапы выполнения удаленного вызова процедуры. Серым цветом выделены заглушки

Важнее всего здесь то, что клиентская процедура, написанная пользователем, выполняет обычный (то есть локальный) вызов клиентской заглушки, имеющей то же имя, что и серверная процедура. Поскольку клиентская процедура и клиентская заглушка существуют в одном адресном пространстве, параметры передаются обычным способом. Аналогично серверная процедура вызывается процедурой, находящейся в том же адресном пространстве, с ожидаемыми параметрами. С точки зрения серверной процедуры не происходит ничего

необычного. Таким образом, вместо ввода/вывода с помощью сокетов сетевая коммуникация осуществляется обычным вызовом процедуры.

Несмотря на элегантность концепции RPC, в ней есть подводные камни. Речь идет прежде всего об использовании указателей в качестве параметров. В обычной ситуации передача указателя процедуре не представляет никаких сложностей. Вызываемая процедура может использовать указатель так же, как и вызывающая, поскольку они обе существуют в одном виртуальном адресном пространстве. При удаленном вызове процедуры передача указателей невозможна, потому что адресные пространства клиента и сервера отличаются.

Иногда с помощью некоторых уловок все же удается передать указатели. Допустим, первым параметром является указатель на целое число *k*. Клиентская заглушка может выполнить маршалинг *k* и передать его серверу. Серверная заглушка создаст указатель на полученную переменную *k* и передаст его серверной процедуре. Именно этого она и ждет. Когда серверная процедура возвращает управление серверной заглушке, последняя отправляет *k* обратно клиенту, где обновленное значение этой переменной записывается вместо старого (если оно было изменено сервером). По сути, стандартная последовательность действий, выполняемая при передаче параметра по ссылке, заменилась прямой и обратной передачей копии параметра. Увы, этот трюк не всегда удается применить, в частности, нельзя это сделать, если указатель ссылается на граф или иную сложную структуру данных. Как мы увидим далее, по этой причине на параметры удаленно вызываемых процедур должны накладываться определенные ограничения.

Вторая проблема заключается в том, что в языках со слабой типизацией данных (например, в C) вполне допустимо написание процедуры, которая подсчитывает скалярное произведение двух векторов (массивов), без указания их размеров. Каждая из этих структур в качестве ограничителя имеет какое-то значение, известное только вызывающей и вызываемой процедурам. В этих обстоятельствах клиентская заглушка не способна запаковать параметры: нет никакой возможности определить их размеры.

Третья проблема состоит в том, что не всегда можно распознать типы параметров по спецификации или по самому коду. В качестве примера приведем процедуру `printf`, у которой может быть любое число параметров (не меньше одного), и они могут представлять собой произвольную смесь различных целочисленных (int, short, long), а также символьных и строковых параметров, чисел различной длины с плавающей запятой и др. Задача удаленного вызова процедуры `printf` может оказаться практически невыполнимой из-за такой своеобразной толерантности языка C. Однако не существует правила, говорящего, что удаленный вызов процедур возможен, только если используется любой другой язык кроме C (C++). Это подорвало бы репутацию метода RPC среди программистов.

Еще одна проблема связана с применением глобальных переменных. В обычной ситуации вызывающая и вызываемая процедуры взаимодействуют не только с помощью параметров, но и посредством глобальных переменных (хотя это не считается хорошей практикой). Если вызываемая процедура переместится на

удаленное устройство, код не сработает, так как эти переменные уже не являются общими.

Все эти проблемы не означают, что RPC безнадежен. В действительности он широко используется, просто нужны некоторые ограничения для его корректной работы на практике.

С точки зрения протоколов транспортного уровня UDP является хорошей основой для реализации RPC. В самом простом случае запросы и ответы можно отправлять в одном UDP-пакете, а обработка может быть очень быстрой. Однако для реализации этой идеи потребуются и другие механизмы. На случай потери запроса или ответа клиенту необходим таймер, отсчитывающий время до повторной отправки пакета. Обратите внимание, что ответ служит неявным подтверждением запроса, поэтому отдельное подтверждение не требуется. Иногда параметры или результаты могут превысить максимальный размер UDP-пакета. Тогда потребуется некий протокол, позволяющий «разбирать» большие сообщения перед отправкой и затем корректно собирать их заново. Если возможно пересечение многочисленных запросов и ответов (как при параллельном программировании), соответствие ответа запросу указывается с помощью специальной метки.

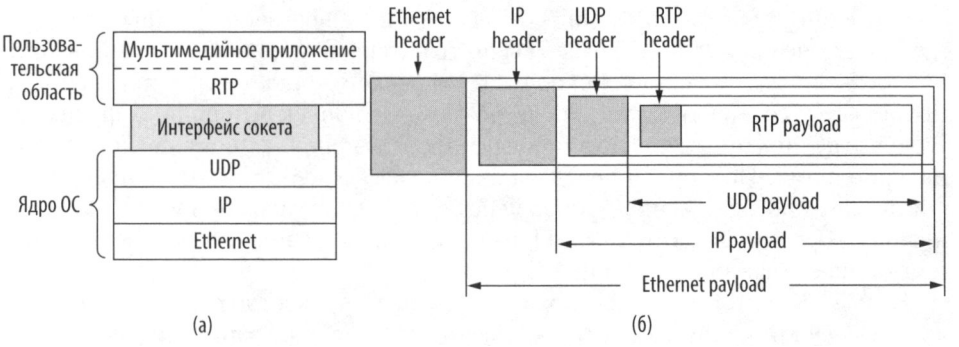
Проблема более высокого уровня связана с тем, что операция может не быть идемпотентной (то есть ее нельзя повторить без риска сбоя). В простом случае мы имеем дело с идемпотентными операциями, такими как DNS-запросы и ответы. Клиент может повторно передавать такие пакеты сколько угодно, ничем не рискуя, до тех пор пока не придет ответ. Пакет может не дойти до сервера, а ответ может потеряться — это неважно. В любом случае, когда ответ придет, он будет тем же (если, конечно, за это время не обновится база данных DNS). Но не все операции идемпотентны: например, если они обладают важными побочными эффектами вроде изменения счетчика. RPC для таких операций требует более сложной семантики: вызванная процедура не должна выполняться несколько раз. В таких случаях может понадобиться установка TCP-соединения и отправка запроса по TCP, а не по UDP.

## 6.4.3. Транспортные протоколы реального времени

Клиент-серверный удаленный вызов процедур — это та область, в которой UDP применяется очень широко. Еще одной такой сферой являются мультимедийные приложения реального времени. С распространением интернет-радио, интернет-телефонии, видеоконференций, музыки и видео по запросу, а также других мультимедийных приложений стало понятно, что все они воссоздают примерно один и тот же транспортный протокол реального времени. Вскоре родилась идея о том, что было бы удобно иметь для них один общий транспортный протокол.

Так возник **транспортный протокол реального времени (Real-Time Transport Protocol, RTP)**. Он описан в RFC 3550 и сегодня широко используется в мультимедиа. Мы опишем два компонента транспортировки данных в реальном времени. Первый из них — протокол RTP, необходимый для пакетной передачи

аудио и видео. Второй — обработка данных, необходимая для воспроизведения (в основном со стороны получателя). Эти функции входят в стек протоколов, представленный на илл. 6.30.




**Илл. 6.30.** (а) Положение RTP в стеке протоколов. (б) Вложение пакетов

RTP обычно работает поверх UDP (в операционной системе). Происходит это так. Мультимедийное приложение может состоять из нескольких аудио-, видео-, текстовых и некоторых других потоков. Они прописываются в библиотеке RTP, которая (как и само приложение) находится в пользовательской области. Библиотека мультиплексирует потоки и вставляет их в пакеты RTP, которые, в свою очередь, отправляются в сокет. На другом конце сокета (в операционной системе) генерируются UDP-пакеты, в которые помещаются RTP-пакеты. Они передаются протоколу IP для отправки по каналу (например, Ethernet). На хосте-получателе происходит обратный процесс. В конечном итоге мультимедийное приложение получает данные из RTP-библиотеки. Оно отвечает за воспроизведение. Стек протоколов для этого сценария показан на илл. 6.30 (а), система вложенных пакетов — на илл. 6.30 (б).

В итоге определить, к какому уровню относится RTP, непросто. Так как он работает в пользовательской области и связан с прикладной программой, он, несомненно, выглядит как прикладной протокол. С другой стороны, он является общим, независимым от приложения протоколом, который просто предоставляет транспортные услуги. С этой точки зрения он может показаться транспортным протоколом. Наверное, лучше всего дать ему следующее определение: RTP — это транспортный протокол, который случайно оказался на прикладном уровне, и именно поэтому мы говорим о нем в этой главе.

### *RTP — транспортный протокол реального масштаба времени*

Базовая функция RTP — мультиплексирование нескольких потоков реального времени в единый поток пакетов UDP. Поток UDP можно направлять либо по одному, либо сразу по нескольким адресам. Поскольку RTP использует обычный UDP, маршрутизаторы не обрабатывают пакет каким-то особым образом, если только не включены функции QoS, свойственные для IP. В частности, нет

никаких гарантий доставки, пакеты могут теряться, задерживаться, повреждаться и т. д.

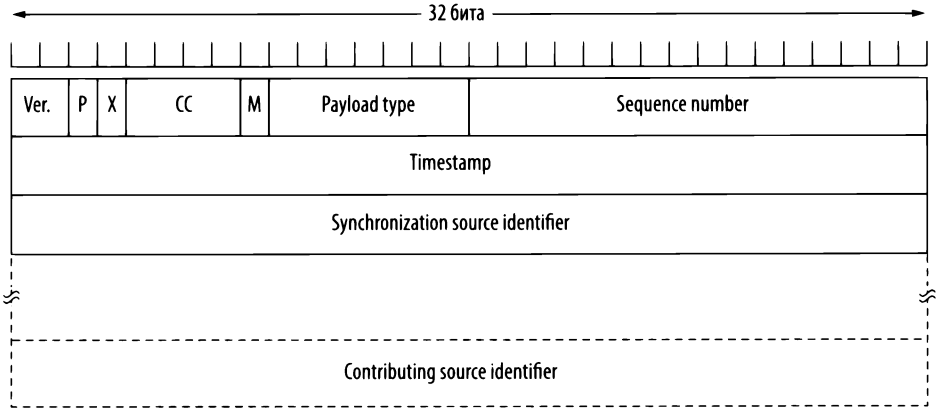
Формат RTP обладает рядом особенностей, которые помогают получателям обрабатывать мультимедийные данные. Каждый пакет потока RTP имеет номер, на единицу превышающий номер своего предшественника. Такой способ нумерации позволяет получателю выявить потерю пакетов. При исчезновении пакета выбор оптимального для получателя решения остается за приложением. Если пакеты содержат видеоданные, возможен пропуск потерянных фреймов. В случае аудиоданных можно аппроксимировать пропущенное значение путем интерполяции. Повторная передача в данном случае не подходит, поскольку она займет много времени и заново переданный пакет окажется уже никому не нужным. Поэтому в протоколе RTP не предусмотрено никаких подтверждений и механизмов запроса повторной передачи.

Каждое поле RTP `Payload` может содержать множество сэмплов, которые кодируются так, как этого хочет приложение. Межсетевое взаимодействие в RTP обеспечивается за счет определения нескольких профилей (например, отдельных аудиопотоков), каждому из которых может соответствовать несколько форматов кодирования. Например, аудиопоток может кодироваться при помощи PCM (8-битными символами с полосой 8 кГц), дельта-кодирования, кодирования с предсказанием, GSM-кодирования, MP3-кодирования и т. д. В RTP имеется специальное поле заголовка, в котором источник может указать метод кодирования, но затем источник никак не влияет на этот процесс.

Еще одна функция, необходимая приложениям реального времени, — это добавление временных меток. Идея в том, чтобы позволить источнику связать метку времени с первым элементом каждого пакета. Метки ставятся относительно момента начала передачи потока, поэтому важны только интервалы между ними. Абсолютные значения никакой роли не играют. Вскоре мы узнаем, что этот механизм позволяет получателю буферизировать небольшой объем данных и проигрывать каждый сэмпл, выждав правильное количество миллисекунд после начала потока, независимо от того, когда пришел пакет с данным сэмплом.

Это не только снижает эффект колебания сетевой задержки, но и дает возможность синхронизации нескольких потоков. Например, в цифровом телевидении может быть видеопоток и два аудиопотока. Второй аудиопоток обычно нужен для стереозвучания либо для звуковой дорожки фильма, дублированной на иностранном языке. У каждого потока свой физический источник, однако с помощью временных меток, генерируемых единым таймером, эти потоки можно синхронизировать при воспроизведении, даже если при передаче и/или получении произошли ошибки.

Заголовок RTP показан на илл. 6.31. Он состоит из трех 32-разрядных слов и некоторых возможных расширений. Первое слово содержит поле `Version`, которое уже имеет значение 2. Будем надеяться, что текущая версия близка к окончательной, поскольку здесь осталась только одна кодовая точка (впрочем, 3 может обозначать, что настоящий номер версии содержится в слове расширения).




**Илл. 6.31.** Заголовок RTP

Бит `P` указывает на то, что размер пакета кратен 4 байтам за счет байтов заполнения. В последнем байте заполнения содержится общее число байтов заполнения. Бит `X` сообщает, что присутствует заголовок расширения. Формат и назначение такого заголовка не определяются. Обязательным для него является только то, что первое слово расширения должно содержать общую длину расширения. Это запасная возможность для разнообразных непредсказуемых требований в будущем.

Поле `CC` сообщает, сколько сотрудничающих источников формируют поток. Их число может колебаться от 0 до 15 (см. ниже). Бит `M` — это маркер, связанный с конкретным приложением. Он может использоваться для обозначения начала видеофрейма, начала слова в аудиоканале или еще для чего-то важного и понятного для приложения. Поле `Payload type` (Тип данных) содержит информацию об использующемся алгоритме кодирования (например, несжатое 8-битное аудио, MP3 и т. д.). Поскольку такое поле есть в каждом пакете, метод кодирования может изменяться прямо во время передачи потока. `Sequence number` (Порядковый номер) — это счетчик, который увеличивается на единицу в каждом пакете RTP. Он используется для выявления потерянных пакетов.

`Timestamp` (Временная метка) генерируется источником потока и служит для записи момента создания первого слова пакета. Метка времени помогает снизить эффект временных колебаний, или **джиттер (jitter)**, на стороне получателя. Это происходит за счет того, что момент воспроизведения не зависит от времени прибытия пакета. `Synchronization source identifier` (Идентификатор источника синхронизации) позволяет определить, какому потоку принадлежит пакет. Это и есть используемый метод мультиплексирования и демультиплексирования нескольких потоков данных, следующих в виде единого потока UDP-пакетов. Наконец, `Contributing source identifiers` (Идентификаторы сотрудничающих источников), если таковые имеются, используются, когда конечный поток формируется несколькими источниками. В этом случае микширующее устройство является источником синхронизации, а в полях идентификаторов источников перечисляются смешанные потоки.

### RTCP — управляющий транспортный протокол реального времени

У протокола RTP есть родственный протокол под названием **управляющий транспортный протокол реального времени (Real-Time Transport Control Protocol, RTCP)**. Он описан в RFC 3550 вместе с RTP. В его задачи входит поддержка обратной связи, синхронизация, обеспечение пользовательского интерфейса, однако передачей медиаданных он не занимается.

Первая функция RTCP может использоваться для обратной связи по задержкам, колебаниям времени задержки (или джиттеру), пропускной способности, перегрузке и другим свойствам сети, сведения о которых сообщаются источникам. Полученная информация может приниматься во внимание кодировщиком для увеличения скорости передачи данных (что улучшит качество), когда состояние сети позволяет это сделать, или для уменьшения скорости при возникновении в сети каких-либо проблем. Благодаря постоянной обратной связи алгоритмы кодирования динамически настраиваются, чтобы обеспечивать наилучшее качество при текущих обстоятельствах. Например, пропускная способность при передаче потока может как увеличиваться, так и уменьшаться, и в соответствии с этим могут изменяться методы кодирования (MP3 может заменяться 8-битным PCM или дельта-кодированием). Поле `Payload type` сообщает получателю, какой алгоритм кодирования применяется для данного пакета, что позволяет менять их по мере необходимости.

Проблема обратной связи заключается в том, что сообщения RTCP рассылаются всем участникам. Если группа большая, то пропускная способность, которая требуется протоколу, резко возрастет. Чтобы этого не произошло, источники RTCP снижают скорость отправки своих сообщений так, чтобы все вместе они потребляли, скажем, 5 % пропускной способности, используемой для передачи медиаданных. А для этого каждый участник должен знать эту пропускную способность (эту информацию он может получить от отправителя) и общее число участников (которое он вычисляет на основании сообщений RTCP).

RTCP также обеспечивает межпотоковую синхронизацию. Проблема в том, что потоки могут использовать различные таймеры с разной степенью разрешения и разными скоростями дрейфа. RTCP помогает решить эти проблемы и синхронизировать потоки с разными параметрами.

Наконец, RTCP позволяет называть источники (например, с помощью обычного ASCII-текста). Эта информация может отображаться на экране получателя, позволяя определить источник текущего потока.

Более подробно о протоколе RTP можно прочитать в работе Перкинса (Perkins, 2002).

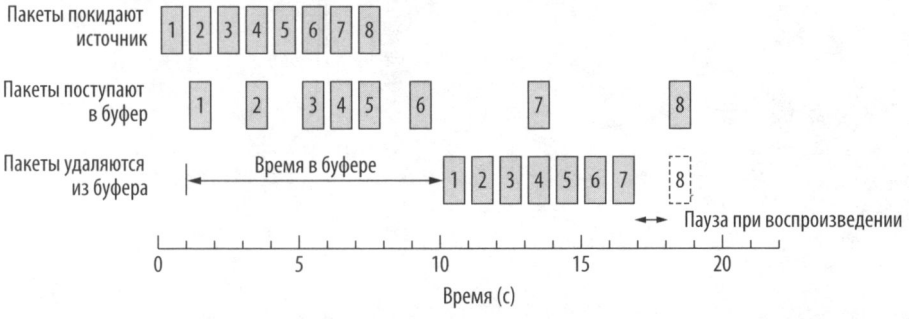
### Буферизация и контроль джиттера при воспроизведении

Когда информация приходит к получателю, необходимо начать ее воспроизведение в правильный момент времени. Обычно этот момент не совпадает со временем прибытия RTP-пакета, поскольку время прохождения через сеть для разных пакетов немного различается. Даже если отправитель будет передавать

их в сеть через одинаковые промежутки времени, они все равно будут приходить с разным интервалом.

Если медиаданные будут проигрываться сразу же по прибытии, даже небольшой джиттер может вызвать серьезные помехи: изображение может быть прерывистым, а звук — неразборчивым.

Решение заключается в **буферизации** пакетов на стороне получателя перед их воспроизведением для снижения джиттера. На илл. 6.32 мы видим поток пакетов, прибывающих к адресату с достаточно большим джиттером. Пакет 1 передан с сервера в момент времени $t = 0$ с и доставлен в $t = 1$ с. Пакет 2 задерживается и прибывает через 2 с. По прибытии пакеты помещаются в буфер устройства клиента.

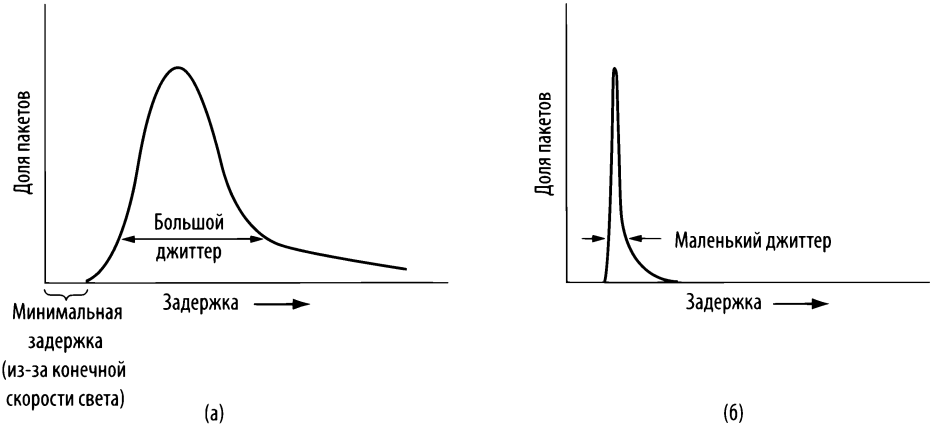



**Илл. 6.32.** Выравнивание выходного потока с помощью буферизации пакетов

В момент времени $t = 10$ с начинается воспроизведение. В буфере уже находятся пакеты с 1-го по 6-й, поэтому их можно доставать оттуда через равные интервалы и тем самым обеспечить равномерное воспроизведение. Как правило, равные промежутки времени использовать не обязательно, так как метки времени RTP содержат информацию о том, когда следует начать воспроизведение.

К сожалению, пакет 8 задерживается настолько, что не успевает прибыть к тому моменту, когда он должен быть воспроизведен. В таком случае возможны два варианта. Проигрыватель может пропустить 8-й пакет и перейти к следующим пакетам. Или же он может остановить воспроизведение, пока не придет 8-й пакет, и тем самым создать неприятную паузу в музыке или фильме. Если приложение работает в режиме реального времени (например, при звонке через интернет), то пакет, скорее всего, будет пропущен. Такие приложения не очень хорошо функционируют в режиме ожидания. В приложениях, работающих с потоковым мультимедиа, проигрыватель может на время остановить воспроизведение. Чтобы улучшить ситуацию, можно начать воспроизведение еще позже и использовать буфер большего размера. Проигрыватели потокового аудио и видео часто используют буферы размером около 10 с, чтобы все пакеты (кроме утерянных) успели прийти вовремя. Приложения, работающие в реальном времени (например, видеоконференции) и требующие быстрой ответной реакции, используют маленькие буферы.

Для равномерного воспроизведения очень важна **задержка воспроизведения (playback point)**. Это время, в течение которого получатель ожидает медиаданные, прежде чем начать воспроизведение. Этот параметр зависит от джиттера. Различие между соединением с маленьким и большим джиттером показано на илл. 6.33. Средняя задержка может не слишком отличаться, но при большом джиттере может потребоваться задержка воспроизведения, захватывающая 99 % пакетов (намного больше, чем при маленьком джиттере).




**Илл. 6.33.** Джиттер: (а) большой; (б) маленький

Чтобы правильно подобрать задержку воспроизведения, приложение может измерить джиттер, вычислив разницу между значениями меток времени RTP и временем прибытия. Эта разница (плюс некоторая константа) и составляет задержку каждого отдельного пакета. Однако некоторые факторы, такие как конкурирующий трафик и изменение маршрутов, могут стать причиной изменения времени задержки. Приложения должны это учитывать и при необходимости менять задержку воспроизведения. Но при неправильной реализации такое изменение может вызвать сильные помехи. Одно из возможных решений для аудио состоит в том, чтобы корректировать задержку воспроизведения между **сегментами речи (talkspurts)**, то есть во время пауз. Разница между короткой паузой и чуть более длинной вряд ли будет заметна. Чтобы это было возможным, RTP позволяет приложениям отмечать начало нового сегмента речи с помощью маркера *M*.

Ситуация, в которой медиаданные не проигрываются слишком долго, представляет серьезную проблему для приложений в реальном времени. Невозможно уменьшить задержку распространения, если уже и так используется кратчайший путь. Задержку воспроизведения можно снизить за счет увеличения доли пакетов, опаздывающих к началу проигрывания. Если это недопустимо, остается последний вариант: уменьшить джиттер, запросив более высокий уровень QoS, например дифференцированное обслуживание с приоритетной доставкой. Иными словами, для этого требуется сеть с лучшими параметрами.

## 6.5. ТРАНСПОРТНЫЕ ПРОТОКОЛЫ ИНТЕРНЕТА: TCP

UDP — это простой протокол, предназначенный для таких важных областей применения, как клиент-серверные взаимодействия и мультимедиа. Однако большинству интернет-приложений требуется надежная, последовательная передача, которую UDP обеспечить не может. Для них следует использовать другой протокол — TCP, «рабочую лошадку» интернета. Далее мы подробно его рассмотрим.

### 6.5.1. Основы TCP

**Протокол управления передачей (Transmission Control Protocol, TCP)** был разработан специально для обеспечения надежного сквозного байтового потока по ненадежной интерсети. Интерсеть отличается от отдельной сети тем, что ее участки могут сильно различаться по топологии, пропускной способности, значениям времени задержки, размерам пакетов и другим параметрам. При разработке TCP основное внимание уделялось способности протокола адаптироваться к свойствам интерсети и отказоустойчивости при возникновении всевозможных проблем.

TCP был описан в RFC 793 в сентябре 1981 года. Со временем он был во многом усовершенствован, были исправлены различные ошибки и неточности. На сегодняшний день существует множество других RFC, являющихся дополнениями к RFC 793 (что позволяет судить о распространенности этого протокола). Уточнения и исправления описаны в RFC 1122, расширения для высокой производительности — в RFC 1323, выборочные подтверждения — в RFC 2018, контроль перегрузки — в RFC 2581, использование полей заголовка для QoS — в RFC 2873, усовершенствованные таймеры повторной передачи — в RFC 2988, явные уведомления о перегрузке — в RFC 3168. Поскольку это далеко не полный список, для удобной работы со всеми этими RFC был создан специальный указатель (конечно же, в виде еще одного RFC-документа) — RFC 4614.

Каждый компьютер, поддерживающий TCP, имеет транспортную подсистему TCP, которая является либо библиотечной процедурой, либо пользовательским процессом, либо (чаще всего) частью ядра системы. В любом случае транспортная подсистема управляет TCP-потоками и интерфейсом с IP-уровнем. Она принимает потоки пользовательских данных от локальных процессов, делит их на части, не превышающие 64 Кбайт (на практике это число обычно равно 1460 байтам данных, что позволяет поместить их в один фрейм Ethernet с заголовками IP и TCP), и отправляет их в виде отдельных IP-дейтаграмм. Когда IP-дейтаграммы с TCP-данными приходят на компьютер, они передаются TCP-подсистеме, которая восстанавливает исходный байтовый поток. Для простоты мы иногда будем употреблять «TCP» для обозначения транспортной подсистемы TCP (части программного обеспечения) или протокола TCP (набора правил). Из контекста будет понятно, что имеется в виду. Например, в выражении «Пользователь передает данные TCP» подразумевается, естественно, транспортная подсистема TCP.

Уровень IP не гарантирует правильной доставки дейтаграмм и не накладывает ограничений на скорость их отправки. Именно TCP приходится выбирать

правильную скорость отправки (согласно целям эффективного использования пропускной способности и предотвращения перегрузок), следить за истекшими интервалами ожидания и в случае необходимости заниматься повторной передачей дейтаграмм, не достигших адресата. Иногда дейтаграммы доставляются в неправильном порядке. Восстанавливать из них сообщения также обязан TCP. Таким образом, протокол TCP призван обеспечить хорошую производительность и надежность, о которой мечтают многие приложения и которая не предоставляется протоколом IP.

## 6.5.2. Модель службы TCP

В основе службы TCP лежат **сокеты (sockets)**, создаваемые как отправителем, так и получателем. Они обсуждались в разделе 6.1.3. У каждого сокета есть номер (адрес), состоящий из IP-адреса хоста и 16-битного номера, локального по отношению к хосту и называемого **портом**. Порт в TCP – это TSAP-адрес. Для обращения к службе TCP между сокетами двух компьютеров должно быть явно установлено соединение. Вызовы сокетов перечислены на илл. 6.5.

Один сокет может использоваться одновременно для нескольких соединений. Другими словами, два и более соединения могут оканчиваться одним сокетом. Соединения различаются по идентификаторам сокетов на обоих концах: (*socket1*, *socket2*). Номера виртуальных каналов или другие идентификаторы не используются.

Номера портов со значениями ниже 1024 зарезервированы стандартными службами и доступны только привилегированным пользователям (например, root в UNIX-системах). Они называются **известными портами (well-known ports)**. К примеру, любой процесс, желающий удаленно загрузить почту с хоста, может связаться с портом 143 хоста-адресата и обратиться, таким образом, к его IMAP-демону. Список известных портов приведен на сайте www.iana.org. На данный момент их насчитывается более 700. Некоторые из них перечислены на илл. 6.34.

| Порт | Протокол | Использование |
|---|---|---|
| 20, 21 | FTP | Передача файлов |
| 22 | SSH | Дистанционный вход в систему, замена Telnet |
| 25 | SMTP | Электронная почта |
| 80 | HTTP | Всемирная паутина (World Wide Web) |
| 110 | POP-3 | Удаленный доступ к электронной почте |
| 143 | IMAP | Удаленный доступ к электронной почте |
| 443 | HTTPS | Защита от угроз (HTPP через SSL/TLS) |
| 543 | RTSP | Контроль воспроизведения мультимедиа |
| 631 | IPP | Коллективное использование принтера |

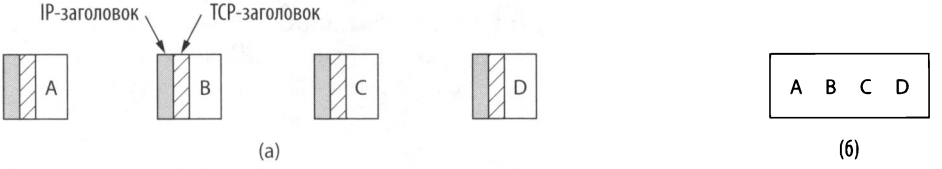
**Илл. 6.34.** Некоторые зарезервированные порты

Порты с номерами от 1024 до 49151 можно зарегистрировать через IANA для непривилегированных пользователей, однако приложения могут выбирать свои собственные порты (что они обычно и делают). К примеру, приложение BitTorrent для одноранговой совместной доступа к файлам использует (неофициально) порты 6881–6887, но другие порты также возможны.

Конечно, можно было бы еще во время загрузки связать FTP-демон с портом 21, SSH-демон с портом 22 и т. д. Но тогда бы память была забита демонами, которые чаще всего простаивают. Вместо этого обычно используется один демон, называемый в UNIX **inetd (Internet daemon)**. Он связывается с несколькими портами и ожидает первое входящее соединение. Когда оно возникает, *inetd* создает новый процесс и вызывает подходящий демон для обработки запроса. Таким образом, постоянно активен только *inetd*, остальные вызываются, только когда для них есть работа. *Inetd* узнает, какие порты нужно использовать, из конфигурационного файла. Это означает, что системный администратор может настроить систему так, чтобы с самыми загруженными портами (например, 80) были связаны постоянные демоны, а с остальными — *inetd*.

Все TCP-соединения являются полнодуплексными и двухточечными. «Полнодуплексное» означает, что трафик может следовать одновременно в обе стороны, а «двухточечное» — что у него есть две конечные точки. Широковещание и многоадресная рассылка протоколом TCP не поддерживаются.

TCP-соединение представляет собой байтовый поток, а не поток сообщений. Границы между сообщениями не сохраняются. Например, если отправляющий процесс записывает в TCP-поток четыре 512-байтные порции данных, эти данные могут быть доставлены получающему процессу в виде четырех 512-байтных порций, двух 1024-байтных порций, одной 2048-байтной порции (илл. 6.35) или как-то еще. Способа, с помощью которого получатель мог бы определить, как записывались данные, не существует.




**Илл. 6.35.** (а) Четыре 512-байтных сегмента, отправленные как отдельные IP-дейтаграммы. (б) 2048 байт данных, доставленные приложению с помощью одного вызова процедуры READ

Файлы в системе UNIX также обладают этим свойством. Программа, читающая файл, не может определить, как был записан этот файл: поблочно, побайтно или целиком. Как и файлы UNIX, TCP-программы не имеют представления о назначении байтов и не интересуются этим. Байт для них — просто байт.

Получив данные от приложения, протокол TCP может отправить их сразу или поместить в буфер (чтобы собрать больше данных и отправить их за один раз) по своему усмотрению. Но иногда приложению необходимо, чтобы

данные были переданы немедленно. Допустим, пользователь интерактивной игры хочет отправить поток обновлений. Важно, чтобы они передавались сразу же, а не сохранялись в буфере до появления других обновлений. Для ускорения передачи данных в TCP существует флаг PUSH (толкнуть), который включается в пакеты. Изначально предполагалось, что с его помощью приложения будут сообщать TCP, что не нужно задерживать передачу пакета. Однако приложения не могут сами устанавливать PUSH при отправке данных. Вместо этого в различных операционных системах используются специальные параметры, позволяющие ускорить передачу данных (например, TCP_NONDELAY в Windows и Linux).

Для тех, кто интересуется историей интернета, мы расскажем о любопытной функции службы TCP. Эта функция все еще входит в состав протокола, но используется редко. Речь пойдет о **срочных данных (urgent data)**. Предположим, что у приложения есть данные с высоким приоритетом (значит, они должны обрабатываться сразу): например, интерактивный пользователь нажимает Ctrl-C, чтобы прервать начавшийся удаленный процесс. Тогда передающее приложение помещает в выходной поток управляющую информацию и отсылает ее TCP-службе вместе с флагом URGENT (срочно). Этот флаг заставляет TCP-подсистему прекратить накопление данных и без промедления передать в сеть все, что у нее есть для данного соединения.

Когда срочные данные приходят по назначению, получающее приложение прерывается (то есть, в терминологии UNIX, «получает сигнал»), затем оно считывает данные из входного потока и ищет среди них срочные. Конец срочных данных маркируется, но их начало приложение должно отыскать самостоятельно.

Эта схема является грубым сигнальным механизмом, который оставляет все прочие задачи приложению. Хотя теоретически использование срочных данных выглядит целесообразным, на заре своего появления эта схема была неудачно реализована и поэтому быстро вышла из употребления. Сейчас использовать ее не рекомендуется из-за различий в имплементации, поэтому приложения вынуждены прибегать к собственным системам сигналов. Возможно, в последующих транспортных протоколах эта идея будет воплощена лучше.

## 6.5.3. Протокол TCP

В данном разделе мы обсудим TCP в общих чертах, а в следующем — подробно изучим его заголовок.

Ключевым свойством TCP, определяющим всю структуру протокола, является то, что в TCP-соединении у каждого байта есть свой 32-разрядный порядковый номер. В прежние годы, когда типичная скорость выделенных линий между маршрутизаторами составляла 56 Кбит/с, хосту, постоянно работающему на полной скорости, потребовалось бы больше недели на то, чтобы перебрать все порядковые номера. При современных скоростях они могут закончиться пугающе быстро. Отдельные 32-разрядные порядковые номера используются для указания позиции раздвижного окна в одном направлении и для подтверждений в обратном. Все это мы обсудим далее.

Отправляющая и принимающая TCP-подсистемы обмениваются данными в виде сегментов. **Сегмент TCP** состоит из фиксированного 20-байтного заголовка (плюс необязательная часть), за которым могут следовать байты данных. Размер сегментов определяется программным обеспечением TCP. Оно может объединять в один сегмент данные, полученные в результате нескольких операций записи, или, наоборот, распределять результат одной записи между несколькими сегментами. Их размер ограничен двумя пределами. Во-первых, каждый сегмент, включая TCP-заголовок, должен помещаться в 65 515-байтное поле пользовательских данных IP-пакета. Во-вторых, в каждом канале есть **максимальный размер передаваемого блока (Maximum Transfer Unit, MTU)**. На сторонах отправителя и получателя каждый сегмент должен помещаться в MTU, чтобы он мог передаваться и приниматься в отдельном пакете, не разделенном на фрагменты. На практике MTU обычно составляет 1500 байт (что соответствует размеру поля пользовательских данных Ethernet), и таким образом определяется верхний предел размера сегмента.

Тем не менее фрагментация IP-пакета, содержащего TCP-сегменты, возможна, если на его пути у одного из каналов слишком низкий MTU. Но в таком случае снижается производительность, а также возникают другие проблемы (Кент и Могул; Kent and Mogul, 1987). Вместо этого современные реализации TCP выполняют **обнаружение MTU маршрута (path MTU discovery)**. При этом используется метод, описанный в RFC 1191 (мы говорили о нем в разделе 5.5.6). Этот метод вычисляет минимальное значение MTU по всем каналам пути, используя сообщения об ошибках ICMP. На основе этого значения TCP выбирает размер сегмента, позволяющий избежать фрагментации.

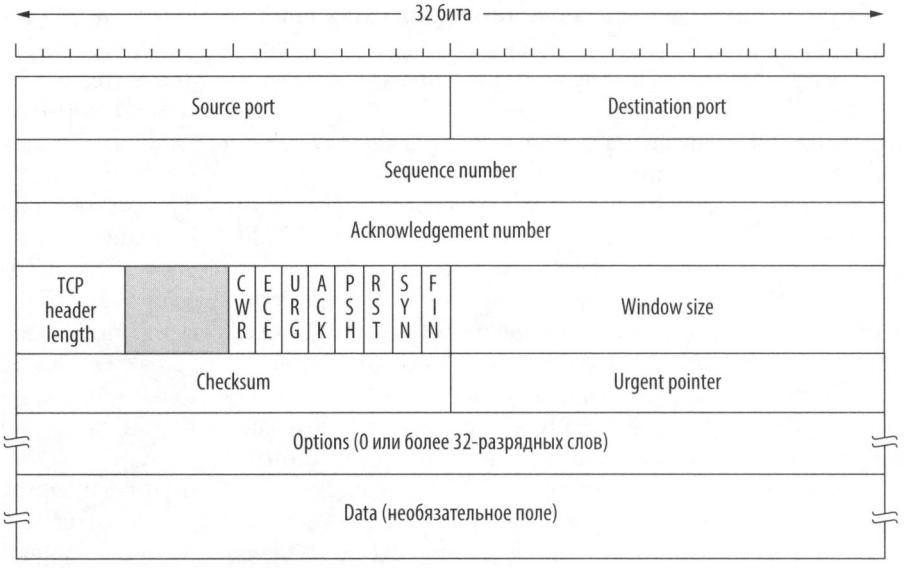
Основной протокол, используемый TCP-подсистемами, — это протокол раздвижного окна с динамическим размером окна. При передаче сегмента отправитель включает таймер. Когда сегмент приходит по назначению, принимающая TCP-подсистема высылает обратно сегмент (с данными, если они есть, или без) с номером подтверждения (он равен порядковому номеру следующего ожидаемого сегмента) и новым размером окна. Если время ожидания подтверждения истекает, отправитель передает сегмент еще раз.

Этот протокол кажется простым, но в нем есть несколько деталей, которые следует рассмотреть подробнее. Сегменты могут приходить в неверном порядке. Например, возможна ситуация, в которой байты с 3072-го по 4095-й уже прибыли, но подтверждение для них не может быть выслано, так как байты с 2048-го по 3071-й еще не получены. К тому же сегменты могут задержаться в сети настолько, что у отправителя истечет время ожидания, и он передаст их снова. Переданный повторно сегмент может включать в себя уже другие диапазоны фрагментов, тогда потребуется очень аккуратное администрирование для определения номеров байтов, которые уже были приняты корректно. Но поскольку каждый байт в потоке имеет свое уникальное смещение, эта задача выполнима.

Протокол TCP должен уметь эффективно решать такие проблемы. На оптимизацию производительности TCP-потоков было потрачено много усилий. В следующем разделе мы обсудим несколько алгоритмов, используемых в различных реализациях TCP.

## 6.5.4. Заголовок TCP-сегмента

На илл. 6.36 показана структура заголовка TCP-сегмента. Каждый сегмент начинается с 20-байтного заголовка фиксированного формата. За ним могут следовать дополнительные параметры. Далее может располагаться до 65 535 – 20 – 20 = = 65 495 байт данных (первые 20 байт это IP-заголовок, а вторые — TCP-заголовок). Сегмент может и не содержать данных. Такие сегменты часто применяются для передачи подтверждений и управляющих сообщений.




**Илл. 6.36.** Заголовок TCP

Рассмотрим поля TCP-заголовка одно за другим. Поля `Source port` и `Destination port` указывают локальные конечные точки соединения. TCP-порт вместе с IP-адресом хоста образуют уникальный 48-битный идентификатор конечной точки. Пара конечных точек получателя и отправителя идентифицируют соединение. Такой идентификатор соединения называется **кортежем из пяти компонентов (5 tuple)**, так как он включает пять информационных составляющих: протокол (TCP), IP-адрес отправителя, порт отправителя, IP-адрес получателя и порт получателя.

Поля `Sequence number` и `Acknowledgement number` (Номер подтверждения) выполняют свою обычную функцию. Обратите внимание: поле `Acknowledgement number` относится к следующему по порядку ожидаемому байту, а не к последнему полученному. Это **накопительное подтверждение (cumulative acknowledgement)**, так как один номер объединяет в себе информацию обо всех полученных данных. Сфера его применения не выходит за рамки потерянных данных. Оба поля 32-разрядные, поскольку в TCP-потоке нумеруется каждый байт данных.

Поле TCP header length (Длина TCP-заголовка) сообщает, сколько 32-разрядных слов содержится в TCP-заголовке. Эта информация необходима, так как поле Options, а вместе с ним и весь заголовок имеет переменную длину. По сути, TCP header length указывает смещение от начала сегмента до поля данных, измеренное в 32-битных словах. Это то же самое, что длина заголовка.

Следом идет неиспользуемое 4-битное поле. Тот факт, что эти биты не используются уже 30 лет (изначально поле было 6-битным и из них были задействованы только 2 бита), свидетельствует о том, насколько хорошо продуман дизайн TCP. Иначе протоколы использовали бы эти биты, чтобы справиться с его недостатками.

Затем следуют восемь 1-битных флагов. CWR и ECE сообщают о перегрузках сети в случае, если используется явное уведомление о перегрузке (см. RFC 3168). Когда TCP-получатель узнает, что сеть перегружена, он с помощью флага ECE передает TCP-отправителю сигнал ECN-Echo (ECN-эхо), предлагая ему снизить скорость отправки. Уменьшив скорость, TCP-отправитель сообщает об этом TCP-получателю с помощью флага CWR с сигналом Congestion Window Reduced (Окно перегрузки уменьшено), после чего получатель перестает передавать сигнал ECN-Echo. Подробнее о роли ECN и CWR при контроле перегрузки в TCP мы поговорим в разделе 6.5.10.

Бит URG устанавливается в 1 в случае использования поля Urgent pointer (Указатель срочности), где указано байтовое смещение от текущего порядкового номера до срочных данных. Таким образом, в TCP реализуются прерывающие сообщения. Как уже упоминалось, этот метод позволяет отправителю передать получателю сигнал, не вовлекая в это TCP; он используется редко.

Если бит ACK установлен в 1, значит, поле Acknowledgement number действует. Это справедливо для большинства пакетов. Если ACK установлен в 0, значит, сегмент не содержит подтверждения, и поле Acknowledgement number игнорируется.

Бит PSH является, по сути, PUSH-флагом, с помощью которого отправитель вежливо просит получателя доставить данные приложению сразу, а не хранить их в буфере, пока тот не наполнится (получатель может это делать в целях эффективности).

Бит RST используется для внезапного сброса состояния соединения, которое из-за сбоя хоста или по другой причине попало в тупиковую ситуацию. Также он применяется для отказа от неверного сегмента или от попытки создать соединение. Если в сегменте установлен бит RST, это означает проблему.

Бит SYN применяется для установки соединения. У запроса соединения бит SYN = 1, а бит ACK = 0, то есть поле подтверждения не задействовано. Но в ответе на этот запрос подтверждение есть, поэтому значения этих битов таковы: SYN = 1, ACK = 1. Таким образом, бит SYN используется для обозначения как сегмента CONNECTION REQUEST, так и CONNECTION ACCEPTED, а бит ACK — чтобы различать их.

Бит FIN используется для разрыва соединения. Он сообщает, что у отправителя больше нет данных для *передачи*. Однако, даже закрыв соединение, процесс может продолжать *получать* данные в течение неопределенного времени. У сегментов с битами FIN и SYN есть порядковые номера, что гарантирует правильный порядок их выполнения.

Управление потоком в TCP осуществляется при помощи раздвижного окна переменного размера. Поле `Window size` (Размер окна) сообщает, сколько байтов может быть отправлено после подтвержденного байта. Нулевое значение `Window size` означает, что все байты до `Acknowledgement number` – 1 включительно пришли, но получатель их еще не обработал, и поэтому остальные байты он пока принять не может. Позже получатель может разрешить дальнейшую передачу, отправив сегмент с таким же значением `Acknowledgement number` и ненулевым значением `Window size`.

В главе 3 мы изучали протоколы, в которых подтверждения приема фреймов были связаны с разрешениями на продолжение передачи. Это следствие фиксированного размера раздвижного окна в этих протоколах. В TCP подтверждения отделены от разрешений на передачу. В сущности, получатель может сказать: «Я получил байты вплоть до $k$-го, пока что достаточно, спасибо». Такое разделение (а если точнее, окно переменного размера) придает протоколу дополнительную гибкость. Далее мы обсудим его более детально.

Поле `Checksum` служит для повышения надежности. Как и в UDP, оно содержит контрольную сумму заголовка, данных и псевдозаголовка. Но в отличие от UDP псевдозаголовок содержит номер протокола TCP (6), а контрольная сумма является обязательной. Более подробная информация приведена в разделе 6.4.1.

Поле `Options` предоставляет дополнительные возможности, не покрываемые стандартным заголовком. Существует множество параметров, и некоторые из них широко используются. Они имеют разную длину, кратную 32 битам (лишнее место заполняется нулями), и могут доходить до отметки в 40 байт — максимального размера заголовка TCP. При установлении соединения факультативные поля могут использоваться для того, чтобы договориться с противоположной стороной или просто сообщить ей о характеристиках этого соединения. Существуют поля, сохраняющиеся в течение всего времени жизни соединения. Все факультативные поля имеют формат Тип-Длина-Значение (Type-Length-Value).

С помощью одного из таких полей хост может указать **максимальный размер сегмента (Maximum Segment Size, MSS)**, который он может принять. Чем больше размер, тем выше эффективность, так как при этом снижается удельный вес накладных расходов в виде 20-байтных заголовков, однако не все хосты способны принимать крупные сегменты. Хосты могут сообщить друг другу MSS во время установки соединения. По умолчанию он равен 536 байтам. Все хосты обязаны принимать TCP-сегменты размером 536 + 20 = 556 байт. Для каждого направления можно установить свой MSS.

Для линий с высокой скоростью передачи и/или большой задержкой окно в 64 Кбайт, соответствующее 16-битному полю, оказывается слишком маленьким. Так, на линии OC-12 (со скоростью приблизительно 600 Мбит/с) для вывода полного окна в 64 Кбайт потребуется менее 1 мс. Если время распространения сигнала в оба конца составляет 50 мс (что типично для трансконтинентального оптического кабеля), 98 % времени отправитель будет ждать подтверждения. Больший размер окна мог бы повысить эффективность. Параметр **масштаб окна (window scale)** позволяет двум хостам договориться о масштабе окна при установке соединения. С его помощью стороны могут сдвигать поле `Window size` до

14 разрядов влево, расширяя окна до $2^{30}$ байт (1 Гбайт). Большинство реализаций TCP поддерживают эту возможность.

Для **временных меток (timestamps)**, передаваемых от отправителя к получателю и обратно, существует одноименный параметр. Если во время настройки соединения было решено использовать этот параметр, он добавляется в каждый пакет. Он позволяет получить данные об RTT, необходимые для выявления потери пакетов. Также он используется в качестве логического расширения 32-битного порядкового номера. При высокоскоростном соединении порядковые номера могут проходить полный круг очень быстро, и в результате новые и старые данные будет невозможно отличить. Описанная ранее схема PAWS удаляет сегменты со старыми временными метками, позволяя избежать этой проблемы.

Наконец, с помощью **выборочного подтверждения (Selective ACKnowledgement, SACK)** получатель может сообщать отправителю диапазоны порядковых номеров доставленных пакетов. Этот параметр является дополнением к `Acknowledgement number` и используется, если после потери пакета данные все равно были доставлены (возможно, в виде копии). Новые данные не отражены в поле заголовка `Acknowledgement number`, так как оно содержит только следующий по порядку ожидаемый байт. Благодаря SACK отправитель всегда будет знать, какие данные есть у получателя, и повторит передачу, только если это действительно нужно. SACK описан в RFC 2108 и RFC 2883. В последнее время эта схема используется все чаще. О ее применении при контроле перегрузки мы поговорим в разделе 6.5.10.

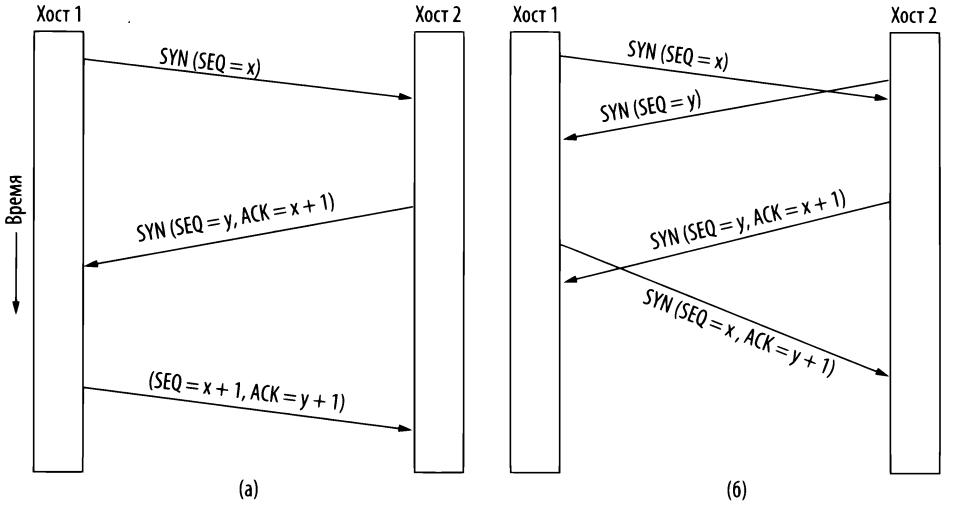
## 6.5.5. Установка TCP-соединения

В TCP соединения устанавливаются с помощью «тройного рукопожатия», как было описано в разделе 6.2.2. Чтобы создать соединение, одна сторона (например, сервер) пассивно ожидает входящего соединения, выполняя примитивы `LISTEN` и `ACCEPT` с указанием конкретного источника либо без него.

Другая сторона (например, клиент) выполняет примитив `CONNECT`, сообщая IP-адрес и порт, с которым она хочет установить соединение, максимальный размер TCP-сегмента, который она может принять, и, по желанию, некоторые данные пользователя (например, пароль). `CONNECT` отправляет TCP-сегмент с установленным битом `SYN` и сброшенным битом `ACK` и ждет ответа от другой стороны.

Когда этот сегмент приходит по назначению, TCP-подсистема проверяет, выполнил ли какой-нибудь процесс примитив `LISTEN`, указав в качестве параметра тот же порт, который содержится в поле `Destination port`. Если такого процесса нет, она отвечает отправкой сегмента с установленным битом `RST` для отказа от соединения.

Если какой-то процесс прослушивает указанный порт, то TCP-сегмент передается этому процессу. Он может принять соединение или отказаться от него. Если процесс принимает соединение, он отвечает подтверждением. Последовательность TCP-сегментов, отправляемых в обычном случае, показана на илл. 6.37 (а). Обратите внимание, что сегмент с установленным битом `SYN` занимает 1 байт пространства порядковых номеров, что позволяет избежать неоднозначности в их подтверждениях.




**Илл. 6.37.** (а) Установка TCP-соединения в обычном случае. (б) Одновременная установка соединения обеими сторонами

Когда два хоста одновременно пытаются установить соединение друг с другом, то события происходят в иной последовательности (см. илл. 6.37 (б)). В результате будет установлено только одно соединение, а не два, так как соединения идентифицируются по паре конечных точек. То есть если они оба обозначают себя с помощью пары $(x, y)$, делается всего одна табличная запись $(x, y)$.

Начальное значение порядкового номера, выбранное каждым хостом, должно медленно меняться, а не равняться константе (например, нулю). Как мы уже говорили в разделе 6.2.2, это правило обеспечивает защиту от задержавшихся копий пакетов. Изначально эта схема была реализована с помощью таймера, изменяющего свое состояние каждые 4 мкс.

Однако проблема реализации схемы «тройного рукопожатия» состоит в том, что слушающий процесс должен помнить свой порядковый номер до тех пор, пока он не отправит собственный SYN-сегмент. Это значит, что злонамеренный отправитель может блокировать ресурсы хоста, отправляя на него поток SYN-сегментов и не разрывая соединение. Такие атаки называются **лавинной адресацией SYN-сегментов (SYN flood)**. В 1990-е годы многие веб-серверы оказались парализованными из-за них. Сегодня уже существуют методы защиты от таких атак.

В частности, для защиты от них можно использовать метод под названием **SYN cookies**. Вместо того чтобы запоминать порядковый номер, хост генерирует криптографическое значение номера, записывает его в исходящий сегмент и забывает. Если «тройное рукопожатие» завершается, этот номер (увеличенный на единицу) вернется на хост. Хост может повторно сгенерировать правильный порядковый номер, вычислив значение той же криптографической функции, при условии, что известны входные данные (это может быть IP-адрес и порт другого хоста, а также какое-то секретное значение). С помощью этой процедуры хост

может проверять правильность подтвержденного порядкового номера, не запоминая его. Одна из тонкостей этого метода состоит в том, что он не работает с дополнительными параметрами TCP. Поэтому SYN cookies можно использовать только в случае лавинной адресации SYN-сегментов. Но в целом это очень интересный прием. Более подробно см. RFC 4987 и работу Лемона (Lemon, 2002).

## 6.5.6. Разрыв TCP-соединения

Хотя TCP-соединения полнодуплексные, чтобы понять, как происходит их разъединение, лучше считать их парами симплексных соединений. Каждое симплексное соединение разрывается независимо от своего напарника. Чтобы его разорвать, любая из сторон может отправить TCP-сегмент с установленным битом FIN; это означает, что у него больше нет данных для передачи. После подтверждения TCP-сегмента это направление закрывается. Тем не менее данные могут продолжать передаваться неопределенно долго в противоположную сторону. Соединение разрывается, когда закрываются оба направления. Обычно для разрыва требуются четыре TCP-сегмента: по одному с битом FIN и по одному с битом ACK в каждом направлении. Первый бит ACK и второй бит FIN могут также содержаться в одном TCP-сегменте, что уменьшит количество сегментов до трех.

Как при телефонном разговоре, когда оба участника могут одновременно попрощаться и повесить трубку, оба конца TCP-соединения могут отправить FIN-сегменты в одно и то же время. Они оба получают обычные подтверждения, и соединение закрывается. По сути, между одновременным и последовательным разъединением нет никакой разницы.

Чтобы избежать проблемы «двух армий» (см. раздел 6.2.3), используются таймеры. Если ответ на отправленный FIN-сегмент не приходит в течение двух максимальных интервалов времени жизни пакета, отправитель FIN-сегмента разрывает соединение. Другая сторона в конце концов заметит, что ей никто не отвечает, и также отсоединится. Эта схема несовершенна, однако, учитывая недостижимость идеала, приходится пользоваться тем, что есть. На практике проблемы возникают довольно редко.

## 6.5.7. Модель управления TCP-соединением

Этапы, необходимые для установления и разрыва соединения, могут быть представлены в виде модели конечного автомата, 11 состояний которого перечислены на илл. 6.38. В каждом из этих состояний разрешены определенные события, в ответ на которые могут осуществляться действия. При возникновении каких-либо других событий сообщается об ошибке.

Каждое соединение начинается в состоянии *CLOSED* (закрыто). Оно может покинуть это состояние, предпринимая либо активную (CONNECT), либо пассивную (LISTEN) попытку открыть соединение. Если другая сторона осуществляет противоположное действие, соединение устанавливается и переходит в состояние *ESTABLISHED*. Инициатором разрыва соединения может выступить любая сторона. По завершении этого процесса соединение возвращается в состояние *CLOSED*.

| Состояние | Описание |
|---|---|
| CLOSED | Закрыто. Соединение не является активным и не находится в процессе установления |
| LISTEN | Ожидание. Сервер ожидает входящего запроса |
| SYN RCVD | Прибыл запрос соединения. Ожидание подтверждения |
| SYN SENT | Запрос соединения отправлен. Приложение начало открывать соединение |
| ESTABLISHED | Установлено. Нормальное состояние передачи данных |
| FIN WAIT 1 | Приложение сообщило, что ему больше нечего передавать |
| FIN WAIT 2 | Другая сторона согласна разорвать соединение |
| TIME WAIT | Ожидание, пока из сети не исчезнут все пакеты |
| CLOSING | Обе стороны попытались одновременно закрыть соединение |
| CLOSE WAIT | Другая сторона инициировала разъединение |
| LAST ACK | Ожидание, пока из сети не исчезнут все пакеты |

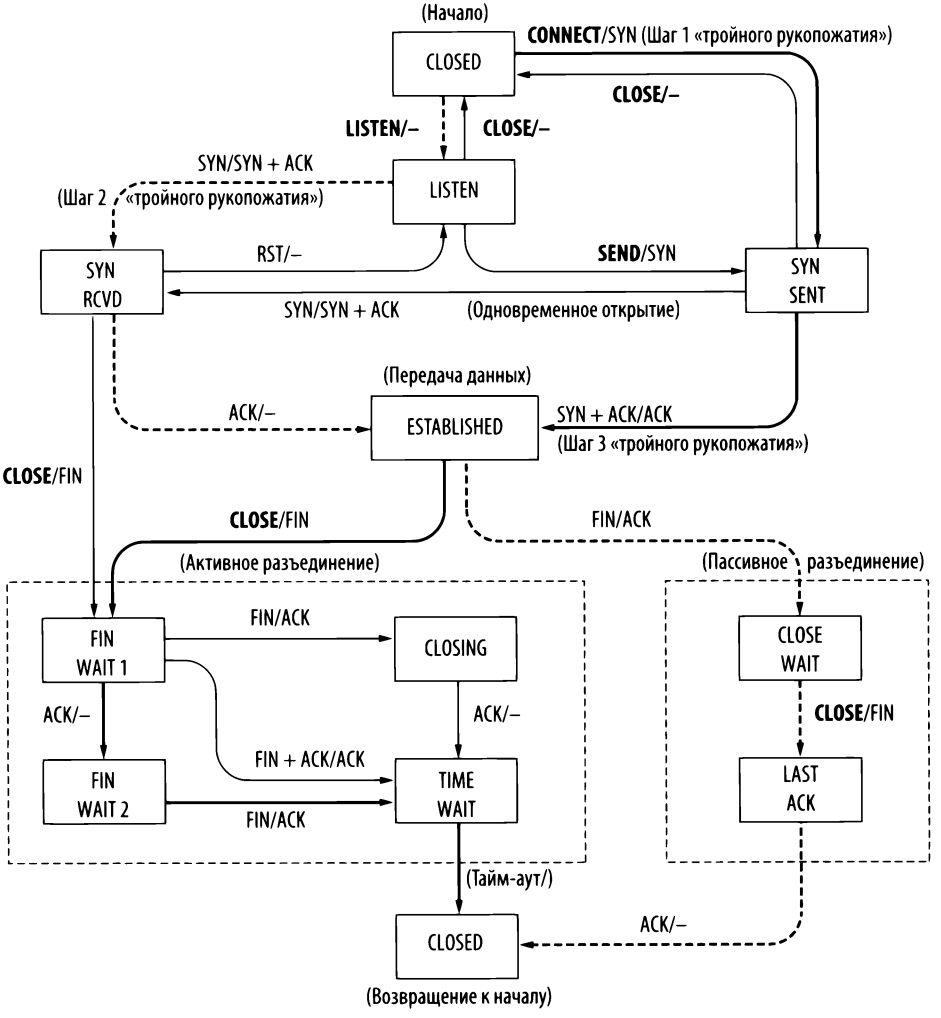
**Илл. 6.38.** Состояния конечного автомата, управляющего TCP-соединением

Конечный автомат показан на илл. 6.39. Типичный случай клиента, активно соединяющегося с пассивным сервером, показан жирными линиями — сплошными для клиента и пунктирными для сервера. Тонкие линии обозначают необычные последовательности событий. Каждая линия на илл. 6.39 маркирована парой *событие/действие*. Событие может представлять собой либо обращение пользователя к системной процедуре (`CONNECT`, `LISTEN`, `SEND` или `CLOSE`), либо прибытие сегмента (`SYN`, `FIN`, `ACK` или `RST`), либо, в одном случае, окончание периода ожидания, равного двойному времени жизни пакетов. Действие может состоять в отправке управляющего сегмента (`SYN`, `FIN` или `RST`). Впрочем, может не предприниматься никакого действия, что обозначается прочерком. В скобках приводятся комментарии.

Диаграмму проще понять, если сначала проследовать по пути клиента (сплошная жирная линия), а затем — по пути сервера (жирный пунктир). Когда приложение на устройстве клиента вызывает операцию `CONNECT`, локальная TCP-подсистема создает запись соединения, помечает его состояние как *SYN SENT* и отправляет `SYN`-сегмент. Обратите внимание, что несколько приложений одновременно могут открыть множество соединений, и состояние каждого из них хранится в записи соединения. Когда прибывает сегмент `SYN` + `ACK`, TCP-подсистема отправляет последний `ACK`-сегмент «тройного рукопожатия» и переключается в состояние *ESTABLISHED*. В этом состоянии можно пересылать и получать данные.

Когда у приложения заканчиваются данные для передачи, оно выполняет операцию `CLOSE`, заставляющую локальную TCP-подсистему отправить `FIN`-сегмент и ждать ответного `ACK`-сегмента (пунктирный прямоугольник с пометкой «активное разъединение»). Когда прибывает подтверждение, происходит переход в состояние

*FIN WAIT 2*, и одно направление соединения закрывается. Когда приходит встречный `FIN`-сегмент, в ответ на него также высылается подтверждение, после чего закрывается второе направление. Теперь обе стороны соединения закрыты, но TCP-подсистема ожидает в течение удвоенного максимального времени жизни пакета. Таким образом гарантируется, что ни один пакет этого соединения больше не перемещается по сети, даже если подтверждение было потеряно. Когда период ожидания истекает, TCP-подсистема удаляет запись о соединении.




**Илл. 6.39.** Конечный автомат TCP-соединения. Жирная сплошная линия показывает нормальный путь клиента. Жирным пунктиром показан нормальный путь сервера. Тонкими линиями обозначены необычные события. Для каждого перехода через косую черту указано, какое событие его вызывает и к выполнению какого действия он приводит

Далее рассмотрим управление соединением с точки зрения сервера. Он выполняет LISTEN и переходит в режим ожидания запросов соединения. Когда приходит SYN-сегмент, в ответ на него высылается подтверждение, после чего сервер переходит в состояние *SYN RCVD* (запрос соединения получен). Когда в ответ на SYN-подтверждение от клиента приходит ACK-сегмент, процедура «тройного рукопожатия» завершается и сервер переходит в состояние *ESTABLISHED*. Теперь можно передавать данные.

По окончании передачи данных клиент выполняет операцию CLOSE, в результате чего на сервер приходит FIN-сегмент (пунктирный прямоугольник, обозначенный как пассивное разъединение). Получив оповещение, сервер тоже выполняет CLOSE, и клиенту отправляется FIN-сегмент. Когда от клиента прибывает подтверждение, сервер разрывает соединение и удаляет запись о нем.

## 6.5.8. Раздвижное окно TCP

Как уже было сказано выше, управление окном в TCP решает проблемы подтверждения корректной доставки сегментов и выделения буферов получателя. Предположим, у получателя есть 4096-байтный буфер (илл. 6.40). Если отправитель передает 2048-байтный сегмент, который успешно принимается получателем, то последний подтверждает его получение. Однако при этом у получателя остается всего лишь 2048 байт свободного буферного пространства (пока приложение не заберет какое-то количество данных из буфера), о чем он и сообщает отправителю, указывая соответствующий размер окна (2048) и номер следующего ожидаемого байта.

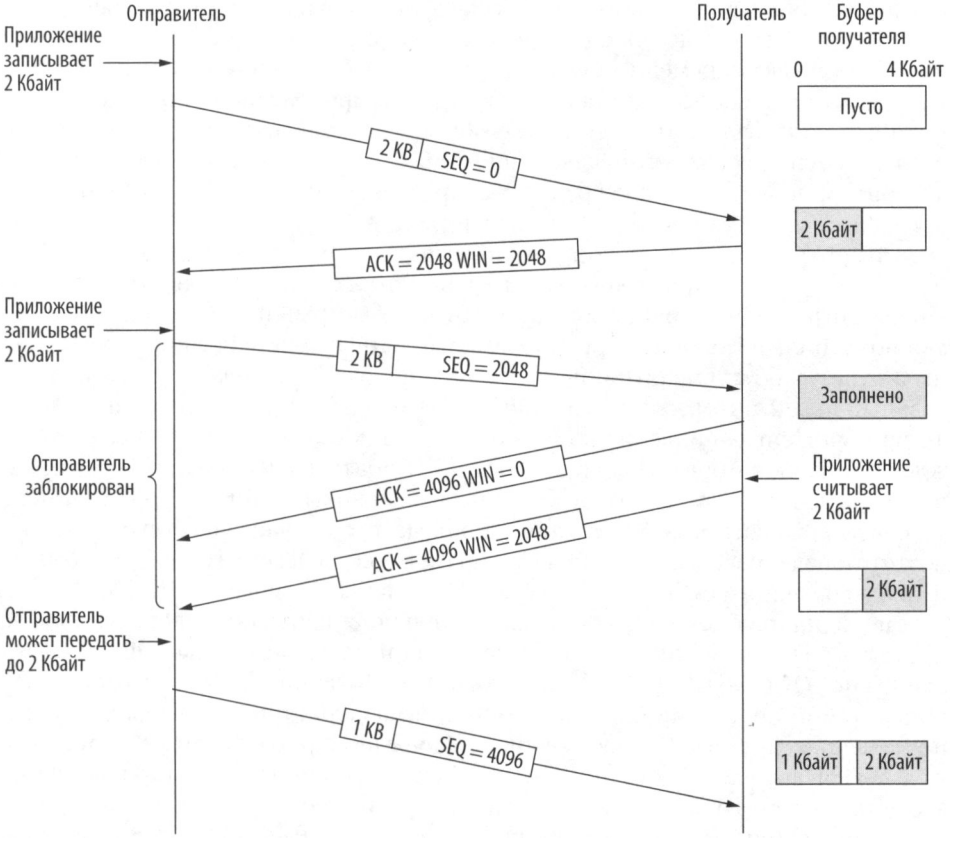
После этого отправитель отправляет еще 2048 байт, получение которых подтверждается, но размер окна объявляется равным нулю. Отправитель должен прекратить передачу до тех пор, пока получающий хост не освободит место в буфере и не увеличит размер окна.

При нулевом размере окна отправитель не может отправлять сегменты, за исключением двух случаев. Во-первых, разрешается передавать срочные данные, например, чтобы пользователь мог уничтожить процесс, выполняющийся на удаленном компьютере. Во-вторых, отправитель может отправить 1-байтный сегмент, прося получателя повторить информацию о размере окна и ожидаемом следующем байте. Такой пакет называется **пробным сегментом (window probe)**. Стандарт TCP прямо предусматривает эту возможность для предотвращения тупиковых ситуаций в случае потери объявления о размере окна.

Отправители не обязаны передавать данные сразу же, как только они приходят от приложения. Также никто не требует от получателей отправлять подтверждения как можно скорее. Например, было бы абсолютно логично, если бы TCP-подсистема на илл. 6.40, получив от приложения первые 2 Кбайт данных и зная, что размер окна равен 4 Кбайт, сохранила бы полученные данные в буфере до тех пор, пока не придут еще 2 Кбайт, чтобы передать сегмент из 4 Кбайт. Такая свобода действий может улучшить производительность.

Рассмотрим соединение (к примеру, telnet или SSH) с удаленным терминалом, реагирующим на каждое нажатие клавиши. При наихудшем сценарии, когда символ прибывает к передающей TCP-подсистеме, она создает 21-байтный

TCP-сегмент и передает его IP-уровню, который, в свою очередь, отправляет 41-байтную IP-дейтаграмму. На принимающей стороне TCP-подсистема немедленно отвечает 40-байтным подтверждением (20 байт TCP-заголовка и 20 байт IP-заголовка). Затем, когда удаленный терминал прочитает этот байт из буфера, TCP-подсистема отправит обновленную информацию о размере буфера, передвинув окно на 1 байт вправо. Размер этого пакета также составляет 40 байт. Наконец, когда удаленный терминал обработает этот символ, он отправит обратно эхо, включенное в 41-байтный пакет. Итого для каждого введенного с клавиатуры символа пересылается четыре пакета общим размером 162 байта. При дефиците пропускной способности линий этот метод работы нежелателен.




**Илл. 6.40.** Управление окном в TCP

Чтобы улучшить ситуацию, многие реализации TCP используют **отложенные подтверждения (delayed acknowledgements)**. Идея в том, чтобы задерживать подтверждения и обновления размера окна на время до 500 мс в надежде получить дополнительные данные и отправить подтверждение вместе с ними.

Если терминал успеет выдать эхо в течение 500 мс, удаленной стороне нужно будет выслать только один 41-байтный пакет, таким образом, нагрузка на сеть снизится вдвое.

Хотя отложенные подтверждения и снижают нагрузку на сеть, тем не менее отправитель, передающий множество маленьких пакетов (к примеру, 41-байтные пакеты с 1 байтом реальных данных), по-прежнему работает неэффективно. Метод, позволяющий повысить эффективность, известен как **алгоритм Нейгла (Nagle's algorithm)** (Nagle, 1984). Предложение Нейгла звучит просто: если данные поступают отправителю маленькими порциями, он просто передает первый фрагмент, а остальные помещает в буфер, пока не получит подтверждение приема первого фрагмента. После этого можно переслать все накопленные в буфере данные в виде одного TCP-сегмента и снова начать буферизацию до получения подтверждения о доставке следующего сегмента. Таким образом, в каждый момент времени может передаваться только один небольшой пакет. Если за время прохождения пакета в обе стороны приложение отправляет много порций данных, алгоритм Нейгла объединяет несколько таких порций в один сегмент, и нагрузка на сеть существенно снижается. Кроме того, согласно этому алгоритму, новый пакет должен быть отправлен, если объем данных в буфере превышает максимальный размер сегмента.

Алгоритм Нейгла широко применяется различными реализациями TCP, однако бывают ситуации, в которых его лучше отключить. В частности, интерактивным играм по интернету обычно требуется быстрый поток мелких пакетов с обновлениями. Если буферизировать эти данные для пакетной пересылки, игра будет работать неправильно, что не порадует пользователей. Дополнительный нюанс в том, что иногда при задержке подтверждений использование алгоритма Нейгла приводит к временным тупиковым ситуациям: получатель ждет данные, к которым можно присоединить подтверждение, а источник ждет подтверждение, без которого не будут переданы новые данные. Так, например, может задерживаться загрузка веб-страниц. На этот случай существует возможность отключения алгоритма Нейгла (параметр `TCP_NODELAY`). Подробнее об этих и других решениях см. работу Могула и Миншалла (Mogul and Minshall, 2001).

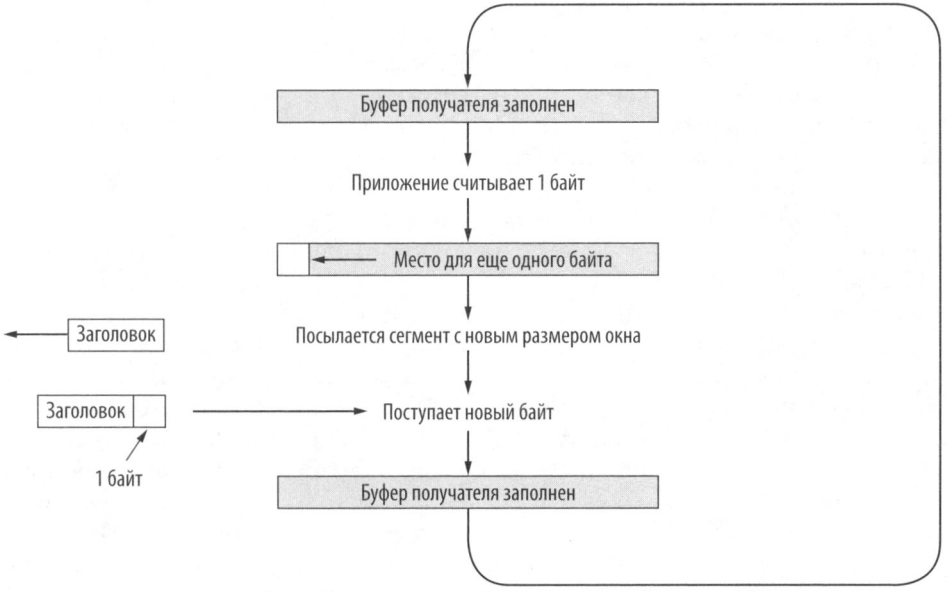
Еще одна проблема, способная значительно снизить производительность протокола TCP, известна под названием **синдрома глупого окна (silly window syndrome)** (Кларк; Clark, 1982). Ее суть в том, что данные пересылаются TCP-подсистемой крупными блоками, но принимающая сторона интерактивного приложения считывает их посимвольно. Чтобы разобраться в этом, рассмотрим илл. 6.41. Начальное состояние таково: TCP-буфер приемной стороны полон (то есть размер его окна равен нулю), и отправителю это известно. Затем интерактивное приложение читает один символ из TCP-потока. Принимающая TCP-подсистема радостно сообщает отправителю, что размер окна увеличился и что он теперь может отправить 1 байт. Отправитель повинуется и передает 1 байт. Буфер снова заполняется, о чем получатель сообщает с помощью подтверждения для 1-байтного сегмента с нулевым размером окна. И так может продолжаться вечно.

Дэвид Кларк предложил запретить принимающей стороне отправлять информацию об однобайтовом размере окна. Вместо этого получатель должен

подождать, пока в буфере не накопится значительное количество свободного места. В частности, получатель не должен отправлять сведения о новом размере окна, пока не сможет принять сегмент максимального размера (который он объявлял при установке соединения) или пока его буфер не освободится хотя бы наполовину. Кроме того, увеличению эффективности передачи может способствовать сам отправитель, отказываясь от отправки слишком маленьких сегментов. Вместо этого он должен подождать, пока размер окна не станет достаточно большим для отправки полного сегмента (или хотя бы равного половине размера буфера получателя).

В задаче избавления от синдрома глупого окна алгоритм Нейгла и решение Кларка дополняют друг друга. Нейгл пытался решить проблему приложения, предоставляющего данные TCP-подсистеме посимвольно. Кларк старался разрешить проблему приложения, посимвольно получающего данные у TCP. Оба решения хороши и могут работать одновременно. Суть их заключается в том, чтобы не отправлять и не просить передавать данные слишком малыми порциями.

Для повышения производительности принимающая TCP-подсистема может делать нечто большее, чем просто обновлять информацию о размере окна крупными порциями. Как и отправляющая TCP-подсистема, она может буферизировать данные и блокировать запрос `READ` (чтение данных), поступающий от приложения, пока у нее не накопится значительный объем данных. Таким образом снижается количество обращений к TCP-подсистеме (и вместе с ними накладные расходы). Конечно, такой подход увеличивает время ответа, но для неинтерактивных приложений, например, при передаче файла, эффективность может быть важнее увеличения времени ответа на отдельные запросы.




**Илл. 6.41.** Синдром глупого окна

Еще одна проблема получателя состоит в том, что сегменты могут приходить не по порядку. Он будет хранить данные в буфере, пока не сможет передать их приложению в нужной последовательности. В принципе, нет ничего плохого в том, чтобы отвергать пакеты, прибывшие не в свою очередь, ведь они все равно будут повторно переданы отправителем, однако это неэффективно.

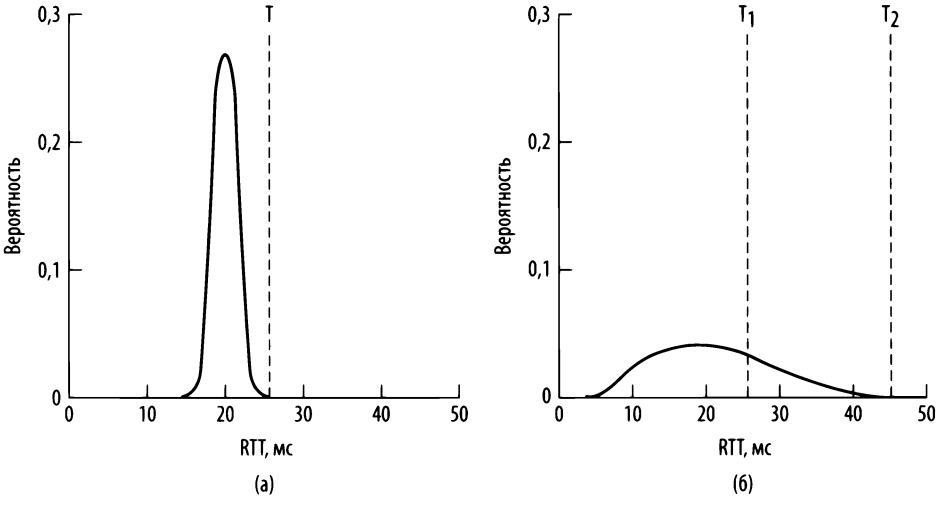
Подтверждение может быть выслано, только если все данные, включая подтверждаемый байт, получены. Это накопительное подтверждение. Если до получателя доходят сегменты 0, 1, 2, 4, 5, 6 и 7, он может подтвердить получение данных вплоть до последнего байта сегмента 2. Когда у отправителя истечет время ожидания, он передаст сегмент 3 еще раз. Если к прибытию сегмента 3 получатель сохранит в буфере сегменты с 4-го по 7-й, он сможет подтвердить получение всех байтов, включая последний байт сегмента 7.

## 6.5.9. Управление таймерами в TCP

В TCP используется множество таймеров (по крайней мере, в теории). Наиболее важным из них является **таймер повторной передачи (Retransmission TimeOut, RTO)**, который запускается при отправке сегмента. Если подтверждение получения сегмента придет раньше окончания заданного интервала, таймер останавливается. И наоборот, если период ожидания истечет раньше, чем прибудет подтверждение, сегмент передается еще раз (а таймер запускается снова). Соответственно, возникает вопрос: насколько долгим должен быть интервал времени ожидания?

На транспортном уровне эта проблема значительно сложнее, чем в протоколах канального уровня. Например, в 802.11 величина ожидаемой задержки измеряется в микросекундах, и ее довольно легко предсказать (у нее небольшой разброс), поэтому таймер можно установить на момент чуть позднее ожидаемого прибытия подтверждения (илл. 6.42 (а)). Поскольку перегрузок нет, подтверждения на канальном уровне задерживаются редко, поэтому их отсутствие в течение установленного временного интервала с большой вероятностью означает потерю фрейма или подтверждения.

TCP вынужден работать в совершенно иных условиях. Функция плотности вероятности для времени, необходимого для доставки подтверждения TCP, выглядит скорее как график на илл. 6.42 (б), чем на илл. 6.42 (а). Она более пологая и вариативная. Поэтому предсказать, сколько времени потребуется для прохождения данных от отправителя к получателю и обратно, весьма непросто. Даже если бы мы знали, каким должно быть это время, есть еще одна сложность — выбор подходящего интервала ожидания. Если он слишком короткий (например, $T_1$ на илл. 6.42 (б)), возникнут излишние повторные передачи, заполняющие интернет бесполезными пакетами. Если же установить слишком большое значение ($T_2$), то из-за увеличения времени ожидания в случае потери пакета пострадает производительность. При этом среднее значение и величина дисперсии времени прибытия подтверждений могут измениться за несколько секунд при возникновении и устранении перегрузки.




**Илл. 6.42.** Плотность вероятности времени прибытия подтверждения: (а) на канальном уровне; (б) для TCP

Решение состоит в использовании динамического алгоритма, который постоянно меняет период ожидания, основываясь на измерениях производительности сети. Алгоритм, широко применяемый в TCP, разработан Джейкобсоном (Jacobson) в 1988 году и работает следующим образом. Для каждого соединения в TCP предусмотрена переменная `SRTT` (Smoothed Round-Trip Time — усредненное время в пути туда-обратно), в которой хранится текущее наилучшее ожидаемое время получения подтверждения для данного соединения. При отправке сегмента запускается таймер, который измеряет время получения подтверждения и повторяет передачу, если оно не приходит в срок. Если подтверждение успевает вернуться до истечения периода ожидания, TCP-подсистема подсчитывает время, которое понадобилось для его получения ($R$). Затем значение переменной `SRTT` обновляется по следующей формуле:

$$SRTT = \alpha\, SRTT + (1-\alpha)R,$$

где $\alpha$ — весовой коэффициент, определяющий, насколько быстро забываются старые значения. Обычно $\alpha = 7/8$. Это формула вычисления **взвешенного скользящего среднего (Exponentially Weighed Moving Average, EWMA)**, или фильтра низких частот, с помощью которого можно удалять шум.

Даже при известном значении `SRTT` выбор периода ожидания подтверждения — нетривиальная задача. В первых реализациях TCP это значение вычислялось как 2x`RTT`, но опыт показал, что постоянный множитель слишком негибкий и не реагирует на увеличение разброса. В частности, модели очередей случайного (то есть пуассоновского) трафика показывают, что когда нагрузка приближается к пропускной способности, задержка растет и становится крайне изменчивой. В результате может сработать таймер повторной передачи, после чего будет отправлена копия пакета, хотя оригинальный пакет все еще будет находиться в сети. Как правило, такие ситуации возникают

именно при высокой нагрузке — и это не самое лучшее время для отправки в сеть лишних пакетов.

Чтобы решить эту проблему, Джейкобсон предложил сделать интервал времени ожидания чувствительным к отклонению RTT и к усредненному RTT. Для этого потребовалась еще одна сглаженная переменная — RTTVAR (RoundTrip Time Variation — изменение времени в пути туда-обратно), которая вычисляется по формуле:

$$RTTVAR = \beta\, RTTVAR + (1-\beta)\left| SRTT - R \right|.$$

Как и в предыдущем случае, это взвешенное скользящее среднее. Обычно $\beta = 3/4$. Значение интервала ожидания повторной передачи (*RTO*) устанавливается по формуле:

$$RTO = SRTT + 4 \times RTTVAR.$$

Множитель 4 выбран произвольно, однако умножение целого числа на 4 может быть выполнено одной командой сдвига, при этом менее 1 % всех пакетов придет с опозданием, превышающим четыре среднеквадратичных отклонения. Обратите внимание, что RTTVAR является не среднеквадратичным, а средним отклонением, но на практике это довольно близкие значения. В своей работе Джейкобсон приводит множество хитрых способов вычисления значений интервала ожидания с помощью только целочисленного сложения, вычитания и сдвига. Для современных хостов такая экономия не требуется, но она стала элементом культуры повсеместного применения TCP: он должен работать как на суперкомпьютерах, так и на небольших устройствах. Для RFID-чипов его пока еще не реализовали, но ведь все может быть.

Более подробные сведения о том, как вычислять этот интервал ожидания, а также начальные значения переменных, можно найти в RFC 2988. Для таймера повторной передачи минимальное значение также устанавливается равным 1 с, независимо от предварительной оценки. Это значение, выбранное с запасом (и в основном эмпирически), требуется, чтобы избежать выполнения лишних повторных передач на основании измерений (Оллман и Паксон; Allman and Paxson, 1999).

При сборе данных (*R*) для вычисления RTT возникает вопрос, что делать при повторной передаче сегмента. Когда для него приходит подтверждение, неясно, к какой передаче оно относится, первой или последней. Неверная догадка может серьезно нарушить работу RTO. Эта проблема была обнаружена радиолюбителем Филом Карном (Phil Karn). Его интересовал вопрос передачи TCP/IP-пакетов с помощью коротковолновой любительской радиосвязи, известной своей ненадежностью. Предложение Карна было очень простым: не обновлять оценки для сегментов, переданных повторно. Кроме того, при каждой повторной передаче время ожидания можно удваивать до тех пор, пока сегменты не пройдут с первой попытки. Это исправление получило название **алгоритма Карна (Karn's algorithm)** (Карн и Партридж; Karn and Partridge, 1987) и применяется в большинстве реализаций TCP.

В протоколе TCP используется не только таймер повторной передачи, но и **таймер настойчивости (persistence timer)**. Он предназначен для предотвращения следующей тупиковой ситуации. Получатель отправляет подтверждение, в котором указывает окно нулевого размера (это значит, что отправитель должен подождать). Через некоторое время получатель передает пакет с новым размером окна, но этот пакет теряется. Теперь обе стороны ожидают действий друг от друга. Когда срабатывает таймер настойчивости, отправитель высылает получателю пакет с вопросом, не изменилось ли текущее состояние. В ответ получатель сообщает текущий размер окна. Если он все еще равен нулю, таймер настойчивости запускается снова, и весь цикл повторяется. В случае увеличения окна отправитель может передавать данные.

В некоторых реализациях протокола используется третий таймер, называемый **таймером проверки активности (keepalive timer)**. Он срабатывает, если соединение простаивает в течение долгого времени, заставляя одну сторону проверить, активна ли другая сторона. Если проверяющая сторона не получает ответа, соединение разрывается. Это свойство протокола довольно противоречиво, поскольку привносит дополнительные накладные расходы и может разорвать вполне жизнеспособное соединение из-за кратковременной потери связи.

И наконец, в каждом TCP-соединении используется таймер, запускаемый в состоянии *TIME WAIT* при закрытии соединения. Он отсчитывает двойное время жизни пакета, чтобы гарантировать, что после закрытия соединения созданные им пакеты исчезли.

## 6.5.10. Контроль перегрузки в TCP

Напоследок мы оставили одну из ключевых функций TCP: контроль перегрузки. Когда в сеть (в том числе в интернет) поступает больше данных, чем она способна обработать, возникают перегрузки. Если сетевой уровень узнает, что на маршрутизаторах скопились длинные очереди, он пытается справиться с этой ситуацией (пусть даже простым удалением пакетов). Транспортный уровень получает обратную связь от сетевого уровня, что позволяет ему следить за перегрузкой и при необходимости снижать скорость отправки. В интернете протокол TCP так же незаменим при контроле перегрузки, как и при транспортировке данных. Именно это делает его особенным.

Общие вопросы контроля перегрузки мы обсуждали в разделе 6.3. Основная мысль заключается в следующем: транспортный протокол, использующий закон управления AIMD при получении двоичных сигналов сети о перегрузке, сходится к справедливому и эффективному распределению пропускной способности. Контроль перегрузки в TCP реализует этот подход с помощью окна, а в качестве сигнала используется потеря пакетов. TCP поддерживает **окно перегрузки (congestion window)**, размер которого равен числу байтов, которое отправитель может передавать по сети в любой момент времени. Соответственно, скорость отправки равна размеру окна, деленному на RTT. Размер окна задается в соответствии с правилом AIMD.

Напомним, что окно перегрузки существует *в дополнение* к окну управления потоком, определяющему количество байтов, которое получатель может поместить в буфер. Они отслеживаются параллельно, и число байтов, которое отправитель может передать в сеть, равно размеру меньшего из этих окон. Таким образом, эффективное окно — наименьшее из подходящих отправителю и получателю. Здесь необходимо участие обеих сторон. TCP останавливает отправку данных, если одно из окон временно заполнено. Если получатель говорит: «Высылайте 64 Кбайт», но при этом источник знает, что отправка более 32 Кбайт засорит сеть, он все же передаст 32 Кбайт. Если же отправителю известно, что сеть способна пропустить и большее количество данных, например 128 Кбайт, он передаст столько, сколько просит получатель (то есть 64 Кбайт). Окно управления потоком было описано ранее, поэтому в дальнейшем мы будем говорить только об окне перегрузки.

Современная схема контроля перегрузки была реализована в TCP во многом благодаря стараниям Ван Джейкобсона (Van Jacobson, 1988). Это поистине захватывающая история. Начиная с 1986 года рост популярности интернета привел к возникновению ситуаций, которые позже стали называть **отказом сети из-за перегрузки (congestion collapse)**, — длительных периодов, во время которых полезная пропускная способность резко падала (более чем в 100 раз) из-за перегрузки сети. Джейкобсон (и многие другие) решил разобраться в ситуации и придумать конструктивное решение.

В результате Джейкобсону удалось реализовать высокоуровневое решение, состоявшее в использовании метода AIMD для выбора окна перегрузки. Особенно интересно, что при всей сложности контроля перегрузки в TCP он смог добавить его в уже существующий протокол, не изменив ни одного формата сообщений. Благодаря этому новое решение можно было сразу применять на практике. Сначала Джейкобсон заметил, что потеря пакетов является надежным сигналом перегрузки, даже несмотря на то что эта информация приходит с небольшим опозданием (когда сеть уже перегружена). В конце концов, трудно представить себе маршрутизатор, который не удаляет пакеты при перегрузке, и в дальнейшем это вряд ли изменится. Даже когда буферная память будет исчисляться терабайтами, вероятно, мы будем использовать терабитные сети, которые будут ее заполнять.

Здесь есть одна тонкость. Дело в том, что использование потери пакетов в качестве сигнала перегрузки предполагает, что ошибки передачи происходят сравнительно редко. В случае беспроводных сетей (таких, как 802.11) это не так, поэтому в них используются собственные механизмы повторной передачи данных на канальном уровне. Из-за особенностей повторной передачи в таких сетях потеря пакетов на сетевом уровне, вызванная ошибками передачи, обычно не учитывается. Столь же редко это происходит в проводных и оптоволоконных сетях, поскольку их частота ошибок по битам обычно низкая.

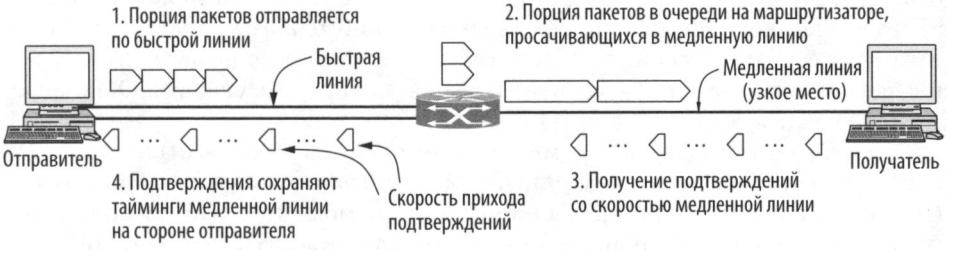
Все алгоритмы TCP для интернета основаны на том предположении, что пакеты теряются из-за перегрузок. Поэтому они внимательно отслеживают тайм-ауты и пытаются обнаружить любые признаки проблемы подобно тому, как шахтеры следят за своими канарейками[1]. Чтобы узнавать о потере пакетов

[1] Долгое время шахтеры использовали канареек в качестве средства обнаружения опасного для жизни рудничного газа. — *Примеч. ред.*

вовремя и с высокой точностью, необходим хороший таймер повторной передачи. Мы уже говорили о том, как такие таймеры в TCP учитывают среднее значение и отклонение RTT. Усовершенствование таймеров путем учета отклонений стало важным шагом в работе Джейкобсона. Если время ожидания повторной передачи выбрано правильно, TCP-отправитель может отследить количество исходящих байтов, нагружающих сеть, — достаточно сравнить порядковые номера переданных и подтвержденных пакетов.

Теперь наша задача выглядит просто. Все, что нам нужно, — это следить за размером окна перегрузки (с помощью порядковых номеров и номеров подтверждений) и менять его, следуя правилу AIMD. Но как вы уже догадались, на самом деле все гораздо сложнее. Во-первых, способ отправки пакетов в сеть (даже через короткие промежутки времени) должен соответствовать сетевому пути, иначе возникнет перегрузка. Допустим, хост с окном перегрузки 64 Кбайт подключен к коммутируемой сети Ethernet, работающей на скорости 1 Гбит/с. Если хост отправит целое окно за один раз, всплеск трафика может пройти через медленную ADSL-линию (1 Мбит/с), расположенную далее на пути. Всплеск, который длился всего половину миллисекунды на гигабитной линии, парализует медленную линию на целых полсекунды, полностью блокируя такие протоколы, как VoIP. В итоге мы получим отличный протокол для создания перегрузок, а не для борьбы с ними.

Однако отправка небольших порций пакетов может быть полезной. На илл. 6.43 показано, что произойдет, если хост-отправитель, подключенный к быстрой линии (1 Гбит/с), отправит небольшую порцию пакетов (4) получателю, находящемуся в медленной сети (1 Мбит/с), которая является узким местом пути или его самой медленной частью. Сначала эти четыре пакета перемещаются по сети с той скоростью, с которой они были отправлены. Затем маршрутизатор помещает их в очередь, так как они приходят по высокоскоростной линии быстрее, чем передаются по медленной. Это не слишком длинная очередь, поскольку число пакетов, отправленных за один раз, невелико. Обратите внимание, что на медленной линии одни и те же пакеты выглядят длиннее, чем на быстрой, так как их передача длится дольше.




**Илл. 6.43.** Порция пакетов, переданная отправителем, и скорость прихода подтверждений

Наконец, пакеты попадают к адресату, и он подтверждает их получение. Время отправки подтверждения зависит от времени прибытия пакета по медленному каналу. Поэтому на обратном пути расстояние между пакетами будет больше,

чем в самом начале, когда исходные пакеты перемещались по быстрой линии. Оно не изменится на протяжении всего прохождения подтверждений через сеть и обратно.

Здесь особенно важно следующее: подтверждения приходят к отправителю примерно с той же скоростью, с которой пакеты могут передаваться по самому медленному каналу пути. Именно она и нужна отправителю. Если он будет передавать пакеты в сеть с такой скоростью, они будут перемещаться настолько быстро, насколько позволяет самая медленная линия, но зато не будут застревать в очередях на маршрутизаторах. Эта скорость называется **скоростью прихода подтверждений (ack clock)** и является неотъемлемой частью TCP. Данный параметр позволяет TCP выровнять трафик и избежать ненужных очередей на маршрутизаторах.

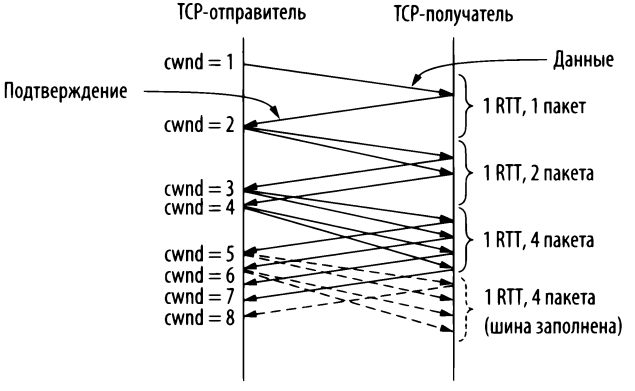
Вторая сложность состоит в том, что достижение хорошего рабочего режима согласно AIMD в быстрых сетях потребует очень много времени, если изначально выбрано маленькое окно перегрузки. Рассмотрим средний сетевой путь, позволяющий передавать трафик со скоростью 10 Мбит/с и RTT 100 мс. Здесь удобно использовать окно перегрузки, равное произведению пропускной способности и времени задержки, то есть 1 Мбит, или 100 пакетов по 1250 байт. Если изначально взять окно размером один пакет и увеличивать его на один пакет через интервал времени, равный времени в пути туда-обратно, соединение начнет работать с нормальной скоростью только через 100 RTT, то есть через 10 с. Это очень долго. Теоретически мы можем начать с большего окна — скажем, размером 50 пакетов. Но для медленных линий это значение будет слишком большим, и при отправке 50 пакетов за один раз возникнет перегрузка — о таком сценарии мы говорили выше.

Решение, предложенное Джейкобсоном, объединяет линейное и мультипликативное увеличение. При установлении соединения отправитель задает маленькое окно размером не более четырех сегментов. (Изначально исходный размер окна не превышал один сегмент, но впоследствии он был увеличен до четырех исходя из опыта.) Подробнее об этом рассказывается в RFC 3390. Затем отправитель передает в сеть начальное окно. Получение пакетов подтвердится через временной интервал, равный RTT. Каждый раз, когда подтверждение о получении сегмента приходит до срабатывания таймера повторной передачи, отправитель увеличивает окно перегрузки на длину одного сегмента (в байтах). К тому же если сегмент был получен, в сети становится на один сегмент меньше, и каждый подтвержденный сегмент позволяет отправить еще два. Окно перегрузки удваивается на каждом RTT.

Этот алгоритм называется **медленным стартом (slow start)**, однако на самом деле он совсем не медленный — это метод экспоненциального роста (особенно в сравнении с предыдущим алгоритмом, который позволяет отправлять целое окно управления потоком за один раз). Медленный старт показан на илл. 6.44. Во время первого RTT отправитель передает в сеть один пакет (и адресат получает один пакет). На втором RTT передается два пакета, на третьем — четыре.

Медленный старт хорошо работает для широкого диапазона значений скорости и RTT. Чтобы регулировать скорость отправки в зависимости от

сетевого пути, он использует скорость прихода подтверждений. Посмотрим, как подтверждения возвращаются от отправителя к получателю (илл. 6.44). Когда отправитель получает подтверждение, он увеличивает окно перегрузки на единицу и сразу же передает в сеть два пакета. (Один из них соответствует увеличению окна на единицу, а второй передается взамен пакета, доставленного получателю и, таким образом, покинувшего сеть. В каждый момент времени число неподтвержденных пакетов определяется окном перегрузки.) Однако эти два пакета не обязательно придут на хост-получатель с тем же интервалом, с каким они были отправлены. Допустим, отправитель подключен к сети Ethernet мощностью 100 Мбит/с. На отправку каждого 1250-байтного пакета уходит 100 мкс. Поэтому интервал между пакетами может быть небольшим, от 100 мкс. Ситуация меняется, если путь проходит через ADSL-линию мощностью 1 Мбит/с. Теперь для отправки такого же пакета требуется 10 мс. Таким образом, минимальный интервал между пакетами возрастает по меньшей мере в 100 раз. Он так и останется большим, если только в какой-то момент пакеты не окажутся все вместе в одном буфере.




**Илл. 6.44.** Медленный старт с начальным окном перегрузки в один сегмент

На илл. 6.44 описанный эффект можно увидеть на примере интервала прибытия пакетов к получателю. Он сохраняется при отправке подтверждений и, следовательно, при их получении отправителем. Если сетевой путь медленный, подтверждения приходят медленно, если быстрый — быстро. В обоих случаях они прибывают через один RTT. Отправитель должен просто учитывать скорость прихода подтверждений при отправке новых пакетов, — это и делает алгоритм медленного старта.

Поскольку алгоритм медленного старта приводит к экспоненциальному росту, в какой-то момент (скорее рано, чем поздно) в сеть будет слишком быстро отправлено чрезвычайно много пакетов. В результате на маршрутизаторах выстроятся очереди. Когда очереди переполняются, происходит потеря пакетов. В этом случае подтверждение для пакета не приходит вовремя и время ожидания TCP-отправителя истекает. На илл. 6.44 можно увидеть слишком быстрый рост

алгоритма медленного старта. Через три RTT в сети находится четыре пакета. Чтобы добраться до получателя, им требуется время, равное целому RTT. Это значит, что для данного соединения подходит окно перегрузки размером в четыре пакета. Но поскольку получение пакетов подтверждается, алгоритм медленного старта продолжает увеличивать окно перегрузки, достигнув восьми пакетов за один RTT. Независимо от того, сколько пакетов отправлено, только четыре из них успевают дойти до места назначения за один RTT. Это значит, что сетевая шина заполнена. Новые пакеты, попадая в сеть, будут застревать в очередях на маршрутизаторах, так как сеть не может достаточно быстро доставлять их получателю. Вскоре возникнет перегрузка и потеря пакетов.

Чтобы контролировать медленный старт, отправитель хранит в памяти пороговое значение для каждого соединения — **порог медленного старта (slow start threshold)**. Изначально устанавливается произвольное высокое значение, не превышающее размер окна управления потоком, чтобы не ограничивать возможности соединения. Используя алгоритм медленного старта, TCP продолжает увеличивать окно перегрузки, пока не произойдет тайм-аут или размер окна не превысит порог (либо пока не заполнится окно получателя).

При обнаружении потери пакета (например, при тайм-ауте) порог медленного старта устанавливается в половину окна перегрузки, и весь процесс начинается заново. Дело в том, что текущее окно слишком велико и вызвало перегрузку, которую удалось зафиксировать с опозданием. Вдвое меньший размер окна, успешно применявшийся ранее, дает лучший результат: пропускная способность используется довольно эффективно, а потерь нет. В нашем примере на илл. 6.44 увеличение окна до восьми пакетов может вызвать потери, а окно, равное четырем пакетам, подходит хорошо. В итоге устанавливается исходное значение окна перегрузки, и алгоритм медленного старта выполняется с начала.

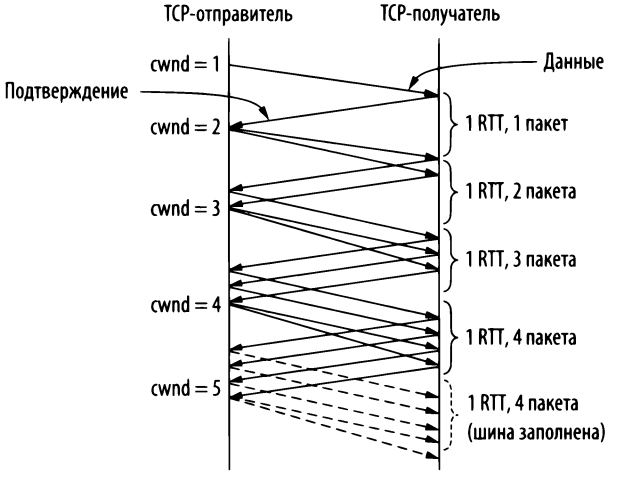
При превышении порога медленного старта TCP переключается на аддитивное увеличение. В этом режиме окно перегрузки возрастает на один сегмент через интервалы времени, равные RTT. Как и при медленном старте, увеличение происходит по мере получения подтверждений о доставке, а не на один сегмент на каждом круге. Пусть *cwnd* — окно перегрузки, а *MSS* — максимальный размер сегмента. Обычно увеличение окна производится с коэффициентом (*MSS* × *MSS*)/*cwnd* для каждого из *cwnd*/*MSS* пакетов, которые можно подтвердить. Этот рост не должен быть быстрым. Вся идея в том, чтобы TCP-соединение максимально долго работало с размером окна, близким к оптимальному, — не слишком маленьким, чтобы пропускная способность не была низкой, и не слишком большим, чтобы не было перегрузок.

Аддитивное увеличение показано на илл. 6.45. Ситуация та же, что и для медленного старта. В конце каждого круга окно перегрузки отправителя увеличивается настолько, что в сеть может быть передан один дополнительный пакет. По сравнению с медленным стартом линейная скорость роста очень низкая. Для маленьких окон разница не слишком существенна, но она станет ощутимой, если, к примеру, понадобится увеличить окно на 100 сегментов.

Для улучшения производительности можно сделать еще кое-что. Недостаток этой схемы — ожидание тайм-аута. Тайм-ауты могут быть относительно долгими, так что они должны быть минимальными. При потере пакета получатель не

подтверждает его, так что номер подтверждения не меняется, а у отправителя нет возможности передавать в сеть новые пакеты, так как его окно перегрузки все еще заполнено. В таком состоянии хост может пробыть довольно долго, пока не сработает таймер и не произойдет повторная передача пакета. На этом этапе медленный старт начинается заново.

Отправитель может быстро выяснить, что один из его пакетов потерян. Новые пакеты, следующие за потерянным, приходят к получателю и вызывают отправку подтверждений, которые имеют один и тот же номер и называются **дубликатами подтверждений (duplicate acknowledgements)**. Каждый раз, когда отправитель получает дубликат подтверждения, есть вероятность, что другой пакет уже пришел к адресату, а потерянный — нет.

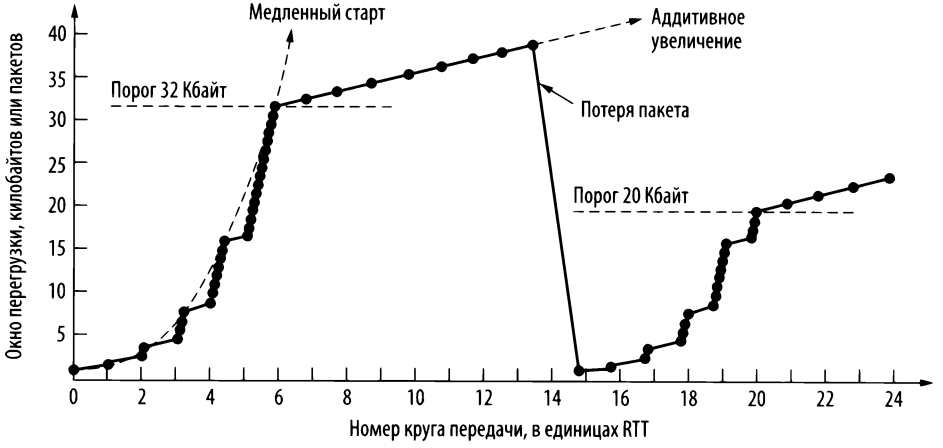



**Илл. 6.45.** Аддитивное увеличение при начальном размере окна в один сегмент

Пакеты могут идти разными путями, поэтому они часто приходят в неправильном порядке. В этом случае дубликаты подтверждений не означают потерю пакетов. Однако в интернете такое случается достаточно редко. Следование пакетов разными маршрутами не слишком нарушает порядок их получения. Поэтому в TCP условно считается, что три дубликата подтверждений сигнализируют о потере пакета. Также по номеру подтверждения можно установить, какой именно пакет потерян. Это следующий по порядку пакет. Его повторную передачу можно выполнить сразу, не дожидаясь срабатывания таймера.

Этот эвристический метод получил название **быстрого повтора передачи (fast retransmisson)**. Когда он происходит, порог медленного старта все равно устанавливается в половину текущего окна перегрузки, как и в случае тайм-аута. Медленный старт можно начать заново, взяв окно размером в один сегмент. Новый пакет будет отправлен через один RTT, за который успеет прийти подтверждение для повторно переданного пакета, а также все данные, переданные в сеть до обнаружения потери пакета.

Существующий на данный момент алгоритм контроля перегрузки проиллюстрирован на илл. 6.46. Эта версия называется TCP Tahoe в честь 4.2BSD Tahoe 1988 года, куда она входила. Максимальный размер сегмента в данном примере равен 1 Кбайт. Сначала окно перегрузки было равно 64 Кбайт, но затем произошел тайм-аут, и порог стал равен 32 Кбайт, а окно перегрузки — 1 Кбайт (передача 0). Окна перегрузки удваивается по экспоненте, пока не достигает порога (32 Кбайт).




**Илл. 6.46.** Медленный старт и последующее аддитивное увеличение в TCP Tahoe

Окно увеличивается каждый раз, когда приходит новое подтверждение, то есть не непрерывно, поэтому мы имеем дискретный ступенчатый график. Однако после превышения порога рост окна приобретает линейный характер. На каждом круге размер окна увеличивается на один сегмент.

Передачи на круге 13 оказываются неудачными (как и положено), и одна из них заканчивается потерей пакета. Отправитель обнаруживает это после получения трех дубликатов подтверждений. Потерянный пакет передается повторно, а пороговое значение устанавливается в половину текущего размера окна (на данный момент это 40 Кбайт, то есть половина составляет 20 Кбайт), и снова запускается медленный старт. Для нового запуска с окном в один сегмент требуется еще один круг. За это время все ранее переданные данные, включая копию потерянного пакета, успевают покинуть сеть. Окно перегрузки снова увеличивается в соответствии с алгоритмом медленного старта до тех пор, пока оно не дойдет до порогового значения в 20 Кбайт. После этого рост окна снова становится линейным. Так будет продолжаться до следующей потери пакета, которая будет выявлена с помощью дубликатов подтверждений или после наступления тайм-аута (или же до заполнения окна получателя).

Версия TCP Tahoe (в которой, кстати, имеются хорошие таймеры повторной передачи) реализует работающий алгоритм контроля перегрузки, который решает проблему отказа сети из-за перегрузки. Однако Джейкобсон

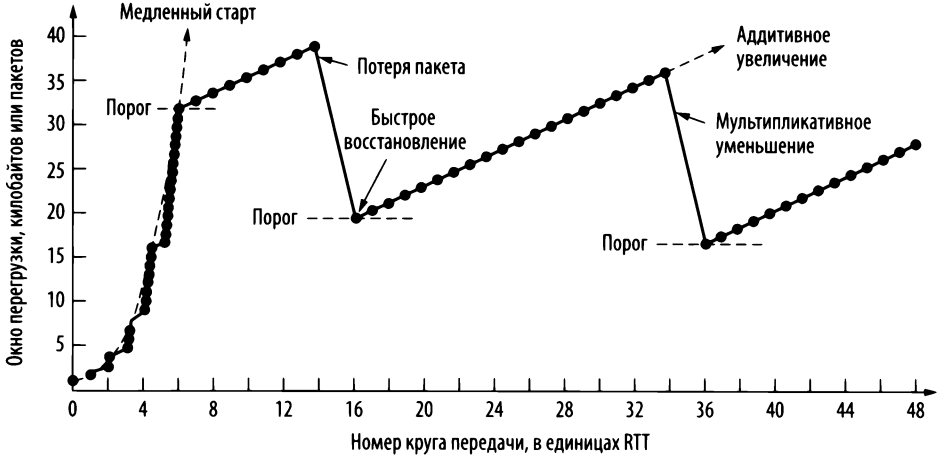
придумал, как добиться большего. Во время быстрой повторной передачи соединение работает с окном слишком большого размера, но скорость прихода подтверждений продолжает учитываться. Каждый раз, когда приходит дубликат подтверждения, велика вероятность того, что еще один пакет покинул сеть. Это позволяет подсчитывать общее количество пакетов в сети и продолжать отправку нового пакета при получении каждого дополнительного дубликата подтверждения.

Эвристический метод реализации этой идеи называется **быстрым восстановлением (fast recovery)**. Это временный режим, направленный на поддержание учета скорости прихода подтверждений в тот момент, когда порогом медленного старта становится текущий размер окна или его половина (во время быстрой повторной передачи). Для этого дубликаты подтверждений (включая те три, которые инициировали быструю повторную передачу) подсчитываются до тех пор, пока число пакетов в сети не снизится до нового порогового значения. На это уходит примерно половина RTT. С этого момента на каждый полученный дубликат подтверждения отправитель может передавать в сеть новый пакет. Через один RTT после быстрой повторной передачи получение потерянного пакета подтвердится. В это время поток дубликатов подтверждений прекратится, и алгоритм выйдет из режима быстрого восстановления. Окно перегрузки будет равняться новому порогу медленного старта и начнет увеличиваться линейно.

Этот метод позволяет TCP избегать медленного старта в большинстве ситуаций, за исключением случаев установления нового соединения и возникновения тайм-аутов. Последнее может произойти, если теряется больше одного пакета, а быстрая повторная передача не помогает. Вместо повтора медленного старта окно перегрузки активного соединения перемещается по **пилообразным (sawtooth)** линиям аддитивного увеличения (на один сегмент за RTT) и мультипликативного уменьшения (в полтора раза за RTT). Это и есть правило AIMD, которое мы с самого начала хотели реализовать.

Такое пилообразное движение показано на илл. 6.47. Данный метод используется в протоколе TCP Reno, названном в честь выпущенного в 1990 году дистрибутива 4.3BSD Reno. По сути, это TCP Tahoe с быстрым восстановлением. После начального медленного старта окно перегрузки растет линейно, пока отправитель не обнаружит потерю пакета, получив нужное количество дубликатов подтверждения. Потерянный пакет передается повторно, и далее алгоритм работает в режиме быстрого восстановления. Этот режим дает возможность продолжать учет скорости прихода подтверждений, пока не придет подтверждение доставки повторно переданного пакета. После этого окно перегрузки принимает значение, равное новому порогу медленного старта, а не единице. Это продолжается неопределенно долго. Почти все время размер окна перегрузки близок к оптимальному значению произведения пропускной способности и времени задержки.

Механизмы выбора размера окна, использующиеся в TCP Reno, составляли основу контроля перегрузки в TCP более двух десятилетий. За это время они претерпели ряд незначительных изменений, например появились новые способы




**Илл. 6.47.** Быстрое восстановление и пилообразный график TCP Reno

выбора начального окна, были устранены различные примеры неоднозначности. Усовершенствования коснулись и механизмов восстановления после потери двух или более пакетов. Так, версия TCP NewReno использует номера частичных подтверждений, полученных после повторной передачи одного потерянного пакета для восстановления другого (Xo; Hoe, 1996) (см. RFC 3782). С середины 1990-х годов стали появляться варианты описанного выше алгоритма, основанные на других законах управления. К примеру, в системе Linux используется CUBIC TCP (Ха и др.; Ha et al., 2008), а в Windows — Compound TCP (Тань и др.; Tan et al., 2006).

Два более серьезных нововведения касаются реализаций TCP. Во-первых, сложность TCP состоит в том, что по потоку дубликатов подтверждений нужно определить, какие пакеты были потеряны, а какие — нет. Номер накопительного подтверждения не содержит такой информации. Простым решением стало применение выборочных подтверждений SACK, в которых может содержаться до трех диапазонов успешно полученных байтов. Эти сведения позволяют отправителю более точно определять, какие пакеты следует передавать повторно, а также следить за еще не доставленными пакетами.

При установлении соединения отправитель и получатель передают друг другу параметр `SACK permitted`, сообщая о возможности работы с выборочными подтверждениями. Когда SACK включены, обмен данными происходит, как показано на илл. 6.48. Получатель использует поле `Acknowledgement number` обычным способом — как накопительное подтверждение последнего по порядку байта, который был принят. Когда пакет 3 приходит вне очереди (так как пакет 2 потерян), получатель отправляет `SACK option` для полученных данных вместе с накопительным подтверждением (дубликатом) для пакета 1. `SACK option` содержит диапазоны байтов, которые были получены сверх числа, заданного накопительным подтверждением. Первый из них — пакет, к которому относится дубликат подтверждения. Следующие диапазоны, если они

есть, относятся к последующим блокам. Обычно применяется не более трех диапазонов. К моменту прихода пакета 6 были использованы два байтовых диапазона, указывающих на получение пакета 6, а также 3 и 4 (в дополнение к тем, которые пришли до пакета 1). Учитывая все принятые `SACK option`, отправитель решает, какие пакеты передать заново. В данном случае неплохо было бы повторить пакеты 2 и 5.

**Илл. 6.48.** Выборочные подтверждения

SACK содержат рекомендательную информацию. Фактическое обнаружение потерь пакетов по дубликатам подтверждений и изменение окна перегрузки происходят так же, как и раньше. Тем не менее SACK упрощают процесс восстановления TCP в ситуациях, когда несколько пакетов теряются примерно в одно и то же время, поскольку TCP-отправитель знает, какие пакеты не дошли до адресата. Сегодня SACK широко распространены. Они описаны в RFC 2883, а контроль управления TCP с использованием SACK — в RFC 3517.

Второе изменение заключается в использовании явных уведомлений о перегрузке ECN в качестве дополнительного сигнала помимо потери пакета. ECN — это механизм IP-уровня, позволяющий сообщать хостам о перегрузке (см. раздел 5.3.2). С их помощью TCP-получатель принимает сигналы перегрузки от IP.

ECN включены для TCP-соединения, если при его установлении отправитель и получатель сообщили друг другу, что они поддерживают такие уведомления, — с помощью битов `ECE` и `CWR`. В заголовке каждого пакета с TCP-сегментом указывается, что этот пакет может передавать ECN. При угрозе перегрузки маршрутизаторы с поддержкой ECN помещают соответствующие сигналы в подходящие для этого пакеты, вместо того чтобы удалять эти пакеты, когда перегрузка действительно происходит.

Если один из входящих пакетов содержит ECN, TCP-получатель узнает об этом и с помощью флага `ECE` (ECN-эхо) сообщает отправителю о перегрузке. Отправитель подтверждает получение этого сигнала с помощью флага `CWR` (Окно перегрузки уменьшено).

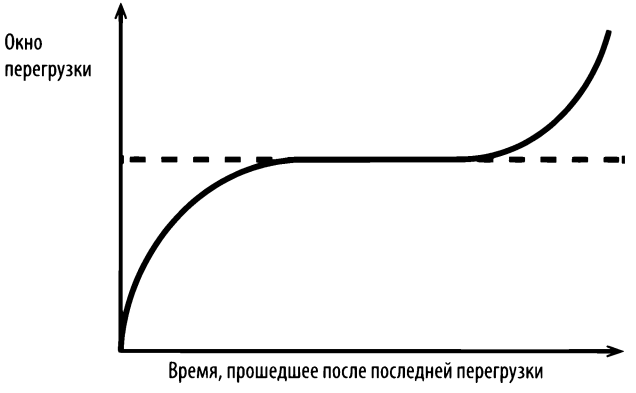
На такие сигналы отправитель реагирует так же, как и на потерю пакетов. Но теперь результат выглядит лучше: перегрузка обнаружена, хотя ни один пакет не пострадал. ECN описаны в RFC 3168. Так как им требуется поддержка как хостов, так и маршрутизаторов, в интернете они не слишком распространены.

Больше информации обо всех методах контроля перегрузки в TCP вы найдете в RFC 5681.

## 6.5.11. CUBIC TCP

Чтобы справиться с растущим значением произведения пропускной способности на задержку, была разработана версия протокола TCP под названием **CUBIC TCP** (Ха и др.; Ha et al., 2008). Как уже упоминалось, сетям с большой величиной этого параметра необходимо множество RTT, чтобы обеспечить максимальную пропускную способность для сквозного пути. Суть CUBIC TCP сводится к тому, что окно перегрузки растет в зависимости от времени с момента прибытия последнего дубликата подтверждения (а не просто на основании поступления подтверждений).

Корректировка окна перегрузки в зависимости от времени также производится несколько иным образом. В отличие от описанного ранее стандартного контроля перегрузки по правилу AIMD, окно растет как кубическая функция; при этом после начального роста оно выходит на «плато», после которого следует период еще более быстрого увеличения. Рост окна перегрузки при использовании протокола CUBIC TCP показан на илл. 6.49. Одно из главных отличий CUBIC от других версий TCP — окно меняется как функция времени, прошедшего после последней перегрузки. Сначала оно быстро увеличивается, затем выходит на «плато» с размером, достигнутым отправителем перед последней перегрузкой, после чего снова растет для обеспечения максимально возможной скорости, пока не возникнет новая перегрузка.




**Илл. 6.49.** Процесс изменения окна перегрузки с течением времени при использовании CUBIC TCP

Протокол CUBIC TCP по умолчанию реализован в ядре Linux версий 2.6.19 и выше, а также в современных версиях Windows.

# 6.6. ТРАНСПОРТНЫЕ ПРОТОКОЛЫ И КОНТРОЛЬ ПЕРЕГРУЗКИ

По мере роста пропускной способности сети некоторые стандартные режимы TCP перестают обеспечивать оптимальную производительность. Протоколам, ориентированным на установление соединения (в том числе TCP) свойственны

высокие издержки на установление соединения, а также проблемы производительности в сетях с большими буферными пространствами. В оставшейся части данного раздела мы обсудим ряд новшеств в области транспортных протоколов, призванных решить эти проблемы.

## 6.6.1. QUIC

Протокол **QUIC (Quick UDP Internet Connections — быстрые интернет-соединения UDP)** был изначально предложен в качестве транспортного протокола, призванного улучшить ряд характеристик TCP относительно пропускной способности и величины задержки. Еще до стандартизации он уже использовался более чем в половине соединений между браузером Chrome и сервисами Google. Несмотря на это, он не поддерживается большинством остальных веб-браузеров.

Как и предполагает его название, протокол QUIC работает поверх UDP и служит главным образом для ускорения прикладных протоколов, в частности веб-протоколов, о которых пойдет речь в главе 7. В ней мы подробно поговорим о том, как QUIC взаимодействует с прикладными протоколами интернета. Как мы вскоре убедимся, интернет требует установления множества параллельных соединений для загрузки отдельной веб-страницы. Поскольку многие из этих соединений представляют собой соединения с общим сервером, установление нового соединения для загрузки каждого отдельного веб-объекта может привести к значительным тратам ресурсов. В результате QUIC стремится мультиплексировать такие соединения в единый UDP-поток, гарантируя при этом, что откладывание пересылки одного веб-объекта не заблокирует передачу остальных.

Базируясь на протоколе UDP, QUIC не обеспечивает надежную передачу автоматически. При потере данных одного потока он может продолжить независимую транспортировку данных других потоков, что в итоге повышает производительность каналов с высокой частотой ошибок передачи. В QUIC также применяется ряд других методов оптимизации для повышения производительности. Среди них вложение информации о шифровании прикладного уровня при установлении транспортного соединения и независимое шифрование каждого пакета (чтобы потеря одного из них не препятствовала декодированию последующих пакетов). Кроме того, QUIC может ускорять передачу обслуживания между сетями (например, переключение с мобильного соединения на Wi-Fi). Он использует идентификатор соединения для сохранения состояния, когда конечные точки меняют сеть.

## 6.6.2. BBR: контроль перегрузки на основе пропускной способности узких мест

Если буферы узких мест слишком велики, алгоритмы контроля перегрузки на основе потерь (включая описанные выше) в итоге заполняют эти буферы, приводя к **излишней сетевой буферизации (bufferbloat)**. Механизм ее возникновения прост: если у сетевых устройств на пути очень крупные буферы, TCP-отправитель с большим окном перегрузки будет передавать данные со

скоростью, намного превышающей пропускную способность сети, прежде чем получит сигнал о потере. Буферы в середине сети заполнятся, что отсрочит наступление перегрузки для отправителей, передающих данные слишком быстро (вместо того, чтобы удалить пакеты). Также это увеличивает сетевую задержку для других отправителей, пакеты которых помещаются в очередь данного буфера (Геттис; Gettys, 2011).

Решить проблему излишней сетевой буферизации можно несколькими способами. Один из них заключается в уменьшении буферов. К сожалению, в этом пришлось бы убедить поставщиков и производителей всех сетевых устройств, от беспроводных точек доступа до магистральных маршрутизаторов. Даже если бы это было возможно, в сетях существует слишком много устаревших устройств, и полагаться лишь на этот подход нельзя. Другое решение — использовать альтернативный метод контроля перегрузки, не основанный на потере данных. Именно таким методом является алгоритм **BBR (Bottleneck Bandwidth and RTT — пропускная способность узких мест и время в пути туда-обратно)**.

Задача BBR — оценить пропускную способность узких мест и величину RTT и выбрать подходящую рабочую точку для отправки данных, исходя из этих оценок. Соответственно, BBR *непрерывно* отслеживает эти параметры. Поскольку TCP уже учитывает RTT, BBR лишь расширяет имеющийся функционал, проверяя, как с течением времени меняется скорость доставки транспортного протокола. Фактически в качестве пропускной способности узких мест принимается максимальная оценка скорости доставки, полученная в течение заданного времени (обычно это 6–10 RTT).

Общая идея алгоритма BBR сводится к следующему. Пока объем данных не превышает произведение пропускной способности на задержку, RTT не увеличивается, поскольку при этом не происходит дополнительной буферизации. Скорость доставки обратно пропорциональна RTT и пропорциональна объему передаваемых пакетов (размеру окна). Когда объем передаваемых пакетов превышает произведение пропускной способности на задержку, величина задержки начинает расти, так как пакеты помещаются в очередь, а скорость доставки перестает увеличиваться. Именно в этот момент требуется алгоритм BBR. На илл. 6.50 показана зависимость RTT и скорости доставки от объема передаваемых данных (еще не подтвержденных). Оптимальной рабочей точкой для BBR является то место, где рост объема трафика ведет к увеличению RTT (но не скорости доставки).

Таким образом, ключевая составляющая алгоритма BBR — это постоянное обновление оценок пропускной способности узких мест и RTT по мере изменения этих параметров. Каждое подтверждение предоставляет актуальную информацию о величине RTT и средней скорости доставки. При этом проверяется, не ограничивается ли скорость доставки приложениями (как часто бывает при использовании запросно-ответных протоколов). Второй составляющей BBR является непосредственно подстройка скорости передачи данных с учетом пропускной способности узких мест. **Подстройка скорости (pacing rate)** — критически важный параметр контроля перегрузки на основе BBR. В устойчивом состоянии используемая алгоритмом скорость отправки

данных просто представляет собой функцию от пропускной способности узких мест и RTT.

**Илл. 6.50.** Рабочая точка алгоритма BBR

Алгоритм BBR минимизирует задержку за счет того, что объем передаваемых данных почти всегда равен произведению пропускной способности на задержку, и эти данные пересылаются со скоростью, в точности равной пропускной способности узких мест. При этом обеспечивается достаточно быстрая адаптация к скорости узких мест.

Компания Google широко использует алгоритм BBR и во внутренней магистральной сети, и во многих своих приложениях. Однако есть один нерешенный вопрос: насколько хорошо контроль перегрузки на базе BBR может соперничать с традиционным контролем перегрузки на базе TCP. Так, в ходе недавно проведенного эксперимента исследователи обнаружили, что BBR-отправитель потреблял 40 % пропускной способности канала при совместном использовании сетевого пути с 16 другими транспортными соединениями, каждое из которых получило менее 4 % от оставшейся пропускной способности (Уэр и др.; Ware et al., 2019). Можно доказать, что BBR часто потребляет фиксированную долю доступной пропускной способности, независимо от количества конкурирующих TCP-потоков. К сожалению, лучший способ проверить, насколько хорошо новые алгоритмы контроля перегрузки обеспечивают равнодоступность, сводится к тому, чтобы просто применить их и посмотреть, что получится. По-видимому, предстоит значительная работа по обеспечению нормального взаимодействия BBR с TCP-трафиком в интернете.

### 6.6.3. Будущее протокола TCP

TCP — «рабочая лошадка» интернета — используется многими приложениями; с течением времени разработчики видоизменяют и дополняют этот протокол, стараясь добиться высокой производительности в разнообразных сетях. На сегодняшний день существует много версий, каждая из которых добавляет что-то новое к классическим алгоритмам, описанным здесь; особенно это касается контроля перегрузки и устойчивости к сетевым атакам. По всей вероятности, TCP будет развиваться параллельно с интернетом. Здесь мы рассмотрим два частных вопроса.

Во-первых, TCP не решает проблем транспортной семантики, актуальных для многих приложений. К примеру, некоторые из них хотят, чтобы границы передаваемых ими сообщений или записей сохранялись. Другие приложения нацелены на работу с группами взаимосвязанных сообщений (например, веб-браузер, загружающий несколько объектов с одного сервера). Некоторым приложениям нужны дополнительные возможности управления сетевыми путями. TCP со своим стандартным интерфейсом сокетов не в состоянии выполнить эти требования. В результате каждое приложение вынуждено самостоятельно решать проблемы, которые не по силам TCP. Это привело к созданию новых протоколов со слегка измененным интерфейсом, таких как SCTP и SST. Но каждый раз, когда кто-то предлагает изменить проверенный механизм, образуется два противоборствующих лагеря: «Пользователи хотят новых возможностей» и «Если ничего не сломано, не надо ничего чинить».

## 6.7. ВОПРОСЫ ПРОИЗВОДИТЕЛЬНОСТИ

Вопросы производительности играют критически важную роль в компьютерных сетях. Когда сотни или тысячи компьютеров соединены вместе, их взаимодействие становится очень сложным и может привести к непредсказуемым последствиям, например к низкой производительности, причины которой довольно трудно определить. В следующих разделах мы обсудим многие вопросы, связанные с производительностью сетей, определим круг существующих проблем и обсудим методы их решения.

К сожалению, понимание производительности сетей — это скорее искусство, чем наука. Теоретическая база, допускающая хоть какое-то практическое применение, крайне скудна. Лучшее, что мы можем сделать, это представить практические методы, полученные в результате долгих экспериментов, и привести несколько реальных примеров. Мы отложили эту дискуссию до того момента, когда уже изучен транспортный уровень, поскольку производительность, доступная приложениям, зависит от общей производительности транспортного, сетевого и канального уровней. К тому же мы сможем приводить TCP в качестве примера.

В следующих разделах мы обсудим основные вопросы производительности сети:

1. Проблемы производительности.
2. Оценка производительности сети.

3. Оценка пропускной способности сетей доступа.
4. Оценка QoE.
5. Проектирование хостов для быстрых сетей.
6. Быстрая обработка сегментов.
7. Сжатие заголовка.
8. Протоколы для протяженных сетей с высокой пропускной способностью.

Таким образом, мы рассмотрим производительность с точки зрения операций, выполняемых на хостах и при перемещении данных по сети, а также с точки зрения размера и быстродействия сети.

## 6.7.1. Проблемы производительности компьютерных сетей

Некоторые проблемы производительности вызваны временным отсутствием свободных ресурсов. Если на маршрутизатор внезапно прибывает больше трафика, чем он способен обработать, возникает перегрузка, и производительность резко падает. Мы подробно изучали перегрузку в этой и предыдущей главах.

Производительность также снижается, если возникает структурный дисбаланс ресурсов. Например, если гигабитная линия присоединена к компьютеру с низкой производительностью, то слабый хост не сможет достаточно быстро обрабатывать приходящие пакеты, что приведет к потере некоторых из них. Эти пакеты рано или поздно будут переданы повторно, что приведет к увеличению задержек, неэффективному использованию пропускной способности и снижению общей производительности.

Перегрузка может быть синхронной. Например, если сегмент содержит неверный параметр (например, номер порта назначения), как правило, получатель заботливо отсылает обратно сообщение об ошибке. Теперь представьте, что случится, если неверный сегмент будет разослан широковещательным способом на 10 000 компьютеров. Каждый из них может ответить сообщением об ошибке. Образовавшийся в результате **широковещательный шторм (broadcast storm)** может парализовать сеть.

UDP часто сталкивался с подобной проблемой, но затем протокол ICMP был изменен таким образом, чтобы хосты воздерживались от отправки сообщений об ошибке в ответ на широковещательные сегменты UDP. Беспроводным сетям особенно важно не отвечать на непроверенные широковещательные пакеты, поскольку их пропускная способность ограниченна.

Синхронная перегрузка может возникнуть после сбоя электроснабжения. Когда питание восстанавливается, все компьютеры начинают перезагружаться. Типичная последовательность перезапуска может потребовать обращения к какому-нибудь DHCP-серверу, чтобы узнать свой истинный адрес, а затем к файловому серверу, чтобы получить копию операционной системы. Если сотни компьютеров центра обработки данных обратятся к серверу одновременно, он, скорее всего, не справится с нагрузкой.

Даже в отсутствие синхронной перегрузки и при достаточном количестве ресурсов производительность может снижаться из-за неправильных системных

настроек. Допустим, у компьютера мощный процессор и много памяти, но недостаточно ресурса выделено под буфер. В таком случае управление потоком данных замедлит получение сегментов и ограничит производительность. Это было проблемой для многих TCP-соединений, поскольку интернет становился быстрее, а окно управления потоком оставалось фиксированным (64 Кбайт).

Также на производительность могут повлиять ошибочные значения таймеров ожидания. Когда посылается сегмент, обычно включается таймер, на случай, если он потеряется. Выбор слишком короткого интервала ожидания подтверждения приведет к излишним повторным передачам сегментов. Если же его сделать слишком большим, это увеличит задержку в случае потери сегмента. К настраиваемым параметрам также относится интервал ожидания попутного сегмента данных для отправки подтверждения и количество попыток повторной передачи.

Еще одна проблема касается приложений, работающих в реальном времени (например, воспроизводящих аудио и видео), — джиттер. В этом случае для хорошей производительности недостаточно оптимальной средней пропускной способности. Таким приложениям нужна низкая задержка. Для этого требуется эффективное распределение нагрузки на сеть, а также поддержка QoS на канальном и сетевом уровнях.

## 6.7.2. Оценка производительности сети

Производительность сетей оценивают все — и сетевые операторы, и пользователи. Чаще всего измеряется пропускная способность (или просто «скорость») сетей доступа. Многие интернет-пользователи проверяют производительность своих сетей доступа с помощью таких инструментов, как Speedtest (www.speedtest.net). Традиционная проверка сводилась к попытке отправить в сеть как можно больше трафика с максимально возможной скоростью (тем самым полностью заполнив канал связи). Но по мере увеличения скорости сетей задача оценки скорости канала доступа становится все более сложной. Теперь для полного заполнения канала требуется гораздо больше данных, а узкие места на тестируемом пути между клиентом и сервером могут смещаться в другие области сети. Пожалуй, важнее всего то, что скорость становится менее актуальным параметром производительности сети. На передний план выходят качество пользовательского опыта и производительность приложения. Как следствие, методы оценки производительности сетей продолжают развиваться, особенно в отношении гигабитных сетей доступа.

## 6.7.3. Оценка пропускной способности сетей доступа

Традиционный подход к оценке пропускной способности сети состоит в следующем. По сетевому пути передается максимально допустимое количество данных в течение заданного периода времени. Затем объем отправленных данных делится на время, затраченное на их передачу, и выводится среднее значение пропускной способности. Будучи простым и пригодным для большинства случаев, данный

подход все же имеет ряд недостатков. Главным минусом является то, что одно TCP-соединение часто не может исчерпать емкость сетевого канала, особенно с учетом того, что скорость каналов доступа продолжает расти. Кроме того, когда тест охватывает начальную стадию процесса передачи, он может отражать скорость до перехода в установившийся режим (например, при медленном старте TCP). В итоге это может привести к занижению пропускной способности. Наконец, проверки на стороне клиента (такие, как speedtest.net или любой другой тест пропускной способности, который можно запустить с клиентского устройства) очень часто выявляют другие проблемы производительности, помимо ограничений сетей доступа (например, ограничения канала радиосвязи устройства или беспроводной сети).

Эти проблемы проявляются все более остро по мере того, как скорость сетей доступа начинает преодолевать гигабитный рубеж. Для их решения был разработан ряд практических рекомендаций по оценке пропускной способности сетей доступа (Фимстер и др.; Feamster et al., 2020). Первая рекомендация состоит в том, чтобы использовать несколько параллельных TCP-подключений для заполнения емкости канала доступа. Проверка первых версий тестов скорости показала, что для заполнения емкости сети обычно достаточно четырех TCP-соединений (Сундаресан и др.; Sundaresan et al., 2011). В большинстве современных клиентских инструментов, включая Speedtest, а также тест пропускной способности, используемый Федеральной комиссией по связи в США, для оценки емкости сети применяется минимум четыре параллельных соединения. Некоторые из этих инструментов даже позволяют увеличивать число параллельных подключений для проверки каналов с высокой емкостью.

Пропускная способность канала доступа интернет-провайдера начинает превышать емкость домашней сети (и других компонентов сквозного пути), поэтому становится все более актуальной вторая практическая рекомендация. Она сводится к оценке пропускной способности сети доступа непосредственно из домашнего маршрутизатора. Это минимизирует риск искажения результатов теста из-за внешних ограничивающих факторов (например, ограничений клиентского устройства или беспроводной сети пользователя).

По мере дальнейшего роста скоростей могут появиться дополнительные рекомендации, к примеру оценивать скорость соединения сразу с несколькими хостами в интернете, используя один канал доступа. Такой подход особенно актуален, когда проблемы на стороне сервера могут приводить к появлению в сети новых узких мест. Постоянный рост скорости также породил повышенный интерес к разработке так называемых пассивных тестов пропускной способности, которые не вводят в сеть большие объемы дополнительного трафика, а просто отслеживают движение данных по сети и пытаются оценить ее емкость просто на основе наблюдений. На данный момент еще не существует надежного пассивного метода оценки пропускной способности сетей доступа, но со временем этот подход, вероятно, позволит оценивать пропускную способность узких мест примерно так же, как алгоритм BBR оценивает величину задержки и скорость доставки.

### 6.7.4. Оценка QoE

Рано или поздно с увеличением скорости сетей самым важным показателем производительности, вероятно, станет не их скорость (то есть пропускная способность), а соответствие работы приложений ожиданиям пользователя. Как правило, в случае видео пользовательский опыт в какой-то момент перестает зависеть от пропускной способности (Рамачандран и др.; Ramachandran et al., 2019). При потоковой видеопередаче он в основном определяется тем, как быстро начинается воспроизведение (то есть насколько велика задержка запуска), требует ли видео повторной буферизации и каково его разрешение. При этом если скорость превышает 50 Мбит/с, все эти факторы главным образом зависят не от пропускной способности канала доступа, а от других свойств сети (задержка, джиттер и т. д.).

Соответственно, современные средства оценки производительности сети выходят за рамки простого тестирования скорости. Их задача — оценить качество пользовательского опыта, QoE; обычно это происходит путем пассивного наблюдения за сетевым трафиком. Подобные средства оценки находят все более широкое распространение в случае потокового видео (Ахмед и др.; Ahmed et al., 2017; Кришнамурти и др; Krishnamoorthy et al., 2017; Мангла и др.; Mangla et al., 2018; Бронзино и др; Bronzino et al., 2020). Проблемой пока является распространение такой оптимизации на большее число видеосервисов, а в конечном итоге и на более широкий круг приложений (включая игры, виртуальную реальность и т. д.).

Очевидно, что QoE является критерием того, насколько человек доволен предоставляемым ему сервисом. В конечном счете этот показатель носит субъективный характер и часто требует обратной связи от пользователя (это могут быть опросы в реальном времени или иные методы). Интернет-провайдеры попрежнему заинтересованы в механизмах, способных вывести логическим путем или спрогнозировать QoE и степень вовлеченности пользователей на основе параметров, которые измеряются напрямую (пропускная способность приложения, частота потери пакетов, интервал между поступлениями пакетов и т. д.).

Методы автоматической оценки QoE на основе пассивного измерения параметров сетевого трафика появятся не скоро, однако эта область является благодатной почвой для исследований на стыке машинного обучения и сетевых технологий. Рано или поздно приложения могут выйти за рамки сетевых технологий, и транспортные протоколы (а также сетевые операторы) тоже будут оптимизировать использование ресурсов для пользователей, нуждающихся в более высоком качестве. Например, человека, который запустил потоковое видео и ушел в другую комнату, гораздо меньше волнует качество потока по сравнению с тем, кто поглощен просмотром фильма. Конечно, при этом возникает вопрос, как определить, просматривает ли пользователь видео со всем вниманием или вышел на кухню за стаканом воды, не удосужившись нажать на паузу.

## 6.7.5. Проектирование хостов для быстрых сетей

Измерение и настройка могут значительно улучшить производительность, но это не заменит качественного дизайна сети. Плохо спроектированная сеть может быть усовершенствована только до определенного уровня. Для дальнейшего увеличения производительности ее придется переделать.

В данном разделе мы рассмотрим несколько эмпирических правил, относящихся к программной реализации сетевых протоколов на хостах. Опыт показывает, что именно эти правила часто являются ограничивающим фактором производительности, даже если сеть сама по себе работает быстро. Так происходит по двум причинам. Во-первых, сетевые карты и маршрутизаторы разрабатываются таким образом, чтобы работать со скоростью физического соединения. Это значит, что они могут обрабатывать пакеты с той скоростью, с которой пакеты поступают. Во-вторых, нас интересует производительность, доступная для приложений. А она вычисляется не на основе мощности канала, а исходя из пропускной способности и задержки, которые являются результатом работы сетевого и транспортного уровней.

Уменьшение накладных расходов из-за программного обеспечения улучшает производительность за счет повышения пропускной способности и снижения задержки. Оно также позволяет снизить затраты энергии, необходимые для передачи данных по сети, что актуально для портативных компьютеров. Большинство этих идей известны разработчикам сетей уже много лет. Впервые они были четко сформулированы Могулом (Mogul, 1993), и мы во многом опираемся на его работу. Другой источник по этой теме — книга Меткалфа (Metcalfe, 1993).

### *Скорость центрального процессора важнее скорости сети*

Многолетний опыт показывает, что почти во всех быстрых сетях накладные расходы операционной системы и протокола составляют основное время задержки. Например, теоретически минимальное время удаленного вызова процедур в сети Ethernet мощностью 1 Гбит/с составляет 1 мкс, что соответствует минимальному запросу (64 байта), на который приходит минимальный (64-байтный) ответ. На практике значительным достижением считается даже небольшое снижение накладных расходов программного обеспечения при вызове удаленной процедуры (что происходит редко).

Аналогично при работе с гигабитной сетью основная задача состоит в достаточно быстрой передаче битов из буфера пользователя на линию, а также в том, чтобы получатель смог обработать их с той скоростью, с какой они приходят. Если удвоить производительность процессора и быстродействие памяти, это нередко может дать практически двойную пропускную способность. При этом удвоение пропускной способности линии часто не дает никакого эффекта, если узким местом являются хосты.

### Уменьшайте число пакетов, чтобы снизить накладные расходы

Каждый сегмент содержит определенный объем накладных расходов (например, заголовок) и некоторое количество данных (например, пользовательские данные). Оба этих компонента требуют пропускной способности. Также каждому из них требуется обработка (например, обработка заголовка и вычисление контрольной суммы). При отправке 1 млн байт побайтовые затраты времени процессора на обработку не зависят от размера сегмента. Однако при использовании 128-байтных сегментов затраты на обработку заголовков будут в 32 раза выше, чем для сегментов размером 4 Кбайт. И эти затраты растут очень быстро.

Накладные расходы более низких уровней усиливают этот эффект. Прибытие каждого пакета вызывает новое прерывание, если хост работает в активном режиме. В современных конвейерных процессорах каждое прерывание нарушает работу процессорного конвейера, снижает эффективность работы кэша, требует изменения контекста управления памятью, аннулирует таблицу предсказания переходов и требует сохранения в стеке значительного числа регистров процессора. Таким образом, уменьшение количества отправляемых сегментов на $n$ дает снижение числа прерываний и накладных расходов в целом в $n$ раз.

Можно сказать, что компьютер, как и человек, плохо справляется с многозадачностью. Это объясняет стремление отправлять MTU-пакеты максимального размера и обходиться без фрагментации. Алгоритм Нейгла и метод Кларка также являются попытками избежать отправки маленьких пакетов.

### Минимизируйте число операций с данными

Самый простой способ реализовать многоуровневый стек протоколов — использовать один модуль для каждого уровня. К сожалению, это приводит к многократному копированию данных (или, по крайней мере, многократному доступу к ним). К примеру, после того как пакет принимается сетевой картой, он обычно копируется в системный буфер ядра. Затем он должен быть обработан на сетевом и на транспортном уровнях. Поэтому он поочередно копируется в буферы этих уровней и, наконец, передается получающему приложению. Часто входящий пакет копируется три или четыре раза, прежде чем содержащийся в нем сегмент доставляется по назначению.

Такое копирование может сильно снизить производительность, поскольку операции с памятью обычно в десятки раз медленнее, чем операции с использованием только внутренних регистров. К примеру, если 20 % инструкций действительно связаны с обращением к памяти (то есть данных нет в кэше), — а это вполне вероятная цифра при обработке входящих пакетов, — среднее время выполнения инструкции снизится в 2,8 раза ($0{,}8 \times 1 + 0{,}2 \times 10$). Аппаратные улучшения здесь не помогут — проблема в слишком большом числе операций копирования, выполняемых операционной системой.

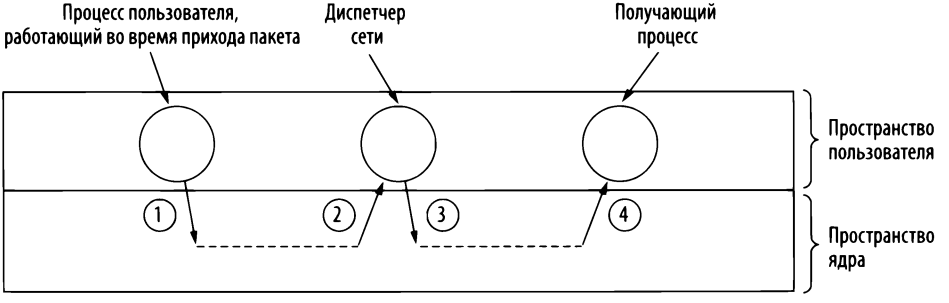
Продуманная операционная система постарается уменьшить число операций копирования, объединяя процессы обработки на разных уровнях. К примеру, TCP и IP обычно работают вместе («TCP/IP»), поэтому когда обработка переходит от сетевого уровня к транспортному, копировать пользовательские данные

пакета не нужно. Еще один популярный прием состоит в том, чтобы выполнять несколько операций на уровне за один проход. Например, контрольные суммы часто вычисляются во время копирования данных (когда это действительно необходимо), и новое значение добавляется в конец.

### Минимизируйте число переключений контекста

Схожее правило заключается в том, чтобы по возможности избегать переключений контекста (например, из режима ядра в режим пользователя), поскольку они объединяют в себе негативные свойства прерываний и копирования. Именно во избежание этих затрат транспортные протоколы часто реализуются в ядре. Как и число пакетов, количество переключений контекста можно снизить с помощью библиотечной процедуры, передающей данные во внутренний буфер до тех пор, пока их не наберется достаточное количество. Аналогично на стороне получателя небольшие сегменты следует собирать вместе и передавать пользователю за один раз, минимизируя число переключений контекста.

В идеале входящий пакет вызывает одно переключение контекста из текущего пользовательского процесса в режим ядра, а затем еще одно — при передаче управления принимающему процессу и предоставлении ему прибывших данных. К сожалению, во многих операционных системах происходит еще одно переключение. Например, если диспетчер сети работает в виде отдельного процесса в пространстве пользователя, поступление пакета вызывает передачу управления от процесса пользователя ядру, затем от ядра диспетчеру сети, затем снова ядру и, наконец, от ядра получающему процессу. Эта последовательность переключений контекста показана на илл. 6.51. Все эти переключения для каждого пакета сильно расходуют время центрального процессора и существенно снижают производительность сети.




**Илл. 6.51.** Четыре переключения контекста для обработки одного пакета в случае, если сетевой менеджер находится в пространстве пользователя

### Лучше избегать перегрузки, чем восстанавливаться после нее

Старая пословица, гласящая, что профилактика лучше лечения, справедлива и в деле борьбы с перегрузками в сетях. Когда сеть перегружена, пакеты теряются, пропускная способность растрачивается впустую, увеличивается задержка и т. п.

Все эти издержки нежелательны, а процесс восстановления после перегрузки требует времени и терпения. Гораздо более эффективной стратегией является предотвращение перегрузки, напоминающее прививку от болезни, — неприятно, зато избавляет от более крупных неприятностей.

### *Избегайте тайм-аутов*

Таймеры в сетях необходимы, но их следует применять умеренно и стремиться минимизировать количество тайм-аутов. Когда срабатывает таймер, обычно повторяется какое-то действие. Если его повтор необходим, так и следует поступить, однако повторение действия без особой необходимости является расточительным.

Чтобы избегать излишней работы, следует устанавливать период ожидания с небольшим запасом. Таймер, срабатывающий слишком поздно, несколько увеличивает задержку в случае (маловероятном) потери сегмента. Преждевременно срабатывающий таймер впустую тратит время процессора и пропускную способность, а также напрасно увеличивает нагрузку на, возможно, десятки маршрутизаторов.

## 6.7.6. Быстрая обработка сегментов

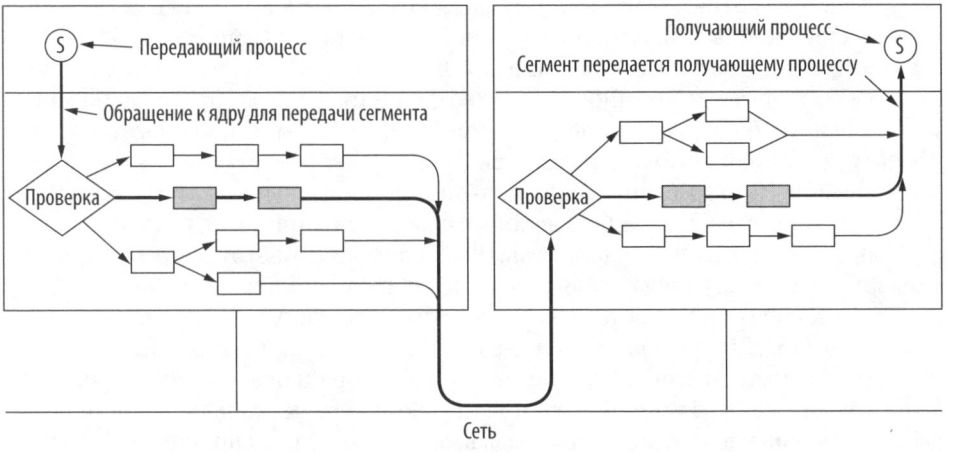
Теперь, когда мы поговорили об общих правилах, рассмотрим несколько методов, позволяющих ускорить обработку сегментов. Дополнительные сведения по этой теме можно найти в следующих источниках: Кларк и др. (Clark et al., 1989); Чейз и др. (Chase et al., 2001).

Накладные расходы по обработке сегментов состоят из двух компонентов: затраты на сегмент и на каждый байт. Нужно действовать сразу на обоих направлениях. Ключ к быстрой обработке сегментов — отделить стандартный, успешный случай (одностороннюю передачу данных) и разобраться с ним. Многие протоколы придают особое значение действиям в нештатной ситуации (например, при потере пакета). Но для быстрой работы протокола разработчик должен пытаться минимизировать время обработки данных в нормальном режиме. Снижение времени обработки при возникновении ошибок является второстепенной задачей.

Для перехода в состояние *ESTABLISHED* требуется передача последовательности специальных сегментов, но как только оно достигнуто, обработка сегментов не вызывает затруднений (пока одна из сторон не начнет закрывать соединение). Допустим, отправитель находится в состоянии *ESTABLISHED* и у него есть данные для передачи. Для простоты мы предположим, что транспортная подсистема расположена в ядре, хотя тот же принцип действует, если это процесс в пространстве пользователя или библиотека в передающем процессе. На илл. 6.52 передающий процесс вклинивается в процессы ядра, выполняя примитив `SEND`. Прежде всего транспортная подсистема проверяет, является ли этот случай нормальным: установлено состояние *ESTABLISHED*, ни одна сторона не пытается закрыть соединение, передается стандартный полный сегмент (без флага `URGENT`) и у получателя достаточный размер окна. Если все эти условия

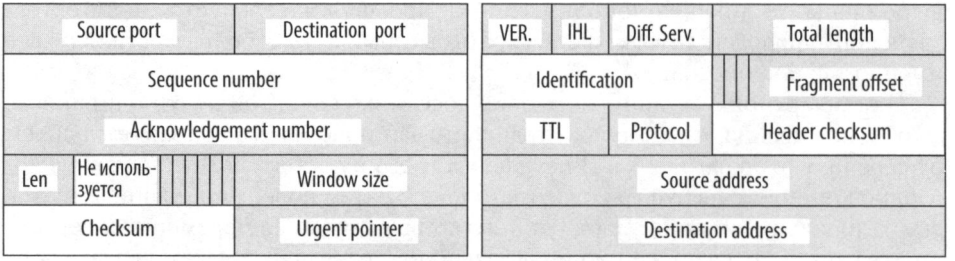
выполнены, то никаких дополнительных проверок не требуется, и алгоритм транспортной подсистемы может выбрать быстрый путь. В большинстве случаев именно так и происходит.

В стандартной ситуации заголовки соседних сегментов почти одинаковы. Чтобы извлечь из этого пользу, транспортная подсистема сохраняет в своем буфере прототип заголовка. В начале быстрого пути он как можно скорее пословно копируется в буфер нового заголовка. Затем поверх перезаписываются все отличающиеся поля. Часто они легко выводятся из переменных состояния — например, следующий порядковый номер сегмента. Затем указатель на заполненный заголовок сегмента и указатель на данные пользователя передаются сетевому уровню. Здесь можно применить ту же стратегию (на илл. 6.52 этого нет). Наконец, сетевой уровень передает полученный в результате пакет канальному уровню для отправки.




**Илл. 6.52.** Быстрый путь от отправителя к получателю обозначен жирной линией. Серым цветом выделены прямоугольники, обозначающие шаги обработки вдоль этого пути

Чтобы увидеть, как работает этот принцип на практике, рассмотрим случай TCP/IP. На илл. 6.53 (а) изображен TCP-заголовок. Поля, одинаковые для заголовков последующих сегментов в однонаправленном потоке, выделены серым цветом. Все, что нужно сделать передающей транспортной подсистеме, — это скопировать пять слов заголовка-прототипа в выходной буфер, заполнить поле порядкового номера (скопировав одно слово), сосчитать контрольную сумму и увеличить на единицу значение переменной, хранящей текущий порядковый номер. Затем она передает заголовок и данные специальной IP-процедуре, предназначенной для отправки стандартного максимального сегмента. Затем IP-процедура копирует свой заголовок-прототип из пяти слов (илл. 6.53 (б)) в буфер, заполняет поле `Identification` и вычисляет контрольную сумму заголовка. Теперь пакет готов к передаче.




**Илл. 6.53.** (а) TCP-заголовок. (б) IP-заголовок. Серым цветом выделены поля, взятые из прототипа без изменений

Теперь рассмотрим быстрый путь обработки пакета получающей стороной на илл. 6.52. Первый шаг состоит в поиске записи соединения для входящего сегмента. В TCP запись соединения может храниться в хеш-таблице, ключом к которой является какая-нибудь простая функция двух IP-адресов и двух портов. После ее обнаружения следует проверить адреса и номера портов, чтобы убедиться, что найдена нужная запись.

Процесс поиска требуемой записи можно дополнительно ускорить, если установить указатель на последнюю использовавшуюся запись и сначала проверить ее. Кларк и его соавторы (Clark et al.,1989) исследовали этот вопрос и пришли к выводу, что в этом случае доля успешных обращений превысит 90 %.

Затем сегмент проверяется на соответствие стандарту: соединение в состоянии *ESTABLISHED*, ни одна сторона не пытается его разорвать, сегмент является полным, специальные флаги не установлены, и порядковый номер именно тот, который ожидался. Такие проверки включают всего несколько условий. Если все эти условия удовлетворяются, вызывается специальная процедура быстрого пути TCP-уровня.

Процедура быстрого пути обновляет запись соединения и копирует данные для пользователя. Во время копирования она подсчитывает контрольную сумму, что уменьшает количество циклов обработки. Если контрольная сумма верна, запись соединения обновляется и отправляется подтверждение. Метод, реализованный в виде отдельной процедуры, которая сначала быстро проверяет заголовок на предмет его соответствия ожиданиям, называется **прогнозированием заголовков (header prediction)**. Он применяется в большинстве реализаций TCP. Использование этого метода оптимизации вместе с остальными, описанными в данном разделе, позволяет протоколу TCP достичь 90 % от скорости копирования в локальной памяти, при условии, что сама сеть достаточно быстрая.

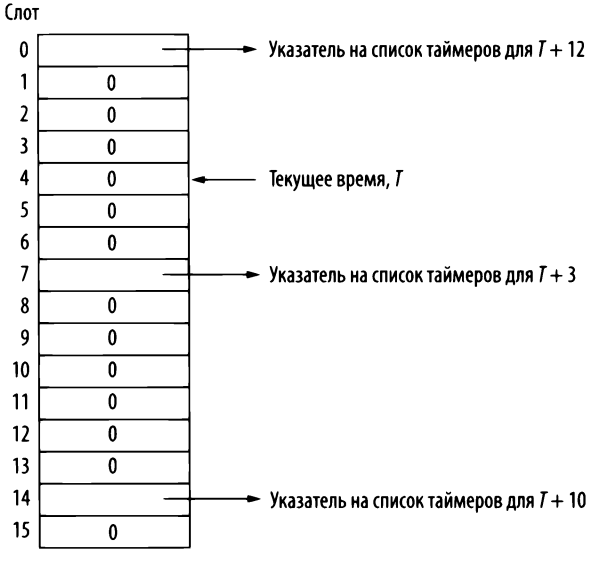
Еще две области, в которых возможны существенные улучшения производительности, — управление буферами и таймерами. Как уже было сказано, в управлении буферами необходимо избегать излишнего копирования. В управлении таймерами следует учитывать, что они почти никогда не срабатывают. Они предназначены для обработки нестандартного случая потерь сегментов,

но большинство сегментов и их подтверждений приходят успешно, и поэтому истечение периода ожидания является редким событием.

В программе таймеры обычно реализуются в виде связанного списка таймеров, отсортированного по времени срабатывания. Начальная запись содержит счетчик минимальных интервалов времени (тактовых импульсов — ticks), оставшихся до истечения периода ожидания. В каждой последующей записи находится счетчик, указывающий, сколько тактовых импульсов остается с учетом предыдущей записи. Например, если таймеры сработают через 3, 10 и 12 импульсов, счетчики списка будут содержать значения 3, 7 и 2 соответственно.

При каждом импульсе таймера счетчик начальной записи уменьшается на единицу. Когда он достигает нуля, обрабатывается связанное с этим таймером событие, а начальной записью становится следующий элемент списка. Значение счетчика можно оставить без изменений. В такой схеме добавление и удаление таймера требуют затрат ресурсов, при этом время выполнения операции пропорционально длине списка.

Если максимальное значение таймера ограничено и известно заранее, можно применить более эффективный метод — использовать массив под названием **циклическое расписание (timing wheel)** (илл. 6.54). Каждый слот соответствует одному тактовому импульсу. Текущее время, показанное на рисунке, — $T = 4$. Таймеры должны сработать через 3, 10 и 12 импульсов. Если новый таймер должен сработать через 7 импульсов, требуется сделать запись в слоте 11. Аналогично, если нужно отключить таймер, установленный на $T + 10$, нужно проверить список, начинающийся в слоте 14, и удалить соответствующую запись. Следует отметить, что массив на илл. 6.54 не может поддерживать таймеры за пределами $T + 15$.




**Илл. 6.54.** Циклическое расписание

Когда работает таймер, указатель текущего времени перемещается по циклическому расписанию вперед на один слот. Если запись, на которую он указывает, не нулевая, обрабатываются все таймеры этого списка. Многочисленные варианты этой концепции обсуждаются у Варгезе и Лаука (Varghese and Lauck, 1987).

## 6.7.7. Сжатие заголовков

Перейдем к вопросу о производительности беспроводных сетей и сетей других типов, где пропускная способность ограниченна. При уменьшении издержек, возникающих из-за программного обеспечения, портативные компьютеры будут работать эффективнее, но это не поможет улучшить производительность в тех случаях, когда проблемой являются сетевые каналы.

Чтобы эффективно использовать пропускную способность, заголовок протокола и пользовательские данные должны занимать как можно меньше битов. В случае пользовательских данных этого можно достичь путем компактного кодирования информации: к примеру, изображения лучше передавать в формате JPEG, а не в растровых форматах, а документы — в форматах со сжатием, таких как PDF. Также для этого необходимы механизмы кэширования прикладного уровня (например, кэш веб-страниц), уменьшающие число передач.

А что делать с заголовками протоколов? В беспроводных сетях на канальном уровне заголовки обычно занимают мало места, поскольку изначально рассчитаны на низкую пропускную способность. В сетях, ориентированных на установление соединения, пакеты имеют короткие идентификаторы соединения вместо длинных адресов. Но протоколы более высоких уровней (IP, TCP или UDP) универсальны для всех типов сетей, поэтому в них не используются компактные заголовки. На самом деле потоковая обработка, уменьшающая затраты на программное обеспечение, часто приводит к еще большему увеличению длины заголовка (например, в IPv6 используются более разреженные заголовки, чем в IPv4).

Заголовки более высоких уровней могут сильно влиять на производительность. Например, рассмотрим процесс передачи VoIP-данных через IP, UDP и RTP. Этим протоколам требуется заголовок длиной 40 байт (20 байт для IPv4, 8 — для UDP, 12 — для RTP). В случае IPv6 ситуация еще хуже: 60 байт, включая 40-байтный заголовок IPv6. Эти заголовки могут становиться еще длиннее, потребляя более половины пропускной способности.

Чтобы уменьшить пропускную способность, потребляемую заголовками высоких уровней, применяется **сжатие заголовков (header compression)**. При этом используются не универсальные, а специально разработанные методы. Дело в том, что по отдельности заголовки сжимаются плохо (поскольку они и так короткие), а для декомпрессии требуется, чтобы исходные данные пришли на место назначения. Последнее может и не произойти из-за потери пакетов.

Сжатие заголовков можно эффективно реализовать, если учитывать формат протокола. Одна из первых схем была разработана Ван Джейкобсоном (1990)

и применялась для сжатия TCP/IP-заголовков при передаче по медленным последовательным каналам. Она позволяет уменьшить обычный 40-байтный TCP/IP-заголовок в среднем до 3 байт. Если посмотреть на илл. 6.53, можно догадаться, на чем основан этот метод. Дело в том, что многие поля заголовка не меняются от пакета к пакету. К примеру, совсем не обязательно каждый раз передавать одно и то же значение времени жизни пакета или одинаковый номер TCP-порта. Эти поля можно пропустить при передаче пакета и заполнить при получении.

Остальные поля меняются по предсказуемому сценарию. К примеру, если нет потерь, порядковые номера TCP увеличиваются по мере отправки данных. Следовательно, получатель может предсказать наиболее вероятное значение. Номер необходимо указывать, только если он отличается от ожидаемого. Но даже тогда можно передавать разность между данным номером и предыдущим (как в случае с номером подтверждения, который увеличивается, когда новые данные приходят в обратном направлении).

С помощью сжатия заголовков можно добиться, чтобы в протоколах высоких уровней заголовки были простыми, а при передаче пакета по каналам с низкой пропускной способностью применялось компактное кодирование. **Надежное сжатие заголовков (Robust Header Compression, ROHC)** — современный вариант сжатия, который определен в RFC 5795 как фреймворк. По замыслу разработчиков, он не реагирует на потерю пакетов в беспроводных каналах. Для каждого набора протоколов, например IP/UDP/RTP, существует отдельный профиль. Сжатые заголовки передаются с помощью отсылки к контексту, то есть фактически к соединению; значения полей можно легко предсказать для пакетов данного соединения, но не для пакетов других соединений. При нормальной работе ROHC размер заголовков для IP/UDP/RTP снижается с 40 байт до 1–3 байт.

Основная цель сжатия заголовков — уменьшить потребляемую пропускную способность. Но с помощью этого механизма можно также снизить задержку, которая складывается из задержки распространения (фиксированной для данного сетевого пути) и задержки передачи (она зависит от пропускной способности и объема передаваемых данных). Например, канал мощностью 1 Мбит/с отправляет 1 бит за 1 мкс. Если мы имеем дело с передачей мультимедиа по беспроводной сети, такой канал считается достаточно медленным, поэтому задержка передачи составляет существенную часть общей задержки, и стабильно низкая задержка важна для QoS.

При сжатии заголовков объем передаваемых данных уменьшается, и вместе с ним снижается задержка передачи. Того же эффекта можно добиться, отправляя пакеты меньшего размера. Так мы фактически меняем повышенные издержки программного обеспечения на сниженную задержку передачи. Стоит отметить, что еще одним источником задержки является время ожидания в очереди для доступа к беспроводному каналу. Это может стать важным фактором, так как беспроводные сети обычно сильно загружены. В таком случае канал должен включать механизмы QoS, обеспечивающие низкую задержку для пакетов в реальном времени. Одного сжатия заголовков будет недостаточно.

### 6.7.8. Протоколы для протяженных сетей с высокой пропускной способностью

В начале 1990-х годов появились гигабитные сети, передающие данные на большие расстояния. Их также называют **протяженными сетями с высокой пропускной способностью (long fat networks)**. Сначала к ним пытались применить старые протоколы, но из-за этого возникло множество трудностей. В данном разделе мы обсудим некоторые из этих проблем с увеличением скорости и задержки сетевых протоколов.

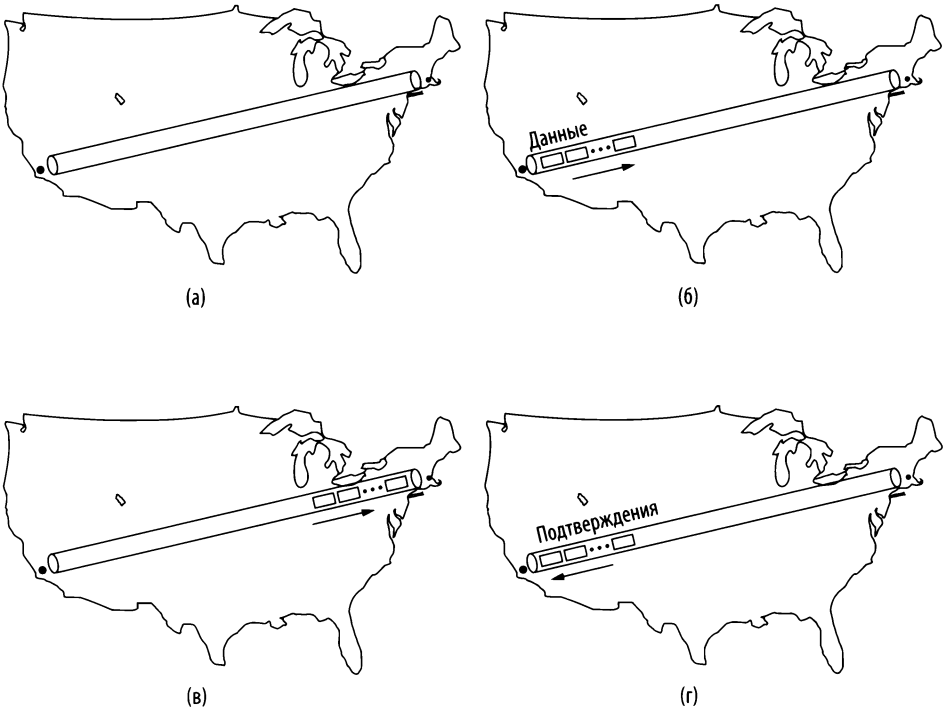
Первая проблема состоит в том, что многие протоколы используют 32-разрядные порядковые номера пакетов. На заре интернета стандартная скорость выделенных линий между маршрутизаторами равнялась 56 Кбит/с. В этих условиях хосту, который постоянно выдавал бы данные в сеть на полной скорости, потребовалось бы больше недели на то, чтобы пройти по всем порядковым номерам. С точки зрения разработчиков TCP, $2^{32}$ считалось неплохим приближением к бесконечности, поскольку вероятность блуждания пакетов по сети в течение недели практически равна нулю. При 10-мегабитном Ethernet время обхода всех порядковых номеров пакетов снизилось с одной недели до 57 минут — гораздо меньше, но все еще приемлемо. Однако когда Ethernet выдает в интернет данные со скоростью 1 Гбит/с, порядковые номера могут закончиться примерно через 34 с, при этом максимальное время жизни пакета в интернете равно 120 с. Оказалось, что $2^{32}$ совершенно не подходит в качестве значения, приближенного к бесконечности: быстрый отправитель может циклически проходить по пространству последовательностей в то время, как старые пакеты все еще будут блуждать по сети.

К сожалению, многие разработчики протоколов предполагали, что время цикла пространства порядковых номеров значительно превосходит максимальное время жизни пакетов. Следовательно, сама возможность того, что старые пакеты еще могут находиться где-то в сети, когда порядковые номера опишут полный круг, даже не рассматривалась. Но для гигабитных сетей это предположение оказалось неверным. Однако решение нашлось: было предложено расширить эффективный порядковый номер, взяв в качестве старших битов метку времени. Она добавляется в TCP-заголовок пакета как дополнительный параметр. Как уже упоминалось, данный механизм называется PAWS.

Вторая проблема заключается в необходимости существенного увеличения окна управления потоком. Например, рассмотрим, процесс отправки 64 Кбайт данных из Сан-Диего в Бостон при размере буфера получателя в 64 Кбайт. Пусть скорость передачи данных по линии составляет 1 Гбит/с, а время прохождения сигнала в одну сторону, ограниченное скоростью света в стекле оптоволокна, равно 20 мс. Вначале, при $t = 0$, канал пуст (илл. 6.55 (а)). Только 500 мкс спустя все сегменты попадут в канал (илл. 6.55 (б)). Первый сегмент оказывается где-то в окрестностях Броли, все еще в Южной Калифорнии. Тем не менее отправитель должен остановиться, пока не получит новую информацию об окне.

Через 20 мс первый сегмент, как показано на илл. 6.55 (в), достигнет Бостона, и в ответ будет передано подтверждение. Наконец, через 40 мс после

начала операции первое подтверждение возвращается к отправителю, после чего передается следующая порция данных. Поскольку линия передачи использовалась в течение всего 0,5 мс из 40 мс, производительность составит около 1,25 %. Такая ситуация типична для старых протоколов, работающих на гигабитных линиях.




**Илл. 6.55.** Передача половины мегабита из Сан-Диего в Бостон. (а) В момент времени t = 0. (б) Через 500 мкс. (в) Через 20 мс. (г) Через 40 мс

При анализе производительности сетей полезно обращать внимание на **произведение пропускной способности на задержку (bandwidth-delay product)**. Пропускная способность канала (в битах в секунду) умножается на время прохождения сигнала в оба конца (в секундах). В результате получается емкость канала в битах.

На илл. 6.55 произведение пропускной способности на задержку равно 40 млн бит. Другими словами, отправитель к моменту получения ответа успеет переслать 40 млн бит, если будет работать на максимальной скорости. Столько битов потребуется, чтобы заполнить канал в обоих направлениях. Вот почему порция данных в полмиллиона битов достигает всего 1,25 % эффективности: это лишь 1,25 % емкости канала.

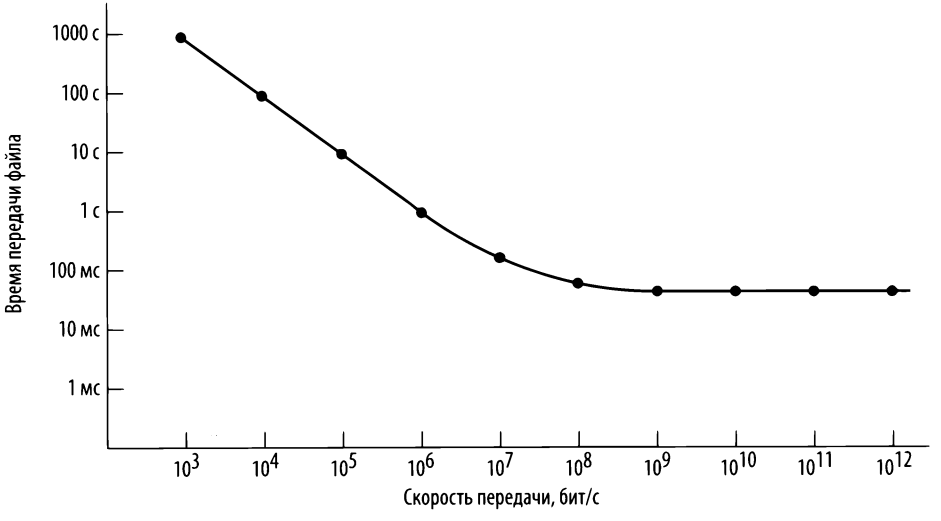
Отсюда следует, что для эффективного использования канала размер окна получателя должен быть по меньшей мере равен произведению пропускной

способности и времени задержки, а лучше — превосходить его, так как получатель может ответить не сразу. Для трансконтинентальной гигабитной линии каждому соединению потребуется минимум по 5 Мбайт.

Третья проблема, связанная с предыдущей, состоит в том, что простые схемы повторной передачи, например протокол с возвращением на $N$ блоков, плохо работают на линиях с большим значением произведения пропускной способности на задержку.

Рассмотрим трансконтинентальную гигабитную линию. Время прохождения сигнала в обе стороны равно 40 мс, за которые отправитель успевает передать 5 Мбайт. Если обнаруживается ошибка, требуется 40 мс, чтобы оповестить об этом отправителя. При использовании протокола с возвращением на $N$ блоков отправителю потребуется повторять передачу не только поврежденного пакета, но также и до 5 Мбайт пакетов, переданных после поврежденного. Очевидно, что этот протокол использует ресурсы очень неэффективно. Поэтому необходимы более сложные механизмы, такие как метод выборочного повтора.

Суть четвертой проблемы заключается в принципиальном различии между гигабитными и мегабитными линиями: в длинных гигабитных каналах главным ограничивающим фактором является не пропускная способность, а задержка. На илл. 6.56 изображена зависимость времени, необходимого для передачи файла размером 1 Мбит по линии длиной в 4000 км, от скорости передачи. На скорости до 1 Мбит/с время передачи в основном зависит от скорости передачи данных. При скорости 1 Гбит/с задержка в 40 мс значительно превосходит 1 мс (время, требующееся для передачи битов по оптоволоконному кабелю). Дальнейшее увеличение скорости не оказывает на время передачи файла почти никакого воздействия.




**Илл. 6.56.** Время передачи и подтверждения файла размером 1 Мбит по линии длиной 4000 км

Изображенная на илл. 6.56 зависимость демонстрирует ограничения сетевых протоколов. Она показывает, что протоколы с ожиданием подтверждений (например, удаленный вызов процедур) имеют врожденное ограничение на производительность. Оно связано со скоростью света. Никакие технологические прорывы в области оптики здесь не помогут (хотя могли бы помочь новые законы физики). Если в периоды времени, пока хост ждет ответа, гигабитная линия будет простаивать, она будет ничем не лучше простой мегабитной линии (и при этом дороже).

Пятая проблема заключается в том, что скорости передачи данных растут гораздо быстрее, чем скорости обработки данных. (Обращение к разработчикам компьютеров: вы должны поставить этих разработчиков средств связи на место! Мы рассчитываем на вас.) В 1970-е годы объединенная сеть ARPANET работала на скорости 56 Кбит/с и состояла из компьютеров с производительностью около 1 MIPS (1 млн инструкций/с). Сравните эти цифры с современными показателями (производительность 1000 MIPS при скорости 1 Гбит/с). Число инструкций на один байт уменьшилось более чем в десять раз. Точные цифры зависят от времени и конкретной ситуации, но вывод такой: у протоколов стало меньше времени на обработку пакетов, поэтому они должны стать проще.

Теперь рассмотрим методы решения всех этих проблем. Главный принцип, который каждый разработчик высокоскоростных сетей должен выучить наизусть, звучит так:

*Проектируя, стремись увеличить скорость, а не оптимизировать пропускную способность.*

При разработке старых протоколов обычно ставилась задача минимизировать количество битов, переданных в линию, часто за счет использования полей малого размера и упаковывания нескольких полей вместе в один байт или слово. Сегодня экономить пропускную способность смысла нет: ее более чем достаточно. Проблема заключается в обработке пакетов, поэтому при разработке новых протоколов минимизировать нужно именно время обработки. Разработчики IPv6, к счастью, хорошо понимают это.

Возникает соблазн создать быстрый сетевой интерфейс на аппаратном уровне. Сложность в том, что если протокол не является совсем простым, аппаратный интерфейс представляет собой дополнительную плату с отдельным процессором и своей программой. Чтобы сетевой сопроцессор не был столь же дорогим, как и центральный CPU, он часто представляет собой медленную микросхему. В результате центральный CPU ждет, пока сопроцессор выполнит свою работу. Было бы ошибкой полагать, что у центрального CPU обязательно найдется другая задача на время ожидания. Кроме того, для общения двух процессоров общего назначения потребуются тщательно продуманные протоколы, обеспечивающие их корректную синхронизацию. Как правило, наилучший метод заключается в создании простых протоколов, чтобы всю работу мог выполнять центральный процессор.

Для гигабитных сетей крайне важна структура пакета. В заголовок следует включить как можно меньше полей — это позволит снизить время его обработки.

Сами поля должны быть достаточно большими, чтобы справляться с задачей, при этом они не должны пересекать границы слов; тогда их будет проще обработать. Под «достаточно большим» подразумевается размер, который исключает зацикливание порядковых номеров при существовании в сети старых пакетов, неспособность получателя сообщить доступный размер окна из-за мелкого служебного поля и другие проблемы.

Максимальный объем данных должен быть большим, чтобы снизить программные накладные расходы и обеспечить эффективную работу сети. 1500 байт — недостаточный размер пакета для высокоскоростных сетей, поэтому гигабитная сеть Ethernet позволяет передавать сверхдлинные фреймы размером до 9 Кбайт, а IPv6 поддерживает джамбограммы, превышающие 64 Кбайт.

Далее мы обсудим вопрос обратной связи в высокоскоростных протоколах. Из-за относительно длинного цикла задержки желательно отказаться от обратной связи: на оповещение отправителя уходит слишком много времени. Пример обратной связи — управление скоростью передачи с помощью протокола раздвижного окна. Чтобы избежать задержек (которые могут быть долгими), связанных с передачей сведений о состоянии окна от получателя отправителю, лучше использовать протокол, основанный на скорости. В этом случае отправитель будет передавать данные на той скорости, с которой получатель заранее согласился (но не быстрее).

Еще один пример обратной связи — алгоритм медленного старта, разработанный Джейкобсоном. Этот алгоритм проводит многочисленные проверки, пытаясь узнать пропускную способность сети. В высокоскоростных сетях на проведение пяти-шести тестов для определения отклика сети тратится огромное количество сетевых ресурсов. Более эффективная схема состоит в резервировании ресурсов отправителем, получателем и сетью в момент установления соединения. Кроме того, заблаговременное резервирование ресурсов позволяет несколько снизить джиттер. Иными словами, рост скоростей передачи неумолимо подталкивает разработчиков к использованию сетей, ориентированных на установление соединения.

Еще одно полезное свойство для протокола — возможность отправлять нормальное количество данных вместе с запросом соединения. Благодаря этому можно сэкономить один RTT.

## 6.8. РЕЗЮМЕ

Транспортный уровень — это ключ к пониманию многоуровневых протоколов. Он предоставляет различные службы, важнейшей из которых является сквозной, надежный, требующий соединения поток байтов от отправителя к получателю. Доступ к нему осуществляется с помощью служебных примитивов: с их помощью можно устанавливать, использовать и разрывать соединения. Общепринятый интерфейс транспортного уровня обеспечивается сокетами Беркли.

Транспортные протоколы должны управлять соединением в ненадежных сетях. Установление соединения осложняется возможностью наличия дубликатов пакетов, которые могут появляться в самый неподходящий момент. Для

борьбы с ними при установлении соединения применяется алгоритм «тройного рукопожатия». Разрыв соединения — более простой, но далеко не тривиальный процесс из-за проблемы «двух армий».

Даже если сетевой уровень абсолютно надежен, у транспортного уровня немало работы. Он должен обрабатывать все служебные примитивы, управлять соединениями и таймерами, распределять пропускную способность и осуществлять контроль перегрузки, а также использовать раздвижное окно переменного размера для управления потоком данных.

Контроль перегрузки должен справедливо распределять пропускную способность между конкурирующими потоками и отслеживать изменения в использовании сети. Закон управления AIMD позволяет получить эффективное и справедливое распределение.

Главными транспортными интернет-протоколами являются TCP и UDP. UDP — протокол без установления соединения. Он работает с IP-пакетами и обеспечивает мультиплексирование и демультиплексирование нескольких процессов с использованием единого IP-адреса. UDP может применяться при клиент-серверных взаимодействиях, например при удаленном вызове процедур (RPC). Кроме того, на его основе можно создавать протоколы реального времени (например, RTP).

TCP — наиболее распространенный интернет-протокол. Он обеспечивает надежный дуплексный поток байтов с контролем перегрузки. Для всех сегментов применяется 20-байтный заголовок. Оптимизации производительности TCP было уделено много внимания. Для этого в нем применяются алгоритмы Нейгла (Nagle), Кларка (Clark), Джейкобсона (Jacobson), Карна (Karn) и др.

Хотя UDP и TCP прекрасно справляются со своей задачей на протяжении многих лет, они оставляют большой простор для внесения улучшений с целью повышения производительности и решения проблем, порождаемых современными высокоскоростными сетями. Такими современными доработками, в частности, являются CUBIC TCP, QUIC и BBR.

Производительность сети обычно зависит от протокола и накладных расходов по обработке сегментов, а с увеличением скорости передачи данных ситуация ухудшается. Протоколы должны разрабатываться так, чтобы минимизировать число сегментов и оставаться работоспособными при больших значениях задержки передачи. Для гигабитных сетей лучше всего подходят простые протоколы и потоковая обработка.

## ВОПРОСЫ И ЗАДАЧИ

1. В нашем примере транспортных примитивов, приведенных на илл. 6.2, `LISTEN` является блокирующим вызовом. Обязательно ли это? Если нет, объясните, как следует пользоваться неблокирующим примитивом. Какое преимущество это даст по сравнению с представленной нами схемой?
2. Приложение для общения в чате, использующее протокол TCP, многократно вызывает функцию получения данных `receive()` и выводит полученные

данные как новое сообщение. К какой проблеме может привести такой подход?

3. В модели, лежащей в основе диаграммы состояний на илл. 6.4, предполагается, что пакеты могут теряться на сетевом уровне и поэтому должны подтверждаться индивидуально. Допустим, сетевой уровень обеспечивает 100 %-ную надежность доставки и никогда не теряет пакеты. Нужны ли какие-нибудь изменения в диаграмме состояний на илл. 6.4, и если да, то какие?
4. В обеих частях илл. 6.6 значение *SERVER_PORT* должно быть одинаковым у клиента и у сервера. Почему это так важно?
5. Предположим, что используется управляемая таймером схема генерирования начальных порядковых номеров с 15-разрядным счетчиком тактовых импульсов. Таймер срабатывает один раз в 100 мс, а максимальное время жизни пакета равно 60 с. Как часто должна производиться ресинхронизация:

   а) в наихудшем случае?

   б) если на данные тратится 240 порядковых номеров в минуту?
6. Почему максимальное время жизни пакета $T$ должно быть достаточно большим, чтобы гарантировать, что не только пакет, но и его подтверждение исчезли?
7. Представьте протокол транспортного уровня, ориентированный на установление соединения, который выбирает порядковые номера для пакетов на основе времени суток. Таймер использует 10-битный счетчик и срабатывает с интервалом в 125 мс. Максимальное время жизни пакета — 64 с. Если отправитель передает по 4 пакета/с, как долго соединение будет работать без захода в запретную зону?
8. Объясните, в чем состоит разница между использованием протокола раздвижного окна на канальном уровне и его использованием на транспортном уровне с точки зрения тайм-аутов протокола.
9. Рассмотрим проблему восстановления после сбоев хостов (илл. 6.18). Если бы интервал между записью и отправкой подтверждения (или наоборот) можно было сделать относительно небольшим, какими были бы две лучшие стратегии отправителя и получателя, минимизирующие риск ошибки протокола?
10. В сеть на илл. 6.20 добавляется новый поток *E*, который идет через маршрутизаторы *R1*, *R2* и *R6*. Как изменится распределение пропускной способности по максиминному критерию для пяти потоков?
11. Допустим, потоки на илл. 6.20 реорганизованы так, что поток *A* проходит через маршрутизаторы *R1*, *R2*, *R3*, *B* — через *R1*, *R2*, *R5*, *R6*, *C* — через *R4*, *R2*, *R3*, а *D* — через *R4*, *R2*, *R3*. Как при этом будет выглядеть распределение пропускной способности по максиминному критерию?
12. Обсудите преимущества и недостатки схемы кредитного протокола по сравнению с протоколами раздвижного окна.
13. Существуют и другие стратегии, обеспечивающие равнодоступность при контроле перегрузки: аддитивное увеличение, аддитивное уменьшение

(Additive Increase Additive Decrease, AIAD); мультипликативное увеличение, аддитивное уменьшение (Multiplicative Increase Additive Decrease, MIAD); мультипликативное увеличение, мультипликативное уменьшение (Multiplicative Increase Multiplicative Decrease, MIMD). Что вы можете сказать об их сходимости и стабильности?

14. Имеется протокол транспортного уровня, который использует правило AISRD (Additive Increase Square Root Decrease – аддитивное увеличение, уменьшение по закону квадратного корня). Сходится ли эта версия протокола к справедливому распределению пропускной способности?
15. Два хоста одновременно передают данные через сеть с пропускной способностью 1 Мбит/с. Хост *A* использует UDP и передает 100-байтный пакет за 1 мс. Хост *B* генерирует данные со скоростью 600 Кбит/с и использует TCP. Какой хост получит более высокую пропускную способность?
16. Зачем нужен протокол UDP? Разве не достаточно было бы просто позволить пользовательским процессам отправлять необработанные IP-пакеты?
17. Рассмотрим простой протокол прикладного уровня на основе UDP, позволяющий клиенту запрашивать файл с удаленного сервера, расположенного по общеизвестному адресу. Клиент отправляет запрос с именем файла, а сервер отвечает последовательностью информационных пакетов с разными частями запрошенного файла. Для обеспечения надежности и доставки частей в правильном порядке клиент и сервер используют протокол с ожиданием. Какие сложности могут возникнуть с таким протоколом, кроме очевидных проблем с производительностью? Учитывайте вероятность сбоя процессов.
18. Клиент отправляет 128-байтный запрос на сервер, удаленный от него на 100 км, по оптоволокну со скоростью 1 Гбит/с. Какова эффективность линии во время выполнения удаленного вызова процедуры?
19. Вновь рассмотрите ситуацию, описанную в предыдущем вопросе. Вычислите минимально возможное время ответа для данной линии со скоростью 1 Гбит/с и для линии со скоростью 1 Мбит/с. Какой вывод можно сделать, исходя из полученных значений?
20. Как в UDP, так и в TCP номера портов используются для идентификации принимающей подсистемы при доставке сообщения. Назовите две причины того, почему для этих протоколов были изобретены новые абстрактные идентификаторы (номера портов) и не использовались идентификаторы процессов, уже существовавшие на момент появления данных протоколов.
21. Почему протокол RTP обычно реализуется поверх UDP, а не TCP? При каких условиях приложение может использовать RTP на основе TCP?
22. Рассмотрим две сети, *N1* и *N2*, с одинаковой средней задержкой при передаче пакетов от источника *A* к получателю *D*. В *N1* задержка распределена равномерно с максимальным значением 10 с, а в *N2* 99 % пакетов имеют задержку менее 1 с, но максимальная задержка может быть сколь угодно большой. Подумайте, как в таких ситуациях можно использовать RTP, если вы планируете передавать аудио-/видеопоток в режиме реального времени.

23. Каков суммарный размер минимального MTU протокола TCP, включая накладные расходы TCP и IP, но не учитывая накладные расходы канального уровня?
24. Фрагментация и дефрагментация дейтаграмм выполняется протоколом IP и невидима для TCP. Означает ли это, что TCP не должен беспокоиться о данных, приходящих в неверном порядке?
25. RTP используется для передачи аудио с таким же качеством звука, как на компакт-диске, что соответствует паре 16-битных сэмплов на частоте 44 100 Гц, по одному сэмплу на каждый канал стерео. Сколько пакетов в секунду должен уметь передавать RTP?
26. Процессу хоста 1 был назначен порт $p$, а процессу хоста 2 — порт $q$. Могут ли несколько соединений одновременно существовать между этими портами?
27. На илл. 6.36 мы видели, что в дополнение к 32-разрядному полю `Acknowledgement number` в четвертом слове имеется бит `ACK`. Приносит ли он какую-либо пользу? Ответ поясните.
28. Имеется TCP-соединение, которое передает данные с такой высокой скоростью, что начинает повторно использовать порядковые номера в течение максимального времени жизни сегмента. Можно ли предотвратить это, увеличив размер сегмента? Объясните почему.
29. Опишите два способа, с помощью которых можно достичь состояния *SYN RCVD* (илл. 6.39).
30. Какой возможный недостаток несет в себе использование алгоритма Нейгла в сильно загруженной сети?
31. Вы играете в онлайн-игру по сети с высокой задержкой. Игра требует, чтобы вы быстро нажимали на объекты на экране. В то же время результаты ваших действий игра отображает в режиме слайд-шоу. Может ли определенный параметр протокола TCP привести к такому результату? Какой еще причиной (связанной с сетью) это может быть вызвано?
32. Рассмотрим эффект использования алгоритма медленного старта на линии с RTT равным 10 мс, без перегрузок. Размер окна получателя 24 Кбайт, а максимальный размер сегмента равен 2 Кбайт. Через какое время может быть передано полное окно?
33. Предположим, окно перегрузки протокола TCP установлено на 18 Кбайт, когда происходит тайм-аут. Каким будет размер окна, если четыре последующие передачи будут успешными? Предполагается, что максимальный размер равен 1 Кбайт.
34. Имеется соединение, использующее протокол TCP Reno. Исходный размер окна перегрузки — 1 Кбайт, исходный порог — 64 Кбайт. Допустим, аддитивное увеличение производится с шагом в 1 Кбайт. Каким будет размер окна перегрузки на восьмом круге передачи, если первым является нулевой круг?
35. Текущее значение RTT протокола TCP равно 30 мс, а следующие подтверждения приходят через 26, 32 и 24 мс. Каково будет новое значение RTT? Используйте $\alpha = 0{,}9$.

36. TCP-устройство передает окна по 65 535 байт по гигабитному каналу, в котором время прохождения сигнала в один конец равно 10 мс. Какова максимальная достижимая пропускная способность канала? Чему равна эффективность использования линии?
37. Какова максимальная скорость, с которой хост может отправлять в линию TCP-пакеты, содержащие 1500 байт пользовательских данных, если максимальное время жизни пакета в сети равно 120 с? Требуется, чтобы порядковые номера не зацикливались. При расчете учитывайте накладные расходы на TCP, IP и Ethernet. Предполагается, что фреймы Ethernet могут передаваться непрерывно.
38. Чтобы устранить ограничения в IPv4, IETF приложила немало усилий и разработала IPv6, однако эта версия до сих пор внедряется неохотно. При этом для решения проблем, связанных с ограничениями в TCP, настолько серьезных усилий не требуется. Объясните почему.
39. Чему равна максимальная скорость передачи данных для каждого соединения, если максимальный размер сегмента равен 128 байт, максимальное время жизни сегмента равно 30 с, при этом используются 8-разрядные порядковые номера сегментов?
40. Имеется TCP-соединение, в котором максимальное время жизни сегмента составляет 128 с. Предполагается, что параметр временной метки *не используется*. Что в таком случае можно сказать о максимальной скорости передачи данных?
41. Рассмотрим TCP-соединение между отправителем и получателем с такими параметрами: отправителю нужно передать получателю ровно 30 сегментов; порог медленного старта *ssthresh* — 4; исходный размер окна перегрузки *cwnd* (на нулевом круге передачи) — 1; RTT для пути между отправителем и получателем — 500 мс; максимальный размер сегмента — 1000 байт; пропускная способность узких мест — 64 Кбит/с. Представьте, что:

    а) отправитель получает три дубликата подтверждения для 14-го сегмента и успешно повторяет его передачу на следующем круге;

    б) в ходе первой попытки сегменты 25–30 теряются в одном сеансе передачи;

    в) не происходит никаких других потерь.

    Чему равна средняя пропускная способность данного соединения на этапе предотвращения перегрузки (в килобитах в секунду)? Чему равна средняя пропускная способность (в килобитах) всего соединения? Чему равен средний коэффициент потерь в рамках всей передачи? На каких кругах передачи заполняется буфер узкого места? На каком круге передачи в этом буфере находится больше всего пакетов? Чему равна максимальная дополнительная задержка, которая добавляется к сквозной задержке из-за такой буферизации (в миллисекундах)?
42. Предположим, вы измеряете время, необходимое для получения сегмента. Когда возникает прерывание, вы считываете показания системного таймера в миллисекундах. После полной обработки сегмента вы снова проверяете его

показания. В результате миллиона измерений вы получаете значения 0 мс 270 000 раз и 1 мс 730 000 раз. Какой вывод можно сделать на основании этих результатов?

43. CPU выполняет 1000 MIPS. Данные могут копироваться 64-разрядными словами. На копирование каждого слова требуется 10 инструкций. Может ли такая система управлять гигабитной линией, если каждый приходящий пакет должен быть скопирован четыре раза? Для простоты предположим, что все инструкции, даже обращения к памяти, выполняются с максимальной скоростью — 1000 MIPS.

44. Для решения проблемы повторного использования старых порядковых номеров пакетов (в то время как старые пакеты еще существуют) можно использовать 64-разрядные порядковые номера. Однако теоретически оптоволоконный кабель может обладать пропускной способностью до 75 Тбит/с. Какое максимальное время жизни пакетов следует выбрать, чтобы гарантировать отсутствие в сетях будущего пакетов с одинаковыми номерами при скорости линий 75 Тбит/с и 64-разрядных порядковых номерах? Предполагается, что порядковый номер присваивается каждому байту, как в TCP.

45. Имеется компьютер, работающий со скоростью 1000 MIPS (1 инструкция за 1 нс). Он должен выполнить 50 инструкций для обработки заголовка пакетов, независимо от объема пользовательских данных, и по 10 инструкций на каждые 8 байт этих данных. Сколько пакетов в секунду может обработать компьютер, если размер пакетов составляет: а) 128 байт и б) 1024 байта? Чему в обоих случаях равна полезная пропускная способность в байтах в секунду?

46. В гигабитной линии протяженностью более 4000 км ограничивающим фактором является не пропускная способность, а время задержки. Рассмотрим MAN со средней удаленностью отправителя от получателя 20 км. При какой скорости передачи данных RTT из-за конечности скорости света будет равно времени передачи одного пакета размером 1 Кбайт?

47. Рассчитайте произведение пропускной способности на задержку в следующих сетях:

    а) T1 (1,5 Мбит/с);

    б) Ethernet (10 Мбит/с);

    в) T3 (45 Мбит/с);

    г) STS-3 (155 Мбит/с).

    Предполагается, что RTT = 100 мс. Не забудьте, что в TCP-заголовке на размер окна отводится 16-разрядное поле. Как этот факт отразится на результатах вычислений?

48. Чему равно произведение пропускной способности на задержку для канала геостационарной спутниковой связи с пропускной способностью 50 Мбит/с? Если все пакеты имеют размер 1500 байт (включая накладные расходы), какого размера должно быть окно в пакетах?

49. По каким причинам тест пропускной способности сети доступа, выполняемый на стороне клиента, может не отражать реальную скорость канала доступа?
50. Рассмотрите TCP-заголовок (илл. 6.36). Каждый отправляемый TCP-сегмент включает четыре неиспользуемых бита. Как скажется на производительности удаление этих битов со смещением всех последующих полей на четыре бита влево?
51. Файловый сервер, код которого представлен на илл. 6.6, далек от совершенства. Неплохо было бы внести в него некоторые улучшения. Проделайте следующие изменения:
    а) пусть у клиента появится третий аргумент, указывающий байтовый диапазон;
    б) добавьте флаг `-w` в программу клиента, который позволил бы записывать файл на сервер.
52. Почти все сетевые протоколы должны уметь работать с сообщениями. Если вы помните, протоколы передают их путем добавления/отделения заголовков. Некоторые протоколы могут разбивать сообщение на несколько фрагментов, а потом восстанавливать его. Попробуйте разработать библиотеку управления сообщениями с поддержкой создания нового сообщения, добавления/отделения заголовка, разбиения одного сообщения на два, объединения двух сообщений в одно и сохранения копии сообщения. Минимизируйте, насколько это возможно, копирование данных из одного буфера в другой. Важно, чтобы эти операции работали только с указателями, не затрагивая данные в сообщении.
53. Разработайте и реализуйте систему сетевого общения (чат) для нескольких групп пользователей. Координатор чата располагается по общеизвестному сетевому адресу, использует для связи с клиентами UDP, настраивает чат-серверы перед каждой сессией общения и поддерживает каталог чат-сессий. На каждую сессию выделяется один обслуживающий сервер. Для связи с клиентами сервер использует TCP. Клиентская программа позволяет пользователям начинать разговор, присоединяться к уже ведущейся дискуссии и покидать сессию. Разработайте и реализуйте код координатора, сервера и клиента.

# ГЛАВА 7
# Прикладной уровень

Изучив основы компьютерных сетей, мы переходим к уровню, на котором находятся приложения. Все уровни, расположенные ниже прикладного, занимаются доставкой данных, но не выполняют никакой практической работы для пользователя. В этой главе мы изучим реальные сетевые приложения.

Прикладной уровень также нуждается в служебных протоколах, способных обеспечить функционирование многочисленных приложений. И прежде чем изучать сами приложения, мы познакомимся с одним из таких протоколов. Речь идет о службе имен доменов, которая сопоставляет доменные имена с IP-адресами. Далее будут рассмотрены три реальных приложения: электронная почта, Всемирная паутина (или просто «интернет») и мультимедийные приложения, включая современную потоковую видеопередачу. В конце главы мы поговорим о распределении контента, в том числе об одноранговых (пиринговых) сетях и сетях доставки контента.

## 7.1. СЛУЖБА ИМЕН ДОМЕНОВ DNS

Теоретически программа может обращаться к веб-страницам, почтовым ящикам и другим ресурсам, используя сетевые адреса (то есть IP-адреса) компьютеров, на которых они хранятся, однако пользователям тяжело запоминать эти адреса. Кроме того, размещение веб-страницы компании по адресу *128.111.24.41* будет означать, что в случае переезда сервера компании на новый компьютер потребуется сообщить новый IP-адрес всем заинтересованным лицам. Хотя такой переезд веб-сайта с одного IP-адреса на другой кажется маловероятным, в действительности это вполне распространенное явление при балансировке нагрузки. Например, контент многих современных сайтов размещается на большом количестве компьютеров, часто в территориально разных кластерах. А хостинг-провайдерам иногда нужно переместить соединение какого-нибудь клиента с одного веб-сервера на другой. Проще всего это сделать с помощью **службы имен доменов (Domain Name System, DNS)**.

Имена компьютеров отделяются от их адресов путем использования удобочитаемых высокоуровневых имен. Таким образом, к веб-серверу компании можно обращаться по имени www.cs.uchicago.edu, независимо от того, какой у него IP-адрес. Поскольку устройства на сетевом пути пересылают трафик

к получателю, руководствуясь его IP-адресом, эти удобочитаемые доменные имена нужно конвертировать в IP-адреса; именно это и делает DNS. В следующих разделах мы узнаем, как DNS выполняет это преобразование и каким изменениям она подверглась за прошедшие десятилетия. В частности, серьезное развитие в последние годы получила защита личной информации пользователей. Мы подробно обсудим эти вопросы и последние разработки в области шифрования DNS, связанные с обеспечением конфиденциальности.

## 7.1.1. История и общие сведения

Во времена сети ARPANET имена всех компьютеров сети и их IP-адреса сохранялись в файле hosts.txt. Каждую ночь все хосты получали этот файл с сайта, на котором он хранился. В сети, состоящей из нескольких сотен больших компьютеров, работающих в системе с разделением времени, такой подход оправдывал себя.

Однако еще задолго до того, как к сети были подключены миллионы компьютеров, всем стало ясно, что этот способ не может работать вечно. Во-первых, размер файла рано или поздно стал бы слишком большим. Что еще важнее, если не управлять именами хостов централизованно, неизбежно возникнут конфликты имен. В то же время в гигантской международной сети такое управление вряд ли возможно. Для решения всех этих проблем в 1983 году и была разработана DNS. С тех пор она является ключевой составляющей интернета.

DNS представляет собой сочетание иерархической схемы именования с распределенной базой данных, реализующей эту схему. В первую очередь DNS служит для преобразования имен хостов в IP-адреса, но у нее есть и ряд иных применений, которые мы подробно обсудим ниже. Система DNS является одним из наиболее активно развивающихся протоколов интернета. Она описана в документах RFC 1034, RFC 1035, RFC 2181 и доработана в ряде других RFC.

## 7.1.2. Процесс поиска DNS

DNS работает следующим образом. Для преобразования имени в IP-адрес прикладная программа вызывает библиотечную процедуру (обычно это `gethostbyname` или ее аналог), передавая ей имя в качестве параметра. Этот процесс иногда называют **оконечным распознавателем (stub resolver)**. Он отправляет запрос с именем локальному DNS-распознавателю, или, как его еще часто называют, **локальному рекурсивному распознавателю (local recursive resolver)** или просто **локальному распознавателю**, который выполняет **рекурсивный поиск (recursive lookup)** имени, используя некоторый набор DNS-распознавателей. В итоге локальный распознаватель возвращает ответ с соответствующим IP-адресом оконечному распознавателю, который затем передает результат той функции, от которой поступил исходный запрос. И запрос, и ответ передаются как UDP-пакеты. После этого, уже имея IP-адрес, программа может связаться с хостом, соответствующим тому DNS-имени, которое она искала. Далее в этой главе мы рассмотрим этот процесс более подробно.

Обычно оконечный распознаватель направляет запрос рекурсивного поиска локальному распознавателю, то есть он просто выдает запрос и ожидает ответа. Локальный распознаватель, в свою очередь, направляет ряд запросов соответствующим серверам имен для каждого элемента именной иерархии. Сервер имен, отвечающий за определенную часть этой иерархии, при этом считается **авторитетным сервером имен (authoritative name server)** для этого домена. Как мы увидим позже, система DNS использует кэширование, но кэш при этом может устаревать. Авторитетный сервер имен обладает, как понятно из названия, непререкаемым авторитетом. Он по определению всегда прав. Перед тем как переходить к более детальному рассмотрению работы DNS, давайте немного поговорим об иерархии серверов имен DNS и процессе выделения имен.

Получив запрос от оконечного распознавателя хоста, локальный распознаватель осуществляет процесс разрешения имен, пока не найдет нужный ответ (или не найдет его вовсе). Он *не* возвращает частичные ответы. С другой стороны, корневой сервер имен (и каждый последующий) не отправляет запрос локальному серверу имен рекурсивно. Он возвращает частичный ответ и переходит к следующему запросу. Локальный распознаватель обеспечивает продолжение процесса разрешения имен путем выдачи дальнейших итеративных запросов.

В ходе процесса разрешения имен обычно используются оба механизма. Рекурсивные запросы практически всегда кажутся предпочтительными, но многие серверы имен (особенно корневые) их не обрабатывают. Они слишком загружены. Итеративные запросы перекладывают ответственность на их источник. Для локального сервера имен разумно поддерживать рекурсивные запросы, чтобы предоставлять сервис хостам на своем домене. Эти хосты не обязательно должны быть сконфигурированы для запуска полного сервера имен, достаточно возможности обращения к локальному серверу. Каждый запрос содержит 16-битный идентификатор транзакции; он копируется в ответ, и, таким образом, сервер имен может выдавать необходимые ответы, даже когда поступает много запросов одновременно.

Все ответы, в том числе все возвращенные частичные ответы, хранятся в кэше. Так, если компьютер, расположенный в домене cs.vu.nl, запросит имя cs.uchicago.edu, ответ будет сохранен в кэше. Если вскоре после этого другой хост в домене cs.vu.nl тоже запросит имя cs.uchicago.edu, ответ уже будет известен. Более того, если хост запрашивает другой хост в том же домене, например noise.cs.uchicago.edu, запрос может быть направлен напрямую на авторитетный сервер имен для домена cs.uchicago.edu. Сходным образом, при запросе других доменов по адресу uchicago.edu запрос может быть передан напрямую серверу имен домена uchicago.edu. Использование ответов, сохраненных в кэше, серьезно сокращает количество шагов в запросе и повышает производительность. Такой сценарий на самом деле является худшим из возможных вариантов, так как в кэше нет полезной информации.

Кэшированные ответы не являются авторитетными, так как изменения в домене cs.uchicago.edu не распространяются автоматически на все кэши по всему

миру, хранящие информацию об этом домене. В силу этого время жизни записей кэша не должно быть слишком большим. Именно поэтому каждый элемент базы данных DNS, о которой мы поговорим чуть позже, называемый записью ресурсов DNS, содержит поле `Time_to_live`. Оно сообщает удаленным серверам имен, как долго хранить эту запись в кэше. Если какой-либо компьютер обладает постоянным IP-адресом в течение многих лет, такую информацию можно хранить в кэше в течение суток. В случае более изменчивой информации запись безопаснее удалить спустя несколько секунд или одну минуту.

DNS-запросы имеют простой формат, который содержит разную информацию, включая запрашиваемое имя (QNAME), и вспомогательные сведения, такие как идентификатор транзакции (он часто используется для сопоставления запросов с ответами). Изначально этот идентификатор состоял только из 16 бит, а запросы и ответы были незащищенными. Такой подход делал систему DNS уязвимой для различных видов атак, в том числе так называемого отравления кэша, о котором мы подробно поговорим в главе 8. При выполнении ряда итеративных операций поиска рекурсивный DNS-распознаватель может отправить все содержимое поля `QNAME` ряду авторитетных серверов имен, возвращающих ответы. В какой-то момент разработчики протокола заметили, что отправка всего содержимого поля `QNAME` каждому авторитетному серверу среди итеративных распознавателей была небезопасной. Теперь многие рекурсивные распознаватели используют **QNAME-минимизацию**, при которой локальный распознаватель отправляет ту часть запроса, которую способен обработать соответствующий авторитетный сервер имен. Например, при разрешении с помощью QNAME-минимизации имени www.cs.uchicago.edu локальный распознаватель отправит авторитетному серверу для домена uchicago.edu только строку cs.uchicago.edu, а не полностью квалифицированное доменное имя (FQDN), чтобы избежать его раскрытия для авторитетного сервера. Более подробную информацию о QNAME-минимизации вы найдете в RFC 7816.

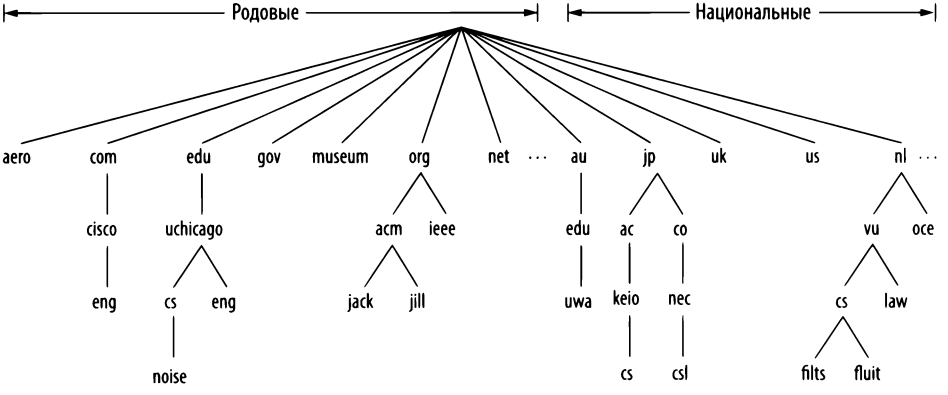
До недавнего времени в качестве транспортного протокола DNS-запросы и DNS-ответы использовали UDP, исходя из тех соображений, что такие запросы и ответы должны быть быстрыми и легковесными и не могут позволить себе накладные расходы «тройного рукопожатия» в TCP. Но после выявления недостаточной безопасности протокола DNS, что сопровождалось множеством попыток использовать его уязвимость (начиная с атак отравления кэша и заканчивая DDoS-атаками), отмечается растущая тенденция к использованию TCP в качестве транспортного протокола для DNS. Со временем это позволило системе DNS эффективно применять современные безопасные протоколы транспортного и прикладного уровней, такие как DNS поверх TLS (DNS-over-TLS, DoT) и DNS поверх HTTPS (DNS-over-HTTPS, DoH). Мы подробно коснемся этих моментов далее.

Если оконечный DNS-распознаватель не получает ответа в течение сравнительно короткого периода времени (периода ожидания), DNS-клиент направляет этот запрос к другому серверу домена после нескольких повторных попыток. Эта возможность предусмотрена на случай, если сервер выйдет из строя или будет потерян ответный пакет.

## 7.1.3. Пространство имен и иерархия DNS

Управление большим и постоянно меняющимся набором имен — нетривиальная задача. На обычных письмах требуется, так или иначе, указывать страну, регион, город, улицу, номер дома, квартиру и фамилию получателя. Благодаря использованию такой иерархической схемы не возникает путаницы между Марвином Андерсоном, живущим на Мейн-стрит в Уайт-Плейнс, штат Нью-Йорк, и Марвином Андерсоном с Мейн-стрит в Остине, штат Техас. Система DNS работает аналогично.

Основа иерархии доменных имен разработана **Корпорацией по управлению доменными именами и IP-адресами (ICANN)**. Она была создана именно для этих целей в 1998 году, поскольку интернет превратился в мировою экономическую среду. Весь интернет разделен на более чем 250 **доменов верхнего уровня (top-level domains)**, каждый из которых охватывает множество хостов. Все домены делятся на поддомены, которые тоже разделены на поддомены, и т. д. Все эти домены составляют иерархию пространства имен, которую можно представить в виде дерева (илл. 7.1). Его листьями являются домены, которые не имеют поддоменов (но, безусловно, содержат хосты). Конечный домен может состоять из одного хоста или представлять компанию и содержать в себе тысячи хостов.




**Илл. 7.1.** Часть доменного пространства интернета

Существует несколько типов доменов верхнего уровня: **родовой домен верхнего уровня, рДВУ (generic Top Level Domain, gTLD)**, **национальный домен верхнего уровня, нДВУ (country code Top Level Doman, ccTLD)**, и др. Некоторые из представленных на илл. 7.2 исходных рДВУ появились еще в 1980-х годах; впоследствии они были дополнены рядом других доменов верхнего уровня, предложенных организации ICANN. За каждым государством в соответствии с международным стандартом ISO 3166 закреплен один национальный домен. Интернационализированные доменные имена стран, в которых используется алфавит, отличный от латинского, были введены в 2010 году. Эти

домены позволяют именовать хосты, используя символы арабской, кириллической, китайской и других видов письменности.

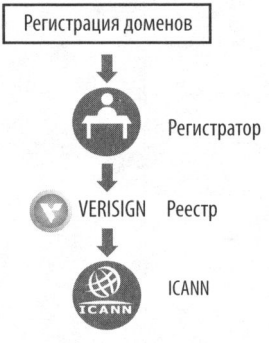
| Домен | Использование | Дата основания | Ограниченный? |
|---|---|---|---|
| com | Коммерция | 1985 | Нет |
| edu | Образовательные учреждения | 1985 | Да |
| gov | Правительство | 1985 | Да |
| int | Международные организации | 1988 | Да |
| mil | Военные | 1985 | Да |
| net | Сетевые провайдеры | 1985 | Нет |
| org | Некоммерческие организации | 1985 | Нет |
| aero | Авиатранспорт | 2001 | Да |
| biz | Бизнес | 2001 | Нет |
| coop | Кооперативы | 2001 | Да |
| info | Информация | 2002 | Нет |
| museum | Музеи | 2002 | Да |
| name | Люди | 2002 | Нет |
| pro | Профессионалы | 2002 | Да |
| cat | Каталония | 2005 | Да |
| jobs | Занятость | 2005 | Да |
| mobi | Мобильные устройства | 2005 | Да |
| tel | Контактная информация | 2005 | Да |
| travel | Индустрия путешествий | 2005 | Да |
| xxx | Секс-индустрия | 2010 | Нет |

**Илл. 7.2.** Исходные родовые ДВУ по состоянию на 2010 год. На 2020 год существовало уже более 1200 рДВУ

В 2011 году существовало только 22 рДВУ, но в июне 2011 года члены ICANN проголосовали за снятие ограничений на создание дополнительных рДВУ, что позволило компаниям и другим организациям выбирать практически произвольные домены верхнего уровня, в том числе содержащие нелатинские символы (например, кириллические). ICANN начала принимать заявки на оформление таких ДВУ нового типа в начале 2012 года. При этом изначально стоимость оформления ДВУ составляла около $200 000. Ряд новых рДВУ был введен в действие в 2013 году.

Так, в июле 2013 года начали действовать первые четыре рДВУ, введенные на основе соглашения, подписанного в Дурбане (Южная Африка). Все четыре новых домена содержали нелатинские символы: слово «интернет» по-арабски, «онлайн» и «сайт» по-русски и «игра» по-китайски. Множество заявок на оформление рДВУ подали и некоторые технологические гиганты, например, компании Google и Amazon запросили примерно по 100 новых рДВУ. На сегодняшний день широкое распространение получили такие рДВУ, как top, loan, xyz и т. д.

Что касается доменов второго уровня наподобие name-of-company.com, то получить такое имя достаточно просто. Управление доменами верхнего уровня осуществляют компании, называемые **реестрами (registries)**. Их назначает ICANN. Например, реестром, отвечающим за домен com, является компания Verisign. Непосредственную продажу доменных имен пользователям осуществляют **регистраторы (registrars)**. Существует много таких компаний, которые конкурируют друг с другом по стоимости и уровню обслуживания. К широко известным регистраторам относятся компании Domain.com, GoDaddy и NameCheap. На илл. 7.3 показаны отношения между реестрами и регистраторами в рамках процесса регистрации доменного имени.




**Илл. 7.3.** Отношения между реестрами и регистраторами

Доменное имя, которое ищет хост (такое, как www.cs.uchicago.edu или cisco.com), обычно называют **полностью квалифицированным доменным именем (Fully Qualified Domain Name, FQDN)**. Оно начинается с наиболее конкретизированной части доменного имени, с отделением каждой части иерархии с помощью символа «.». (Строго говоря, все FQDN также заканчиваются символом «.», который означает корень иерархии DNS, однако большинство операционных систем подставляет эту часть имени автоматически.)

Имя каждого домена, подобно полному пути к файлу в файловой системе, состоит из пути от этого домена до корня (безымянного). Компоненты пути разделяются точками. В отличие от того, как это принято в UNIX (/com/cisco/eng), домен технического отдела корпорации Cisco может выглядеть как eng.cisco.com. Следует отметить, что при такой иерархической системе имя eng.cisco.com. не конфликтует с потенциальным использованием eng в домене eng.uchicago.edu., который мог бы использоваться факультетом английского языка Чикагского университета.

Имена доменов могут быть абсолютными и относительными. Абсолютное имя домена всегда оканчивается точкой (например, eng.cisco.com.), тогда как относительное имя — нет. Чтобы однозначно определить истинное значение относительного имени, его нужно интерпретировать в некотором контексте. В обоих случаях именованный домен означает определенный узел дерева и все узлы под ним.

Имена доменов нечувствительны к регистру символов: edu, Edu и EDU означают одно и то же. Длина отдельных компонентов имени может составлять до 63 символов, но длина полного имени не должна превышать 255. Тот факт, что система DNS нечувствительна к регистру, используется для защиты от различных видов DNS-атак (включая отравление кэша DNS) с помощью метода 0x20-кодирования (Дейгон и др.; Dagon et al., 2008), который мы подробно обсудим далее в этой главе.

Как правило, новые домены могут добавляться в иерархию как родовых, так и национальных доменов. Так, домен cc.gatech.edu можно без проблем поместить (зачастую так и происходит) в список национального домена us в виде cc.gt.atl.ga.us. Однако на практике большинство организаций в США используют родовые домены, а большинство организаций за пределами США — домены страны, к которой они относятся. Не существует каких-либо правил, запрещающих регистрацию нескольких доменов верхнего уровня. Крупные компании часто именно так и поступают (например: sony.com, sony.net и sony.nl).

Каждый домен управляет распределением доменов, расположенных под ним. Например, в Японии домены ac.jp и co.jp используются как аналог доменов edu и com. В Голландии вообще не проводится деление по такому принципу, и все домены организаций находятся непосредственно под доменом nl. Все австралийские университеты располагаются в домене edu.au. В качестве примера можно привести следующие домены, выбранные для факультетов вычислительной техники и электротехники трех университетов:

1) cs.uchicago.edu (Чикагский университет, США);
2) cs.vu.nl (Университет Врийе, Нидерланды);
3) ee.uwa.edu.au (Университет Западной Австралии).

Для создания нового домена требуется разрешение домена, в который он будет включен. Например, если в Чикагском университете появится исследовательская группа по вопросам безопасности, которая захочет использовать домен security.cs.uchicago.edu, то ей нужно будет получить разрешение от тех, кто управляет доменом cs.uchicago.edu. (К счастью, этих людей, как правило, не так сложно найти благодаря федеративной архитектуре управления DNS.) Аналогично, если будет создан, скажем, новый Университет Северной и Южной Дакоты, то для присвоения ему домена unsd.edu (при условии, что он свободен) потребуется обратиться за разрешением к менеджеру домена edu. Так удается избежать конфликта имен, а каждый домен отслеживает состояние всех своих поддоменов. После создания и регистрации домена в нем могут создаваться такие поддомены, как cs.unsd.edu, для чего уже не требуется разрешение вышестоящих доменов.

Структура доменов отражает не физическое строение сети, а логическое разделение между организациями и их внутренними подразделениями. Так, факультеты вычислительной техники и электротехники могут использовать разные домены, даже если они находятся в одном здании и пользуются общей LAN. И наоборот, если факультет вычислительной техники расположен в двух корпусах университета, хосты обоих зданий, скорее всего, принадлежат к одному и тому же домену.

## 7.1.4. DNS-запросы и DNS-ответы

Теперь рассмотрим структуру, формат и назначение DNS-запросов, а также разберемся с тем, каким образом DNS-серверы выдают ответы на них.

### *DNS-запросы*

Как было сказано выше, DNS-клиент обычно направляет запрос локальному рекурсивному распознавателю, который выполняет итеративные операции поиска, чтобы в конечном итоге выполнить запрос. Наиболее распространенный тип запроса — запрос A-записи, позволяющей сопоставить доменное имя с IP-адресом соответствующей конечной точки интернета. В DNS есть разные записи ресурсов (с другими запросами); мы обсудим их в следующем разделе, посвященном записям ресурсов (то есть ответам).

Хотя DNS почти всегда использовалась в основном для сопоставления удобочитаемых имен с IP-адресами, за прошедшие годы DNS-запросы использовались для множества других задач. Еще один популярный вид DNS-запроса — проверка на наличие домена в **черном списке DNS (Domain Name Service Black List, DNSBL)**. Этот список используется для отслеживания IP-адресов, связанных с распространением спама и вредоносных программ. Для поиска доменного имени в DNSBL клиент может направить DNS-запрос A-записи специальному DNS-серверу, например pbl.spamhaus.org. Он производит сопоставление со списком IP-адресов, которые не должны соединяться с почтовыми серверами. Чтобы проверить конкретный IP-адрес, нужно просто изменить порядок октетов в IP-адресе на обратный и поставить полученный результат перед именем сервера pbl.spamhaus.org.

Так, например, чтобы проверить адрес 127.0.0.2, нужно сделать запрос для 2.0.0.127.pbl.spamhaus.org. При наличии этого IP-адреса в списке DNS-запрос возвращает IP-адрес, в котором, как правило, закодирована дополнительная информация (например, сведения о том, как данная запись попала в список). Если данного IP-адреса в списке нет, DNS-сервер сообщает об этом с помощью соответствующего ответа NXDOMAIN, означающего «такого домена нет» («no such domain»).

### *Расширения и улучшения DNS-запросов*

Со временем DNS-запросы стали более изощренными и сложными, поскольку клиенты нуждались во все более конкретизированной и актуальной

информации и вместе с тем возрастали требования безопасности. Существенным расширением DNS-запросов за последние годы стало использование **клиентской подсети EDNS (EDNS0 CS Extended DNS Client Subnet**, или просто **EDNS Client Subnet)**, что позволяет локальному рекурсивному распознавателю клиента передавать авторитетному серверу имен IP-адрес подсети оконечного распознавателя.

Благодаря механизму клиентской подсети EDNS авторитетный сервер имен для доменного имени может узнать IP-адрес клиента, направившего исходный запрос. Обычно это позволяет ему произвести более эффективное сопоставление с ближайшей копией реплицируемого сервиса. Например, если клиент выдаст запрос для домена google.com, авторитетный сервер имен для Google в большинстве случаев вернет имя, соответствующее ближайшему к клиенту интерфейсному серверу. Конечно, сделать это можно, лишь зная, в какой части сети (а в идеале и в какой части мира) находится клиент. Обычно авторитетный сервер имен видит только IP-адрес локального рекурсивного распознавателя.

Если клиент, инициировавший запрос, находится рядом с соответствующим локальным распознавателем, то авторитетный сервер этого домена может сопоставить клиента, просто руководствуясь местоположением локального рекурсивного DNS-распознавателя. Но клиенты все чаще используют локальные рекурсивные распознаватели, IP-адрес которых не позволяет определить местоположение клиента. Например, компании Google и CloudFlare используют общие DNS-распознаватели (с адресами 8.8.8.8 и 1.1.1.1 соответственно). Если клиент настроен на использование одного из таких распознавателей, авторитетный сервер не сможет извлечь полезную информацию из сведений об IP-адресе этого распознавателя. Механизм клиентской подсети EDNS решает эту проблему, включая IP-адрес подсети в запрос от локального рекурсивного распознавателя, чтобы авторитетный сервер видел IP-адрес подсети клиента, который инициировал запрос.

Как отмечалось ранее, имена в DNS-запросах нечувствительны к регистру. Это позволило современным DNS-распознавателям включить в запрос дополнительные биты идентификатора транзакции, установив символы в поле QNAME в произвольный регистр. 16-битный идентификатор уязвим к различным видам атаки отравления кэша, включая атаку Каминского, о которой пойдет речь в главе 8. Отчасти эта уязвимость обусловлена тем, что длина DNS-идентификатора транзакции составляет лишь 16 бит. Увеличение длины потребовало бы изменений в спецификации протокола DNS, а это непростая задача.

В итоге было разработано альтернативное решение — метод **0x20-кодирования**, который сводится к следующему. Локальный рекурсивный распознаватель переключает регистр в каждом поле QNAME («uchicago.edu» может превратиться в «uCHicaGO.EDu» и т. п.), что позволяет использовать каждый символ доменного имени в качестве дополнительного бита DNS-идентификатора транзакции. Конечно, важно, чтобы никакой другой распознаватель не менял регистр поля QNAME в последующих итеративных запросах или ответах. При сохранении

регистра соответствующий ответ содержит поле QNAME с тем регистром, который ему присвоил локальный рекурсивный распознаватель, и в результате эти биты добавляются в идентификатор транзакции. Конечно, все это выглядит как лишенные какой-либо элегантности костыли, но именно так происходит внесение изменений в широко используемое программное обеспечение с сохранением обратной совместимости.

### *DNS-ответы и записи ресурсов DNS*

У каждого домена, независимо от того, является ли он единственным хостом или доменом верхнего уровня, может быть набор ассоциированных с ним **записей ресурсов (resource records)**. Эти записи являются базой данных DNS. Для одного хоста запись ресурсов чаще всего представляет собой просто его IP-адрес, но существует еще много других записей. Когда распознаватель передает имя домена DNS-серверу, то, что он получает обратно, представляет собой записи ресурсов, ассоциированные с этим именем. Таким образом, основная функция системы DNS заключается в преобразовании доменных имен в записи ресурсов.

Записи ресурса состоят из пяти частей. Хотя для эффективности они часто перекодируются в двоичную форму, в большинстве описаний они представлены в виде ASCII-текста, по одной строке на каждую запись:

```
Domain_name Time_to_live Class Type Value
```

Поле `Domain_name` (Имя домена) сообщает, к какому домену относится текущая запись. Обычно для каждого домена имеется несколько записей ресурсов, и каждая копия базы данных хранит информацию о множестве доменов. Поле имени домена используется в качестве первичного ключа поиска при выполнении запросов. Порядок записей в базе данных значения не имеет.

Поле `Time_to_live` указывает, насколько стабильно состояние записи. Редко меняющимся данным присваивается высокое значение этого поля, например 86 400 (число секунд в сутках). Если же информация непостоянна (как, например, курсы акций) или ее вынуждены часто менять операторы сетей (например, для балансировки нагрузки при использовании одного имени для нескольких IP-адресов), то ей может быть присвоено небольшое значение, такое как 60 с (1 мин). Мы вернемся к этому вопросу, когда будем обсуждать кэширование.

Третье поле в записи ресурсов — `Class` (Класс). Для интернет-информации это всегда `IN`. Для прочей информации применяются другие коды, но на практике они встречаются редко.

Поле `Type` сообщает тип записи. Существует много типов DNS-записей. Наиболее важные из них перечислены на илл. 7.4.

Запись `SOA` (Start Of Authority — начальная точка полномочий) сообщает имя первичного источника информации о зоне сервера имен (описанного ниже), адрес электронной почты его администратора, уникальный порядковый номер, различные флаги и тайм-ауты.

| Тип | Значение | Значение |
|---|---|---|
| SOA | Начальная запись зоны | Параметры для этой зоны |
| A | IPv4-адрес хоста | 32-битное целое число |
| AAAA | IPv6-адрес хоста | 128-битное целое число |
| MX | Обмен почтой | Приоритет, с которым домен хочет принимать электронную почту |
| NS | Сервер имен | Имя сервера для этого домена |
| CNAME | Каноническое имя | Имя домена |
| PTR | Указатель | Псевдоним IP-адреса |
| SPF | Правила отправки почты | Правила отправки почты, закодированные в текстовом виде |
| SRV | Служба | Хост, предоставляющий данную службу |
| TXT | Текст | Описательный ASCII-текст |

**Илл. 7.4.** Основные типы записей ресурсов DNS

### *Распространенные типы записей*

Самым важным типом записей является A (Address — адрес). Эти записи содержат 32-разрядный IPv4-адрес интерфейса для хоста. Соответствующая запись AAAA («quad A» — «четыре A») содержит 128-разрядный IPv6-адрес. У каждого хоста в интернете должен быть как минимум один IP-адрес, чтобы другие компьютеры могли с ним взаимодействовать. Иногда у хостов есть несколько сетевых интерфейсов. В таком случае они обладают несколькими записями ресурсов типа A или AAAA. Кроме того, один сервис (например, google.com) может быть размещен на множестве компьютеров, рассредоточенных по всему миру (Колдер и др.; Calder et al., 2013). В таких случаях DNS-распознаватель может возвращать несколько IP-адресов для одного доменного имени. В случае географически распределенного сервиса для улучшения производительности и балансировки нагрузки распознаватель может возвращать клиенту один или несколько IP-адресов ближайшего к нему (с точки зрения географии или топологии) сервера.

Еще один существенный тип записей — NS. Они несут информацию о сервере имен для домена или поддомена. Это хост, на котором содержится копия базы данных для домена. Он используется в ходе процесса поиска имен, о котором мы вскоре поговорим подробнее.

Записи MX сообщают имя хоста, готового принимать электронную почту для указанного домена. Дело в том, что не каждый компьютер готов это делать. Если письмо отправлено, скажем, на адрес bill@microsoft.com, то передающий хост прежде всего должен найти в домене microsoft.com почтовый сервер, готовый к получению почты. Эту информацию может предоставить запись MX.

Записи CNAME позволяют создавать псевдонимы. Допустим, что кто-то, имеющий представление о том, как формируются имена в интернете, хочет

отправить сообщение пользователю *paul* на факультете вычислительной техники Чикагского университета. Действуя наудачу, этот человек пытается использовать адрес paul@cs.chicago.edu. Однако адрес не сработает, поскольку на самом деле факультет использует домен cs.uchicago.edu. В таком случае для удобства тех, кто этого не знает, Чикагский университет может создать запись CNAME, которая будет направлять пользователей и программы по нужному пути. Для этого можно использовать запись следующего вида:

```
www.cs.uchicago.edu 120 IN CNAME hnd.cs.uchicago.edu
```

Записи CNAME широко используются для присвоения псевдонимов веб-сайтам, поскольку адреса веб-серверов (которые часто начинаются с префикса www), как правило, размещаются на устройствах, используемых для нескольких целей, при этом www не является их основным именем.

Записи PTR указывают на другое имя и обычно используются для связывания IP-адреса с соответствующим именем. Просмотр записи PTR, связывающей имя с соответствующим IP-адресом, называется **обратным просмотром (reverse lookup)**.

SRV представляют собой более новый тип записей, позволяющий идентифицировать хост как определенный сервис в рамках домена. Например, веб-сервер для домена www.cs.uchicago.edu может быть идентифицирован как hnd.cs.uchicago.edu. Данный тип записей является обобщением записей типа MX, которые выполняют ту же задачу, но только для почтовых серверов.

Записи SPF позволяют домену закодировать информацию о том, какие компьютеры в рамках домена будут отправлять письма в остальную часть интернета. Это помогает принимающим устройствам отсеивать недопустимую почту. Если почта поступает от компьютера с именем dodgy, в то время как согласно записям домена почта должна поступать только от компьютера с именем smtp, велика вероятность того, что данные сообщения являются спамом.

Последний в списке тип записей, TXT, изначально предназначался для того, чтобы позволить доменам идентифицировать себя произвольным образом. Сегодня эти записи обычно используются для кодирования машиночитаемой информации; обычно это SPF-информация.

Наконец, рассмотрим последнее поле записи ресурсов — Value (Значение). Оно может содержать число, имя домена или текстовую ASCII-строку — это зависит от типа записи. Краткое описание полей Value для каждого из основных типов записей дано на илл. 7.4.

### *Записи DNSSEC*

В ходе изначального внедрения DNS безопасность этого протокола не учитывалась. В частности, серверы имен и распознаватели DNS могли манипулировать содержимым любой DNS-записи, в результате чего клиент мог получать некорректную информацию. В спецификации RFC 3833 описаны некоторые проблемы безопасности DNS, а также их решение с помощью **записей DNSSEC**. Записи DNSSEC позволяют включать в ответы серверов имен DNS цифровую подпись, которую

впоследствии может проверить распознаватель (локальный или оконечный), чтобы убедиться в том, что DNS-запись не подвергалась изменениям или искажениям. Каждый DNS-сервер вычисляет хеш (своего рода длинную контрольную сумму) **набора записей ресурсов (Resource Record Set, RRSET)** для каждого набора записей ресурсов одного типа, используя свои закрытые криптографические ключи. Соответствующие открытые ключи можно использовать для проверки подписей наборов RRSET. (Если вы не знакомы с шифрованием, в главе 8 будет предоставлена более подробная техническая информация.)

Прежде чем проверять подпись набора RRSET с помощью соответствующего открытого ключа сервера имен, естественно, нужно убедиться в подлинности самого ключа. Это можно сделать, если открытый ключ авторитетного сервера имен подписан родительским сервером в иерархии имени. Например, авторитетный сервер имен домена .edu может подписывать открытый ключ, привязанный к авторитетному серверу имен домена uchicago.edu, и т. д.

Технология DNSSEC подразумевает наличие двух записей ресурсов, привязанных к открытым ключам: `RRSIG` и `DNSKEY`. Запись `RRSIG` ассоциируется с подписью набора RRSET, который подписывается соответствующим закрытым ключом авторитетного сервера имен. Запись `DNSKEY` представляет собой открытый ключ для соответствующего набора RRSET, подписываемого закрытым ключом родителя. Такая иерархическая структура подписей делает возможным групповое распространение открытых ключей DNSSEC для иерархии серверов имен. Отдельно требуется распространять только открытые ключи корневого уровня, и они могут распространяться так же, как распознаватели получают информацию об IP-адресах корневых серверов имен. В главе 8 мы подробнее остановимся на технологии DNSSEC.

### *DNS-зоны*

На илл. 7.5 показано, какую информацию может содержать типичная запись ресурсов DNS для определенного доменного имени. Здесь представлена часть базы данных (гипотетической) для домена cs.vu.nl (см. илл. 7.1), которую часто называют **файлом DNS-зоны (DNS zone file)**, или просто **DNS-зоной (DNS zone)**. Данный файл зоны содержит семь типов записей ресурсов.

Первая незакомментированная строка листинга на илл. 7.5 содержит базовую информацию о домене, которая для нас не важна. Далее две строки указывают на два хоста, на которые нужно попытаться доставить электронную почту для person@cs.vu.nl. Сначала следует связаться с хостом zephyr. В случае неудачи следует попробовать доставить письмо хосту top. Следующая строка задает в качестве сервера имен домена хост star.

После пустой строки, добавленной для удобства чтения, следуют строки, сообщающие IP-адреса хостов star, zephyr и top. Затем следует псевдоним www.cs.vu.nl, позволяющий не обращаться к какому-то конкретному компьютеру. Создание этого псевдонима позволяет домену cs.vu.nl изменять свой веб-сервер, не меняя адрес, по которому люди к нему обращаются. С той же целью в следующей строке создается псевдоним ftp.cs.vu.nl для FTP-сервера.

```
; Authoritative data for cs.vu.nl
cs.vu.nl.        86400   IN   SOA     star boss (9527,7200,7200,241920,86400)
cs.vu.nl.        86400   IN   MX      1 zephyr
cs.vu.nl.        86400   IN   MX      2 top
cs.vu.nl.        86400   IN   NS      star

star             86400   IN   A       130.37.56.205
zephyr           86400   IN   A       130.37.20.10
top              86400   IN   A       130.37.20.11
www              86400   IN   CNAME   star.cs.vu.nl
ftp              86400   IN   CNAME   zephyr.cs.vu.nl

flits            86400   IN   A      130.37.16.112
flits            86400   IN   A      192.31.231.165
flits            86400   IN   MX     1 flits
flits            86400   IN   MX     2 zephyr
flits            86400   IN   MX     3 top

rowboat                  IN   A      130.37.56.201
                         IN   MX     1 rowboat
                         IN   MX     2 zephyr

little-sister            IN   A      130.37.62.23

laserjet                 IN   A      192.31.231.216
```

**Илл. 7.5.** Часть гипотетической базы данных DNS (файла зоны) домена cs.vu.nl

В секции, относящейся к хосту flits, указаны два IP-адреса и три возможных варианта адреса для обработки почты, направляемой на адрес flits.cs.vu.nl. Разумеется, прежде всего нужно попробовать доставить письмо самому хосту flits. Но если он выключен, необходимо обратиться к хостам zephyr и top.

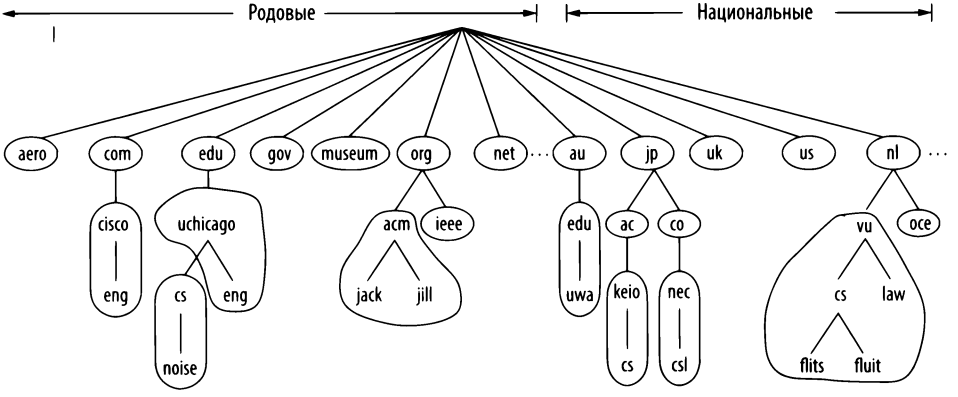
Следующие три строки содержат типичные записи для компьютера, в данном случае для rowboat.cs.vu.nl. Эти строки содержат его IP-адрес, а также имена первого и второго хостов для доставки почты. Следом идет запись о компьютере, неспособном самостоятельно получать почту. Последняя строка, вероятно, описывает подключенный к интернету лазерный принтер.

Теоретически один сервер мог бы содержать всю базу данных DNS и отвечать на все запросы к ней. На практике он оказался бы настолько перегружен, что был бы бесполезен. Более того, если бы он когда-нибудь отключился, то весь интернет перестал бы работать.

Чтобы избежать проблем, связанных с хранением всей информации в одном месте, пространство имен DNS разделено на непересекающиеся **зоны**. На илл. 7.6 показан один из способов разделения пространства имен с илл. 7.1. Здесь каждая очерченная зона содержит некоторую часть общего дерева доменов.

Расстановка границ внутри каждой зоны определяется ее администратором. Это решение зависит от того, сколько серверов имен требуется в той или иной зоне. Например, у Чикагского университета (илл. 7.6) есть зона для домена

uchicago.edu, которая управляет доменом cs.uchicago.edu, но не доменом eng.uchicago.edu, расположенным в отдельной зоне со своими серверами имен. Подобное решение может быть принято, когда факультет английского языка не хочет управлять собственным сервером имен, но этого хочет факультет вычислительной техники.



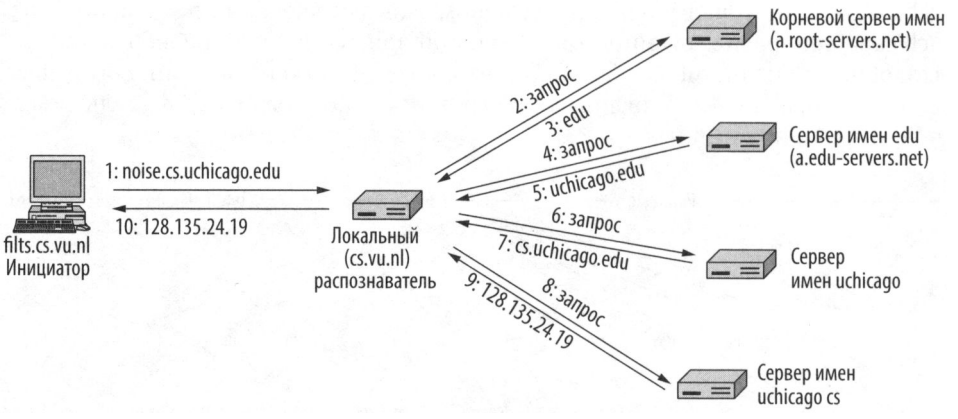

**Илл. 7.6.** Часть пространства имен DNS, разделенная на зоны (они обведены)

## 7.1.5. Разрешение имен

Каждая зона ассоциируется с одним или несколькими серверами имен. Это хосты, на которых находится база данных для зоны. Обычно у зоны есть один основной сервер имен, который берет информацию из файла на своем диске, и один или несколько второстепенных серверов, получающих данные с основного сервера. Для повышения надежности некоторые серверы имен могут быть расположены вне зоны.

Процесс поиска адреса по имени называется **разрешением имен (name resolution)**. Распознаватель обращается с запросом разрешения имени домена к локальному серверу имен. Если искомый домен относится к сфере ответственности данного сервера (к примеру, домен top.cs.vu.nl относится к домену cs.vu.nl), тогда сервер возвращает **авторитетную запись (authoritative record)** ресурса (так называют запись, которая поступает из управляющего ею авторитетного источника). В отличие от **кэшируемых записей (cached records)**, которые могут устаревать, авторитетные записи всегда считаются верными.

Но что происходит, если домен является удаленным, как, например, в случае, когда flits.cs.vu.nl пытается найти IP-адрес для домена cs.uchicago.edu в Чикагском университете? В этом случае, при отсутствии информации о запрашиваемом домене в локальном кэше, сервер имен отправляет удаленный запрос. Этот запрос выполняется, как показано на илл. 7.7. На первом шаге запрос передается локальному серверу имен. В нем указывается имя искомого домена, тип (A) и класс (IN).




**Илл. 7.7.** Пример поиска распознавателем имени удаленного хоста за десять шагов

На следующем шаге запрос направляется одному из **корневых серверов имен (root name servers)** на вершине иерархии. На таких серверах хранится информация о каждом домене верхнего уровня. На рис 7.7 этот шаг обозначен цифрой 2. Чтобы связаться с корневым сервером, каждый сервер имен должен обладать информацией об одном или нескольких подобных серверах. Обычно эти сведения находятся в файле системной конфигурации, который загружается в кэш DNS на этапе запуска сервера DNS. Это просто список записей `NS` для корня, вместе с соответствующими записями `A`.

Существует 13 корневых серверов DNS, которым без всякого воображения были присвоены имена от a-root-servers.net до m.root-servers.net. Каждый корневой сервер логически представляет собой просто отдельный хост. Но поскольку весь интернет зависит от корневых серверов, для них используются мощные и активно реплицируемые компьютеры. Большинство этих серверов расположено в разных географических точках, доступ к которым осуществляется путем произвольной маршрутизации (она сводится к доставке пакета ближайшему экземпляру адреса получателя). Произвольная рассылка была подробно описана в главе 5. Репликация этих серверов повышает надежность и производительность.

Корневой сервер имен вряд ли знает адрес искомого хоста в домене uchicago.edu, и, возможно, ему даже неизвестно, к какому серверу имен этот домен привязан. Но он должен знать сервер имен для домена edu, где расположен домен cs.uchicago.edu. Он возвращает имя и IP-адрес для этой части ответа на третьем шаге.

Затем локальный сервер имен проводит дальнейший поиск. Он отправляет весь запрос серверу имен домена edu (a.edu-servers.net). Этот сервер имен возвращает информацию о сервере имен домена uchicago.edu. На рисунке это шаги 4 и 5. Теперь, уже приблизившись к цели, локальный сервер имен отсылает запрос серверу имен домена uchicago.edu (шаг 6). Если искомое доменное имя расположено на факультете английского языка, то ответ будет найден уже на этом этапе, поскольку зона uchicago.edu включает в себя этот факультет. Однако факультет

вычислительной техники использует собственный сервер имен. Если нужное доменное имя находится на факультете вычислительной техники, то запрос вернет имя и IP-адрес сервера имен этого факультета в зоне uchicago.edu (шаг 7).

Наконец, локальный сервер имен направляет запрос серверу имен факультета вычислительной техники в зоне uchicago.edu (шаг 8). Поскольку этот сервер отвечает за домен cs.uchicago.edu, он должен выдать нужный ответ. Он вернет окончательный ответ на шаге 9, после чего на шаге 10 локальный сервер имен передаст его хосту flits.cs.vu.nl.

## 7.1.6. Практическое ознакомление с DNS

Вы можете изучить этот процесс с помощью таких стандартных инструментов, как программа *dig*, устанавливаемая на большинстве UNIX-систем. Например, напечатав

```
dig ns @a.edu-servers.net cs.uchicago.edu
```

вы отправите запрос имени cs.uchicago.edu на сервер имен a.edu-servers.net и выведете результат. Вы увидите информацию, которая была получена на четвертом шаге в приведенном выше примере, и узнаете имя и IP-адрес серверов имен для домена uchicago.edu. Большинство организаций использует несколько серверов имен на тот случай, если один из них даст сбой, часто около пяти-шести. Если у вас есть доступ к системе UNIX, Linux или MacOS, поэкспериментируйте с программой *dig* и посмотрите, чего можно добиться с ее помощью. Это позволит вам существенно углубить свое понимание системы DNS. (Программу *dig* также можно использовать и в Windows, но в этом случае вам придется установить ее самостоятельно.)

Хотя цель создания DNS проста и понятна, вполне очевидно, что это большая и сложная распределенная система, включающая в себя миллионы совместно работающих серверов имен. DNS формирует ключевую связь между удобочитаемыми доменными именами и IP-адресами хостов. Она использует репликацию и кэширование для повышения производительности и надежности и сама по себе максимально устойчива.

Иногда приложениям нужно использовать имена более гибко, например, присвоить контенту имя, а затем преобразовать его в IP-адрес ближайшего хоста, содержащего этот контент. По такой схеме производится поиск и скачивание фильмов. Главное — найти конкретный фильм, и совершенно неважно, на каком компьютере находится его копия. Соответственно, нам нужен IP-адрес *любого* ближайшего компьютера, содержащего копию искомого фильма. Такого преобразования можно достичь, к примеру, с помощью сетей доставки контента. Об их использовании поверх DNS подробно рассказано в разделе 7.5.

## 7.1.7. DNS и конфиденциальность

Изначально запросы и ответы системы DNS не шифровались. В результате любое устройство или средство подслушивания в сети (другое устройство, системный администратор, сеть кофейных автоматов и т. д.) теоретически могло

просматривать трафик пользователя системы DNS и извлекать информацию об этом человеке. Например, изучив трафик сайта uchicago.edu, можно выяснить, что пользователь посетил сайт Чикагского университета. Эта информация вполне безобидна, но в случае сайта webmd.com просмотр трафика позволяет узнать, что пользователь провел некое медицинское исследование. Сочетая эти данные с другой информацией, часто можно получить более конкретные сведения и узнать, какой именно веб-сайт посещает пользователь.

Проблемы конфиденциальности DNS-запросов становятся все более актуальными по мере развития таких новых областей применения, как IoT и умные дома. Например, DNS-запросы от устройств могут раскрывать сведения о том, какие именно устройства используют люди в своих умных домах и насколько активно. Так, DNS-запросы, отправляемые подключенной к интернету камерой или находящимся в спящем режиме монитором, позволяют однозначно идентифицировать это устройство (Апторп и др; Apthorpe et al., 2019). Учитывая растущую конфиденциальность активности пользователей, применяющих устройства с выходом в интернет (будь то браузер или умное устройство), потребность в шифровании запросов и ответов DNS становится все более актуальной.

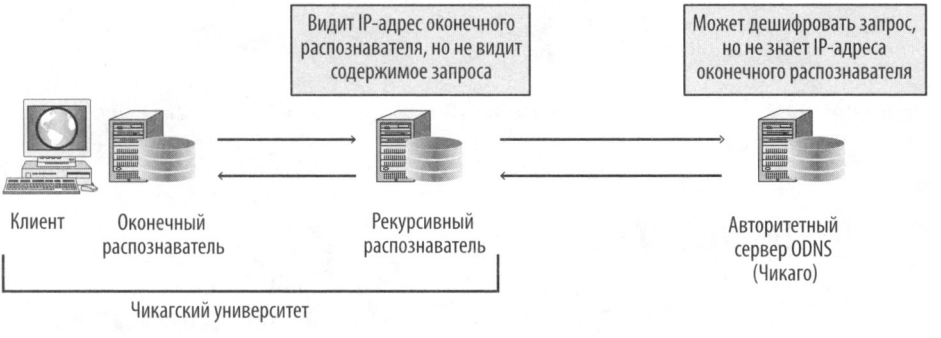
Некоторые последние тенденции, например тренд в сторону шифрования DNS-запросов и DNS-ответов, могут привести к полному видоизменению DNS. Многие компании, включая CloudFlare, Google и ряд других, предоставляют пользователям возможность направить DNS-трафик на локальные рекурсивные распознаватели этих компаний, при этом позволяя шифровать трафик (к примеру, с помощью TLS и HTTPS) между оконечным DNS-распознавателем и локальным распознавателем компании. Некоторые из этих компаний сотрудничают с разработчиками браузеров (например, браузера Mozilla), чтобы в конечном итоге весь DNS-трафик по умолчанию передавался на их локальные распознаватели.

Если пересылка всех запросов и ответов DNS будет осуществляться облачным провайдером по шифруемому каналу, это может иметь очень серьезные последствия для архитектуры интернета в будущем. В частности, интернет-провайдеры не смогут отслеживать DNS-запросы, поступающие из домашних сетей абонентов, в то время как раньше это был один из основных способов проверки сетей на предмет вредоносных программ (Антонакакис и др.; Antonakakis et al., 2010). Просмотр содержимого DNS-трафика требуется и для многих услуг, предлагаемых провайдерами, например для родительского контроля.

В итоге мы имеем дело с двумя не зависящими друг от друга проблемами. Одна из них — смещение DNS в сторону шифруемой передачи, что почти все считают положительной переменой (изначально это вызвало ряд вопросов о производительности, но по большей части они решены). Вторая, более сложная проблема — вопрос о том, кто должен управлять локальными рекурсивными распознавателями. Раньше это, как правило, был провайдер пользователя. Но если процесс DNS-разрешения переместится в браузер за счет протокола DoH, то браузеры смогут управлять доступом к DNS-трафику (а на сегодня два

самых популярных браузера почти полностью контролируются одним главным поставщиком, компанией Google). В конечном счете оператор локального рекурсивного распознавателя сможет просматривать содержимое DNS-запросов пользователя и ассоциировать их с IP-адресом. При этом пользователь сможет выбрать, кому доверить просмотр своего DNS-трафика — своему провайдеру или крупной рекламной компании. Но от стандартных настроек браузера будет зависеть, кто в итоге просмотрит большую часть этого трафика. В настоящее время многие организации, начиная с интернет-провайдеров и заканчивая провайдерами контента и рекламными компаниями, пытаются ввести в практику **доверенные рекурсивные распознаватели (Trusted Recursive Resolvers, TRR)**. Это локальные рекурсивные распознаватели, которые используют для разрешения клиентских запросов DoT или DoH. Как эти тенденции повлияют на архитектуру DNS, покажет время.

Даже протоколы DoT и DoH не могут полностью решить проблемы DNS в сфере конфиденциальности, поскольку оператору локального распознавателя все равно приходится предоставлять частную информацию, а именно содержимое DNS-запросов и IP-адреса клиентов, от которых они поступают. Недавно в качестве альтернативы были предложены улучшенные версии DNS и DoH, в частности **«забывчивый DNS» (oblivious DNS, ODNS)** (Шмитт и др.; Schmitt et al., 2019) и **«забывчивый DoH» (oblivious DoH, ODoH)** (Киннир и др.; Kinnear et al., 2019). Благодаря этому оконечный распознаватель шифрует исходный запрос перед тем, как передать его локальному распознавателю, а тот передает его авторитетному серверу имен. Сервер может выполнить дешифрование и разрешить запрос, но при этом ему неизвестен идентификатор или IP-адрес оконечного распознавателя, который инициировал запрос. Схема этих взаимодействий представлена на илл. 7.8.




**Илл. 7.8.** «Забывчивый DNS»

Большая часть этих реализаций пока находится на начальном этапе развития, в виде первых прототипов и проектов стандартов, обсуждаемых в рамках рабочей группы по конфиденциальности DNS комитета IETF.

### 7.1.8. Разногласия по поводу способов именования

По мере того как интернет распространяется по всему миру и все больше развивается с точки зрения коммерции, растет число спорных вопросов (особенно в отношении именования доменов). Разногласия возникают и в самой ICANN. Например, создание домена xxx заняло несколько лет и повлекло за собой несколько судебных разбирательств. Размещение контента для взрослых на отдельном домене — это хорошо или плохо? (Многие хотели вообще запретить публикацию подобных материалов в интернете; другая точка зрения сводилась к тому, что лучше выделить для них отдельный домен, чтобы облегчить их поиск и блокировку для фильтров родительского контроля.) Некоторые домены самоорганизуются, другие имеют ограничения в отношении того, кто может получить имя (см. илл. 7.2). Вопрос в том, какие ограничения оправданны. Возьмем для примера домен pro, предназначенный для квалифицированных специалистов. Кто именно является таким специалистом? Врачи и адвокаты точно входят в эту категорию. А что насчет самозанятых фотографов, учителей музыки, фокусников, сантехников, парикмахеров, дезинсекторов, татуировщиков, наемников и проституток? Можно ли отнести к этой категории представителей данных профессий? И кто должен это определять?

Также можно зарабатывать на именах доменов. Например, Тувалу, небольшая островная страна между Гавайями и Австралией, сдала в аренду права на свой домен tv за $50 млн, так как он отлично подходит для телевизионных сайтов. Практически все общеупотребительные английские слова используются в качестве имен поддоменов com (вместе с самыми распространенными опечатками). Попробуйте набрать любое слово, касающееся домашнего хозяйства, животных, растений, частей тела и т. д. Регистрация доменных имен с целью их выгодной перепродажи заинтересованной стороне даже получила специальное название — **киберсквоттинг (cybersquatting)**. Не успев зарегистрироваться в начале эры интернета, многие компании обнаружили, что самые очевидные доменные имена уже заняты. В целом, если не были нарушены права на товарный знак и не было замечено мошенничества, здесь работает правило «кто не успел, тот опоздал». Но политика разрешения споров по поводу имен доменов пока еще не устоялась.

## 7.2. ЭЛЕКТРОННАЯ ПОЧТА

Электронная почта, или как ее часто называют — **e-mail**, существует уже более четырех десятилетий. Она быстрее и дешевле обычной почты и стала популярной с первых дней интернета. До 1990 года электронная почта использовалась преимущественно в научных организациях. В 1990-е годы с ней познакомились широкие слои населения, и ее использование выросло в геометрической прогрессии. В результате число ежедневно отправляемых электронных писем во много раз превышает количество бумажных. За последнее десятилетие стали популярными и другие формы интернет-коммуникации, например

обмен мгновенными сообщениями и IP-телефония, но электронная почта по-прежнему остается основным средством сетевого общения. К примеру, она широко применяется для внутрикорпоративной связи на предприятиях, что позволяет обеспечить участие географически удаленных сотрудников в работе над сложными проектами. К сожалению, как и в случае обычной почты, девять из десяти электронных писем — это рекламная рассылка, то есть **спам**. Хотя современные почтовые системы удаляют большую часть таких сообщений, многие из них все же достигают адресата. Поиск способов, позволяющих обнаружить весь спам, продолжается (Дэн и др.; Dan et al., 2019; Чжан и др.; Zhang et al., 2019).

Электронной почте, как и любой другой форме коммуникации, свойствен определенный стиль и условные обозначения. Этот вид общения носит очень неформальный и демократичный характер. Скажем, человек, который никогда не осмелился бы позвонить или даже написать бумажное письмо какой-нибудь высокопоставленной персоне, запросто может отправить ему (или ей) небрежно написанное электронное письмо. Устраняя большинство указаний на должность, возраст и пол, электронная почта позволяет сфокусироваться исключительно на содержании, не принимая в расчет различия в статусе. Благодаря электронной почте блестящая идея студента-практиканта может произвести более серьезный эффект, чем слабая идея, предложенная вице-президентом компании.

В электронной почте широко используются жаргонные сокращения, например BTW (By The Way — «кстати»), ROFL (Rolling On the Floor Laughing — «катаюсь по полу от смеха») и IMHO (In My Humble Opinion — «на мой взгляд»). Многие пользователи также используют комбинации символов ASCII, то есть **смайлы**. Например, комбинацию «:-)» можно увидеть повсюду. Этот и другие **эмотиконы** (символы эмоций) помогают передать тон сообщения. Они широко применяются и в других видах коммуникации, таких как обмен мгновенными сообщениями, обычно в виде графических знаков — **эмодзи**. Современные смартфоны позволяют использовать сотни эмодзи.

За время существования электронной почты ее протоколы существенно изменились. Первые системы e-mail состояли из протокола передачи файлов с условием, что первая строка каждого сообщения (то есть файла) должна содержать адрес получателя. Постепенно электронная почта отошла от передачи файлов и получила множество новых функций, таких как рассылка сообщения сразу нескольким адресатам. В 1990-х годах добавилась передача мультимедийных данных — писем с изображениями и другой нетекстовой информацией. Серьезно изменились и программы для чтения электронной почты: на смену текстовому пришел графический пользовательский интерфейс. Также у пользователей появился доступ к почте с ноутбука вне зависимости от местонахождения. Наконец, из-за повсеместного засилия спама почтовые программы научились худо-бедно находить и удалять нежелательные письма.

При рассмотрении электронной почты мы сфокусируемся на том, как производится передача сообщений от одного пользователя другому, не касаясь

внешнего вида и особенностей программ для чтения таких писем. Однако, обсудив общую архитектуру, мы коснемся и интерфейса почтовой системы, поскольку она знакома большинству читателей.

## 7.2.1. Архитектура и службы

В данном разделе мы рассмотрим возможности и организацию систем электронной почты; ее архитектура показана на илл. 7.9. Система e-mail состоит из двух подсистем: **пользовательских агентов (user agents)**, позволяющих пользователям читать и отправлять письма, и **агентов передачи сообщений (message transfer agents)**, пересылающих письма от отправителя к получателю; последних мы будем неформально называть **почтовыми серверами (mail servers)**.

**Илл. 7.9.** Архитектура системы e-mail

Пользовательский агент — это программа, предоставляющая графический интерфейс (а иногда интерфейс, основанный на тексте или командах), через который пользователи взаимодействуют с системой электронной почты. Он включает средства написания новых сообщений и ответов, отображения входящих писем и их организации (путем распределения по папкам), поиска и удаления. Отправка новых сообщений в почтовую систему называется **подачей почтового сообщения (mail submission)**.

Обработка сообщений может частично автоматизироваться с учетом пожеланий пользователя. Например, входящая почта может фильтроваться с целью выявления или снижения приоритета сообщений, похожих на спам. Некоторые пользовательские агенты предлагают такие продвинутые возможности, как автоматическая отправка ответных сообщений («Я в отпуске и смогу связаться с Вами лишь спустя некоторое время»). Пользовательский агент работает на том же компьютере, на котором пользователь читает свою почту. Это обычная программа, которую можно запускать время от времени.

Агенты передачи сообщений, как правило, являются системными процессами, которые работают в фоновом режиме на устройствах почтовых серверов и всегда должны быть доступны. Они автоматически перемещают почтовые сообщения

в системе от отправителя к получателю с помощью протокола SMTP, который мы обсудим в разделе 7.2.4. На этом этапе передается сообщение.

SMTP был впервые описан в RFC 821. Позже в него вносились изменения вплоть до текущей редакции RFC 5321. Он отправляет сообщения по соединениям и высылает обратно отчеты о статусе доставки и любых возникших ошибках. Существует множество ситуаций, в которых подтверждение доставки имеет большую важность и даже может иметь юридическое значение («Ваша честь, моя электронная система не очень надежна, и я полагаю, что электронная повестка в суд где-то потерялась»).

Агенты передачи также применяют **списки рассылки (mailing lists)**, чтобы доставлять идентичные копии сообщений всем получателям в списке. Среди других полезных функций можно назвать следующие: копия письма, скрытая копия, высокий приоритет письма, секретная (то есть зашифрованная) почта, доставка сообщения альтернативному получателю (если основной временно недоступен), а также возможность предоставить доступ к почте своему секретарю.

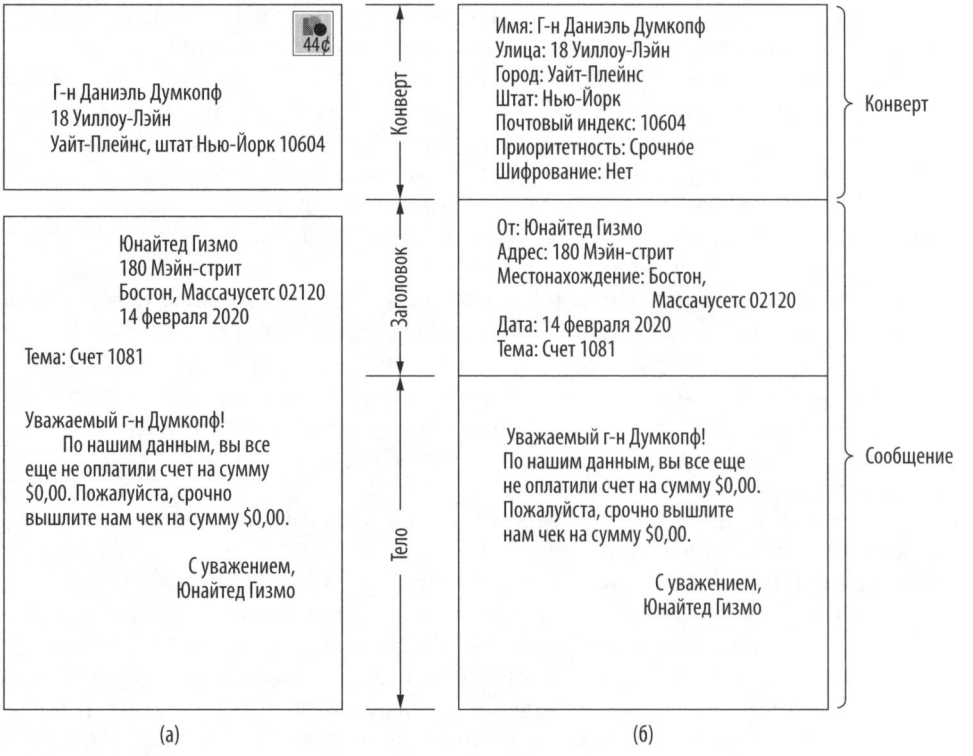
Соединение пользовательских агентов и агентов передачи сообщений обеспечивают почтовые ящики и стандартный формат почтовых сообщений. **Почтовые ящики (mailboxes)** хранят доставленную почту. Они обслуживаются почтовыми серверами. Пользовательские агенты просто предоставляют людям возможность увидеть содержимое их почтовых ящиков. Для этого агент отправляет команды почтовым серверам и получает возможность управлять почтовыми ящиками (проверять содержимое, удалять сообщения и т. д.). Получение почты — это ее доставка конечному пользователю на илл. 7.9 (шаг 3). При такой архитектуре один пользователь может использовать различные агенты на разных устройствах, чтобы получить доступ к одному и тому же почтовому ящику.

Почта пересылается между агентами передачи сообщений в стандартном формате. Первоначальный формат, RFC 822, был существенно переработан. Текущая версия носит название RFC 5322; в нее включена поддержка мультимедиа-контента и международный текст. Эта схема получила название «MIME». Однако многие по-прежнему называют электронную почту «RFC 822».

В основе данного формата сообщений лежит четкое разграничение между **конвертом (envelope)** и его наполнением. Конверт содержит в себе сообщение, а также всю информацию, необходимую для его доставки: адрес получателя, приоритет, уровень секретности и т. п. Все эти сведения отделены от самого сообщения. Агенты передачи сообщений используют конверт для маршрутизации, аналогично тому, как это делает обычная почтовая служба.

Сообщение внутри конверта состоит из двух отдельных частей: **заголовка (header)** и **тела (body)**. В заголовке указана управляющая информация для пользовательских агентов. Тело письма целиком предназначается для человека-адресата. Примеры конвертов и сообщений показаны на илл. 7.10.

Мы изучим эту архитектуру более детально и рассмотрим шаги, необходимые для того, чтобы передать сообщение от одного пользователя другому. Наше путешествие начинается с пользовательского агента.



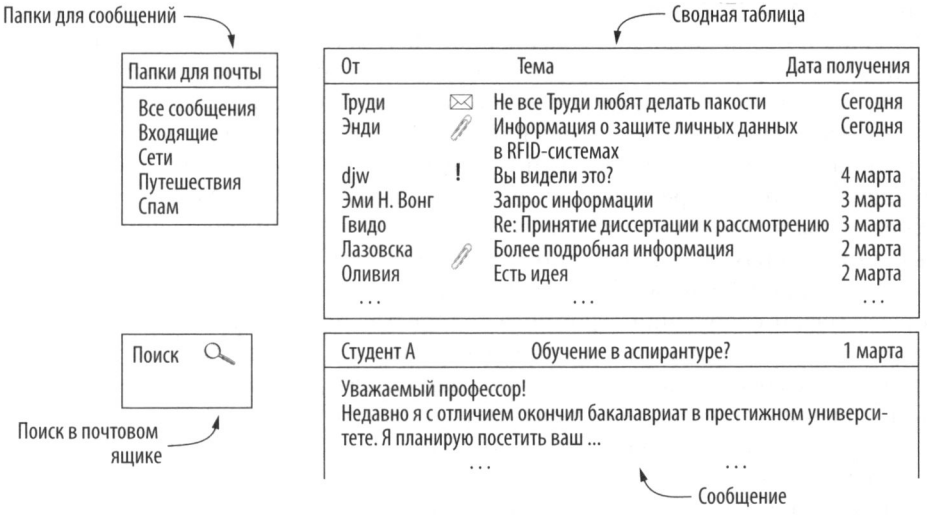

**Илл. 7.10.** Конверты и сообщения. (а) Обычное письмо. (б) Электронное письмо

## 7.2.2. Пользовательский агент

Пользовательский агент — это программа (иногда ее называют **«почтовиком»** — **email reader**), выполняющая разнообразные команды для составления и получения сообщений, ответа на них, а также для управления почтовыми ящиками. Существует множество популярных пользовательских агентов, включая Google Gmail, Microsoft Outlook, Mozilla Thunderbird и Apple Mail. Внешне они могут сильно различаться. Графический интерфейс часто основан на меню или на значках и требует наличия мыши или сенсорного интерфейса (на небольших мобильных устройствах). Более ранние пользовательские агенты (Elm, mh или Pine) имеют текстовый интерфейс и работают при помощи ввода с клавиатуры однобуквенных команд. С точки зрения функциональности это то же самое, по крайней мере в отношении текстовых сообщений.

Типичные элементы интерфейса пользовательского агента показаны на илл. 7.11. Агент, которым пользуетесь вы, наверняка выглядит гораздо более современно, но функции, скорее всего, совпадают. Когда агент запущен, обычно он отображает сводную таблицу сообщений в почтовом ящике. Часто каждому сообщению соответствует одна строка, а идут они в каком-либо порядке,

выбранном при сортировке. В сводной таблице выделены ключевые поля сообщения, извлеченные из конверта или заголовка.




**Илл. 7.11.** Стандартные элементы интерфейса пользовательского агента

На илл. 7.11 представлены семь строк сводной таблицы. В строках используются поля `From` (От), `Subject` (Тема) и `Received` (Получено); именно в таком порядке. В них указано, от кого это сообщение, о чем оно и когда пришло. Вся информация удобно отформатирована; она не отображает содержание полей сообщения полностью, но основана на них. Таким образом, те, кто отправляет сообщения без темы, часто сталкиваются с тем, что их письмам присваивается низкий приоритет.

Поля и маркеры бывают очень разнообразными. Значки рядом с темой (см. илл. 7.11) могут обозначать, например, непрочитанное письмо (конверт), прикрепленный файл (скрепка) или важное, по крайней мере с точки зрения отправителя, сообщение (восклицательный знак).

Также возможны несколько вариантов сортировки. Самый распространенный — по времени получения; сначала более свежие письма, с маркером, указывающим на то, прочитано сообщение или нет. Поля сводной таблицы и порядок сортировки можно настроить по желанию пользователя.

Пользовательские агенты должны отображать входящие сообщения так, чтобы почту можно было читать. Обычно предоставляется предварительный просмотр письма (см. илл. 7.11), чтобы пользователь мог решить, стоит ли открыть его или пометить как спам. При предварительном просмотре могут использоваться маленькие значки или изображения, чтобы описать содержание письма. Обработка отображения также включает форматирование сообщения таким образом, чтобы оно помещалось на экране, перевод, конвертацию содержания в более удобный формат (например, оцифрованной речи в распознанный текст).

Прочитав сообщение, вы можете решить, что с ним делать. Это называется **размещением сообщения (message disposition)**. Письмо можно удалить, написать ответ, переслать его другому пользователю и оставить в ящике для дальнейшей работы. В большинстве агентов присутствует один почтовый ящик для входящих и набор папок для сохраненной почты. Это позволяет распределять сообщения по разным категориям в зависимости от отправителя, темы и т. д.

Распределение по папкам также может происходить автоматически, до того, как пользователь прочтет сообщение. Например, агент проверяет поля и содержание писем, а также анализирует реакцию пользователя на предыдущие сообщения, чтобы определить, является ли письмо спамом. Многие интернет-провайдеры и компании используют софт, помечающий сообщения как важные или как спам, так что пользовательский агент может распределить их по соответствующим папкам. У провайдеров и компаний есть преимущество: они обрабатывают сообщения огромного количества пользователей, так что у них могут быть списки известных спамеров. Если сотни пользователей получают одно и то же сообщение, вероятно, это спам (хотя с другой стороны, это может быть и письмо от генерального директора для всех сотрудников). Предварительно сортируя письма и помечая часть из них как «похоже на спам», агент может избавить пользователей от массы работы по отделению нужной почты от мусора.

Но какой же спам встречается чаще всего? Спам генерируется сетью взломанных вычислительных устройств — **ботнетом (botnet)**, а его содержание зависит от страны, в которой вы живете. В Азии часто предлагают приобрести поддельные дипломы, в США — дешевые лекарства и другую сомнительную продукцию. В большом количестве предлагаются невостребованные счета в нигерийском банке. Ну а таблетки для увеличения определенных частей тела входят в топ-лист, где бы вы ни жили.

Правила распределения писем по папкам пользователь может установить сам. Каждое правило определяет состояние и действие. Например, оно может гласить, что письмо от руководителя должно перемещаться в папку для немедленного прочтения, а письма от определенного списка людей должны помещаться в другую папку, чтобы их можно было прочитать позже. На илл. 7.11 изображено несколько папок. Самые важные — это Входящие (для почты, которая не была никуда распределена при получении) и Спам (для сообщений, которые программа посчитала нежелательными).

## 7.2.3. Форматы сообщений

Перейдем от рассмотрения пользовательского интерфейса к формату самих сообщений электронной почты. Чтобы сообщения, отсылаемые пользовательским агентом, обрабатывались агентами передачи сообщений, они должны быть оформлены в соответствии с определенными стандартами. Прежде всего мы рассмотрим базовый ASCII-формат электронного письма стандарта RFC 5322. Это последний вариант оригинального формата интернет-сообщений, описанного в стандарте RFC 822 и его многочисленных обновлениях. Затем мы познакомимся с мультимедийным расширением этого первоначального стандарта.

### *RFC 5322 — формат интернет-сообщений*

Сообщения состоят из примитивного конверта (в RFC 5321 он описан как часть SMTP), нескольких полей заголовка, пустой строки и тела сообщения. Каждое поле заголовка состоит (логически) из одной строки ASCII-текста, содержащей имя поля, двоеточие и значение поля (в большинстве случаев). Первоначальный RFC 822 был создан несколько десятилетий назад, и в нем нет четкого разграничения конверта и заголовка. Хотя частично стандарт был пересмотрен в RFC 5322, целиком обновить его было невозможно, поскольку RFC 822 уже широко применялся. Обычно пользовательский агент формирует сообщение и передает его агенту передачи сообщений, который с помощью одного из полей заголовка создает конверт. Получается несколько старомодное сочетание письма и конверта.

Основные поля заголовка, связанные с транспортировкой сообщения, перечислены на илл. 7.12. Поле `To:` (Кому) содержит адрес электронной почты основного получателя (их может быть несколько). В поле `Cc:` (Carbon copy — Копия; буквально «под копирку») указываются адреса дополнительных получателей. С точки зрения доставки никаких отличий между основным и дополнительными получателями нет. Разница между ними чисто психологическая и важна только для людей, но не для почтовой системы. Обозначение `Cc:` несколько устарело, ведь компьютеры не используют копировальную бумагу, но термин прижился. Поле `Bcc:` (Blind carbon copy — Скрытая копия) аналогично предыдущему, но в этом случае строка поля удаляется из всех экземпляров сообщения, отправленных как основному, так и дополнительным получателям. Это позволяет рассылать письмо одновременно нескольким получателям, и они не будут знать, что это письмо отправлено кому-то еще.

| Поле | Значение |
|---|---|
| To: | Электронный адрес (адреса) основного получателя (получателей) |
| Cc: | Электронный адрес (адреса) дополнительного получателя (получателей) |
| Bcc: | Электронный адрес (адреса) получателей скрытой копии |
| From: | Автор (авторы) сообщения |
| Sender: | Электронный адрес отправителя |
| Received: | Строка, добавляемая каждым агентом передачи сообщений на пути |
| Return-Path: | Может быть использовано для идентификации обратного пути к отправителю |

**Илл. 7.12.** Поля заголовка стандарта RFC 5322, связанные с транспортировкой сообщения

Следующие два поля, `From:` (От) и `Sender:` (Отправитель), сообщают, соответственно, кто составил и отправил сообщение. Это могут быть разные люди. Например, написать письмо может руководитель предприятия, а отослать — его секретарь. В этом случае в поле `From:` будет указано имя руководителя, а в поле

`Sender:` — имя секретаря. Поле `From:` является обязательным, тогда как поле `Sender:` может быть опущено, если его содержимое не отличается от содержимого поля `From:`. Эти поля нужны на случай, если сообщение доставить невозможно и об этом нужно проинформировать отправителя. Кроме того, на адреса, указанные в этих полях, может быть выслан ответ.

Строка, содержащая поле `Received:` (Получено), добавляется каждым агентом передачи сообщений на пути следования письма. В это поле помещается идентификатор агента, дата и время получения сообщения, а также другая информация, которая может быть использована для исправления неисправностей в системе маршрутизации. Поле `Return-Path:` (Обратный путь) добавляется последним агентом передачи сообщений. Предполагалось, что это поле будет сообщать, как добраться до отправителя. Теоретически эта информация может быть собрана из всех полей `Received:` (кроме имени почтового ящика отправителя), но на практике оно редко заполняется и обычно просто содержит адрес отправителя.

Помимо полей, показанных на илл. 7.12, сообщения стандарта RFC 5322 могут также включать широкий спектр полей заголовка, используемых пользовательским агентом или самим пользователем. Наиболее распространенные поля заголовка приведены на илл. 7.13. Информации в таблице достаточно, чтобы понять назначение полей, поэтому мы не будем рассматривать их подробно.

| Поле | Значение |
|---|---|
| Date: | Дата и время отправки сообщения |
| Reply-to: | Электронный адрес, на который следует присылать ответ |
| Message-Id: | Уникальный номер для последующей ссылки на это сообщение |
| In-Reply-To: | Идентификатор Message-Id сообщения, в ответ на которое отправляется это сообщение |
| References: | Другие важные ссылки (идентификаторы Message-Id) |
| Keywords: | Ключевые слова, выбираемые пользователем |
| Subject: | Краткое изложение сообщения для отображения в одной строке (тема письма) |

**Илл. 7.13.** Некоторые поля, используемые в заголовке сообщения стандарта RFC 5322

Поле `Reply-to:` (Обратный адрес) иногда применяется в случае, если ни составитель письма, ни его отправитель не хотят получать на него ответ. Например, управляющий отделом сбыта может написать письмо, информирующее клиентов о новом продукте. Это письмо отправляется его секретарем, но в поле `Reply-to:` указан адрес менеджера отдела продаж, который может ответить на вопросы и принять заказы. Это поле также может пригодиться, если у отправителя есть два электронных адреса и он хочет, чтобы ответы приходили на второй адрес.

`Message-Id:` (Идентификатор сообщения) — автоматически генерируемое число, которое используется, чтобы связывать сообщения (например, при ответе на письмо) и избежать повторной доставки.

В спецификации RFC 5322 пользователям напрямую разрешается создавать дополнительные заголовки в своих целях. В RFC 822 и последующих стандартах эти заголовки начинаются со строки `X-`. Гарантируется, что в будущем ни один стандартный заголовок не будет начинаться с этих символов, чтобы избежать конфликтов между официальными и частными заголовками. Иногда умники-студенты включают поля вроде `X-Fruit-of-the-Day:` («фрукт дня») или `X-Disease-of-the-Week:` («недуг недели»), что вполне допустимо, хотя и не всегда понятно.

После заголовков идет тело самого сообщения. Пользователь может разместить в нем все, что ему угодно. Некоторые люди завершают свои послания сложными подписями с популярными или малоизвестными цитатами, политическими заявлениями и разнообразными объявлениями (например, «Корпорация АБВ не несет ответственности за высказанное выше мнение. Собственно, она даже не в силах его постичь»).

### *MIME — многоцелевые расширения интернет-почты*

На заре существования сети ARPANET электронная почта состояла исключительно из текстовых сообщений на английском языке, представленных символами ASCII. В таких условиях первоначального стандарта RFC 822 было вполне достаточно: он определял формат заголовков, но оставлял содержимое сообщения полностью на усмотрение пользователей. В 1990-е годы повсеместное распространение интернета и необходимость передавать через почтовую систему более разнообразный контент показали, что такой подход уже не отвечает требованиям. К примеру, нужно было обеспечить передачу сообщений на разных языках: с диакритическими знаками (французском или немецком), с алфавитом, отличным от латинского (иврите или русском), или вовсе без алфавита (китайском или японском). Также требовалась возможность отправки сообщений, не являющихся текстом (например, аудио, изображений или бинарных документов и программ).

Решением стала разработка стандарта **многоцелевых расширений интернет-почты (Multipurpose Internet Mail Extensions, MIME)**. Он широко применяется как для сообщений электронной почты, передаваемых по интернету, так и для описания контента в других случаях, например при просмотре сайтов. Изначально MIME описан в RFC 2045, а последующие его версии — в RFC 4288 и 4289.

Основная идея стандарта MIME сводилась к тому, чтобы продолжить использование формата RFC 822, но при этом придать структуру телу сообщения и определить правила кодирования для пересылки сообщений без ASCII-символов. Не отклоняясь от стандарта RFC 822, MIME-сообщения могут передаваться с помощью обычных агентов передачи сообщений и протоколов (ранее основанных на RFC 821, а сейчас на RFC 5321). Все, что нужно было

изменить, — отправляющие и принимающие программы; это пользователи могли сделать самостоятельно.

MIME задает пять новых заголовков сообщений (илл. 7.14). Первый заголовок (`MIME-Version:`) просто информирует пользовательского агента, что он имеет дело с сообщением MIME, и указывает номер версии MIME. Если в сообщении нет такого заголовка, то оно считается написанным на английском языке (или по крайней мере использующим только ASCII-символы) и обрабатывается соответственно.

| Поле | Значение |
|---|---|
| MIME-Version: | Указывает версию MIME |
| Content-Description: | Строка обычного текста, описывающая содержимое сообщения |
| Content-Id: | Уникальный идентификатор |
| Content-Transfer-Encoding: | Указывает способ кодировки тела сообщения для его передачи |
| Content-Type: | Тип и формат содержимого сообщения |

**Илл. 7.14.** Заголовки сообщений, добавленные в стандарте MIME

Заголовок `Content-Description:` (Описание содержимого) представляет собой ASCII-строку, информирующую о том, что находится в сообщении. Он позволяет пользователю принять решение о том, нужно ли ему декодировать и читать сообщение. Если в строке сказано: «Фото хомяка Арона», а получатель не любит хомяков, скорее всего, он сразу удалит это сообщение и не станет перекодировать его в цветную фотографию высокого разрешения.

Заголовок `Content-Id:` (Идентификатор содержимого) идентифицирует данные в сообщении. В нем используется тот же формат, что и в стандартном заголовке `Message-Id:`.

В заголовке `Content-Transfer-Encoding:` (Кодировка содержимого для передачи) сказано, как тело сообщения было упаковано для отправки по сети. При разработке MIME основной проблемой было то, что протоколы электронной почты (SMTP) предполагали использование ASCII-сообщений с длиной строк не более 1000 символов. Символы ASCII задействуют 7 бит из каждого восьмибитного байта. Бинарные данные, такие как исполняемые программы и изображения, используют все 8 бит байта, так же как и расширенные наборы символов. Не было никакой гарантии безопасной передачи таких данных. Требовался метод передачи бинарных данных в виде обычного почтового сообщения ASCII. Расширения SMTP, введенные с момента разработки MIME, позволяют пересылать восьмибитные бинарные данные, но даже сегодня эти данные не всегда корректно передаются почтовой системой, будучи незакодированными.

MIME предоставляет пять схем кодирования для передачи данных (также можно добавлять новые схемы — на всякий случай). Самая простая

схема — обычные текстовые сообщения ASCII. Символы ASCII используют 7 бит и могут передаваться напрямую протоколом электронной почты, при условии, что строка не превышает 1000 символов.

Следующая по простоте схема аналогична предыдущей, но использует 8-битные символы, то есть все значения байта от 0 до 255 включительно. Сообщения в 8-битной кодировке также должны соблюдать правило о максимальной длине строки.

Далее идут сообщения в настоящей двоичной кодировке. К ним относятся произвольные двоичные файлы, которые не только используют все 8 бит, но и не соблюдают ограничение на 1000 символов в строке. К этой категории относятся исполняемые программные файлы. Сегодня почтовые серверы могут проверять, есть ли возможность переслать данные в бинарной (или 8-битной) кодировке, и если это расширение не поддерживается на обеих сторонах, данные будут передаваться в ASCII.

Кодировка бинарных данных в формате ASCII называется **64-символьной кодировкой (base64 encoding)**. При использовании данного метода группы по 24 бита разбиваются на четыре единицы по 6 бит, каждая из которых передается в виде обычного разрешенного ASCII-символа. В этой кодировке 6-битный символ 0 кодируется ASCII-символом `A`, 1 — ASCII-символом `B` и т. д. Затем следуют 26 строчных букв, затем 10 цифр и, наконец, `+` для кодирования 62 и `/` для 63. Последовательности `==` и `=` говорят о том, что последняя группа содержит только 8 или 16 бит соответственно. Символы перевода строки и возврата каретки игнорируются, поэтому их можно вставлять в любом месте закодированного потока символов, чтобы строки были достаточно короткими. Таким способом можно передать любой двоичный код, хотя и не слишком эффективно. Эта кодировка была крайне популярна до того, как стали широко применяться почтовые серверы, способные передавать бинарную информацию. Она часто встречается и сегодня.

Особый интерес представляет последний заголовок на илл. 7.14, `Content-Type:` (Тип содержания). В нем указан тип тела сообщения. Применение этого заголовка выходит далеко за пределы электронной почты. Например, контент, загружаемый из интернета, получает метку MIME-типа, что позволяет браузеру корректно отображать информацию. Так же обстоит дело с данными, которые передаются через потоковое мультимедиа и в реальном времени (например, в IP-телефонии).

Изначально в RFC 1521 были определены семь MIME-типов. У каждого из них есть один или несколько доступных подтипов. Подтип отделяется от типа косой чертой, например: `Content-Type: video/mpeg`. Впоследствии было добавлено более 2700 подтипов, а также два новых типа (`font` и `model`). Новые сущности добавляются постоянно, по мере появления новых типов контента.

Список утвержденных типов и подтипов поддерживается организацией IANA и расположен по адресу www.iana.org/assignments/media-types. Существующие типы, а также несколько примеров часто используемых подтипов показаны на илл. 7.15.

| Тип | Примеры подтипов | Описание |
|---|---|---|
| text | plain, html, xml, css | Текст в различных форматах |
| image | gif, jpeg, tiff | Изображения |
| audio | basic, mpeg, mp4 | Звуки |
| video | mpeg, mp4, quicktime | Видеофильмы |
| font | otf, ttf | Шрифты для форматирования текста |
| model | vrml | 3D-модель |
| application | onoctet-stream, pdf, javascript, zip | Данные, производимые приложениями |
| message | http, rfc822 | Инкапсулированное сообщение |
| multipart | mixed, alternative, parallel, digest | Комбинация нескольких типов |

**Илл. 7.15.** Типы содержания MIME и примеры подтипов

Назначение MIME-типов на илл. 7.15 вполне очевидно, за исключением, возможно, последнего. Тип `multipart` служит для передачи сообщений с несколькими вложениями разного MIME-типа.

## 7.2.4. Пересылка сообщений

Теперь, когда мы описали пользовательские агенты и сообщения электронной почты, пора разобраться в том, как агенты передачи сообщений доставляют их от отправителя получателю. Передача почты осуществляется с помощью протокола SMTP.

Проще всего установить транспортное соединение между компьютером-источником и компьютером-получателем, а затем передать по нему сообщение. Так изначально и работал протокол SMTP. Однако со временем возникли два разных способа его применения. Первый — подача почтового сообщения, шаг 1 в архитектуре e-mail на илл. 7.9. Так пользовательские агенты передают данные в почтовую систему для их дальнейшей отправки. Вторым способом является пересылка сообщений между агентами передачи сообщений (шаг 2 на илл. 7.9). Такая последовательность обеспечивает доставку почты на всем пути от отправляющего до получающего агента передачи сообщений за один шаг. Окончательная доставка происходит при участии других протоколов, которые мы опишем в следующем разделе.

В этом разделе мы расскажем об основах SMTP и механизме его расширения. Затем мы обсудим разные способы его использования для подачи почты и передачи сообщений.

### *SMTP и его расширения*

В интернете для доставки электронной почты компьютер-источник устанавливает TCP-соединение с портом 25 компьютера-получателя. Этот порт

прослушивается почтовым сервером, который поддерживает **простой протокол передачи электронной почты (Simple Mail Transfer Protocol, SMTP)**. Сервер проверяет безопасность входящих соединений, утверждает их и принимает сообщения для передачи. Если письмо невозможно доставить, отправителю возвращается отчет об ошибке, в котором содержится первая часть этого письма.

SMTP представляет собой простой ASCII-протокол. Это не недостаток, а его характерная особенность. Использование текста в формате ASCII позволяет легко модифицировать, тестировать и исправлять ошибки в протоколах. Они могут тестироваться отсылкой команд вручную, при этом записи сообщений легко читать. Сейчас так работает большинство интернет-протоколов прикладного уровня (например, HTTP).

Разберем простую передачу сообщений между почтовыми серверами, которые их доставляют. Установив TCP-соединение с портом 25, передающий компьютер, выступающий в роли клиента, ждет запроса принимающего компьютера, работающего в режиме сервера. Сервер начинает диалог с того, что отсылает текстовую строку с его идентификатором и информацией о том, готов ли он к приему почты. Если нет, клиент разрывает соединение и повторяет попытку позднее.

Если сервер готов принимать почту, клиент сообщает, от кого она поступила и кому предназначается. Если такой получатель существует на этом направлении, сервер дает клиенту добро на пересылку сообщения. Тогда клиент отправляет его, а сервер подтверждает его получение. Контрольные суммы не проверяются, так как TCP обеспечивает надежный байтовый поток. Если у отправителя есть еще почта, она также отсылается. После передачи всей почты в обоих направлениях соединение разрывается. Пример такого диалога представлен на илл. 7.16. Здесь строки, отправленные клиентом (то есть отправителем), снабжены префиксом `C:`. Строки, отправленные сервером (то есть получателем), снабжены префиксом `S:`.

Сначала клиент, естественно, отправляет серверу приветствие — `HELO`. Это наиболее удачный вариант сокращения слова *HELLO* до четырех символов. Зачем все эти команды нужно было сокращать до четырех символов, сейчас уже никто не помнит.

В нашем примере сообщение нужно отправить только одному получателю, поэтому используется только одна команда `RCPT` (от слова «recipient» — «получатель»). Использование этой команды несколько раз позволяет отослать одно сообщение ряду получателей. Каждая передача подтверждается или отвергается индивидуально. Даже если не получится передать сообщение некоторым получателям (из-за их отсутствия на этом направлении), сообщение все равно может быть доставлено по остальным адресам.

Наконец, несмотря на то что синтаксис четырехсимвольных команд строго определен, синтаксис ответов не столь строг. Ограничен только числовой код в начале строки. Все, что следует за этим кодом, может считаться комментарием и зависит от конкретной реализации протокола.

Базовый протокол SMTP хорошо работает, но имеет ряд ограничений. Прежде всего, он не предусматривает аутентификации. Это означает, что команда `FROM` в нашем примере может отобразить какой угодно адрес отправителя, что крайне удобно для рассылки спама. Еще одно ограничение протокола — он

позволяет передавать сообщения в кодировке ASCII, но не бинарные данные. Именно поэтому для передачи MIME-контента приходилось использовать кодировку base64. Однако применение base64 при передаче почты ведет к неэффективному использованию пропускной способности, что становится проблемой в случае больших сообщений. Третье ограничение SMTP заключается в том, что он отсылает сообщение в незашифрованном виде. То есть сообщения никак не защищены от посторонних глаз.

```
                S: 220 ee.uwa.edu.au - служба SMTP готова
C: HELO abcd.com
                S: 250 cs.uchicago.edu приветствует ee.uwa.edu.au
C: MAIL FROM: <alice@cs.uchicago.edu>
                S: 250 подтверждаю отправителя
C: RCPT TO: <bob@ee.uwa.edu.au>
                S: 250 подтверждаю получателя
C: DATA
                S: 354 Отправляем письмо, завершая его символом ".",
расположенным в отдельной строке
C: From: alice@cs.uchicago.edu
C: To: bob@ee.uwa.edu.au
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@ee.uwa.edu.au>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Земля обошла вокруг Солнца целое число раз
C:
C: Это преамбула. Пользовательский агент игнорирует ее. Хорошего дня.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/html
C:
C: <p>С днем рожденья тебя!
C: С днем рожденья тебя!
C: С днем рождения, дорогой <bold>Боб</bold>!
C: С днем рожденья тебя!
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:       access-type="anon-ftp";
C:       site="bicycle.cs.uchicago.edu";
C:       directory="pub";
C:       name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
                S: 250 сообщение принято
C: QUIT
                S: 221 ee.uwa.edu.au разрывает соединение
```

**Илл. 7.16.** Передача сообщения от alice@cs.uchicago.edu для bob@ee.uwa.edu.au

Чтобы справиться с этими и многими другими проблемами, связанными с передачей сообщений, к SMTP было добавлено расширение. Оно является обязательной частью стандарта RFC 5321. Использование SMTP с расширениями называется **расширенным SMTP (Extended SMTP, ESMTP)**.

Клиенты, желающие применить **ESMTP**, на первом этапе высылают `EHLO` вместо `HELO`. Если этот вариант отвергается, сервер работает с обычным SMTP, а пользователь должен идти по стандартному пути. Если `EHLO` принимается, сервер сообщает, какие расширения он поддерживает. После этого клиент может использовать любое из них. Несколько стандартных расширений показаны на илл. 7.17. В таблице даны ключевые слова, в том виде, в каком они используются в механизме расширения, и описание нового функционала. Более подробно рассматривать расширения мы не будем.

| Ключевое слово | Описание |
|---|---|
| AUTH | Аутентификация клиента |
| BINARYMIME | Сервер принимает бинарные сообщения |
| CHUNKING | Сервер принимает большие сообщения по частям |
| SIZE | Проверка размера сообщения перед попыткой отправки |
| STARTTLS | Переключение на безопасный канал (TLS; см. главу 8) |
| UTF8SMTP | Интернационализированный адрес |

**Илл. 7.17.** Некоторые расширения SMTP

Чтобы лучше понять, как работает SMTP и другие рассмотренные в этой главе протоколы, попробуйте поработать с ними самостоятельно. Для начала найдите компьютер, подключенный к интернету. В системе UNIX (или Linux) наберите в командной строке:

```
telnet mail.isp.com 25
```

подставив вместо mail.isp.com DNS-имя почтового сервера провайдера. На компьютерах с Windows сначала, возможно, потребуется установить и запустить программу telnet (или ее аналог). В результате выполнения этой команды будет установлено telnet-соединение (то есть соединение TCP) с портом 25 данного компьютера. Порт 25 используется протоколом SMTP (какие порты задействованы в других стандартных протоколах, см. на илл. 6.34). В ответ на введенную команду вы получите что-то вроде этого:

```
Trying 192.30.200.66...
Connected to mail.isp.com
Escape character is '^]'.
220 mail.isp.com Smail #74 ready at Thu, 25 Sept 2019 13:26 +0200
```

Первые три строки приходят от telnet; они описывают, что делает программа. Последняя строка — от сервера SMTP удаленного компьютера; в ней говорится о готовности к общению с вашим компьютером и приему почты. Чтобы узнать о доступных командах, наберите

```
HELP
```

Начиная с этого момента возможен обмен последовательностями команд, показанными на илл. 7.16, если сервер готов принимать от вас почтовые сообщения. Вам придется вводить команды достаточно быстро, поскольку соединение может прерваться, если оно слишком долго не используется. Кроме того, не каждый почтовый сервер разрешает устанавливать telnet-соединение с неизвестным компьютером.

### *Подача почтовых сообщений*

Первоначально пользовательские агенты запускались на том же компьютере, что и агенты передачи сообщений. В этом случае все, что необходимо для отправки сообщения, — чтобы пользовательский агент связался с локальным почтовым сервером, используя только что описанный диалог. Однако этот вариант уже не так широко распространен.

Пользовательские агенты часто работают на ноутбуках, домашних компьютерах и мобильных телефонах, а они не всегда подключены к интернету. Агенты передачи сообщений работают на серверах провайдеров и крупных компаний, которые имеют постоянный выход в интернет. Это означает, что у пользовательского агента в Бостоне может возникнуть необходимость обратиться к его обычному почтовому серверу в Сиэтле, чтобы передать сообщение, потому что пользователь отправился в путешествие.

Эта удаленная коммуникация не является проблемой сама по себе. Именно для подобных случаев был разработан протокол TCP/IP. Однако провайдер (или компания) обычно без энтузиазма относится к тому, что некий удаленный пользователь может подавать сообщения на их почтовый сервер для доставки куда-то еще. Этот провайдер (или компания) держит сервер не на общественных началах. Кроме того, такой вид **открытой почтовой станции (open mail relay)** привлекает спамеров, поскольку у них есть возможность скрыть настоящего отправителя и таким образом затруднить идентификацию сообщения как спама.

Учитывая эти особенности, SMTP обычно используется для подачи писем с расширением AUTH. Оно позволяет серверу проверять данные отправителя (имя пользователя и пароль) для подтверждения того, что сервер должен предоставить почтовое обслуживание.

Есть еще несколько отличий в том, как SMTP используется при подаче почты. Например, может использоваться порт 587, а не порт 25, и SMTP-сервер может проверять и исправлять формат сообщений, отправленных пользовательским агентом. Чтобы больше узнать об использовании SMTP при подаче писем, вы можете ознакомиться со стандартом RFC 4409.

### *Физическая передача*

Когда отправляющий агент передачи почты получает сообщение от пользовательского агента, он доставляет его получающему агенту передачи, используя SMTP. При этом источник использует адрес получателя. Взгляните на сообщение, отправляемое на адрес bob@ee.uwa.edu.au (на илл. 7.16). На какой почтовый сервер оно должно быть доставлено?

Чтобы правильно выбрать почтовый сервер, нужно обратиться к системе DNS. В предыдущем разделе было упомянуто, что DNS использует различные типы записей, включая MX («mail exchanger» — «обмен почтовыми сообщениями»). В данном случае выполняется DNS-запрос для записей MX домена ee.uwa.edu.au. Запрос возвращает упорядоченный список имен и IP-адресов одного или нескольких почтовых серверов.

Далее отправляющий агент создает TCP-соединение с IP-адресом почтового сервера на порте 25, чтобы связаться с принимающим агентом. Для передачи сообщения используется SMTP. Затем принимающий агент помещает сообщения для пользователя *bob* в нужный почтовый ящик, чтобы позднее Боб мог их прочитать. Этот шаг локальной доставки может включать перемещение сообщения между компьютерами при наличии разветвленной почтовой инфраструктуры.

В этом случае почта проходит от начального до конечного агента передачи за один шаг. На этапе пересылки нет промежуточных серверов. В то же время процесс доставки может повторяться несколько раз. Один из примеров мы уже привели (когда агент передачи сообщений применяет список рассылки). Сообщение приходит для списка адресатов, а затем рассматривается как письмо для каждого из перечисленных получателей и рассылается по индивидуальным адресам.

Еще один пример перенаправления почты выглядит следующим образом. Допустим, Боб окончил Массачусетский технологический институт и доступен по еще одному адресу: bob@alum.mit.edu. Вместо того чтобы собирать почту с разных аккаунтов, Боб может сделать так, чтобы почта, приходящая на данный адрес, переправлялась на bob@ee.uwa.edu. В этом случае сообщения, отправленные на bob@alum.mit.edu, будут доставляться дважды. Сначала они будут отсылаться на почтовый сервер alum.mit.edu, а затем — на сервер ee.uwa.edu.au. Каждый из этих шагов является полноценной отдельной доставкой, если мы рассматриваем их с точки зрения агентов передачи почтовых сообщений.

## 7.2.5. Окончательная доставка

Наше почтовое сообщение почти доставлено. Оно прибыло в почтовый ящик Боба. Осталось только передать копию сообщения пользовательскому агенту Боба, чтобы оно могло отобразиться. Это шаг 3 в архитектуре на илл. 7.9. Когда пользовательский агент и агент передачи почты работали на одном и том же компьютере и представляли собой всего лишь разные процессы, эта задача была несложной. Агент передачи почты просто записывал новые сообщения в конец файла почтового ящика, а пользовательский агент просто проверял файл почтового ящика на наличие новых сообщений.

Сегодня пользовательский агент на ПК, ноутбуке или мобильном телефоне обычно не размещается на почтовом сервере компании или провайдера, а в случае Gmail он точно расположен на другом устройстве. Пользователи хотят иметь доступ к своей почте вне зависимости от того, где они находятся: с рабочего или домашнего компьютера, с ноутбука в командировках или из интернет-кафе. Им нужна возможность работы без постоянного соединения с сетью, с периодическим подключением к интернету для приема и отправки сообщений. Более того, пользователь может запускать несколько агентов в зависимости от того, каким компьютером ему удобно пользоваться в данный момент. Несколько пользовательских агентов даже могут работать одновременно.

В данном случае пользовательский агент должен отображать содержимое почтового ящика и позволять работать с ним удаленно. Для этого может задействоваться несколько разных протоколов, но только не SMTP. SMTP основан на механизме push. Для передачи сообщения ему необходимо соединение с удаленным сервером. Таким образом, окончательная доставка не может быть осуществлена по двум причинам: из-за того, что почтовый ящик должен храниться на агенте передачи почты, и из-за того, что пользовательский агент может быть не подключен к интернету, когда SMTP пытается передать сообщение.

### *Протокол IMAP*

Одним из главных современных протоколов для окончательной доставки является **протокол доступа к электронной почте (Internet Mail Access Protocol, IMAP )**. Четвертая версия этого протокола определена в спецификации RFC 3501 и ее многочисленных обновлениях. Чтобы использовать IMAP, почтовый сервер запускает IMAP-сервер, который проверяет порт 143. Пользовательский агент запускает IMAP-клиент. Этот клиент соединяется с сервером и начинает передавать команды из списка, приведенного на илл. 7.18.

Сначала клиент устанавливает защищенное соединение, если это возможно (для соблюдения конфиденциальности сообщений и команд), а затем вводит логин и пароль или как-то иначе авторизуется на сервере. После входа в систему можно воспользоваться множеством команд, чтобы просматривать папки и сообщения или их фрагменты, помечать письма галочками для дальнейшего удаления и сортировать их по папкам. Обратите внимание, что во избежание неразберихи мы в этой главе используем слово *«папка» (folder)*, имея в виду, что почтовый ящик состоит из нескольких папок. Однако в спецификации IMAP вместо папки используется термин *«почтовый ящик» (mailbox)*. Получается, что у пользователя есть много ящиков IMAP, каждый из которых он видит как папку.

Протокол IMAP предоставляет разнообразные функции, например, можно упорядочить почту не по порядку ее поступления, а по атрибутам писем («показать первое письмо от Алисы»). На сервере также могут быть инструменты поиска сообщений по заданным критериям; клиент увидит только выбранные письма.

| Команда | Описание |
|---|---|
| CAPABILITY | Перечислить возможности сервера |
| STARTTLS | Установить защищенное соединение (TLS, см. главу 8) |
| LOGIN | Войти на сервер, используя имя пользователя и пароль |
| AUTHENTICATE | Авторизоваться иным способом |
| SELECT | Выбрать папку |
| EXAMINE | Выбрать папку, предназначенную только для чтения |
| CREATE | Создать папку |
| DELETE | Удалить папку |
| RENAME | Переименовать папку |
| SUBSCRIBE | Добавить папку к активному набору |
| UNSUBSCRIBE | Удалить папку из активного набора |
| LIST | Перечислить доступные папки |
| LSUB | Перечислить активные папки |
| STATUS | Узнать статус папки |
| APPEND | Добавить сообщение в папку |
| CHECK | Просмотреть состояние выбранной папки (что именно входит в понятие «состояние», зависит от конкретной реализации сервера). Создать контрольную точку для папки |
| FETCH | Просмотреть сообщения в папке |
| SEARCH | Найти сообщения, находящиеся в папке |
| STORE | Изменить метки сообщения |
| COPY | Сделать копию сообщения в папке |
| EXPUNGE | Удалить отмеченные сообщения |
| UID | Вызвать команды, используя уникальные идентификаторы |
| NOOP | Ничего не делать |
| CLOSE | Удалить помеченные сообщения и закрыть папку |
| LOGOUT | Выйти из системы и закрыть соединение |

**Илл. 7.18.** Команды IMAP (версия 4)

IMAP — это улучшенная версия более раннего протокола окончательной доставки — **протокола почтового отделения версии 3 (Post Office Protocol, version 3, POP3)**, описанного в RFC 1939. Протокол POP3 проще, но он поддерживает не так много функций и является менее безопасным при типичном использовании. Обычно почта загружается на компьютер с пользовательским агентом и не остается на почтовом сервере. Это облегчает жизнь серверу, но усложняет ее пользователю. Читать почту с нескольких устройств становится сложнее, а если компьютер с пользовательским агентом сломается, вся почта будет безвозвратно утеряна. Тем не менее POP3 все еще используется.

Кроме того, могут использоваться и частные протоколы, так как протокол работает между почтовым сервером и пользовательским агентом, который может разрабатываться одной компанией — производителем софта. Примером почтовой системы с частным протоколом является Microsoft Exchange.

### *Веб-почта*

Популярной альтернативой IMAP и SMTP в деле предоставления почтовых услуг стало использование веб-интерфейса для отправки и получения сообщений. Широко применяются такие системы **веб-почты (Webmail)**, как Google Mail, Microsoft Hotmail и Yahoo! Mail. Веб-служба электронной почты — это пример программного обеспечения (в данном случае почтового агента пользователя), которое предоставляется как интернет-служба.

В такой системе провайдер управляет почтовыми серверами как обычно, чтобы принимать сообщения для пользователей при помощи SMTP через порт 25. Однако пользовательский агент работает по-другому. Он является не отдельной программой, а пользовательским интерфейсом, который предоставляется через веб-страницы. То есть пользователи могут использовать любой понравившийся им браузер для доступа к своей почте и отправки новых сообщений.

Когда пользователь заходит на страницу Gmail, он видит форму, где нужно ввести учетное имя и пароль. Пароль и учетное имя отправляются на сервер для их подтверждения. Если вход в систему осуществлен корректно, сервер находит почтовый ящик пользователя и строит веб-страницу, на которой отображается содержимое ящика. Эта веб-страница отсылается на браузер клиента.

Пользователь может кликать по многочисленным элементам страницы, отображающей его почтовый ящик, так что сообщения можно читать, удалять и т. д. Чтобы интерфейс лучше откликался на действия пользователя, веб-страницы часто используют код на JavaScript. Они запускаются локально на компьютере пользователя в ответ на определенные события (например, клики мышкой) и могут загружать на сервер и подгружать с сервера сообщения в фоновом режиме, чтобы подготовить к показу следующее письмо или подать новое. В этой модели подача почты происходит при использовании обычных веб-протоколов с отправкой данных по URL. Веб-сервер помещает сообщения в обычную, уже описанную нами систему доставки почты. Для безопасности также могут использоваться стандартные веб-протоколы. Они заботятся о шифровании веб-страниц, а не о том, является ли содержание страницы почтовым сообщением.

## 7.3. ВСЕМИРНАЯ ПАУТИНА

Всемирная паутина (World Wide Web), или просто «веб», — это архитектура, которая является основой для доступа к соединенному ссылками контенту на миллионах компьютеров по всему интернету. За 10 лет своего существования из средства координации разработки экспериментов по физике высоких энергий в Швейцарии она превратилась в то, что миллионы людей с разными интересами теперь называют интернетом. Невероятная популярность Всемирной паутины связана с тем, что она проста в использовании даже для новичка. Кроме того, при помощи многофункционального графического интерфейса она предоставляет доступ к гигантскому количеству информации практически по любому вопросу, от африканских муравьедов до яшмового фарфора.

Всемирная паутина была создана в 1989 году в Европейской организации по ядерным исследованиям ЦЕРН (Conseil Européen pour la Recherche Nucléaire, CERN) в Швейцарии. Изначальная цель заключалась в том, чтобы наладить совместную работу больших групп, участники которых были разбросаны по разным странам и часовым поясам. Исследователям требовался доступ к постоянно меняющимся отчетам, чертежам, рисункам, фотографиям и другим документам, появляющимся по ходу ведения экспериментов в области физики элементарных частиц. Предложение создать паутину из связанных друг с другом документов пришло от физика ЦЕРН Тима Бернерса-Ли (Tim Berners-Lee). Первый (текстовый) прототип был готов к эксплуатации 18 месяцев спустя. Публичная демонстрация, произведенная в декабре 1991 года на конференции Hypertext'91 в Сан-Антонио, штат Техас, привлекла внимание ряда других ученых, в результате чего Марк Андриссен (Marc Andreessen) из Иллинойсского университета начал разработку первого графического браузера — Mosaic. Программа увидела свет в феврале 1993 года.

Остальное, как говорится, уже история. Браузер Mosaic стал настолько популярным, что год спустя Марк Андриссен решил основать собственную компанию Netscape Communications Corp.. Компания занималась разработкой программного обеспечения для Всемирной паутины. В течение последующих трех лет между Netscape Navigator и Internet Explorer от Microsoft развернулась настоящая «война браузеров». Разработчики с обеих сторон пытались захватить новый рынок, лихорадочно добавляя в свои программы как можно больше функций (а заодно и ошибок), чтобы превзойти соперника.

На протяжении 1990-х и 2000-х годов объем веб-контента рос в геометрической прогрессии, пока число веб-сайтов не достигло миллионов, а число веб-страниц — миллиардов. Некоторые из них стали невероятно популярными. Эти сайты и стоящие за ними компании сегодня в значительной мере определяют ландшафт современного интернета. В их числе книжный магазин Amazon (1994), торговая площадка eBay (1995), поисковая система Google (1998) и социальная сеть Facebook (2004). Период в 2000-х, когда стоимость многих веб-компаний могла взлететь до сотен миллионов долларов за одну ночь, только чтобы обрушиться практически на следующий день (когда оказывалось, что их создание было просто пиар-ходом), даже имеет название — **«эра доткомов» (dot com era)**. Новые идеи в Сети и сейчас могут мгновенно озолотить своих авторов. Многие

из них исходят от студентов. Например, Марк Цукерберг (Mark Zuckerberg) был студентом Гарварда, когда придумал и запустил проект Facebook, а Сергей Брин (Sergey Brin) и Ларри Пейдж (Larry Page) учились в Стэнфорде, когда основали Google. Возможно, вы будете следующим.

В 1994 году ЦЕРН и Массачусетский технологический институт (Massachusetts Institute of Technologies, MIT) подписали соглашение об основании **Консорциума Всемирной паутины (World Wide Web Consortium, W3C)**. Цель этой организации — дальнейшее развитие Всемирной паутины, стандартизация протоколов и поощрение взаимодействия между отдельными сайтами. Директором консорциума стал Бернерс-Ли. С тех пор к нему присоединились несколько сотен университетов и компаний. Хотя о Всемирной паутине написано уже очень много книг, получить самую свежую информацию о ней проще всего на официальном сайте. Домашняя страница W3C находится по адресу http://www.w3.org. Здесь заинтересованный читатель найдет ссылки на другие страницы, содержащие информацию обо всех документах консорциума и о его деятельности.

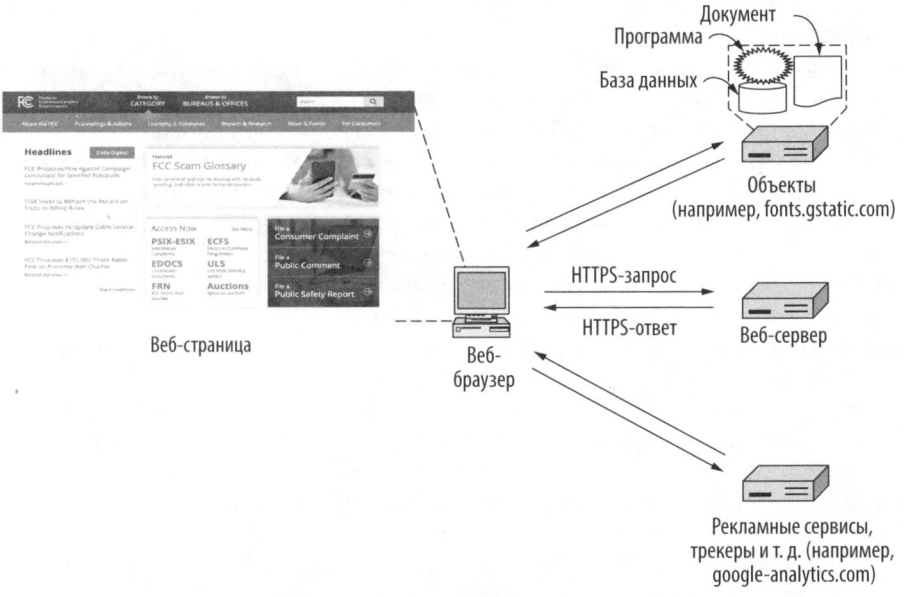
## 7.3.1. Представление об архитектуре

С точки зрения пользователя, Всемирная паутина состоит из огромного количества контента в виде **веб-страниц (Web pages)**. На каждой странице обычно есть ссылки на сотни других объектов, которые могут размещаться на веб-сервере в любой точке мира. Раньше такой объект мог быть еще одним текстом или картинкой, а в настоящее время, помимо этого, существует широкий спектр других объектов, таких как рекламные объявления и скрипты отслеживания. Веб-страница также может содержать **ссылки (links)** на другие страницы, на которые можно перейти, кликнув по такой ссылке. Этот процесс можно повторять бесконечно. Идея использования страниц, направляющих друг на друга, — так называемого **гипертекста (hypertext)** — была впервые предложена в 1945 году Ванневаром Бушем (Vannevar Bush), профессором электротехники Массачусетского технологического института. Это было задолго до возникновения интернета и даже до появления коммерческих компьютеров. (Хотя в некоторых университетах уже были созданы первые грубые прототипы, которые занимали целые помещения и потребляли уйму энергии (сравнимо с небольшим заводом), обладая при этом вычислительной мощностью в миллионы раз меньшей, чем современные умные часы.)

Для просмотра страниц обычно используется специальная программа, называемая **браузером (browser)**. Примерами популярных браузеров могут служить Brave, Chrome, Edge, Firefox, Opera и Safari. Браузер предоставляет пользователю запрашиваемую страницу, интерпретирует ее контент и выводит должным образом отформатированные страницы на экран. Контент может представлять собой сочетание текста, изображений и команд форматирования и выглядеть как обычный документ, видео или программа с графическим интерфейсом.

Пример веб-страницы с большим количеством объектов приведен на илл. 7.19. Это страница сайта Федеральной комиссии по связи США. Она содержит некий текст и графические элементы (большая часть которых является нечитаемой

из-за небольших размеров иллюстрации). Многие элементы содержат ссылки на другие страницы. Загружаемая браузером **индексная страница (index page)** обычно содержит указания для браузера относительно того, где находятся другие необходимые объекты, и того, каким образом и в какой части страницы их нужно отобразить.




**Илл. 7.19.** Загрузка и отображение веб-страницы требует ряда HTTP/HTTPS-запросов к нескольким серверам

Строки текста, иконки, изображения и прочие элементы, представляющие собой ссылки на другие страницы, называются **гиперссылками (hyperlinks)**. Чтобы перейти по ссылке, пользователь стационарного компьютера или ноутбука должен навести указатель мыши на область страницы с гиперссылкой (что приведет к изменению формы указателя) и выполнить щелчок. Пользователю смартфона или планшета потребуется коснуться ссылки. Выполняя переход по ссылке, мы просто сообщаем браузеру, что он должен загрузить другую страницу. На заре Всемирной паутины ссылки выделялись подчеркиванием и другим цветом, чтобы их можно было отличить от остального текста. Сегодня создатели веб-страниц используют **таблицы стилей (style sheets)**, позволяющие управлять внешним видом многих элементов страницы, включая гиперссылки, так что они могут выглядеть как того пожелает дизайнер сайта. Их внешний вид даже может меняться динамически, при наведении указателя мыши. Задача визуального выделения ссылок для обеспечения удобной навигации по сайту лежит на его создателях.

Заинтересовавшись определенной статьей, посетитель страницы может кликнуть по нужной области, чтобы браузер загрузил и отобразил новую страницу.

При этом на первой странице могут быть ссылки на десятки других страниц. Каждая из них может содержать контент, размещенный на том же компьютере, что и первая страница, либо на компьютерах в другой части мира. Для пользователя это не заметно. Как правило, браузер загружает любые объекты, к которым привязаны ссылки, выбираемые пользователем, обеспечивая свободное перемещение между устройствами в ходе просмотра контента.

Браузер отображает веб-страницу на клиентском компьютере, отсылая запрос на один или несколько серверов, которые отвечают, передавая контент страницы. Запросно-ответный протокол для отображения страниц представляет собой простейший текстовый протокол, работающий поверх TCP (подобно SMTP), — **HTTP**. Его защищенная версия **HTTPS** является основным способом доставки контента в современном интернете. В качестве контента может выступать простой документ, который считывается с диска, или результат работы программы или запроса, отправленного в базу данных. Страница называется **статичной (static page)**, если это документ, который всегда отображается одинаково. Если же она создается по требованию программы или сама содержит какую-либо программу, то это **динамическая страница (dynamic page)**.

Контент динамической страницы может быть разным при каждом вызове. К примеру, главная страница интернет-магазина может различаться для каждого посетителя. Если клиент книжного магазина в прошлом покупал детективы, на главной странице он, скорее всего, увидит новые книги этого жанра, а человек, интересующийся рецептами, обнаружит на ней новые кулинарные книги. Мы вскоре расскажем, как веб-сайты следят за тем, что нравится покупателям. Если кратко, это обеспечивается с помощью файлов cookie.

Для загрузки веб-страницы браузер связывается с рядом серверов. Контент индексной страницы может быть загружен непосредственно из файлов, размещенных в домене fcc.gov. Дополнительный контент, например встроенное видео, может размещаться на отдельном сервере (по-прежнему в домене fcc.gov, но, возможно, на инфраструктуре, предназначенной для хранения такого контента). Индексная страница также может содержать ссылки на другие объекты, например невидимые пользователю скрипты отслеживания или размещенные на сторонних серверах рекламные объявления. Браузер загружает все эти объекты, скрипты и т. д. и собирает их для пользователя в виде единого представления страницы.

Представление страницы предполагает обработку, тип которой зависит от контента. Кроме отображения текста и графических элементов, может понадобиться воспроизвести видеофайл или запустить скрипт, в котором прописан отдельный пользовательский интерфейс, являющийся частью страницы. В нашем случае сервер fcc.gov предоставляет главную страницу, fonts.gstatic.com — дополнительные объекты (к примеру, шрифты), а google-analytics.com — невидимую пользователю аналитику посещаемости сайта. Использование трекеров и обеспечение конфиденциальности в интернете мы обсудим чуть позже в этой главе.

### *Сторона клиента*

Теперь мы подробнее рассмотрим весь процесс со стороны веб-браузера (см. илл. 7.19). По сути, браузер — это программа, которая отображает веб-страницу

и реагирует на пользовательские запросы «перехода» к ее элементам. При выборе элемента браузер переходит по гиперссылке и извлекает объект, на который указал пользователь (с помощью щелчка мыши или нажатия на ссылку на экране мобильного устройства).

Как только была создана Всемирная паутина, сразу стало очевидным, что наличие ссылок с одних страниц на другие требует создания механизма именования и расположения страниц. Рассмотрим три самых важных вопроса, на которые нужно ответить, прежде чем отобразить выбранную страницу:

1. Как называется страница?
2. Где она расположена?
3. Как получить доступ к ней?

Если бы каждой странице было приписано уникальное имя, не было бы никакой неоднозначности в их идентификации. Но это не решило бы проблему. Проведем параллель между страницами и людьми. В США практически у каждого взрослого гражданина есть номер социального страхования, который может служить уникальным идентификатором, ведь два разных человека не могут иметь одинаковый номер. Однако если у вас есть только номер социального страхования, вы не сможете выяснить адрес его владельца и уж тем более определить, на каком языке к нему следует обращаться: на английском, испанском или китайском. Во Всемирной паутине возникают примерно те же проблемы.

Найденный способ идентификации страниц решил все три проблемы. Каждой странице приписывается **унифицированный указатель ресурса (Uniform Resource Locator, URL)**, который представляет собой своего рода «имя» страницы во Всемирной паутине. URL-адрес содержит три элемента: протокол (который также называют **схемой** – **scheme**), DNS-имя устройства, на котором расположена страница, и уникальный для каждой страницы путь (файл для чтения или программу для запуска на компьютере). В общем случае у пути есть иерархическое имя, которое моделирует структуру каталогов файлов. При этом интерпретация пути – это работа сервера; действительная структура каталогов может и не отображаться.

В качестве примера возьмем URL-адрес страницы на илл. 7.19:

https://fcc.gov/

Этот URL-адрес включает в себя три элемента: протокол (https), DNS-имя хоста (fcc.gov) и имя пути (/), которое веб-сервер часто воспринимает как некоторый индексный объект по умолчанию.

Когда пользователь выбирает гиперссылку, браузер выполняет ряд действий для загрузки той страницы, на которую она указывает. Рассмотрим последовательность действий при активации ссылки в нашем примере:

1. Браузер определяет URL-адрес (исходя из того, какой элемент страницы выбрал пользователь).
2. Браузер запрашивает у службы DNS IP-адрес сервера fcc.gov.
3. DNS выдает в качестве ответа адрес 23.1.55.196.

4. Браузер устанавливает TCP-соединение с этим IP-адресом; поскольку при этом применяется HTTPS, защищенная версия HTTP, TCP-соединение по умолчанию устанавливается с портом 443 (а не со стандартным портом 80 протокола HTTP, который сегодня используется все реже).
5. Браузер отправляет HTTPS-запрос на получение страницы //, которую веб-сервер обычно интерпретирует как некую индексную страницу (например, index.html, index.php и т. п., как указано в конфигурации веб-сервера хоста fcc.gov).
6. Сервер отправляет страницу как HTTPS-ответ, например, в виде файла /index.html, если таковой определен как индексный объект по умолчанию.
7. Если страница содержит URL-адреса, которые нужно отобразить, то браузер получает их таким же способом. В нашем случае эти URL-адреса содержат ряд встроенных изображений, также загружаемых с данного сервера, встроенные объекты с сайта gstatic.com и скрипт с сайта google-analytics.com (а также с ряда других доменов, которые здесь не показаны).
8. Браузер отображает страницу /index.html в том виде, в каком она представлена на илл. 7.19.
9. Если в течение некоторого времени на те же серверы не поступает других запросов, TCP-соединения обрываются.

Многие браузеры отображают текущее выполняемое ими действие в строке состояния внизу экрана. Это позволяет пользователю понять причину низкой производительности: например, не отвечает служба DNS или сервер либо просто сеть сильно перегружается при передаче страницы.

Получить более детальное представление о выполнении веб-страницы можно с помощью так называемой **каскадной диаграммы (waterfall diagram)** (илл. 7.20).

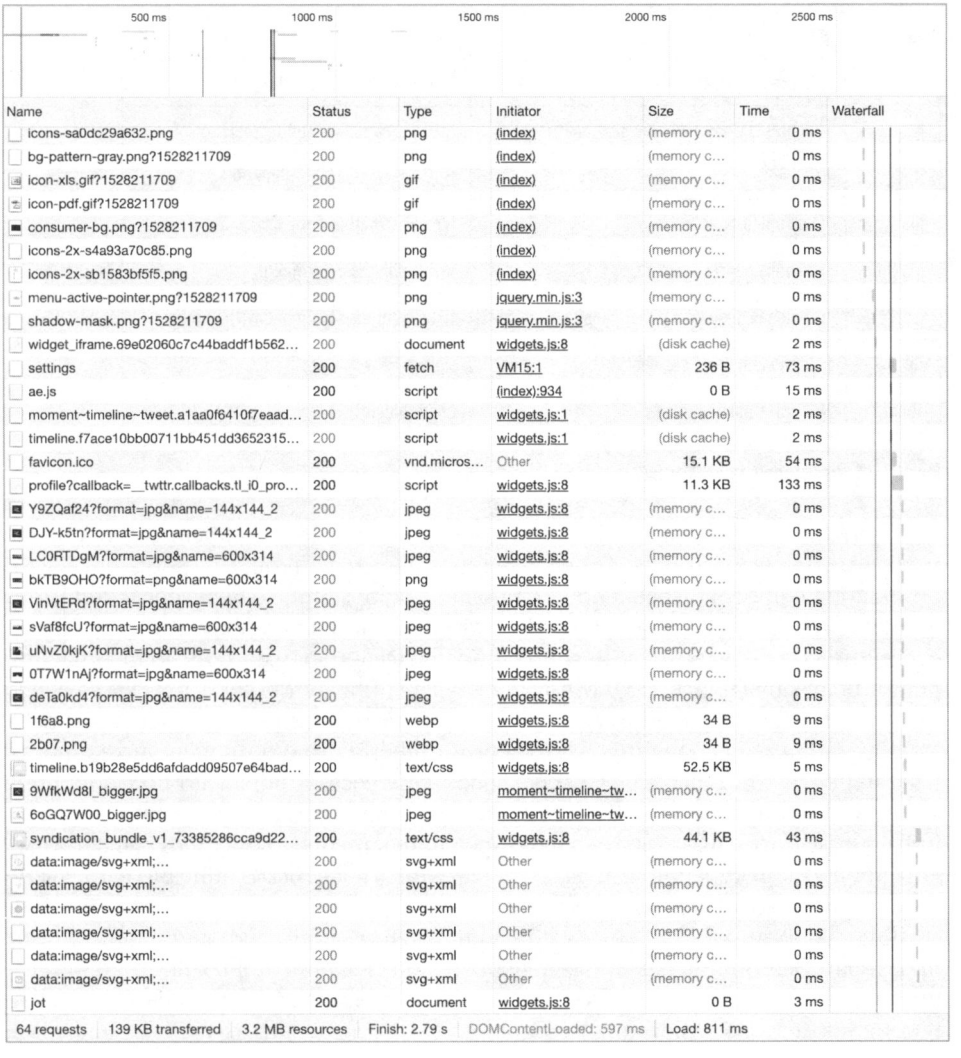
Каскадная диаграмма показывает список всех объектов, отображаемых браузером в ходе загрузки страницы (в данном случае имеется 64 объекта, но многие страницы содержат сотни объектов), временные зависимости, связанные с загрузкой каждого запроса, и операции, связанные с загрузкой каждой страницы (включая DNS-поиск, установление TCP-соединения, непосредственное скачивание контента и т. д.). Такая диаграмма может многое рассказать о поведении браузера. Например, сколько параллельных соединений он устанавливает с тем или иным сервером и используются ли они повторно; какая часть времени тратится на DNS-поиск, а какая — на непосредственную загрузку объектов. С помощью каскадной диаграммы можно определить и другие потенциальные узкие места производительности.

На URL-адреса не накладывается никаких ограничений по использованию браузером различных протоколов для извлечения разных видов ресурсов. На самом деле для других протоколов был определен ряд дополнительных видов URL-адресов. Наиболее распространенные из них представлены на илл. 7.21 в слегка упрощенном виде.

Кратко пройдемся по этому списку. Протокол http — «родной язык» Всемирной паутины, на котором «разговаривают» веб-серверы. Мы подробно

рассмотрим его в этом разделе, уделяя особое внимание его защищенной версии HTTPS. Сегодня она чаще всего используется для доставки объектов в интернете.

Протокол ftp применяется для доступа к FTP-файлам. FTP появился еще до Всемирной паутины и используется уже более четырех десятилетий. Интернет позволяет легко получать файлы, расположенные на различных FTP-серверах по всему миру, предоставляя простой интерактивный интерфейс вместо традиционного интерфейса командной строки. Упрощенный доступ к информации — одна из причин поразительного роста интернета.




**Илл. 7.20.** Каскадная диаграмма для сайта fcc.gov

Получить доступ к локальному файлу как к веб-странице можно, используя протокол file или просто написав его имя. Для применения этого способа не нужен сервер. Конечно, это работает только для локальных файлов, но не для удаленных.

Протокол mailto не загружает веб-страницы, но зато позволяет отправлять почту через браузер. Когда пользователь переходит по ссылке mailto, большинство браузеров запускает пользовательского агента, предоставляющего форму для написания электронного письма с уже заполненным полем адреса.

| Имя | Назначение | Пример |
|---|---|---|
| http | Гипертекст (HTML) | https://www.ee.uwa.edu/~rob/ |
| https | Гипертекст с обеспечением безопасности | https://www.bank.com/accounts/ |
| ftp | FTP | ftp://ftp.cs.vu.nl/pub/minix/README |
| file | Локальный файл | file:///usr/nathan/prog.c |
| mailto | Отправка почты | mailto:JohnUser@acm.org |
| rtsp | Потоковая передача мультимедиа | rtsp://youtube.com/montypython.mpg |
| sip | Мультимедийные вызовы | sip:eve@adversary.com |
| about | Информация браузера | about:plugins |

**Илл. 7.21.** Некоторые стандартные схемы URL

Протоколы rtsp и sip предназначены для установления сеансов передачи мультимедийных потоков, а также аудио- и видеозвонков.

Наконец, протокол about предоставляет информацию о браузере. Например, если вы пройдете по ссылке about:plugins, большинство браузеров отобразит страницу со списком MIME-типов, обрабатываемых ими с помощью расширений браузера (плагинов). Многие браузеры предоставляют в разделе about: очень интересные сведения. Так, в браузере Firefox по ссылке about:telemetry можно найти всю собранную браузером информацию о производительности и активности пользователя. Ссылка about:preferences отображает пользовательские настройки, а about:config — подробные сведения о конфигурации браузера, в том числе о том, производит ли браузер поиск по протоколу DoH (и с какими доверенными рекурсивными распознавателями), как было описано в предыдущем разделе, посвященном DNS.

Таким образом, URL-адреса можно применять не только для навигации по просторам Всемирной паутины, но и для использования старых протоколов (например, FTP и электронной почты), новых протоколов (для аудио- и видеоданных), а также для обеспечения удобного доступа к локальным файлам и информации браузера. Благодаря этому пропадает необходимость в любых специальных интерфейсных программах для вышеперечисленных целей, а интернет-доступ практически полностью интегрируется в одну программу: веб-браузер. Если бы

мы не знали, что эту идею предложил британский физик, работавший в составе многонациональной исследовательской лаборатории в Швейцарии, то можно было бы подумать, что этот план разработан в рекламном отделе какой-нибудь софтверной компании.

### Сторона сервера

О стороне клиента сказано уже достаточно много. Поговорим теперь о стороне сервера. Как мы уже знаем, когда пользователь вводит URL-адрес или кликает по гиперссылке, браузер производит синтаксический разбор URL-адреса и интерпретирует часть, заключенную между https:// и следующей косой чертой, как искомое DNS-имя. Вооружившись IP-адресом сервера, браузер устанавливает TCP-соединение с его портом 443. Затем отсылается команда, содержащая оставшуюся часть URL-адреса (путь к странице на данном сервере). Сервер возвращает браузеру ту страницу, которую он должен отобразить.

Если не вдаваться в детали, простой веб-сервер работает примерно так, как показано на илл. 6.6. Этому серверу передается имя файла, который нужно найти и отправить по сети. В обоих случаях в основном цикле сервер выполняет следующие действия:

1. Принимает входящее TCP-соединение от клиента (браузера).
2. Получает путь к странице, являющийся именем запрашиваемого файла.
3. Получает файл (с диска).
4. Высылает содержимое файла клиенту.
5. Разрывает TCP-соединение.

Современные веб-серверы обладают более широкими возможностями, однако в основе их работы лежат именно перечисленные шаги, которые предпринимаются в случае запроса контента из файла. Если контент динамический, третий шаг может быть заменен выполнением программы (указанной в пути), которая генерирует и возвращает определенный контент.

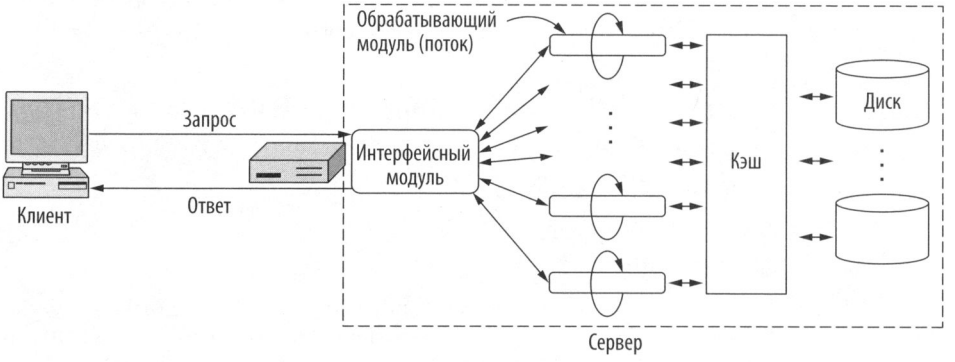
Если нужно обрабатывать сотни и даже тысячи запросов в секунду, веб-серверы работают иначе. Одна из проблем простой реализации состоит в том, что доступ к файлам часто становится узким местом производительности. Чтение с диска идет слишком медленно по сравнению с работой программы, и одни и те же файлы могут считываться несколько раз из-за вызовов операционной системы. Другая проблема заключается в том, что за раз может быть обработан только один запрос. При запросе большого файла обработка других запросов будет блокироваться до окончания его передачи.

Очевидное решение этой проблемы (применяемое на всех веб-серверах) состоит в том, чтобы кэшировать в памяти $n$ последних запрошенных файлов или определенное количество гигабайтов контента. Прежде чем обратиться за файлом к диску, сервер проверяет содержимое кэша. Если файл есть в кэше, его можно сразу выдать клиенту, не обращаясь к диску. Конечно, для эффективного кэширования требуются большие объемы памяти и дополнительное время на

проверку кэша и управление его содержимым, но суммарный выигрыш во времени почти всегда оправдывает эти издержки.

Параллельную обработку запросов также можно осуществить с помощью **многопоточных (multithreaded) серверов**. В одной из реализаций такого подхода сервер состоит из интерфейсного модуля, принимающего все входящие запросы, и $k$ обрабатывающих модулей (илл. 7.22). Все $k + 1$ потоков принадлежат одному и тому же процессу, поэтому у обрабатывающих модулей есть доступ к кэшу в адресном пространстве процесса. Когда приходит запрос, интерфейсный модуль принимает его и создает краткую запись с его описанием. Затем запись передается одному из обрабатывающих модулей.

Сначала обрабатывающий модуль проверяет, нет ли запрашиваемого объекта в кэше. При наличии этого объекта в кэше он обновляет запись, включая в нее указатель на этот файл. Если искомого файла в кэше нет, обрабатывающий модуль обращается к диску и считывает файл в кэш (при этом, возможно, удаляя некоторые хранящиеся там файлы, чтобы освободить место). Считанный с диска файл помещается в кэш и отсылается клиенту.




**Илл. 7.22.** Многопоточный веб-сервер с интерфейсным модулем и набором обрабатывающих модулей

Преимущество такого подхода заключается в том, что пока один или несколько обрабатывающих модулей заблокированы в ожидании окончания дисковой или сетевой операции (при этом они не потребляют мощности центрального процессора), другие модули могут активно обрабатывать другие запросы. Имея $k$ обрабатывающих модулей, производительность можно повысить в $k$ раз по сравнению с однопоточным сервером. Конечно, если ограничивающим фактором являются диск или сеть, то необходимо наличие нескольких дисков или более быстрой сети, чтобы действительно улучшить однопоточную модель.

По сути, все современные веб-архитектуры построены по представленной выше схеме, с разделением на интерфейсную («фронтенд») и серверную («бэкенд») части. Интерфейсный веб-сервер часто называют **обратным прокси-сервером (reverse proxy)**, поскольку он как посредник («прокси») извлекает содержимое из других серверов (обычно относящихся к серверной части)

и доставляет эти объекты клиенту. Слово «обратный» указывает на то, что он действует от имени сервера, а не клиента.

При загрузке веб-страницы клиент сначала часто направляется (с использованием DNS) к обратному прокси-серверу (интерфейсному), который начинает возвращать веб-браузеру клиента статические объекты, чтобы он мог как можно скорее загрузить какое-то содержимое страницы. В это время серверная часть выполняет такие сложные операции, как поиск в интернете или базе данных, либо иное генерирование динамического контента, а затем возвращает клиенту результаты через обратный прокси-сервер.

## 7.3.2. Статичные веб-объекты

Основная идея Всемирной паутины состоит в доставке веб-страниц от сервера клиенту. В самом простом случае веб-объекты статические. Сегодня практически любая веб-страница имеет динамический контент, но в то же время даже на динамических страницах значительная часть содержимого (например, логотип, таблицы стилей, верхний и нижний колонтитулы) по-прежнему носит статичный характер. Статичные объекты — это размещенные на каком-либо сервере файлы, которые при каждом просмотре отображаются одинаково. Обычно их можно кэшировать, и даже на длительное время, поэтому они часто размещаются в ближайшем к пользователю объектном кэше. Но то, что страницы статичны, еще не значит, что при отображении в браузере они ведут себя пассивно. Ведь видеофайлы, к примеру, тоже представляют собой статичные объекты.

Как уже упоминалось, HTML — общепринятый язык Всемирной паутины, на котором написано большинство страниц. Домашние страницы преподавателей университетов, как правило, статичны. Сайты компаний могут быть динамическими, но конечным результатом динамического генерирования контента в любом случае будет HTML-страница. **Язык HTML (HyperText Markup Language — язык разметки гипертекста)** возник одновременно с появлением Всемирной паутины. С его помощью на веб-страницах можно размещать текст, изображения, видео, указатели на другие страницы и т. п. Это язык разметки, то есть язык для описания способа форматирования документов. Термин **разметка (markup)** восходит к тем дням, когда редактор с помощью специальной разметки указывал типографу, какой шрифт использовать для печати. Таким образом, языки разметки позволяют четко задавать команды форматирования. Например, в HTML тег `<b>` **означает начало участка текста, выделенного полужирным шрифтом**, а `</b>` указывает на конец такого участка. В свою очередь, тег `<h1>` указывает начало заголовка уровня 1. Другие языки разметки, хорошо известные академическим авторам, — LaTeX и TeX. Программа Microsoft Word, напротив, *не* является таким языком, поскольку она *не* встраивает в текст команды форматирования.

Главное преимущество языков разметки над языками, не имеющими явных команд форматирования, заключается в том, что они позволяют отделить контент от способа его представления. В большинстве современных веб-страниц для определения гарнитуры, цвета, размера, отступов и многих других атрибутов

текста, списков, таблиц, заголовков, рекламных объявлений и других элементов используются таблицы стилей. Они пишутся на языке **CSS (Cascading Style Sheets — каскадные таблицы стилей)**.

В итоге написание браузера не представляет большого труда: он должен лишь понимать команды разметки и содержимое таблиц стилей и применять все это к контенту. Благодаря встроенным стандартизированным командам разметки в HTML-файлах любой браузер может прочитать и переформатировать какую угодно страницу. Это критически важно, поскольку страницу, созданную, к примеру, на мощном компьютере с разрешением 3840 × 2160 и 24-битным цветом, часто приходится отображать на экране мобильного телефона с разрешением 640 × 320. Простое линейное масштабирование в данном случае плохая идея, поскольку шрифт при этом станет настолько мелким, что его невозможно будет прочитать.

Хотя такие документы, в принципе, можно создавать с помощью обычного текстового редактора, и многие именно так и поступают, лучше все же использовать для этого текстовые процессоры или специальные HTML-редакторы, которые берут большую часть работы на себя (но, соответственно, уменьшают степень контроля над отдельными элементами конечного результата). Также существует множество программ для создания веб-страниц, например Adobe Dreamweaver.

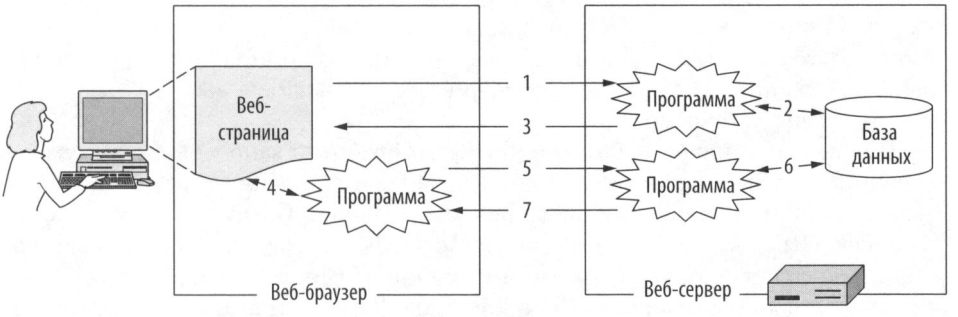
## 7.3.3. Динамические веб-страницы и веб-приложения

Модель статичной страницы, которую мы обсуждали до этого момента, рассматривает страницу как набор удобно связанных между собой документов (в том числе мультимедийных). Эта модель хорошо подходила для раннего интернета, когда в него было загружено огромное количество информации. Сегодня же интернет в основном используется для доступа к приложениям и сервисам, например для покупки товаров в интернет-магазине, поиска книги в каталоге библиотеки, использования онлайн-карт, чтения и отправки почты, а также совместного редактирования документа несколькими пользователями.

Эти новые области применения похожи на традиционное прикладное ПО (например, почтовые клиенты и текстовые редакторы). Однако в данном случае приложения запускаются в браузере, а пользовательские данные хранятся на серверах в центрах обработки данных, подключенных к интернету. Эти приложения используют веб-протоколы для получения информации через интернет и браузер для отображения пользовательского интерфейса. Преимущество состоит в том, что пользователю не нужно устанавливать отдельные приложения и он может получить доступ к своим данным с разных компьютеров, причем данные сохраняются у оператора сервиса. Этот подход оказался настолько успешным, что стал соперничать с использованием традиционного ПО. Конечно, немаловажен и тот факт, что эти приложения предоставляются крупными провайдерами бесплатно. По такой модели строится большая часть **облачных вычислений (cloud computing)**, переводящих процесс вычисления с пользовательских компьютеров на совместно используемые кластеры серверов в интернете.

Чтобы выступать в роли приложения, веб-страницы больше не могут быть статичными. Они должны содержать динамический контент. Например, страница библиотечного каталога должна показывать, какие книги доступны на данный момент, а какие находятся на руках. Полезная страница фондовой биржи должна позволять пользователю взаимодействовать со страницей, чтобы просматривать курсы акций за разные периоды времени и рассчитывать прибыль и убытки. Как можно понять по этим примерам, динамический контент может генерироваться программами, запущенными на сервере или в браузере (или на обоих хостах).

Типичная ситуация показана на илл. 7.23. Например, представим себе картографический сервис, в котором можно ввести название улицы и затем просмотреть карту местности. Получив запрос, веб-сервер должен использовать программу для создания страницы, которая отображает карту запрашиваемой местности с помощью базы данных улиц и другой географической информации. Это шаги 1–3. Запрос (шаг 1) вызывает запуск программы на сервере. Программа опрашивает базу данных, генерирует нужную страницу (шаг 2) и возвращает ее в браузер (шаг 3).




**Илл. 7.23.** Динамические страницы

Но это не весь динамический контент. Сама страница может содержать программы, которые запускаются в браузере. В нашем примере программа позволяет пользователю находить маршруты и исследовать соседние области с разными уровнями детализации. Она обновляет страницу, увеличивая или уменьшая масштаб в соответствии с запросами пользователя (шаг 4). Чтобы выполнить некоторые операции, программе может понадобиться больше данных с сервера. В этом случае она отправит запрос на сервер (шаг 5), который отыщет нужную информацию в базе данных (шаг 6) и вернет ответ (шаг 7). Затем программа продолжит вносить изменения на странице (шаг 4). Запросы и ответы обрабатываются в фоновом режиме; пользователь может и не знать о них, так как URL и название страницы обычно не меняются. Страница с программами, выполняющимися на стороне клиента, может предоставить более удобный интерфейс, чем страница, включающая только программы, выполняющиеся на сервере.

### *Динамическая генерация веб-страниц на стороне сервера*

Давайте кратко обсудим динамическую генерацию веб-страниц на стороне сервера. Когда пользователь активирует в форме ссылку (например, чтобы купить некий товар), серверу по указанному в форме URL-адресу отправляется запрос с введенной пользователем информацией. Эти данные должны быть переданы скрипту или программе для обработки. Таким образом, URL-адрес идентифицирует программу для запуска, и данные из запроса передаются ей в качестве входной информации. Какую страницу вернет этот запрос, зависит от того, что произойдет в ходе его обработки. Результат не фиксирован, как в случае статичной страницы. При успешном оформлении заказа возвращаемая страница может содержать дату доставки товара. В противном случае она сообщит, что товар закончился или что кредитная карта пользователя недействительна.

То, как именно сервер запускает программу вместо поиска файла, зависит от устройства веб-сервера. В самих веб-протоколах это не определено. Именно поэтому интерфейс может быть защищен правом собственности, а браузеру не нужно знать детали. Что касается браузера, он просто создает запрос и получает страницу.

И все же для веб-серверов были разработаны стандартные API, чтобы запускать программы. Существование этих интерфейсов позволяет разработчикам тратить меньше усилий на расширение различных серверов за счет веб-приложений. Мы кратко рассмотрим два API, чтобы вы получили о них некоторое представление.

Первый API представляет собой метод обработки запросов динамических страниц, который был доступен с момента возникновения интернета. Он называется **общим шлюзовым интерфейсом (Common Gateway Interface, CGI)** и описан в RFC 3875. CGI предоставляет интерфейс, позволяющий веб-серверам общаться с серверными программами и скриптами, способными принимать некоторые входные данные (например, из формы) и в ответ генерировать HTML-страницы. Эти программы могут быть написаны на любом удобном для разработчика языке; обычно для упрощения разработки используется скриптовой язык. Можно выбрать Python, Ruby, Perl или любой другой язык на ваш вкус.

Существует договоренность, в соответствии с которой программы, запускаемые через CGI, должны размещаться в каталоге cgi-bin, который включается в URL-адрес. Сервер отображает запрос этого каталога на имя программы и запускает ее как отдельный процесс. В качестве входных данных программе передаются данные, отправленные вместе с запросом. На выходе программа выдает веб-страницу, которая возвращается браузеру.

Второй API предполагает совершенно иной подход. В данном случае в HTML-страницу встраиваются небольшие скрипты, которые выполняются сервером для генерирования страницы. Популярным инструментом для написания таких скриптов является язык **PHP (PHP: Hypertext Preprocessor — PHP: препроцессор гипертекста)**. Для его использования необходимо, чтобы сервер понимал язык PHP (так же, как браузер должен понимать язык CSS, чтобы интерпретировать страницы с таблицами стилей). Обычно веб-страницы с PHP-кодом имеют расширение php, а не htm или html, что позволяет серверам легко

их идентифицировать. В использовании PHP проще, чем CGI, и распространен повсеместно.

Несмотря на простоту в применении, PHP — мощный язык программирования для создания веб-интерфейсов и взаимодействия с серверными базами данных. В PHP есть переменные, строки, массивы и большинство управляющих структур, имеющихся в языке C, но при этом он обеспечивает более мощные возможности ввода/вывода по сравнению с процедурой `printf`. PHP имеет открытый исходный код, распространяется бесплатно и широко используется. PHP был разработан специально для сервера Apache, который также обладает открытым исходным кодом и является самым распространенным веб-сервером в мире.

### *Создание динамических веб-страниц на стороне клиента*

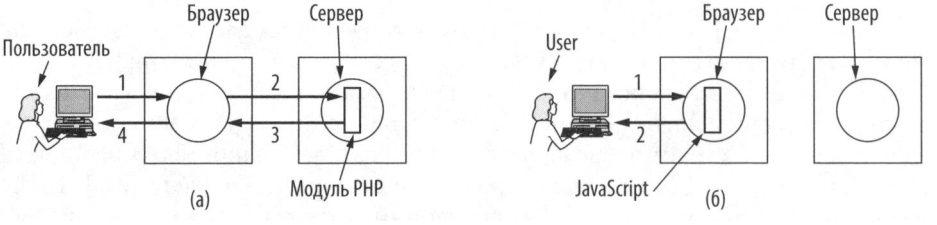
Скрипты CGI и PHP решают вопросы обработки входных данных и взаимодействия с базами данных на сервере. Они могут принимать входящую информацию из форм, осуществлять поиск по одной или нескольким базам данных и в качестве результата генерировать HTML-страницы. Но ни один из этих методов не позволяет напрямую взаимодействовать с пользователем, например реагировать на движения мыши. Для этих целей необходимы скрипты, внедренные в HTML-страницы и выполняющиеся не на серверном, а на клиентском устройстве. Начиная с HTML 4.0, появилась возможность включать скрипты такого типа с помощью тега `<script>`. Текущий стандарт HTML сегодня принято называть **HTML5**. HTML5 содержит множество новых возможностей синтаксиса для встраивания мультимедийного и графического контента, включая теги `<video>`, `<audio>` и `<canvas>`. Элемент `<canvas>`, в частности, облегчает динамическое отображение двумерных форм и растровых изображений. Как ни странно, данный элемент вызывает большое беспокойство относительно конфиденциальности, поскольку свойства тега HTML `<canvas>` зачастую уникальны для разных устройств. Это позволяет операторам веб-сайтов следить за пользователями, даже если те избавятся от всех отслеживающих файлов cookie и скриптов.

Наиболее популярным языком для написания клиентских скриптов является **JavaScript**, поэтому мы немного поговорим о том, что он собой представляет. Об этом языке написано немало книг, к примеру работа Кодинга (Coding, 2019) или Атенсио (Atencio, 2020)[1]. Несмотря на схожесть названий, JavaScript не имеет практически ничего общего с языком программирования Java. Как и другие скриптовые языки, JavaScript — это язык с очень высоким уровнем абстракции: одной строкой JavaScript можно создать диалоговое окно, войти в цикл ожидания пользовательского ввода и сохранить полученную строку в переменной. Такой уровень абстракции идеален для разработки интерактивных веб-страниц. С другой стороны, тот факт, что JavaScript мутирует быстрее, чем мушка дро-

[1] Издательский дом «Питер» предлагает книги по JavaScript как для новичков, так и для специалистов. Вот некоторые из них: Чиннатхамби К. «JavaScript с нуля», 2022; Симпсон К. «{Вы пока еще не знаете JS} Познакомьтесь, JavaScript», 2022; Фрисби М. «JavaScript для профессиональных веб-разработчиков», 2022. — *Примеч. ред.*

зофила под воздействием рентгеновских лучей, сильно усложняет написание на нем программ, способных работать на всех платформах. Возможно, однажды он станет более стабильным.

Хотя языки PHP и JavaScript схожи в том, что они встраивают код в HTML-файлы, важно понимать, что обработка кода при этом выполняется совершенно по-разному. При использовании PHP после щелчка по кнопке подтверждения браузер собирает всю введенную информацию в одну длинную строку и отправляет ее на сервер как запрос PHP-страницы. Сервер загружает PHP-файл и выполняет встроенный в него PHP-скрипт, получая на выходе новую HTML-страницу, которая возвращается браузеру для отображения. При этом браузер даже не знает, что страница была сгенерирована программой. Эта последовательность действий показана как шаги 1–4 на илл. 7.24 (а).




**Илл. 7.24.** (а) Сценарий PHP на стороне сервера. (б) Сценарий JavaScript на стороне клиента

В случае JavaScript после нажатия кнопки подтверждения браузер сам интерпретирует содержащийся на странице JavaScript-код. Вся работа производится локально, внутри браузера, без какого-либо взаимодействия с сервером. Эта последовательность действий показана как шаги 1 и 2 на илл. 7.24 (б). Как следствие, результат отображается практически мгновенно, тогда как при использовании PHP на доставку полученной HTML-страницы клиенту иногда уходит несколько секунд.

Эти отличия вовсе не означают, что JavaScript лучше, чем PHP. Просто у них совершенно разное назначение. PHP применяется для взаимодействия с серверной базой данных, а JavaScript (и другие языки для клиентской стороны) — для взаимодействия с пользователем на клиентском компьютере. Разумеется, как мы вскоре убедимся, PHP и JavaScript можно использовать одновременно.

## 7.3.4. HTTP и HTTPS

Теперь, когда мы имеем представление о веб-контенте и веб-приложениях, пора поговорить о протоколе, используемом для передачи всей этой информации от веб-сервера к клиенту и обратно. Это **протокол передачи гипертекста (HyperText Transfer Protocol, HTTP)**, описанный в спецификации RFC 2616. Прежде чем вдаваться в подробности, стоит упомянуть о некоторых различиях между HTTP и его безопасной версией — **защищенным протоколом передачи гипертекста (Secure HyperText Transfer Protocol, HTTPS)**. В сущности, оба протокола извлекают веб-объекты одинаково, и HTTP-стандарт для извлечения объектов

развивается практически независимо от своей защищенной версии. HTTPS фактически использует HTTP поверх защищенного транспортного протокола **защиты транспортного уровня (Transport Layer Security, TLS)**. В этой главе мы сосредоточимся на особенностях HTTP и истории его развития от первых версий до более современной HTTP/3. В главе 8 будет подробно рассмотрен TLS, который, по сути, является транспортным протоколом для HTTP, что в совокупности дает нам HTTPS. В оставшейся части этого раздела мы обсудим HTTP; HTTPS при этом можно рассматривать просто как HTTP, транспортируемый с помощью TLS.

### *Общие сведения*

HTTP — простой запросно-ответный протокол. Его старые версии обычно запускаются поверх TCP, однако его самая последняя версия, HTTP/3, также нередко работает поверх UDP. Протокол HTTP определяет, какие сообщения клиент может отправлять серверу и что может быть возвращено в качестве ответа. Заголовки запросов и ответов, так же как в SMTP, даны в ASCII. Их содержимое имеет формат, похожий на MIME (опять же, как и в SMTP). Эта модель была одной из причин успеха Всемирной паутины на раннем этапе, так как сильно упрощала процесс разработки и развертывания.

В этом разделе мы поговорим о наиболее важных особенностях протокола HTTP в том виде, в каком он используется сегодня. Прежде чем вдаваться в детали, следует отметить, что подход к использованию этого протокола в интернете активно дорабатывается. HTTP — это протокол прикладного уровня, так как он работает поверх TCP и тесно связан с интернетом. Именно поэтому мы говорим о нем в этой главе. Но с другой стороны, HTTP все больше напоминает транспортный протокол, позволяющий процессам передавать контент из одной сети в другую. В качестве этих процессов могут выступать не только веб-браузер и веб-сервер. Так, медиаплеер может использовать HTTP для того, чтобы запросить у сервера информацию об альбоме. Антивирус — для загрузки последних обновлений вирусной базы. Разработчики — для получения файлов, относящихся к проекту, над которым они работают. Изделия бытовой электроники (например, цифровые фоторамки) часто используют встроенный HTTP-сервер как интерфейс, связывающий их с внешним миром. Коммуникация между компьютерами все чаще осуществляется при помощи HTTP. Например, сервер авиакомпании может связаться с сервером проката автомобилей и забронировать машину в рамках предлагаемого авиакомпанией турпакета.

### *Методы*

HTTP был разработан специально для веб-технологий, но его намеренно сделали более универсальным, чем это было необходимо, в расчете на будущее применение в объектно-ориентированных приложениях. По этой причине HTTP позволяет запрашивать не только веб-страницы, но и операции, называемые **методами**.

Каждый запрос состоит из одной или нескольких строк ASCII-текста, где начальное слово в первой строке является именем вызываемого метода. Встроенные методы перечислены на илл. 7.25. Имена методов чувствительны к регистру символов, то есть метод GET существует, а метод get — нет.

Метод GET запрашивает у сервера страницу (под которой в общем случае подразумевается объект, но на практике это просто файл), закодированную согласно стандарту MIME. Большую часть запросов к веб-серверам составляют именно GET-запросы с простым синтаксисом. Типичный GET-запрос выглядит так:

```
GET filename HTTP/1.1
```

где filename — запрашиваемая страница, а 1.1 — используемая версия протокола.

| Метод | Описание |
|---|---|
| GET | Чтение веб-страницы |
| HEAD | Чтение заголовка веб-страницы |
| POST | Добавить к веб-странице |
| PUT | Сохранить веб-страницу |
| DELETE | Удалить веб-страницу |
| TRACE | Отобразить входящий запрос |
| CONNECT | Подключиться через прокси |
| OPTIONS | Параметры запроса страницы |

**Илл. 7.25.** Встроенные методы HTTP-запросов

Метод HEAD запрашивает только заголовок сообщения, без самой страницы. С его помощью можно собрать индексную информацию или просто проверить работоспособность URL-адреса.

Метод POST используется при подтверждении формы. Как и GET, он использует URL-адрес, но вместо простого извлечения страницы загружает данные (то есть содержимое формы или параметры) на сервер. Сервер выполняет над данными некое действие, которое зависит от URL-адреса, фактически добавляя их к объекту. В итоге происходит покупка указанного товара или вызов необходимой процедуры. Наконец, метод возвращает страницу с сообщением об этом результате.

Другие методы редко используются при просмотре веб-страниц. Метод PUT является противоположностью метода GET: он не читает, а записывает страницу. PUT позволяет создать набор веб-страниц на удаленном сервере. При этом тело запроса содержит страницу. Она может быть закодирована согласно стандарту MIME. В этом случае строки, следующие за командой PUT, могут содержать

заголовки аутентификации, подтверждающие, что абонент обладает правами доступа к запрашиваемой операции.

Метод DELETE, как ни удивительно, удаляет страницу или, по крайней мере, сообщает, что веб-сервер дал согласие на ее удаление. Как и с методом PUT, важную роль здесь играет аутентификация и права доступа.

Метод TRACE предназначен для отладки. Он приказывает серверу отослать запрос обратно. Этот метод особенно полезен, когда запросы обрабатываются некорректно и клиенту необходимо узнать, какой именно запрос получает сервер.

Метод CONNECT предоставляет пользователю возможность подключиться к серверу через промежуточное устройство, такое как веб-кэш.

Метод OPTIONS позволяет клиенту запросить у сервера информацию о том, какие методы и заголовки можно использовать на указанной странице.

В ответ на каждый запрос включается строка состояния, часто вместе с дополнительной информацией (например, веб-страница целиком или ее часть). Эта строка может содержать трехразрядный код состояния, сообщающий об успешном выполнении запроса или о причинах неудачи. Первый разряд предназначен для разделения всех ответов на пять основных групп, перечисленных на илл. 7.26.

| Код | Значение | Примеры |
|---|---|---|
| 1xx | Информация | 100 = сервер согласен обрабатывать запросы клиента |
| 2xx | Успех | 200 = запрос успешно обработан; 204 = содержимое отсутствует |
| 3xx | Перенаправление | 301 = страница перемещена; 304 = кэшированная страница все еще доступна |
| 4xx | Ошибка клиента | 403 = ошибка доступа; 404 = страница не найдена |
| 5xx | Ошибка сервера | 500 = внутренняя ошибка сервера; 503 = попробуйте еще раз позднее |

**Илл. 7.26.** Группы кодов состояния, содержащиеся в ответах сервера

Коды, начинающиеся с 1 (1xx), на практике используются редко. Коды 2xx означают, что запрос был обработан успешно и данные (если их запрашивали) возвращены. Коды 3xx сообщают клиенту о том, что нужно попытать счастья в другом месте — используя либо другой URL, либо свой собственный кэш (мы обсудим его далее). Коды 4xx означают, что запрос по какой-либо причине, связанной с клиентом, потерпел неудачу: например, запрошена несуществующая страница или сам запрос некорректен. Наконец, коды 5xx сообщают о внутренних ошибках сервера, возникших из-за ошибки программы или временной перегрузки.

### *Заголовки сообщений*

За строкой запроса (к примеру, содержащей метод GET) могут следовать другие строки с дополнительной информацией. Они называются **заголовками запросов (request headers)**. Эти сведения можно сравнить с параметрами вызова процедуры. В свою очередь, ответы могут содержать **заголовки ответов (response headers)**. Некоторые заголовки встречаются и там и там. Наиболее важные из

них перечислены на илл. 7.27. Это длинный список, и как вы понимаете, каждый запрос и ответ может содержать целый набор заголовков.

| Заголовок | Тип | Содержимое |
|---|---|---|
| User-Agent | Запрос | Информация о браузере и его платформе |
| Accept | Запрос | Тип страниц, поддерживаемых клиентом |
| Accept-Charset | Запрос | Поддерживаемые клиентом наборы символов |
| Accept-Encoding | Запрос | Поддерживаемые клиентом типы кодирования |
| Accept-Language | Запрос | Естественные языки, доступные клиенту |
| If-Modified-Since | Запрос | Время и дата последнего обновления |
| If-None-Match | Запрос | Теги, отправленные с последнего обновления |
| Host | Запрос | DNS-имя сервера |
| Authorization | Запрос | Список персональных идентификаторов клиента |
| Referer | Запрос | URL, с которого был отправлен предыдущий запрос |
| Cookie | Запрос | Отправка ранее принятого cookie-файла на сервер |
| Set-Cookie | Ответ | Сервер хочет, чтобы клиент сохранил cookie |
| Server | Ответ | Информация о сервере |
| Content-Encoding | Ответ | Тип кодирования содержимого (например, gzip) |
| Content-Language | Ответ | Естественный язык, используемый на странице |
| Content-Length | Ответ | Размер страницы в байтах |
| Content-Type | Ответ | MIME-тип страницы |
| Content-Range | Ответ | Идентифицирует часть контента страницы |
| Last-Modified | Ответ | Время и дата внесения последних изменений в страницу |
| Expires | Ответ | Время и дата, когда страница перестанет считаться действительной |
| Location | Ответ | Команда клиенту на пересылку его запроса по другому адресу |
| Accept-Ranges | Ответ | Сервер готов принимать запросы на страницы указанного размера |
| Date | Запрос/Ответ | Дата и время отправки сообщения |
| Range | Запрос/Ответ | Идентифицирует часть страницы |
| Cache-Control | Запрос/Ответ | Указание на то, как обрабатывать кэш |
| ETag | Запрос/Ответ | Тег для контента страницы |
| Upgrade | Запрос/Ответ | Протокол, на который хочет переключиться отправитель |

**Илл. 7.27.** Некоторые заголовки сообщений протокола HTTP

С помощью заголовка `User-Agent` клиент может сообщить серверу версию своего браузера (например, *Mozilla/5.0* или *Chrome/74.0.3729.169*). Эта информация позволяет серверу адаптировать свои ответы под конкретный браузер, так как логика работы и возможности разных браузеров серьезно отличаются.

Четыре заголовка, начинающиеся с `Accept`, сообщают серверу о том, какие типы информации готов принять клиент (если их набор ограничен). Первый из них указывает допустимые MIME-типы (например, *text/html*). Заголовок `Accept-Charset` указывает используемый клиентом набор символов (к примеру, *ISO-8859-5* или *Unicode-1-1*). Заголовок `Accept-Encoding` указывает допустимые методы сжатия (например, *gzip*). Наконец, заголовок `Accept-Language` указывает естественный язык, используемый клиентом (например, испанский). Если сервер может выбирать из нескольких страниц, он подберет наиболее подходящий для клиента вариант согласно полученной информации. Если запрос удовлетворить невозможно, возвращается код ошибки, и запрос считается неудавшимся.

Заголовки `If-Modified-Since` и `If-None-Match` используются при кэшировании. Они позволяют клиенту запрашивать отправку страницы только в том случае, если в кэше нет доступной копии. Мы обсудим кэширование чуть позже.

Заголовок `Host` указывает имя сервера, содержащееся в URL-адресе. Этот заголовок обязателен, поскольку некоторые IP-адреса могут обслуживать несколько имен DNS одновременно, и серверу необходимо каким-то образом различать, кому передавать запрос.

Заголовок `Authorization` требуется в тех случаях, когда запрашивается защищенная страница. С его помощью клиент может подтвердить свои права на ее просмотр.

Клиент использует заголовок `Referer` (не соответствующий правилам английской орфографии), чтобы сообщить, с какого URL-адреса был выполнен переход на запрашиваемый URL-адрес. Чаще всего это URL-адрес предыдущей страницы. Этот заголовок крайне полезен для отслеживания процесса просмотра веб-страниц, поскольку позволяет серверу узнать, каким образом клиент попал на ту или иную страницу.

**Файлы cookie** представляют собой небольшие файлы, которые размещаются серверами на клиентских устройствах с целью запоминания информации на будущее. Типичный пример — сайт интернет-магазина, использующий файл cookie на стороне клиента для отслеживания того, какие товары он уже успел заказать. При каждом добавлении товара в корзину производится обновление файла cookie, чтобы он содержал сведения обо всех заказах. Несмотря на то что файлы cookie описываются в спецификации RFC 2109, а не в спецификации RFC 2616, в них тоже используются заголовки. Заголовок `Set-cookie` определяет то, как серверы отправляют файлы cookie клиентам. Предполагается, что клиент сохранит у себя cookie и вернет его вместе со следующим запросом на сервер при помощи заголовка `Cookie`. (Обратите внимание, что существует и более поздняя спецификация для файлов cookie с обновленными заголовками, RFC 2965, но она не слишком распространена.)

В ответах используется множество других заголовков. Заголовок `Server` позволяет серверу указать версию своего программного обеспечения. Следующие

пять заголовков, начинающиеся с `Content-`, позволяют серверу описать свойства отправляемой им страницы.

Заголовок `Last-modified` содержит дату и время внесения последних изменений в отправляемую страницу, а заголовок `Expires` сообщает, сколько времени страница будет доступна. Оба они играют важную роль при кэшировании страницы.

С помощью заголовка `Location` сервер сообщает, что клиента нужно перенаправить на другой URL-адрес. Это может потребоваться при «переезде» страницы на другой адрес или если нужно разрешить использование нескольких URL-адресов для ссылки на одну и ту же страницу (возможно, на разные сервера). Этот заголовок также нередко применяется компаниями, главная веб-страница которых прописана в домене `com`, однако клиенты перенаправляются с нее на национальные или региональные страницы, в соответствии с IP-адресом клиента или выбранным языком.

Если страница крупная, мелкий клиент может не захотеть принять ее сразу целиком. Некоторым серверам можно отправлять запросы, ограничивающие размеры страниц, отсылаемых за один раз. Если страница оказывается слишком большой, она будет разбита на более мелкие единицы и выслана в несколько приемов. Заголовок `Accept-Ranges` сообщает о том, что сервер готов работать таким образом.

Теперь перейдем к заголовкам, которые могут быть использованы в обоих направлениях. Заголовок `Date` применяется как в запросах, так и в ответах и содержит время и дату отправки сообщения. Заголовок `Range` сообщает, какой диапазон байтов страницы предоставляется в качестве ответа.

Заголовок `ETag` сообщает короткий тег, выполняющий роль имени контента страницы. Он применяется для кэширования. Заголовок `Cache-Control` выдаст другие четкие указания о том, как кэшировать (а чаще — как не кэшировать) страницы.

Заголовок `Upgrade` используется для переключения на другой протокол обмена данными, например на новую версию HTTP или на защищенный способ передачи данных. Он позволяет клиенту и серверу сообщить, что именно они поддерживают.

### *Кэширование*

Мы часто возвращаемся на страницы, которые уже просматривали ранее, а на связанных веб-страницах нередко размещаются одни и те же ресурсы: например, изображения, которые используются для навигации по сайту, а также стандартные таблицы стилей и скрипты. Было бы крайне неэкономично получать все эти ресурсы страниц каждый раз, когда они отображаются, ведь у браузера уже есть их копии.

Сохранение полученных страниц для дальнейшего использования называется **кэшированием (caching)**. Преимущество этого метода в том, что страница, сохраненная в кэше, может быть использована снова, при этом не обязательно повторять передачу. У HTTP есть встроенная поддержка кэширования, чтобы помочь клиентам узнать, могут ли они безопасно использовать

страницы повторно. Эта поддержка повышает производительность, сокращая и трафик, и время ожидания. Недостаток в том, что теперь браузер должен хранить страницы, но это почти всегда оправданно, так как локальное хранение информации не требует серьезных затрат ресурсов. Обычно страницы хранятся на диске, так что они могут быть использованы, когда браузер будет запущен в следующий раз.

С HTTP-кэшированием связан непростой вопрос: как определить, что сохраненная копия страницы соответствует тому, как она будет выглядеть при следующем вызове. Такой вывод не может быть сделан только на основании совпадения URL. Допустим, URL ведет на страницу с последними новостями. Этот контент будет постоянно обновляться, хотя URL останется тем же. Если же на странице содержится список богов греческой и римской мифологии, она будет меняться гораздо реже.

HTTP использует две стратегии решения этой проблемы. Они отображены на илл. 7.28 как формы обработки между запросом (шаг 1) и ответом (шаг 5). Первая стратегия — это проверка действительности страницы (шаг 2). Запрашивается кэш, и если в нем есть свежая (то есть действительная) копия страницы заданного URL, получать новую копию с сервера нет необходимости. Вместо этого сохраненная в кэше страница может быть возвращена напрямую. Когда кэшированная страница была первоначально получена с сервера, вместе с ней был получен заголовок `Expires`. Его содержание, а также текущие дата и время используются, чтобы сделать заключение о действительности страницы.

**Илл. 7.28.** Кэширование HTTP

Однако не все страницы приходят с удобным заголовком `Expires`, сообщающим, когда страница должна быть получена снова. В конце концов, строить догадки тяжело, особенно о будущем. В этом случае браузер может использовать эвристические правила. Например, если страница не была изменена за последний год (как сказано в заголовке `Last-Modified`), логично предположить, что она не изменится за ближайший час. Хотя гарантий нет, и этот прогноз может не оправдаться. Например, биржа может закрыться на ночь, и ее страница не будет обновляться несколько часов, но как только начнется следующая торговая сессия, изменения будут вноситься постоянно. Таким образом, оправданность кэширования может серьезно варьироваться в зависимости от периода времени. По этой причине эвристические правила нужно использовать с осторожностью, хотя в основном они работают очень хорошо.

Обнаружение страниц, которые не были изменены за определенный промежуток времени, — наиболее выгодный способ использования кэширования, так как в этом случае не нужно связываться с сервером. К сожалению, этот подход не всегда работает. Сервер должен использовать заголовок `Expires` осторожно, поскольку не всегда известно, когда на странице будут внесены изменения. Таким образом, кэшированные копии могут быть свежими, но клиент этого не знает.

В этом случае используется вторая стратегия — с сервера запрашивается информация о действительности сохраненной копии страницы. Такой запрос называют **GET-запросом с условием (conditional GET)**. Он показан на илл. 7.28 как шаг 3. Если сервер знает, что копия в кэше все еще актуальна, он может отправить короткий ответ, подтверждая это (шаг 4а). В ином случае он должен отослать полный ответ (шаг 4б).

Существует еще несколько полей заголовков, которые используются для того, чтобы сервер мог проверить, действительна ли копия на момент запроса. Клиент знает, когда на страницу были внесены последние изменения, благодаря заголовку `Last-Modified`. Он может выслать это время серверу, используя заголовок `If-ModifiedSince`, чтобы запросить страницу только в том случае, если она была изменена после последнего кэширования. О кэшировании можно было бы рассказать еще очень много, поскольку оно сильно влияет на производительность, однако мы не будем раскрывать это в нашей книге. При желании вы легко найдете множество учебных пособий на эту тему в интернете по запросу «веб-кэширование».

### *HTTP/1 и HTTP/1.1*

Обычно при установке связи с сервером браузер устанавливает TCP-соединение с портом 443 сервера для HTTPS (или портом 80 для HTTP), хотя формально эта процедура необязательна. Ценность использования TCP в том, что ни браузерам, ни серверам не приходится беспокоиться о том, что делать со слишком длинными сообщениями, надежностью и контролем перегрузки. Все это обеспечивается протоколом TCP.

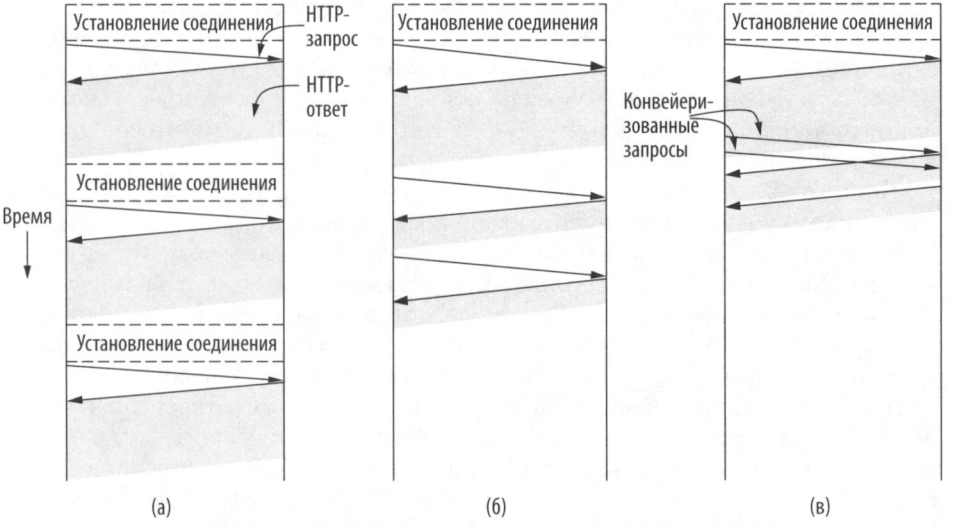
Изначально в интернете использовался протокол HTTP/1.0, и после установления соединения отсылался один запрос, на который приходил один ответ. После этого TCP-соединение разрывалось. Тогда типичная веб-страница целиком состояла из HTML-текста, и такой способ взаимодействия был подходящим. Однако вскоре рядовая веб-страница наполнилась многочисленными встроенными ссылками на различные значки и другие декоративные элементы. Установление отдельного TCP-соединения для передачи каждого значка стало слишком затратным.

Позже был создан протокол HTTP/1.1, поддерживающий **постоянные соединения (persistent connection)**, что позволило устанавливать TCP-соединения, отправлять запросы, получать ответы, а затем передавать и принимать дополнительные запросы и ответы. Эта стратегия называется **повторным использованием соединения (connection reuse)**. Таким образом снизились накладные расходы, возникавшие при постоянных установках и разрывах соединения. Стало

возможным также конвейеризировать запросы, то есть отправлять запрос 2 еще до прибытия ответа на запрос 1.

Разница в производительности между этими тремя случаями показана на илл. 7.29. В части (а) мы видим три запроса, отсылаемых один за другим, при этом для каждого устанавливается отдельное соединение.

Предположим, что это представление страницы с двумя встроенными картинками, которые находятся на одном сервере. URL изображений определяется при получении главной страницы, так что они приходят после нее. Сегодня на среднестатистической странице присутствует около 40 других объектов, которые должны быть получены для ее отображения; но это сделало бы наш рисунок огромным. Так что в данном примере мы используем всего два встроенных объекта.




**Илл. 7.29.** HTTP. (а) Множественные соединения и последовательные запросы. (б) Постоянное соединение и последовательные запросы. (в) Постоянное соединение и конвейеризованные запросы

На илл. 7.29 (б) страница получена через постоянное соединение. То есть TCP-соединение открывается с самого начала, затем отсылаются те же самые три запроса, как и раньше, один за другим, и только после этого соединение закрывается. Заметьте, что загрузка завершается быстрее. Это происходит по двум причинам: во-первых, не тратится время на установление дополнительных соединений (для каждого TCP-соединения требуется дополнительное время, как минимум для его подтверждения). Во-вторых, передача тех же картинок проходит быстрее. Почему же? Все это из-за контроля перегрузок сети в TCP. Сразу после установления соединения TCP использует процедуру медленного старта, чтобы увеличить пропускную способность, пока он не изучит поведение сетевого пути. Вследствие этого периода «разогрева» на передачу информации посредством множественных коротких TCP-соединений уходит гораздо больше времени, чем в случае с одним длительным TCP-соединением.

Наконец, на илл. 7.29 (в) установлено одно постоянное соединение, а запросы передаются конвейеризованно. Второй и третий запросы быстро отсылаются друг за другом, так как была получена достаточно большая часть главной страницы, чтобы можно было принять решение о загрузке и отображении картинок. В конечном итоге приходят ответы на эти запросы. Этот метод сокращает время, в течение которого сервер простаивает, так что это еще больше повышает производительность.

Однако за постоянные соединения приходится платить. Возникает новый вопрос: когда закрывать соединение? Соединение с сервером не должно разрываться, пока загружается страница. А что потом? Высок шанс того, что пользователь нажмет на ссылку, которая запросит еще одну страницу с сервера. Если соединение остается открытым, следующий запрос может быть отправлен немедленно. Но гарантии того, что клиент создаст запрос к серверу в ближайшее время, нет. На практике клиенты и серверы обычно сохраняют постоянное соединение, пока не пройдет какой-то небольшой промежуток времени (например, 60 с), в течение которого не будет отправлено ни одного запроса и не будет принято ни одного ответа, или же если открыто слишком много соединений и некоторые нужно закрыть.

Внимательный читатель мог заметить, что существует одна комбинация, о которой мы пока не упомянули. Можно также отсылать один запрос по одному TCP-соединению, но устанавливать эти соединения одновременно. Метод **параллельных соединений (parallel connection)** широко использовался браузерами до появления постоянных соединений. У него тот же недостаток, что и у последовательных соединений — дополнительные служебные операции, — но производительность гораздо выше. Так происходит из-за того, что параллельное установление и увеличение числа соединений занимает некоторое количество времени. В нашем примере соединения для обоих встроенных изображений могут быть установлены в одно и то же время. Однако запуск большого числа TCP-соединений с одним и тем же сервером не лучшая идея, поскольку TCP отслеживает перегрузки для каждого соединения отдельно. В результате соединения соревнуются друг с другом, вызывая дополнительные потери пакетов, и в общем являются более агрессивными пользователями сети, чем индивидуальные соединения. Постоянные соединения превосходят параллельные и являются более предпочтительными, так как избегают ненужных издержек и не страдают от проблем с перегрузками.

### *HTTP/2*

На заре интернета использовался протокол HTTP/1.0; а в 2007 году была создана версия HTTP/1.1. К 2012 году он уже немного устарел, в связи с чем IETF создал рабочую группу для разработки следующей версии HTTP/2. Отправной точкой при этом служил протокол SPDY, уже разработанный компанией Google. Окончательный результат был опубликован в мае 2015 года в виде спецификации RFC 7540.

Рабочая группа стремилась достигнуть нескольких целей, в том числе:

1. Предоставление клиентам и серверам возможности выбирать, какую версию HTTP использовать.

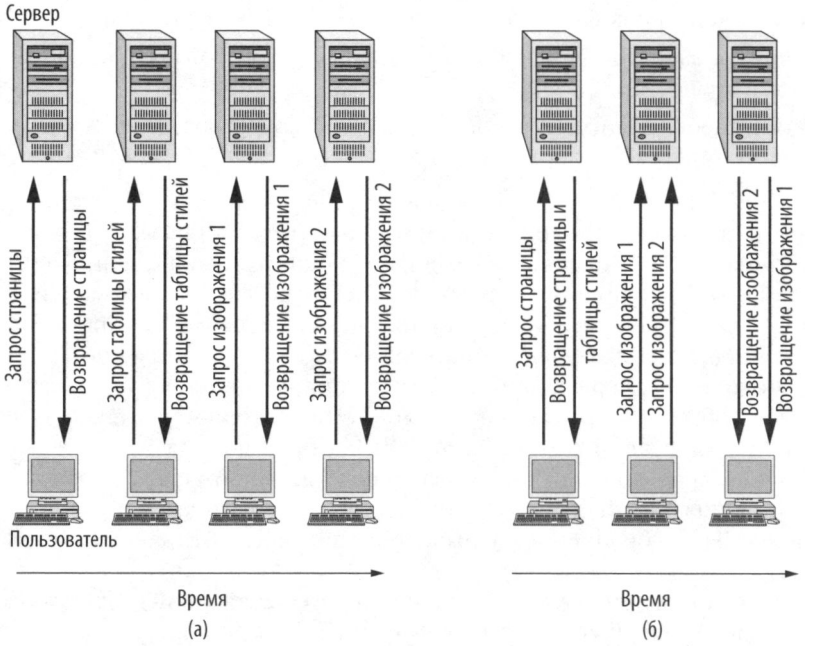
2. Поддержка максимальной совместимости с версией HTTP/1.1.
3. Повышение производительности за счет мультиплексирования, конвейеризации, сжатия и т. д.
4. Поддержка существующих методов, используемых браузерами, серверами, прокси-серверами, сетями доставки и т. д.

Ключевой идеей было обеспечение обратной совместимости. Предполагалось, что существующие приложения смогут работать с HTTP/2, а вновь создаваемые приложения – использовать новые функции для повышения производительности. По этой причине заголовки, URL-адреса и общая семантика почти не претерпели изменений. При этом поменялись способы кодирования и взаимодействия клиентов с серверами. В случае HTTP/1.1 клиент устанавливает TCP-соединение с сервером, отправляет запрос в виде текста, ожидает ответа, после чего обычно разрывает соединение. Это повторяется столько раз, сколько нужно для доставки всей страницы. В HTTP/2 после установления TCP-соединения можно отправить много запросов в двоичном виде, с возможностью их приоритизации, а сервер может отвечать на них в любом удобном для него порядке. TCP-соединение разрывается лишь после отправки ответов на все запросы.

Используя механизм **push на стороне сервера (server push)**, протокол HTTP/2 позволяет серверу «проталкивать» файлы, которые, по его мнению, понадобятся клиенту, хотя изначально тот может об этом и не знать. Например, если сервер видит, что запрошенная клиентом страница использует таблицу стилей и файл JavaScript, то он может отправить их еще до того, как клиент их запросит. Это позволяет устранить часть задержек. На илл. 7.30 показано, как одна и та же информация (веб-страница с таблицей стилей и двумя изображениями) может быть получена с помощью протоколов HTTP/1.1 и HTTP/2.

Обратите внимание, что на илл. 7.30 (а) показан наиболее благоприятный случай для протокола HTTP/1.1, при котором можно последовательно отправить несколько запросов по одному и тому же TCP-соединению, но при этом обработка запросов и возвращение результатов должны выполняться с соблюдением исходного порядка. В HTTP/2 (илл. 7.30 (б)) возвращение ответов может производиться в любом порядке. Например, если картинка 1 очень большая, сервер может сначала вернуть картинку 2, чтобы с ней браузер мог начать отображение страницы еще до того, как получит картинку 1. В HTTP/1.1 это недопустимо. Также обратите внимание, что на илл. 7.30 (б) сервер отправил таблицу стилей еще до того, как браузер ее запросил.

Наряду с конвейеризацией и мультиплексированием запросов в рамках одного TCP-соединения HTTP/2 производит сжатие заголовков и отправляет их в двоичном виде для экономии пропускной способности и уменьшения задержки. Сеанс протокола HTTP/2 состоит из серии фреймов, снабженных отдельными идентификаторами. Ответы могут поступать обратно в ином порядке, нежели запросы (илл. 7.30 (б)), но поскольку каждый ответ содержит идентификатор запроса, браузер может определить, какому из них соответствует тот или иной ответ.




**Илл. 7.30.** (а) Получение веб-страницы с помощью HTTP/1.1. (б) Получение той же страницы с помощью HTTP/2

Острым вопросом в ходе разработки версии HTTP/2 стала необходимость шифрования. Одни разработчики активно выступали за шифрование, другие столь же отчаянно выступали против. Аргументы противников главным образом были связаны с технологией IoT, где каждое устройство имеет весьма ограниченную вычислительную мощность. В итоге шифрование в HTTP/2 не обязательно, но поскольку его требуют все современные браузеры, оно все равно применяется, по крайней мере при просмотре веб-сайтов.

### *HTTP/3*

**HTTP/3**, или просто **H3**, — это третья крупная версия протокола HTTP, призванная заменить HTTP/2. Главное отличие HTTP/3 состоит в использовании другого транспортного протокола для пересылки HTTP-сообщений: вместо TCP здесь используется **QUIC** — версия протокола UDP, дополненная контролем перегрузки пользовательского пространства. Версия HTTP/3, которую изначально назвали просто HTTP-over-QUIC (HTTP поверх QUIC), является последней крупной переработкой протокола HTTP. Существует множество библиотек с открытым исходным кодом, которые обеспечивают поддержку клиентской и серверной логики для QUIC и HTTP/3 в таких языках, как C, C++, Python, Rust и Go. Популярные веб-серверы, включая nginx, теперь тоже позволяют использовать HTTP/3, установив соответствующие патчи.

Транспортный протокол QUIC поддерживает мультиплексирование потоков и примерно такой же механизм управления отдельными потоками, какой предлагался в версии HTTP/2. Надежность на уровне потока и контроль перегрузок в масштабе соединения существенно повышают производительность HTTP. Все потому, что информация о перегрузках может использоваться во время нескольких сеансов, а затраты на обеспечение надежности распределяются между множеством соединений, которые доставляют объекты параллельно. При наличии соединения с конечной точкой сервера HTTP/3 позволяет клиенту повторно использовать то же соединение с множеством различных URL-адресов.

Версия HTTP/3, использующая протокол HTTP поверх QUIC, теоретически может значительно увеличить производительность по сравнению с версией HTTP/2 (в основном за счет преимуществ QUIC над TCP). В некоторой степени протокол QUIC можно рассматривать как следующее поколение TCP. Он обеспечивает настройку соединения без дополнительных кругов передачи между клиентом и сервером. Если между ними уже устанавливалось соединение, его можно восстановить без кругов передачи (при условии, что в ходе предыдущего соединения был создан и сохранен в кэше секретный ключ). Протокол QUIC гарантирует надежную доставку байтов в пределах одного потока в исходном порядке, но не дает никаких гарантий относительно байтов в других QUIC-потоках. Хотя QUIC позволяет доставлять данные в рамках потока без соблюдения изначального порядка, HTTP/3 не использует эту возможность. Версию HTTP/3 на базе QUIC планируется применять исключительно в сочетании с защищенной версией HTTP – HTTPS. URL-адреса с HTTP-запросами (которые теперь используются все реже) не будут обновляться для версии HTTP/3. Более подробные сведения о протоколе HTTP/3 см. по адресу https://http3.net.

## 7.3.5. Конфиденциальность в интернете

Крайне актуальной проблемой в последние годы стало обеспечение конфиденциальности при просмотре веб-страниц. Веб-сайты, веб-приложения и сторонние трекеры часто применяют механизмы HTTP для отслеживания поведения пользователей как в рамках одного веб-сайта или приложения, так и по всему интернету. Также для этого могут использоваться некоторые дополнительные информационные каналы в браузере или устройстве. В этом разделе описаны некоторые из механизмов отслеживания и снятия цифрового отпечатка отдельных пользователей и устройств.

### *Файлы cookie*

Один из традиционных способов отслеживания – размещение на клиентских устройствах **файлов cookie** (по сути, небольшого количества данных), позже высылаемых клиентами обратно при последующем посещении какого-нибудь сайта. Когда пользователь запрашивает веб-объект (например, веб-страницу), веб-сервер может разместить на устройстве пользователя фрагмент персистентного состояния в виде файла cookie, используя директиву «set-cookie» протокола HTTP. Данные, переданные на устройство клиента с помощью этой директивы,

локально сохраняются на нем. При последующем посещении устройством этого веб-домена HTTP-запрос наряду с самим запросом передаст и файл cookie.

Собственные файлы cookie (они устанавливаются доменом посещаемого веб-сайта: интернет-магазина или новостного сайта) применяются для улучшения пользовательского опыта. Например, файлы cookie часто используются для сохранения состояния в рамках веб-сеанса. С их помощью сайт отслеживает полезную информацию о текущем поведении пользователя, например о том, как давно он авторизовался и какие товары он добавил в корзину.

Файлы cookie, установленные одним доменом, обычно видны только этому домену. Например, если некая рекламная сеть установит cookie на устройство пользователя, то он будет виден только ей и никому другому. Эта стратегия веб-безопасности, называемая **политикой одного источника (same-origin policy)**, предотвращает чтение одной из сторон файла cookie, установленного другой стороной, и в какой-то мере ограничивает распространение информации об отдельных пользователях.

Хотя собственные файлы cookie часто используются для улучшения пользовательского опыта, третьи лица (рекламодатели и компании, занимающиеся отслеживанием посетителей) также могут установить cookie на клиентское устройство, чтобы узнать, на какие сайты заходит пользователь. Это происходит так:

1. При посещении веб-сайта помимо контента, который запрашивается напрямую, устройство может загрузить контент со сторонних сайтов, в том числе с доменов рекламных сетей. Загрузка рекламы или скрипта позволяет третьей стороне установить уникальный файл cookie на устройство пользователя.
2. Впоследствии этот пользователь может посетить другие сайты, которые будут загружать веб-объекты той же третьей стороны, что позволит ей установить отслеживающую информацию на разных сайтах.

Приведем типичный пример такой практики: два разных веб-сайта используют одну и ту же рекламную сеть для доставки объявлений. В этом случае рекламная сеть видит следующее:

1. Устройство пользователя возвращает файл cookie, установленный на втором веб-сайте.
2. Заголовок `Referer` HTTP-запроса на загрузку объекта из рекламной сети сообщает, с какого сайта устройство пользователя перешло на текущий сайт. Это называют **межсайтовым отслеживанием (cross-site tracking)**.

**Супер-cookie** и другие локально сохраняемые идентификаторы отслеживания (которые пользователь не может контролировать, в отличие от обычных cookie) позволяют прокси-серверу отслеживать поведение пользователя в течение долгого времени. Уникальные идентификаторы могут включать сторонние идентификаторы отслеживания, кодируемые в заголовках HTTP (точнее, в заголовках **HSTS (HTTP Strict Transport Security — строгая безопасность передачи информации по протоколу HTTP)**, которые не удаляются при чистке файлов cookie, и в тегах, которые третья сторона (например, провайдер мобильного интернета)

может вставлять в незашифрованный трафик определенного сегмента сети. Это позволяет рекламодателям и другим заинтересованным сторонам формировать профиль посещений пользователя подобно отслеживающим файлам cookie, которые используются рекламными сетями и поставщиками приложений.

### Сторонние трекеры

Файлы cookie сторонних доменов, которые используются на нескольких сайтах, позволяют рекламной сети или другой третьей стороне отслеживать поведение пользователя на любом сайте, где эти файлы будут установлены (то есть на любом сайте с рекламными объявлениями, кнопками «Поделиться» и другим встроенным кодом от данной рекламной сети). Отслеживание паттернов просмотра, свойственных пользователю, как правило, происходит в рамках набора сайтов, которые он посещает. При этом зачастую применяется отслеживающее ПО на базе браузера. Иногда третья сторона разрабатывает собственное ПО для отслеживания (например, для веб-аналитики). Также она может использовать сервис другой третьей стороны, позволяющий собирать и агрегировать нужные сведения на разных сайтах.

Некоторые сайты позволяют рекламным сетям и прочим сторонним трекерам работать на своих страницах, собирать аналитические данные, перенаправлять рекламу на другие веб-сайты (осуществлять ретаргетинг) и монетизировать доступное рекламное пространство, размещая тщательно таргетированные объявления. Рекламодатели собирают данные о пользователях с помощью различных механизмов отслеживания, включая файлы cookie, объекты HTML5, JavaScript-код, опознавание устройств и браузеров и другие популярные веб-технологии. Если пользователь посещает ряд веб-сайтов, которые сотрудничают с одной рекламной сетью, она распознает устройство пользователя и отслеживает его поведение в динамике.

С помощью отслеживающего ПО рекламная сеть или другая третья сторона может получить информацию о поведении пользователя в социальных сетях, о его контактах, предпочтениях, интересах, покупках и т. д. Это позволяет точно установить, привела ли реклама к совершению покупки, изучить связи между людьми, создать подробные профили отслеживания, осуществить качественный таргетинг, а также многое другое в силу масштабности отслеживания.

Индивидуальное отслеживание пользователей с помощью сторонних (или собственных) трекеров возможно, даже если человек не зарегистрирован в конкретном сервисе (социальной сети или поисковой системе), перестал его использовать или вышел из системы. При этом на сегодняшний день сторонние трекеры все больше сосредоточиваются в руках крупных провайдеров.

Наряду с применением файлов cookie существуют и другие методы стороннего отслеживания. Среди них снятие цифрового отпечатка пользователя (разновидность цифрового отпечатка браузера), повтор сеанса (позволяющий третьей стороне воспроизвести все взаимодействия пользователя с конкретной страницей) и даже использование функции автозаполнения браузера или менеджера паролей для возвращения данных из веб-форм еще до заполнения пользователем всей формы. Эти усовершенствованные технологии позволяют

получить подробные сведения о поведении и данных пользователя, включая такие детали, как число прокруток страницы и щелчков мыши, а в некоторых случаях даже логин и пароль для конкретного сайта (придуманные самим пользователем или созданные автоматически).

Согласно недавнему исследованию, определенные виды стороннего отслеживающего ПО используются повсеместно. При этом новостные сайты содержат наибольшее количество сторонних трекеров на любом собственном сайте. Кроме того, отслеживание широко применяется на сайтах, посвященных искусству и спорту, а также на торговых площадках. **Перекрестное отслеживание устройств (cross-device tracking)** подразумевает установление связи между действиями пользователя на нескольких устройствах (настольных ПК, планшетах, смартфонах и т. д.). Это позволяет собрать данные о поведении человека, даже если он использует разные устройства.

Перекрестное отслеживание устройств может оказаться полезным. Например, оно способно повысить качество пользовательского опыта. При использовании файлов cookie на одном устройстве или в одном браузере оно обеспечивает комфортный переход между устройствами (например, позволяя продолжить чтение книги или просмотр фильма с того места, где пользователь остановился в прошлый раз). Также перекрестное отслеживание помогает предотвращать мошенничество. Если пользователь авторизовался с незнакомого устройства в совершенно новом месте, поставщик услуг может выполнить дополнительные действия по его аутентификации (например, использовать двухфакторную аутентификацию).

Хотя перекрестное отслеживание устройств больше применяется для собственных сервисов, в частности, провайдерами почтового сервиса, контента (например, потокового видео) или интернет-магазинами, сторонние трекеры тоже все чаще отслеживают поведение пользователей на нескольких устройствах.

1. Перекрестное отслеживание устройств может быть детерминированным, основанным на постоянном идентификаторе (например, логине, привязанном к конкретному пользователю).
2. Оно также может носить вероятностный характер; в качестве вероятностного идентификатора можно использовать IP-адрес. Например, трансляция сетевого адреса может привести к тому, что несколько устройств в сети будут иметь одинаковый публичный IP-адрес. Допустим, пользователь посещает веб-сайт с мобильного устройства (например, смартфона), используя его и дома, и на работе. Третья сторона может задать информацию об IP-адресе в расположенных на устройствах файлах cookie. Тогда этот пользователь сможет появляться с двух публичных IP-адресов (с одного на работе и со второго дома), и эти IP-адреса будут связаны между собой с помощью того же стороннего файла cookie. Если пользователь посетит сайт третьей стороны с других устройств, которые также используют один из этих IP-адресов, их можно будет с большой долей уверенности привязать к тому же пользователю.

Детерминированный и вероятностный подходы часто совмещаются; при этом зачастую для реализации такого отслеживания даже не требуется, чтобы

пользователь авторизовался на каком-либо сайте. Иногда третья сторона предлагает сервисы «аналитики», которые встраиваются на множество веб-сайтов, не имеющих отношения к разработчикам этих сервисов. Это позволяет третьей стороне отслеживать поведение пользователя в рамках нескольких сайтов и устройств. Третьи стороны часто сотрудничают друг с другом с целью перекрестного отслеживания пользователей с помощью синхронизации cookie-файлов (мы подробно рассмотрим ее далее в этом разделе).

Перекрестное отслеживание устройств делает возможным более продвинутый логический вывод общей информации о действиях пользователя за счет сочетания данных, полученных с разных устройств. Данные о местоположении пользователя (полученные с мобильного устройства) можно объединить с историей поиска и сведениями о его активности в социальной сети (например, количестве поставленных лайков). Эти сведения помогут определить, посетил ли он физический магазин после поиска в интернете или после просмотра рекламного объявления.

### *Снятие цифрового отпечатка устройства и браузера*

Даже когда пользователи отключают такие распространенные механизмы отслеживания, как сторонние cookie, веб-сайты и третьи стороны по-прежнему могут их отслеживать, используя информацию, которая возвращается устройством серверу (об окружении, контексте и устройстве). Основываясь на таком наборе сведений, третьей стороне часто удается однозначно идентифицировать пользователя, то есть создать его цифровой отпечаток на разных сайтах в течение некоторого времени.

Один из популярных методов — **снятие цифрового отпечатка хоста (canvas fingerprinting)**, при котором для идентификации устройства используется тег HTML `<canvas>`. Этот тег позволяет веб-приложениям отображать графику в режиме реального времени. Из-за различий в размерах, способе рендеринга шрифтов, подходе к сглаживанию и т. д. устройства могут по-разному отображать картинки, и результирующие пиксели можно использовать в качестве цифрового отпечатка. Данный метод был впервые обнаружен в 2012 году, но широкая общественность узнала о нем только в 2014 году. Несмотря на предпринятые ответные меры, многие трекеры продолжают использовать этот подход, а также схожие методы (например, идентификация устройства с помощью снятия цифрового отпечатка на основе браузерных списков шрифтов). Недавнее исследование показало, что снятие цифрового отпечатка по-прежнему используется на тысячах сайтов. Также устройства можно отследить с помощью API браузеров, например, извлекая информацию о состоянии аккумулятора: уровень его заряда и время разрядки. В ряде исследований описывается, как такая информация помогает отследить устройство и ассоциировать его с пользователем (Олейник и др.; Olejnik et al., 2015).

### *Синхронизация cookie-файлов*

Обмениваясь информацией, сторонние трекеры могут отслеживать отдельных пользователей даже при посещении веб-сайтов с разными механизмами

отслеживания. Трудно поддающийся выявлению метод **синхронизации cookie-файлов (cookie syncing)** позволяет отдельным третьим сторонам легко объединить свои наборы сведений о конкретных пользователях, что создает серьезную угрозу конфиденциальности. Согласно последним исследованиям, сторонние трекеры широко используют данный метод.

## 7.4. ПОТОКОВАЯ ПЕРЕДАЧА АУДИО И ВИДЕО

Помимо электронной почты и веб-приложений, существует еще одна значительная область применения сетей. Для многих краеугольным камнем сетевых технологий стала передача аудио и видео. Слово «мультимедиа» одинаково будоражит и технарей, и коммерсантов. Одни видят в мультимедиа бесконечный источник интересных технических проблем, связанных с обеспечением качественной IP-телефонии или доставки видео по запросу, другие — столь же безграничный источник прибыли.

Хотя идея отправки мультимедиа через интернет появилась еще в 70-х годах прошлого столетия, только в начале XXI века передача **аудио и видео в реальном времени (real-time audio; real-time video)** стала возможной. Трафик в реальном времени отличается от обычного интернет-трафика: для такой передачи нужно обеспечить заранее определенную скорость воспроизведения, иначе передаваемые данные окажутся бесполезными. Никто не захочет смотреть видео в замедленном темпе и с постоянными прерываниями. Между тем при работе в интернете возникают короткие перебои, а на загрузку страницы может уйти некоторое время (в установленных пределах), но обычно это не является серьезной проблемой.

Такому развитию способствовали две вещи. Во-первых, значительно возросла мощность компьютеров; при этом они стали оснащаться микрофонами и камерами, что дает возможность легко вводить, обрабатывать и выводить аудио и видео. Во-вторых, кратно выросла пропускная способность интернета. Скорость на магистральных линиях в основе интернета достигает многих гигабитов в секунду, а широкополосные и беспроводные (802.11ac) сети обслуживают пользователей на его периферии. Все это позволяет провайдерам пересылать огромные объемы трафика, а пользователи могут подключаться к интернету в 100–1000 раз быстрее по сравнению с обычными телефонными модемами, работавшими на скорости 56 Кбит/с.

С расширением пропускной способности вырос аудио- и видеотрафик, хотя и по разным причинам. Телефонные разговоры занимают сравнительно небольшую часть полосы (64 Кбит/с, а при сжатии еще меньше), но телефонные услуги всегда были дорогими. Компании обнаружили, что можно передавать телефонный трафик по интернету, тем самым снижая свои расходы. Такие стартапы, как Skype, когда-то позволили пользователям звонить бесплатно, по интернет-соединению. Расторопные телефонные операторы нашли дешевый способ передавать обычные голосовые звонки, используя оборудование, предназначенное для выхода в сеть. В итоге многократно возрос объем передаваемых по интернету голосовых данных посредством так называемой **интернет-телефонии (Internet telephony)**; мы поговорим о ней в разделе 7.4.4.

В отличие от аудио, видео занимает значительную часть пропускной способности канала. Интернет-видео приемлемого качества кодируется сжатием, что дает поток со скоростью около 8 Мбит/с в 4K (это 7 Гбайт для 2-часового фильма). До появления широкополосного доступа передача видео по сети была невозможна. Но все изменилось: теперь люди с большим удовольствием смотрят потоковые видео хорошего качества, не выходя из дома. Около четверти пользователей каждый день заходят на популярный видеохостинг YouTube. На смену прокату видеофильмов на кассетах и DVD пришло скачивание из интернета. Огромное количество размещенных в сети видеороликов совершенно изменило структуру интернет-трафика. Уже сегодня большая часть информации в интернете — это видеофайлы, а через несколько лет на их долю будет приходиться около 90 % трафика.

Учитывая, что ширины полосы пропускания достаточно для передачи видео и аудио, ключевой проблемой разработки приложений для проведения конференций и передачи мультимедиа является сетевая задержка. Для аудио и видео требуется представление в реальном времени, то есть они должны проигрываться с определенной скоростью, иначе в них не будет смысла. Долгие задержки предполагают, что звонки, которые должны быть интерактивными, таковыми не являются. Если вы хоть раз разговаривали по спутниковому телефону, вы поймете, о чем речь. Ведь в этом случае задержка даже на полсекунды ужасно раздражает. При проигрывании музыки и фильмов по сети абсолютная задержка не играет роли, так как она влияет только на то, когда начнется воспроизведение файла. Но варьирование задержки (джиттер) по-прежнему имеет значение. Оно должно маскироваться плеером, иначе аудио будет неразборчивым, а видео дерганым.

В качестве отступления следует сказать, что термин **«мультимедиа» (multimedia)** часто используется применительно к аудио и видео в интернете. Буквально слово «мультимедиа» означает использование нескольких видов данных. Согласно этому определению, данная книга тоже является мультимедийной презентацией, поскольку содержит текст и графику (иллюстрации). Однако вы вряд ли так себе представляли мультимедийные данные, поэтому здесь под словом «мультимедиа» мы будем понимать использование нескольких видов **непрерывных данных (continuous media)**, для проигрывания которых требуется определенный интервал времени. На практике чаще всего это аудио и видео, то есть звук плюс движущееся изображение. Возможно, когда-нибудь удастся обеспечить непрерывное воспроизведение звука в сочетании с запахами. Многие также ошибочно относят к мультимедийным данным чисто звуковую информацию, используемую, например, в случае интернет-телефонии или интернет-радио. Строго говоря, в таких случаях уместнее термин **«потоковые данные»**. Несмотря на это, мы поддадимся общей тенденции и тоже будем рассматривать аудио в реальном времени как мультимедиа.

## 7.4.1. Цифровой звук

Звуковая волна представляет собой одномерную акустическую волну (давление). Когда она достигает уха, барабанная перепонка вибрирует, вызывая вибрацию тонких костей внутреннего уха, в результате чего в мозг по нерву

идет пульсирующий сигнал. Эта пульсация воспринимается слушателем как звук. Подобным образом, когда акустическая волна воздействует на микрофон, он формирует электрический сигнал, представляющий собой амплитуду звука как функцию времени.

Человеческое ухо способно слышать сигналы в диапазоне частот от 20 до 20 000 Гц, а некоторые животные, например собаки, могут слышать и более высокие частоты. Громкость, воспринимаемая ухом, изменяется логарифмически по отношению к амплитуде, поэтому отношение силы двух звуков с амплитудами $A$ и $B$ обычно измеряется в **децибелах (дБ)**, как $10 \log10(A/B)$. Если принять за 0 дБ нижний порог слышимости (звуковое давление на уровне 20 мкПа) для синусоидальной волны частотой 1 кГц, то громкости обычного разговора будет соответствовать 50 дБ, а болевой порог наступит при силе звука около 120 дБ. Динамический диапазон сигнала (отношение между самыми большими и самыми маленькими значениями) при этом будет больше 1 миллиона.

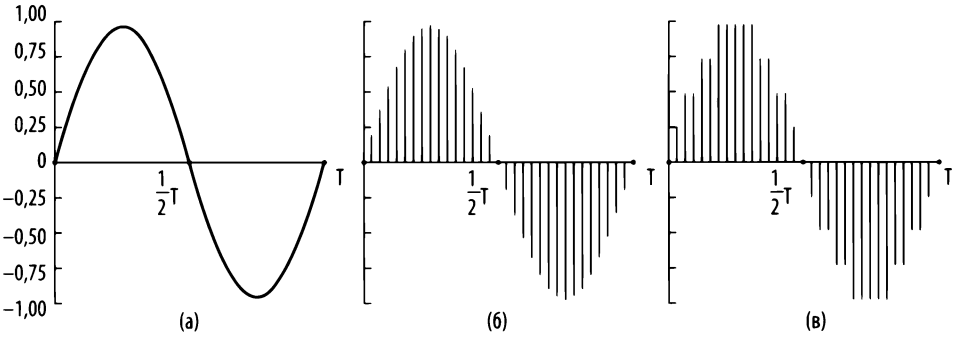
Человеческое ухо очень чувствительно к изменениям звука, длящимся всего несколько миллисекунд. Глаз, напротив, не способен заметить такие кратковременные изменения. Таким образом, джиттер в несколько миллисекунд при передаче мультимедиа влияет в большей степени на качество звука, чем на качество изображения.

Цифровое аудио — это цифровое представление звуковой волны, которое можно использовать для ее воссоздания. Звук можно преобразовывать в цифровую форму при помощи **аналогово-цифрового преобразователя (АЦП)**. На вход АЦП подается электрическое напряжение, а на выходе формируется двоичное число. На илл. 7.31 (а) показан пример синусоидальной волны. Чтобы представить этот сигнал в цифровом виде, мы можем измерять значения сигнала через равные интервалы времени $\Delta T$ (то есть производить квантование сигнала), как показано на илл. 7.31 (б). Если звуковая волна не является чисто синусоидальной, а представляет собой сумму нескольких синусоидальных волн и самая высокая частота ее составляющих равна $f$, тогда, согласно теореме Найквиста (см. главу 2), для последующего восстановления сигнала достаточно измерять его значения с частотой дискретизации $2f$. Производить замеры сигнала с большей частотой нет смысла, так как более высокие частоты в нем отсутствуют.

Обратный процесс заключается в переводе цифровых значений в аналоговое электрическое напряжение с помощью **цифро-аналогового преобразователя (ЦАП)**. Затем репродуктор переводит аналоговое напряжение в акустические волны, и люди слышат звуки.

### *Сжатие звука*

Несмотря на то что аудиоданные требуют не такой большой пропускной способности, как видеоданные, звук часто сжимается для того, чтобы сократить требуемую полосу пропускания канала и время передачи. Во всех системах сжатия должно быть два алгоритма: один для сжатия данных на стороне источника и второй для их распаковки на стороне адресата. В литературе они называются алгоритмами **кодирования (encoding)** и **декодирования (decoding)**. Мы также будем использовать эту терминологию.




**Илл. 7.31.** (а) Синусоидальная волна. (б) Дискретизация. (в) Квантование сэмплов 4 битами

Важно понимать, что алгоритмы сжатия обладают некоторой асимметрией. Хотя сейчас мы говорим об аудио, это так же справедливо и для видео. В первую очередь, асимметрия проявляется в кодировании исходных данных. Обычно кодирование мультимедийного документа производится только один раз (при его сохранении на мультимедийном сервере), а его декодирование — тысячи раз (при его проигрывании пользователями). Эта асимметрия означает, что ситуация, когда алгоритм кодирования работает медленно и нуждается в дорогом оборудовании, вполне допустима, при условии, что алгоритм декодирования будет быстрым и дешевым.

Второе нарушение симметрии состоит в том, что процесс кодирования/декодирования не всегда обратим. То есть обычно ожидается, что после сжатия, передачи и декомпрессии файла пользователь получит точную копию оригинала. В случае мультимедийных данных этого требования нет. Обычно допускается небольшое различие результата декодирования аудио- или видеосигнала и оригинала, при условии, что звучит (или выглядит) он так же. Если результат декодирования отличается от исходных входных данных, значит, использовалась система **с потерями (lossy)**. Если входные данные и результат идентичны, мы имели дело с системой **без потерь (lossless)**. Системы с потерями играют важную роль, поскольку позволяют обеспечить гораздо лучшее сжатие за счет потери небольшой части информации.

Для сжатия аудиофайлов было разработано множество алгоритмов. Пожалуй, наиболее популярными являются формат **MP3 (MPEG audio layer 3 — MPEG[1] аудио, уровень 3)** и формат **ACC (Advanced Audio Coding — усовершенствованное кодирование аудио)** в том виде, как он используется в файлах **MP4 (MPEG-4)**. Чтобы не путаться, запомните, что MPEG определяет сжатие аудио- и видеоданных. MP3 обозначает третью, относящуюся к аудиоданным, часть стандарта MPEG-1, а не третью версию стандарта MPEG, на смену которой пришла версия MPEG-4. AAC — это формат, призванный заменить MP3; он применяется по умолчанию для кодирования аудиоданных в стандарте MPEG-4. MPEG-2

[1] Motion Picture Experts Group — Экспертная группа по кинематографии. — *Примеч. ред.*

позволяет использовать оба варианта кодирования аудиоданных, и MP3, и AAC. Теперь понятно? Что хорошо в стандартах, так это их разнообразие. А если вам не нравится ни один из них, просто подождите год-другой.

Существует два подхода к сжатию звука. При **кодировании формы сигналов (waveform coding)** сигнал раскладывается на компоненты при помощи преобразования Фурье. В главе 2 мы рассматривали пример разложения в ряд Фурье временной функции (см. илл. 2.12 (а)). Амплитуда каждой компоненты кодируется с минимальными искажениями. Задача в том, чтобы довольно точно воспроизвести форму сигнала, используя для этого как можно меньше битов.

Вторым подходом является **перцепционное кодирование (perceptual coding)**. С учетом недостатков слухового аппарата человека сигнал кодируется так, чтобы слушатель не заметил никакой разницы в звучании, даже если на осциллографе результат выглядит совершенно иначе. В основе перцепционного кодирования лежит область науки, изучающая восприятие звука человеком, — **психоакустика (psychoacoustics)**. И MP3, и AAC используют перцепционное кодирование.

Поскольку в современных мультимедийных системах главным образом применяется перцепционное кодирование, мы подробнее остановимся на этом подходе. Его ключевой особенностью является то, что одни звуки могут маскировать другие. Допустим, вы транслируете живой концерт флейты в теплый летний день. Вдруг, откуда ни возьмись, появляется бригада рабочих и начинает вскрывать на улице асфальт отбойными молотками. Расслышать флейту уже невозможно, и вы передаете только частоты отбойных молотков. Слушатели при этом слышат то же самое, как если бы вы передавали и звуки флейты, а вы экономите пропускную способность. Это называется **частотным маскированием (frequency masking)**.

После прекращения работы отбойных молотков какое-то время можно не транслировать частоты флейты, поскольку человеческое ухо снижает свою чувствительность, когда слышит громкие звуки, и для ее восстановления требуется некоторое время. Поскольку передавать звуки низкой амплитуды в течение этого периода восстановления бессмысленно, их лучше опустить, сэкономив тем самым пропускную способность. Это называется **временным маскированием (temporal masking)**. Отказ от кодирования или передачи тех аудиоданных, которые в любом случае не смогут услышать пользователи, является одной из главных составляющих метода перцепционного кодирования.

## 7.4.2. Цифровое видео

Теперь, когда мы узнали все об ушах, пора перейти к глазам. (И нет, в следующем разделе мы не будем обсуждать нос.) У человеческого глаза есть одна особенность: когда изображение появляется на сетчатке, оно сохраняется на ней на несколько миллисекунд, прежде чем исчезнуть. Если картинки сменяются со скоростью 50 изображений в секунду, глаз не замечает, что видит отдельные изображения. Этот принцип получения движущихся изображений используют все видеосистемы с тех пор, как в 1895 году братья Люмьер придумали первый кинопроектор.

Самое простое представление видео — это последовательность кадров, каждый из которых состоит из набора пикселей (прямоугольных элементов, составляющих изображение). Обычно размеры экранов составляют 1280 × 720 (обозначается как **720p**), 1920 × 1080 (**1080p** или **HD video**), 3840 × 2160 (**4K**) или 7680 × 4320 (**8K**).

В большинстве систем используется 24 бита на пиксель, по 8 бит для красного, зеленого и синего (RGB — red, green, blue) компонентов. Красный, зеленый и синий — это первичные аддитивные цвета; любой другой цвет можно получить путем их наложения друг на друга при соответствующей интенсивности.

Раньше частота кадров варьировалась от 24 кадров/с (в традиционном пленочном кино) до 25 кадров/с (в системе PAL, используемой в большинстве стран мира) и 30 кадров/с (в американской системе NTSC). Если быть абсолютно точным, в системе NTSC скорость кадров равна 29,97, а не 30 кадров/с, из-за модификации, проведенной инженерами при переходе от черно-белого телевидения к цветному. Для управления цветом нужно было выделить еще немного полосы пропускания, что удалось сделать, снизив частоту кадров на 0,03 кадра/с. В системе PAL цвет использовался изначально, поэтому здесь частота кадров действительно составляет 25,00 кадра/с. Во Франции применяется слегка отличающаяся система под названием SECAM. Отчасти она была создана, чтобы защитить французские компании от немецких производителей телевизионной техники. Частота кадров в этой системе также составляет ровно 25,00 кадра/с. В 1950-е годы социалистические страны Восточной Европы внедрили у себя SECAM, чтобы население этих стран не могло смотреть передачи западногерманского телевидения (использующие PAL) и поддаваться влиянию «плохих идей».

Чтобы уменьшить полосу пропускания, необходимую для эфирного телевизионного вещания, телестанции взяли на вооружение схему, при которой кадр разделялся на два **поля** (одно с четными и одно с нечетными строками), передаваемых попеременно. При этом частота кадров фактически составляла не 25 кадров/с, а 50 полей/с. Эта схема называется **чересстрочной разверткой (interlacing)**. Она обеспечивает меньший уровень мерцания, чем при последовательной передаче целых кадров. Современные видеосистемы не используют чересстрочную развертку и просто последовательно передают целые кадры, обычно с частотой 50 кадров/с (PAL) или 59,94 кадра/с (NTSC). Это называется **прогрессивной разверткой**.

### *Сжатие видео*

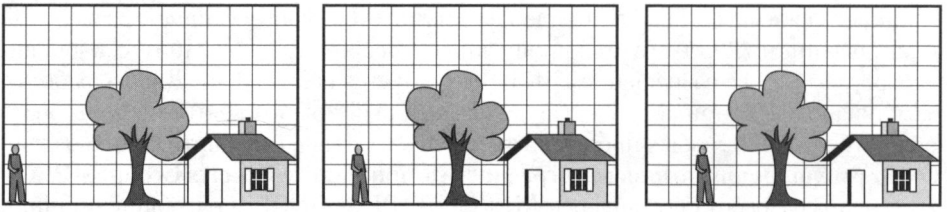
Из сказанного выше очевидно, что при передаче видео по интернету критически важно его сжимать. Даже для видео формата PAL с прогрессивной разверткой и разрешением 720p требуется пропускная способность в 553 Мбит/с, а для видео формата HD с разрешением 4K и 8K требуется еще больше. Для разработки международного стандарта сжатия видео, который можно было бы использовать на платформах от любых производителей, комитеты по стандартизации создали **Экспертную группу по кинематографии (Motion Picture Experts Group, MPEG)**. Эта группа разработала стандарты MPEG-1, MPEG-2 и MPEG-4, общий принцип действия которых выглядит следующим образом. С периодичностью в несколько

секунд передается полный видеокадр. Кадр сжимается с помощью алгоритма, похожего на привычный алгоритм JPEG для неподвижных цифровых изображений. Затем в течение нескольких секунд вместо отправки полных кадров передатчик отправляет различия между текущим и базовым (последним полным) кадром.

Сначала кратко ознакомимся с алгоритмом **JPEG (Joint Photographic Experts Group — Объединенная экспертная группа по фотографии)** для сжатия одиночных неподвижных изображений. Вместо того чтобы работать с RGB-компонентами, он преобразует изображение в компоненты **яркости (luminance)** и **цветности (chrominance)**. Тот факт, что глаз человека намного более чувствителен к яркости, чем к цветности, позволяет использовать для кодирования цветности меньшее количество битов без потери воспринимаемого качества изображения. Затем картинка делится на множество отдельно обрабатываемых блоков, размер которых обычно составляет 8 × 8 или 10 × 10 пикселей. Сигналы яркости и цветности по отдельности подвергаются дискретному косинусному преобразованию Фурье с получением спектра. После этого можно отбросить высокочастотные амплитуды. Чем больше амплитуд отбрасывается, тем более размытым является исходное изображение и тем меньше получаемое на выходе сжатое изображение. Затем к оставшимся амплитудам применяются стандартные методы сжатия без потерь, такие как кодирование по длинам серий или метод Хаффмана. Если этот процесс показался вам сложным, то это так и есть, однако компьютеры легко справляются с выполнением сложных алгоритмов.

Теперь посмотрим, как работают алгоритмы MPEG (в упрощенном виде). Кадр, идущий следом за полным (базовым) кадром JPEG, здесь выглядит почти так же, как в случае алгоритма JPEG. Поэтому вместо того, чтобы кодировать полный кадр, передаются только блоки с отличиями от базового кадра. Блок с фрагментом голубого неба, скорее всего, будет выглядеть так же и через 20 мс, и нет смысла отправлять его снова. Повторно следует передавать только те блоки, в которых произошли изменения.

Рассмотрим пример, когда камера закреплена на штативе, а актер идет к неподвижным дереву и дому. Первые три кадра показаны на илл. 7.32. При кодировании второго кадра отправляются лишь те блоки, в которых произошли изменения. Теоретически, чтобы его принять, получателю нужно просто скопировать первый кадр в буфер, а затем применить изменения. После этого он сохраняет второй кадр в несжатом виде для отображения на экране. Он также использует второй кадр как основу для применения следующих изменений, отражающих различия между третьим и вторым кадром.



**Илл. 7.32.** Три последовательных кадра

В реальности этот процесс чуть сложнее. Если блок (например, с актером) присутствует во втором кадре, но с некоторым смещением, алгоритм MPEG позволяет кодировщику сообщить следующее: «Блок 29 предыдущего кадра присутствует в новом кадре со смещением на расстояние ($\Delta x$, $\Delta y$), и кроме того, шестой пиксель теперь содержит значения *abc*, а 24-й пиксель — значения *xyz*». Это обеспечивает еще большее сжатие.

Ранее мы упоминали об асимметрии между кодированием и декодированием. Здесь мы видим еще один случай такой асимметрии. Кодировщик может сколько угодно искать смещенные и измененные блоки. Так он решает, что лучше: отправить список обновлений предыдущего кадра или новый полный кадр JPEG. При этом поиск смещенного блока требует гораздо больше усилий, чем его копирование из предыдущего кадра и вставка в новый кадр с известным смещением ($\Delta x$, $\Delta y$).

На самом деле для полноты картины в MPEG используются не два, а *три* разных вида кадров:

1. I-кадры (Intracoded — закодированные внутри), которые содержат неподвижные изображения, сжатые независимо от других кадров.
2. P-кадры (Predictive — предиктивные); отражают отличия от *предыдущего* кадра.
3. B-кадры (Bidirectional — двунаправленные); отражают отличия от *следующего* I-кадра.

B-кадры требуют, чтобы получатель останавливал обработку до поступления следующего I-кадра, а затем возвращался назад. Иногда это обеспечивает большую степень сжатия, но когда кодировщик постоянно проверяет, что даст результат с наименьшим размером — отличия от предыдущего кадра или отличия от одного из следующих 30, 50 или 80 кадров, это ведет к большим затратам времени на стороне кодирования без больших затрат на стороне декодирования. Эта асимметрия используется по максимуму для того, чтобы обеспечить на выходе файл минимально возможного размера. Стандарты MPEG не определяют, как и на каком расстоянии должен производиться поиск и какой должна быть степень совпадения, чтобы передавать отличия или новый полный блок. Все зависит от конкретной реализации.

Аудио и видео кодируются по отдельности описанным выше способом. Окончательный результат MPEG-кодирования представляет собой файл, состоящий из ряда сегментов, которые содержат определенное число сжатых изображений и соответствующие им сжатые аудиоданные; они должны воспроизводиться при отображении кадров данного сегмента. Это позволяет избегать рассогласования видео- и аудиоданных.

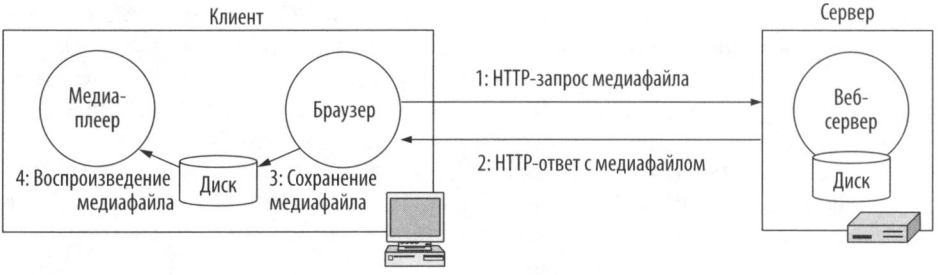
Обратите внимание, что это несколько упрощенное описание. В реальности данный подход использует еще несколько способов повышения степени сжатия, но изложенные выше основные принципы при этом не меняются. Последняя версия данного формата — это MPEG-4 (MP4). Ее формальное описание представлено в стандарте H.264. Следующей версией H.264 (предназначенной для описания разрешений до 8K) является стандарт H.265. Формат H.264

используется большинством потребительских видеокамер. Поскольку камера должна записывать видео на SD-карту или другой носитель в режиме реального времени, у нее остается совсем немного времени на поиск слегка сместившихся блоков. Как следствие, степень сжатия при этом очень далека от того уровня, который может обеспечить голливудская киностудия, динамически выделив на облачном сервере 10 000 компьютеров для кодирования своего творения. Это еще один пример реального проявления асимметрии между кодированием и декодированием.

## 7.4.3. Потоковая передача сохраненных медиафайлов

Теперь перейдем к сетевым приложениям. В первую очередь следует обсудить просмотр потокового видео, которое уже хранится где-то на сервере, как в случае с YouTube или Netflix. Самый распространенный пример — онлайн-просмотр видео. Это одна из форм **видео по запросу (Video on Demand, VoD)**. Другие формы VoD для передачи фильмов используют провайдерскую сеть, которая не является частью интернета (например, сеть кабельного телевидения).

В интернете очень много сайтов с музыкой и видео, предоставляющих потоковый доступ к хранящимся на них медиафайлам. По сути, самый простой способ обработки таких медиафайлов — *не* передавать их в потоковом режиме. Намного проще рассматривать предварительно закодированный видео- или аудиофайл как очень большую веб-страницу и позволить браузеру его скачать. Эта последовательность из четырех шагов показана на илл. 7.33.




**Илл. 7.33.** Воспроизведение медиафайлов по интернету путем обычного скачивания

Браузер начинает действовать, когда пользователь кликает по названию фильма. На первом шаге отсылается HTTP-запрос фильма на веб-сервер, на который указывает ссылка. На втором шаге сервер получает фильм (который представляет собой обычный файл в формате MP4 или каком-нибудь другом) и отправляет его браузеру. Исходя из MIME-типа файла, браузер выбирает способ воспроизведения. На третьем шаге браузер сохраняет весь фильм на диске во временном файле и запускает медиаплеер, которому передается имя временного файла. Наконец, на четвертом шаге медиаплеер начинает читать файл и проигрывать фильм. Теоретически это мало чем отличается от доставки и отображения статической веб-страницы, разница лишь в том, что загруженный

файл «отображается» с помощью медиаплеера, а не просто путем записи пикселей в монитор.

В целом это вполне корректный подход, который позволит нормально воспроизвести файл с фильмом. У вас не будет никаких проблем с передачей данных по сети в режиме реального времени, поскольку в данном случае требуется лишь скачать файл. Правда, до начала воспроизведения необходимо передать по сети всю видеозапись. Большинство клиентов не хочет ждать целый час, пока начнется воспроизведение выбранного ими «видео по запросу», поэтому нужно найти решение получше.

На помощь приходит медиаплеер для потоковой передачи. Это может быть либо компонент веб-браузера, либо внешняя программа, вызываемая браузером, когда требуется воспроизвести видео. Современные браузеры с поддержкой HTML5 имеют встроенный медиаплеер.

Медиаплеер решает пять основных задач:

1. Управление интерфейсом пользователя.
2. Обработка ошибок передачи.
3. Декомпрессия сжатых данных.
4. Устранение джиттера.
5. Дешифрование файла.

Большинство современных медиаплееров обладает привлекательным интерфейсом, иногда имитирующим внешний вид стереосистемы с блестящими кнопками, ручками, ползунками и дисплеями. Зачастую пользователь может менять внешний вид плеера, выбирая разные «лицевые панели», **скины (skins)**. Медиаплеер должен всем этим управлять, обеспечивая взаимодействие с пользователем.

Следующие три задачи связаны друг с другом и зависят от используемых сетевых протоколов. Рассмотрим их по очереди, начиная с обработки ошибок передачи. Ее сложность зависит от того, какой транспортный протокол используется для доставки медиаданных. Это может быть протокол, основанный на TCP (такой, как HTTP) или на UDP, например **протокол реального времени (Real Time Protocol, RTP)**. При использовании транспортного протокола на основе TCP медиаплееру не придется исправлять ошибки, ведь TCP и так обеспечивает высокую надежность за счет повторной передачи. Это упрощает (по крайней мере, для медиаплеера) обработку ошибок, но в то же время усложняет удаление джиттера на более позднем этапе, поскольку тайм-ауты и повторная передача могут приводить к нестабильным и переменным задержкам при воспроизведении фильма.

В качестве альтернативы для передачи данных можно использовать транспортный протокол на основе UDP, например RTP. Такие протоколы не производят повторную передачу данных. То есть потеря пакетов из-за перегрузки или ошибок передачи приведет к тому, что часть медиаданных не будет доставлена. Решать эту проблему приходится медиаплееру. Можно просто игнорировать ее, допуская наличие ошибок в видео и аудио. Если ошибки случаются нечасто, такой подход будет работать и ошибки будут практически не заметны. Другой

подход состоит в том, чтобы использовать **упреждающую коррекцию ошибок (forward error correction)**, в частности, путем кодирования видеофайла с некоторой долей избыточности (например, с использованием кода Хэмминга или кода Рида — Соломона). В таком случае у медиаплеера будет достаточно информации для того, чтобы исправлять ошибки самостоятельно, и ему не потребуется запрашивать повторную передачу данных или пропускать поврежденные участки фильмов.

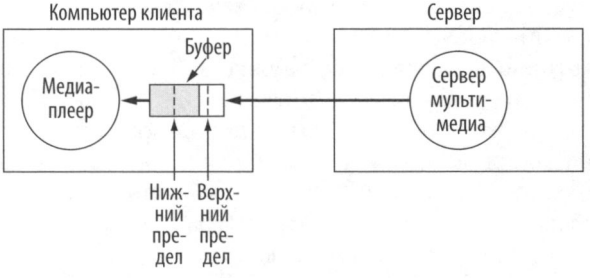
Недостатком этого метода является то, что внесение избыточности в файл ведет к увеличению его размера. Еще один подход сводится к выборочному повтору передачи наиболее важных для воспроизведения контента частей видеопотока. Например, в случае сжатого видеоряда потеря пакета в I-кадре имеет самые серьезные последствия, поскольку ошибки декодирования, возникающие в результате такой потери, могут распространяться на целую группу изображений. С другой стороны, потери в производных кадрах (P- и B-кадрах) наносят гораздо меньший урон. Схожим образом, ценность повторной передачи также зависит от того, успеет ли заново отправленный контент поступить к моменту его воспроизведения. В результате некоторые повторные передачи намного важнее других, и одна из возможных стратегий сводится к тому, чтобы выборочно повторять передачу определенных пакетов (например, пакетов внутри I-кадра, которые придут к моменту воспроизведения). Над RTP и QUIC был надстроен ряд протоколов, обеспечивающих неравномерную защиту от потерь при потоковой передаче видео по UDP; см. работу Фимстера и др. (Feamster et al., 2000), а также Палмера и др. (Palmer et al., 2018).

Третья задача медиаплеера — декомпрессия сжатых данных; это достаточно просто, хотя и затратно с точки зрения вычислений. Сложным моментом является лишь декодирование медиаданных в том случае, когда сетевой протокол не исправляет ошибки передачи. Во многих схемах сжатия данные, полученные позже, невозможно декодировать, пока не будут декодированы предыдущие данные (поскольку более поздние данные кодируются относительно более ранних). Как вы помните, P-кадр строится на основе последнего I-кадра (и всех последующих I-кадров). Если I-кадр поврежден и его не удается декодировать, то все последующие P-кадры бесполезны. При этом медиаплеер вынужден ждать следующего I-кадра, просто пропуская несколько секунд видео.

Из-за этого кодировщик вынужден делать выбор. Если I-кадры идут вплотную, например, с интервалом в 1 с, то пауза в случае ошибки будет незначительной, но видеофайл будет крупнее, поскольку I-кадры намного больше, чем P- или B-кадры. Если I-кадры идут, скажем, с интервалом в 5 с, то видеофайл гораздо меньше, но мы получаем 5-секундную паузу при повреждении I-кадра и паузу меньшего размера при повреждении P-кадра. В силу этого, когда в качестве базового протокола применяется TCP, интервал между I-кадрами может быть намного больше, чем при RTP. Поэтому многие сайты потокового видео используют TCP, чтобы получать файлы меньшего размера с крупными интервалами между I-кадрами и меньшей полосой пропускания, необходимой для плавного воспроизведения.

Четвертой задачей является устранение джиттера, главной проблемы всех систем реального времени. Использование TCP серьезно ее усложняет, поскольку

оно вносит случайные задержки каждый раз, когда требуется выполнить повторную передачу. Распространенное решение, к которому прибегают все системы потоковой передачи, сводится к использованию буфера воспроизведения. Перед проигрыванием система буферизует медиаданные для 5–30 с воспроизведения (илл. 7.34). Медиаданные постоянно извлекаются из буфера для отчетливого и плавного воспроизведения звука и видео. Задержка при запуске позволяет заполнить буфер до **нижнего предела (low-water mark)**. При этом ожидается, что в дальнейшем новые данные будут поступать достаточно регулярно для того, чтобы буфер не оказался пустым. Если это все же произойдет, воспроизведение будет остановлено.




**Илл. 7.34.** Медиаплеер буферизует входящую информацию с медиасервера и воспроизводит медиаданные из буфера, а не напрямую из сети

Буферизация еще больше усложняет процесс. Медиаплеер поддерживает буфер в частично заполненном состоянии, в идеале где-то между нижним и верхним пределами. Это значит, что если объем данных в буфере достигает верхнего предела, плеер сообщает источнику о необходимости прекратить отправку данных, чтобы они не потерялись из-за отсутствия места для их размещения. Верхний предел при этом должен немного не доходить от конца буфера, поскольку потоковая передача данных продолжается, пока медиасервер не получит запрос остановки (`STOP`). После того как сервер прекращает отправку и канал становится пустым, объем данных в буфере начинает уменьшаться. Когда он достигает нижнего предела, плеер отправляет серверу команду запуска (`START`), чтобы возобновить потоковую передачу.

Благодаря протоколу, который позволяет сообщать серверу о необходимости остановки и запуска передачи, плеер может держать в буфере достаточно, но не слишком много медиаданных, чтобы обеспечить плавное воспроизведение. Поскольку ОЗУ на сегодняшний день стоит довольно дешево, медиаплеер даже на смартфоне может выделить достаточно места в буфере для нескольких минут воспроизведения, если нужно.

Механизм запуска и остановки обладает еще одной приятной особенностью: с ним скорость передачи сервера не привязана к скорости воспроизведения. Допустим, плееру нужно воспроизвести видео со скоростью 8 Мбит/с. Когда объем данных в буфере опустится до нижнего предела, плеер запросит у сервера доставку дополнительных данных. Если сервер способен производить доставку со скоростью 100 Мбит/с, это не вызовет проблем. Поступающие данные будут просто сохраняться в буфере. При достижении верхнего предела плеер сообщит

серверу о необходимости остановки. Таким образом, скорость передачи сервера и скорость воспроизведения совершенно не зависят друг от друга. То, что изначально было системой реального времени, стало обычной системой передачи файлов. Избавление от всех требований к передаче в реальном времени — одна из причин, почему Youtube, Netflix, Hulu и другие сервисы потоковой передачи используют TCP. Это существенно упрощает дизайн всей системы.

Определить подходящий размер буфера не так просто. Если доступно много оперативной памяти, может показаться, что лучше использовать большой буфер, позволив серверу держать его почти заполненным, на случай перегрузки сети. Но следует учесть, что люди порой привередливы. Посчитав какую-нибудь сцену скучной, пользователь может нажать кнопку перемотки вперед, и большая часть или все содержимое буфера станет бесполезным. Как бы там ни было, переход к определенному моменту времени вперед или назад сработает, только если этот кадр является I-кадром. В противном случае плееру нужно найти ближайший I-кадр. Если новая точка воспроизведения находится за пределами буфера, его нужно полностью очистить и загрузить заново. То есть если пользователь часто перематывает видео вперед или назад (а так поступают многие), он фактически впустую тратит пропускную способность сети, делая бесполезными данные буфера, загрузка которых требовала определенных затрат. В рамках всей системы наличие пользователей, склонных часто перематывать видео, — серьезный аргумент в пользу ограничения размера буфера, даже несмотря на низкую стоимость больших объемов ОЗУ. В идеале медиаплеер должен понаблюдать за поведением пользователя и выбрать размер буфера, исходя из свойственной этому человеку манеры просмотра.

Поскольку все коммерческие видео шифруются с целью защиты от пиратства, медиаплееры должны уметь дешифровать их по мере их поступления. Это пятая задача в приведенном выше списке.

### DASH и HLS

Далее мы коснемся проблем, возникающих из-за многообразия устройств для просмотра мультимедийного контента. Если пользователь купил себе яркий, блестящий и очень дорогой монитор с разрешением экрана 8K, ему нужно предоставлять видео с разрешением 7680 × 4320 и частотой кадров 100 или 120 кадров/с. Но если в ходе просмотра интересного фильма он отправится к врачу и будет досматривать этот фильм в приемной на смартфоне с разрешением 1280 × 720 и максимальной частотой кадров 25 кадров/с, возникнет проблема. Перед сайтом потокового вещания встает вопрос о том, какое разрешение и частоту кадров использовать при кодировании фильмов.

Самый простой выход — использовать все возможные комбинации. В худшем случае место на диске тратится на кодирование каждого фильма с использованием семи разрешений экрана (в формате смартфона, форматах NTSC, PAL, 720P, HD, 4K, 8K) и шести частот кадров (25, 30, 50, 60, 100 и 120). В совокупности мы получаем 42 варианта, но дисковое пространство сегодня стоит не так уж дорого. Более крупной сопутствующей проблемой является вопрос о том, что произойдет, если зритель останется дома со своим большим блестящим монитором, но

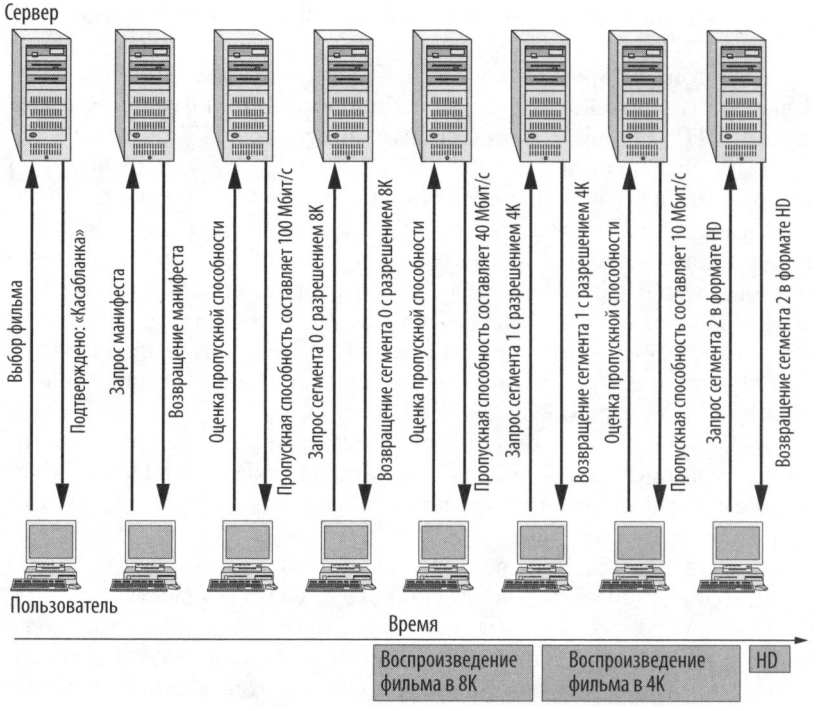
из-за перегрузок сети пропускная способность канала между ним и сервером будет сильно колебаться, не всегда позволяя использовать полное разрешение.

К счастью, в настоящее время уже реализовано несколько решений этой проблемы, в частности протокол **динамической адаптивной потоковой передачи данных по HTTP (Dynamic Adaptive Streaming over HTTP, DASH)**. Он основан на простой идее и совместим с HTTP (и HTTPS), благодаря чему его можно использовать для потоковой передачи на веб-странице. Сначала сервер потокового вещания кодирует свои фильмы с разным разрешением и частотой кадров, сохраняя их на своих хранилищах. Каждая версия сохраняется в виде множества файлов (вместо одного), которые содержат, скажем, 10 с видео и аудио. Это значит, что для 90-минутного фильма, кодируемого с использованием семи разрешений экрана и шести частот кадров (что дает 42 варианта), потребуется $42 \times 540 = 22\,680$ отдельных файлов с контентом для 10 с воспроизведения. Другими словами, каждый файл содержит сегмент фильма в конкретном разрешении и с определенной частотой кадров. С фильмом ассоциируется манифест — **описание представления медиаданных (Media Presentation Description, MPD)** — с именами всех файлов и их параметрами, включая разрешение, частоту кадров и номер кадра в фильме.

Чтобы этот подход работал, протокол DASH должен использоваться и плеером, и сервером. В качестве пользовательской стороны может выступать либо непосредственно браузер, либо плеер, встроенный в него в виде JavaScript-кода, либо специальное приложение (например, для мобильного устройства или телеприставки). Перед просмотром фильма DASH прежде всего доставляет манифест для него. Это небольшой файл, поэтому достаточно обычного HTTPS-запроса `GET`.

Затем плеер запрашивает у устройства, на котором он работает, информацию о максимальном разрешении и, возможно, другие сведения (к примеру, поддерживаемые аудиоформаты и количество динамиков). Затем он проводит ряд проверок, отправляя тестовые сообщения на сервер, чтобы оценить доступную пропускную способность. Получив эти данные и узнав разрешение экрана, плеер сверяется с манифестом, чтобы найти первые 10 с фильма с наилучшим качеством для имеющейся конфигурации.

Но это еще не конец истории. По мере воспроизведения фильма плеер продолжает запускать проверки пропускной способности. Каждый раз, когда ему нужен дополнительный контент (если объем медиаданных в буфере опускается до нижнего предела), он снова сверяется с манифестом и запрашивает подходящий файл в зависимости от того, какая часть фильма требуется и какое разрешение и частоту кадров нужно при этом использовать. Если пропускная способность сильно колеблется по мере воспроизведения, формат фильма может переключаться несколько раз в минуту от 8K при 100 кадрах/с к HD при 25 кадрах/с и обратно. При данном подходе система быстро адаптируется к изменению параметров сети и обеспечивает наилучший опыт просмотра видео в соответствии с имеющимися ресурсами. Такие компании, как Netflix, опубликовали информацию о том, как они корректируют битрейт видеопотока в зависимости от степени заполненности буфера воспроизведения (Хуан и др.; Huang et al., 2014). Пример показан на илл. 7.35.




**Илл. 7.35.** Использование DASH для изменения формата видео во время просмотра фильма

На илл. 7.35 по мере снижения пропускной способности плеер запрашивает версии со все более низким разрешением. Однако он также мог использовать и другие способы уменьшения объема данных. Например, отправка 300 кадров для 10 с воспроизведения потребует гораздо меньше пропускной способности, чем отправка 600 или 1200 кадров для тех же 10 с (даже с высокой степенью сжатия). В крайнем случае плеер мог запросить черно-белую версию с частотой 10 кадров/с, разрешением 480 × 320 и одноканальным звуком при наличии такой версии в манифесте. Протокол DASH позволяет плееру корректировать воспроизведение согласно меняющимся условиям, тем самым обеспечивая наилучший пользовательский опыт в конкретной ситуации. Логика работы плеера и способ запрашивания сегментов варьируются в зависимости от характера сервиса воспроизведения и особенностей устройства. Сервисы, избегающие повторной буферизации, могут запрашивать множество сегментов целыми группами; если же сервис стремится к максимальной интерактивности, он извлекает DASH-сегменты в более постоянном, размеренном темпе.

Протокол DASH продолжает развиваться. Например, ведется работа по снижению задержки (Ле Февр и др.; Le Feuvre et al., 2015), улучшению надежности (Ван и Рен; Wang and Ren, 2019), повышению равнодоступности (Алтамини и Ширмохаммади; Altamini and Shirmohammadi, 2019), обеспечению поддержки виртуальной реальности (Рибеццо и др.; Ribezzo et al., 2018), а также

по эффективной обработке видео формата 4K (Куинлан и Сринан; Quinlan and Sreenan, 2018).

DASH сегодня является самым распространенным протоколом потоковой передачи видео, хотя существуют и другие методы, о которых стоит упомянуть. Протокол **потоковой передачи в реальном времени на основе HTTP (HTTP Live Streaming, HLS)** от компании Apple также работает в браузере с использованием HTTP. Он рекомендуется для просмотра видео в браузере Safari на устройствах компании Apple (iPhone, iPad, MacBook и т. д.). Он также широко используется браузерами Microsoft Edge, Firefox и Chrome, на платформах Windows, Linux и Android. Кроме того, его часто поддерживают игровые консоли, «умные» телевизоры и другие устройства, способные воспроизводить мультимедийный контент.

Как и DASH, HLS требует, чтобы сервер кодировал фильм с разным разрешением и частотой кадров и каждый сегмент охватывал лишь несколько секунд видео. Это позволяет быстро адаптироваться к изменению условий. Протокол HLS также предлагает и ряд дополнительных функций, включая быструю прокрутку вперед и назад, субтитры на нескольких языках и многое другое. Он описан в RFC 8216.

Несмотря на одинаковые базовые принципы, протоколы DASH и HLS все же имеют некоторые различия. DASH инвариантен к кодеку: он может работать с видео, используя любой алгоритм кодирования. HLS работает только с теми алгоритмами, которые поддерживаются Apple, но поскольку туда входят H.264 и H.265, то этим различием можно пренебречь, ведь с их помощью сегодня сжимаются почти все видео. Протокол DASH позволяет третьей стороне легко вставлять рекламу в видеопоток, в то время как HLS не позволяет этого. С DASH можно использовать любую схему управления цифровыми правами, а HLS поддерживает только систему от Apple.

Протокол DASH — это открытый официальный стандарт, а HLS является проприетарным продуктом. При этом и у первого и у второго есть свои плюсы и минусы. Благодаря тому, что за HLS стоит мощный спонсор, он доступен на гораздо большем числе платформ, чем DASH, и его реализации отличаются чрезвычайной стабильностью. Однако, несмотря на то что на устройствах с iOS нет встроенной поддержки DASH, его используют и YouTube, и Netflix. По всей видимости, в ближайшие годы эти два протокола продолжат существовать параллельно.

Потоковое видео было одной из главных движущих сил интернета на протяжении десятилетий. Ретроспективный анализ этой технологии можно найти в работе Ли и др. (Li et al., 2013).

Актуальная проблема потоковой передачи видео — оценка **QoE** (то есть того, насколько пользователь доволен работой приложения). Измерить QoE напрямую довольно сложно: для этого пришлось бы опрашивать пользователей. Однако многие операторы сетей стремятся выявлять ситуации, когда потоковые приложения попадают в условия, влияющие на пользовательский опыт. Обычно операторы стараются оценить разрешение и величину задержки при запуске (как долго начинается воспроизведение видео), а также любые случаи остановки («повторной буферизации»). При зашифрованном видеопотоке получение этой

информации осложняется, особенно если интернет-провайдер не имеет доступа к клиентскому ПО. В таких случаях оценка качества приложения сегодня все чаще производится с помощью методов машинного обучения (Мангла и др.; Mangla et al., 2018; Бронзино и др.; Bronzino et al., 2020).

## 7.4.4. Потоковая передача в реальном времени

Огромной популярностью в интернете пользуются не только готовые видеозаписи, но и потоковая передача видео в реальном времени. Когда появилась технология потоковой передачи аудио- и видеоданных через интернет, коммерческие радиостанции и телеканалы тут же воспользовались этим, чтобы транслировать свой контент не только в эфире, но и онлайн. Вскоре появились университетские интернет-радиостанции. Затем собственные онлайн-трансляции стали вести *студенты*.

Сегодня живую аудио- и видеотрансляцию осуществляют как отдельные люди, так и компании самых разных масштабов. Вещание в реальном времени стало колыбелью инноваций, возникли новые стандарты и технологии. Чтобы обеспечить онлайн-присутствие, крупные телеканалы транслируются в интернете; это называется **IP-телевидением (IP TeleVision, IPTV)**. Радиостанции также работают онлайн; такое вещание называют **интернет-радио**. И IPTV, и интернет-радио используются по всему миру для освещения самых разных событий, от показов мод до мировых чемпионатов по футболу и отборочных состязаний по крикету. Помимо этого, с помощью технологии живого IP-вещания провайдеры кабельного телевидения создают собственные вещательные системы. Эта технология широко используется малобюджетными проектами от сайтов для взрослых до зоопарков. При современном уровне технологий практически любой человек может быстро и без значительных затрат начать живое вещание.

Один из подходов к живому вещанию сводится к тому, чтобы записывать контент на диск. Пользователь может подключиться к архивам сервера, выбрать любую передачу и скачать ее для прослушивания или просмотра. Скачанный выпуск программы называют **подкастом (podcast)**.

Потоковая трансляция событий в реальном времени иногда вносит дополнительные сложности. В случае прямой трансляции спортивных событий, новостей и длинных речей политиков можно по-прежнему использовать метод буферизации (илл. 7.34). После авторизации пользователя на сайте с прямой трансляцией события в течение первых нескольких секунд видео не отображается, пока не заполнится буфер. После этого все происходит так же, как при просмотре фильма. Плеер извлекает данные из буфера, который непрерывно заполняется. По сути, разница лишь в том, что при потоковой передаче фильма сервер может загружать 10 с видеофайла за 1 с, если соединение достаточно быстрое. При прямой трансляции события это невозможно.

### *IP-телефония*

Хорошим примером потоковой трансляции в реальном времени, при которой невозможна буферизация, является передача телефонного разговора по интернету

(иногда с видео, как в случае Skype и FaceTime). Когда-то голосовые звонки передавались коммутируемой телефонной сетью общего пользования, а сетевой трафик был преимущественно голосовым (с небольшим количеством данных). Затем появились интернет и Всемирная паутина. С годами объем данных возрастал, и к 1999 году информационный трафик сравнялся с голосовым (сегодня речь оцифрована, поэтому и то и другое можно измерить в битах). К 2002 году информационный трафик на порядок превысил голосовой, и его экспоненциальный рост продолжается. Между тем объем голосового трафика сохраняется практически неизменным.

В результате телефонные сети стали вытесняться интернетом. На сегодняшний день голосовой трафик передается с помощью интернет-технологий и занимает лишь малую часть пропускной способности сети. Эта прорывная технология известна как **IP-телефония (Voice over IP — передача голоса по протоколу IP)**, или **интернет-телефония**. Это название относится и к видеоконференциям, когда при разговоре используется видео или в нем участвует больше двух человек.

Самое главное отличие интернет-телефонии от потоковой онлайн-передачи фильмов в том, что в данном случае необходим низкий уровень времени ожидания. В телефонной сети оно составляет не более 150 мс в одном направлении; при превышении этого порога задержка становится заметной и начинает раздражать абонентов. (У международных звонков время ожидания иногда доходит до 400 мс, что дает не слишком приятный пользовательский опыт.)

Обеспечить столь низкий уровень задержки трудно. Конечно, буферизация 5–10 с мультимедиа (как это происходит при прямой спортивной радиотрансляции) не сработает. Вместо этого системы видео и голосовой IP-телефонии должны предусматривать разнообразные методы минимизации времени ожидания. Это ведет к выбору UDP вместо TCP, потому что повторные передачи TCP добавляют к задержке как минимум один полный обход.

Однако некоторые виды времени ожидания невозможно снизить даже с UDP. Например, расстояние между Сиэтлом и Амстердамом составляет около 8000 км. Задержка распространения со скоростью света в оптическом волокне для этого расстояния равна 40 мс. Пожелаем удачи тому, кто попробует ее сократить! На практике задержка распространения по сети будет больше, поскольку она покроет еще большее расстояние (биты идут не по кратчайшему пути). Кроме того, имеются задержки передачи, так как каждый IP-маршрутизатор сохраняет и пересылает пакет. Эти фиксированные задержки съедают общее допустимое время задержки.

Другой источник времени ожидания связан с размером пакета. Как правило, большие пакеты являются наилучшим способом использования пропускной способности сети, поскольку они эффективнее. Но для звукового сигнала с частотой дискретизации 64 Кбит/с пакет размером 1 Кбайт будет заполняться 125 мс (и даже дольше, если сэмплы сжаты). Такая задержка превысит общее время ожидания. Кроме того, если пакет в 1 Кбайт будет отправлен по широкополосному каналу, скорость которого равна 1 Мбит/с, передача займет 8 мс. Затем добавятся еще 8 мс, чтобы пакет прошел через широкополосное соединение на другом конце. Очевидно, что большие пакеты не подходят.

Вместо этого в системах IP-телефонии используются небольшие пакеты, чтобы сократить время ожидания за счет снижения эффективности использования пропускной способности. Звуковые сэмплы доставляются небольшими блоками, обычно по 20 мс. При скорости 64 Кбит/с это 160 байт данных (и еще меньше при сжатии). Однако задержка при использовании такого пакета составит всего 20 мс. Задержка передачи также будет меньше, потому что пакет короче. В нашем примере она сократится примерно до 1 мс. Минимальная задержка в одном направлении для небольшого пакета Сиэтл — Амстердам снижается с недопустимой — 181 мс (40 + 125 + 16) до приемлемой — 62 мс (40 + 20 + 2).

Мы даже не затронули издержки программного обеспечения, а оно тоже расходует часть допустимого времени ожидания. Это особенно характерно для видео, так как для его передачи с учетом имеющейся пропускной способности обычно требуется сжатие. В отличие от потоковой передачи сохраненного файла, в этом случае нет времени на работу кодировщика, требующего сложных вычислений и обеспечивающего высокую степень сжатия. И кодировщик, и декодировщик должны действовать быстро.

Буферизация по-прежнему требуется для своевременного проигрывания медиасэмплов (чтобы избежать неразборчивого звука или прерывистого видео), но количество данных в буфере должно быть очень небольшим, так как оставшееся допустимое время задержки измеряется в миллисекундах. Если пакет не приходит слишком долго, проигрыватель пропускает отсутствующие сэмплы. При этом он может заменить их на фоновый шум или повторить фрагмент, чтобы маскировать потерю для пользователя. Существует компромисс между размером буфера, необходимого для управления джиттером, и объемом потерянных данных. Буфер меньшего размера сокращает время ожидания, но увеличивает потери из-за джиттера. Таким образом, чем меньше буфер, тем более заметны потери для потребителя.

Наблюдательные читатели, возможно, заметили, что до сих пор в этом разделе мы не сказали ничего о протоколах *сетевого уровня*. Сеть может сократить время ожидания или хотя бы джиттер, используя механизмы QoS. Причина, по которой эта проблема не поднималась прежде, в том, что потоковая передача может выполняться с существенным временем ожидания даже в случае живой трансляции. Если время ожидания не является поводом для беспокойства, то буфера оконечного хоста достаточно, чтобы решить проблему джиттера. Однако для конференций в реальном времени часто важно снизить как задержку, так и джиттер, чтобы оставаться в пределах допустимой задержки. Это не имеет значения, только если пропускная способность так велика, что каждый получает хорошее обслуживание.

В главе 5 мы описали два механизма QoS, которые помогают достичь этой цели. Первым механизмом являются дифференцированные службы: пакеты распределяются по классам, получают соответствующую метку и обрабатываются в сети по-разному. Для пакетов IP-телефонии подходит метка низкой задержки. На практике системы устанавливают точки кода дифференцированных служб в общеизвестные значения следующим образом: класс обслуживания — `Expedited Forwarding` (Беспрепятственная пересылка); тип обслуживания — `Low Delay` (Низкая задержка). Это особенно полезно для каналов

широкополосного доступа, так как они могут перегружаться из-за конкуренции веб-трафика (или какого-либо другого) за линию. При устойчивом сетевом пути задержка и джиттер увеличиваются из-за перегрузки. Для передачи пакета в 1 Кбайт по каналу со скоростью 1 Мбит/с нужно 8 мс, и пакет IP-телефонии примет на себя эти задержки, если он находится в очереди после веб-трафика. Но при наличии метки низкой задержки пакеты IP-телефонии перейдут в начало очереди, обойдя пакеты веб-трафика и снизив задержку.

Второй механизм снижения задержки состоит в обеспечении достаточной пропускной способности. Если доступная пропускная способность или скорость передачи варьируются (как со сжатым видео) и иногда полосы пропускания не хватает, возникают очереди и задержка увеличивается. Это происходит даже при использовании дифференцированных служб. Чтобы гарантировать достаточную пропускную способность, часть сети можно зарезервировать. Такая возможность обеспечивается интегрированными службами.

К сожалению, этот подход не слишком распространен. Вместо этого либо сети проектируются под ожидаемый уровень трафика, либо клиентам предоставляются соответствующие SLA. Приложения должны действовать ниже этого уровня, чтобы избегать перегрузок и ненужных задержек. Для повседневных видеоконференций с домашнего компьютера пользователь сам выбирает качество видео с учетом пропускной способности сети (или же программное обеспечение проверяет сеть и выбирает нужное качество автоматически).

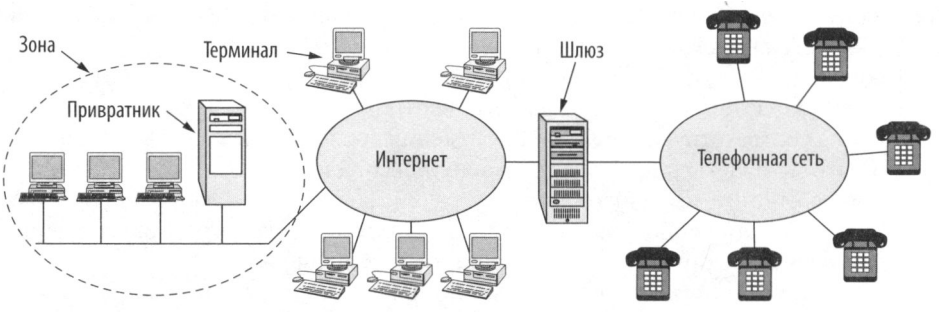
Любой из упомянутых выше факторов может сделать время ожидания неприемлемым, поэтому конференц-связь в реальном времени требует внимания к каждому из них. Краткий обзор IP-телефонии вместе с анализом этих факторов можно найти в работе Сан и др. (Sun et al., 2015).

Теперь, когда мы обсудили проблему времени ожидания для потокового мультимедиа, мы перейдем к другому важному вопросу систем проведения конференций: как устанавливать и прекращать вызовы. Мы рассмотрим два протокола, которые широко используются для этой цели, H.323 и SIP. Еще двумя важными системами являются Skype и FaceTime, но поскольку они проприетарные, информация об их внутреннем устройстве закрыта.

### *H.323*

Еще до того как голосовые и видеозвонки стали совершаться по интернету, всем было понятно, что если каждый производитель изобретет собственный стек протоколов, система никогда не будет работать. Чтобы этого избежать, заинтересованные стороны приступили к разработке единых стандартов под руководством МСЭ. В 1996 году МСЭ выпустил рекомендации **H.323** под заголовком «Видеотелефонные системы и оборудование для локальных вычислительных сетей, не предоставляющих гарантированный уровень QoS». Такое название могло родиться только в телефонной индустрии. С учетом критики, при пересмотре этих рекомендаций в 1998 году им было присвоено новое название: «Системы мультимедийной коммуникации на основе пакетов». Стандарт H.323 лежал в основе первых распространенных в интернете систем конференций и до сих пор широко используется.

H.323 представляет собой скорее общий обзор архитектуры систем интернет-телефонии, чем конкретный протокол. В данном документе можно найти множество ссылок на различные специализированные протоколы кодирования речи, установления вызова, сигнализации, передачи данных и т. п., однако их описание не приводится. Общая модель изображена на илл. 7.36. В центре находится **шлюз (gateway)**, соединяющий интернет с телефонной сетью. Он поддерживает протоколы стандарта H.323 на стороне интернета и протоколы телефонной сети общего пользования на «телефонной» стороне. Коммуникационные устройства называются **терминалами**. В LAN может быть **привратник (gatekeeper)**, управляющий конечными узлами, находящимися в ее «юрисдикции», которая называется **зоной**.




**Илл. 7.36.** Модель архитектуры H.323 для интернет-телефонии

Работу телефонной сети обеспечивает множество протоколов. Во-первых, необходим протокол кодирования и декодирования аудио и видео. Стандартное телефонное представление одного голосового канала кодируется как цифровое аудио с потоком 64 Кбит/с (8000 сэмплов, 8 бит/с), что определено в рекомендации МСЭ **G.711**. Все системы H.323 обязаны поддерживать G.711. Поддержка других протоколов кодирования речи разрешена (но необязательна). Они используют иные алгоритмы сжатия и дают другое соотношение между качеством и пропускной способностью. Для сжатия видео выбран формат MPEG, который мы обсуждали ранее (он поддерживается в том числе в H.264).

Поскольку разрешено несколько алгоритмов сжатия, необходим отдельный протокол, который позволил бы терминалам договориться об использовании одного из них. Такой протокол называется **H.245**. Также он позволяет согласовать другие параметры соединения, например битрейт.

RTCP требуется для управления каналами RTP. Кроме того, нужен протокол для установления и разрыва соединений, обеспечения тонального вызова, генерирования звуков звонков и других стандартных функций телефонной системы. Для этого используется стандарт МСЭ **Q.931**. Терминалам требуется протокол для ведения переговоров с привратником (если он есть). Для этого разработан протокол **H.225**. Он управляет каналом между ПК и привратником, который называется каналом **RAS (Registration/Admission/Status — Регистрация/Доступ/Статус)**. Помимо прочего, RAS позволяет терминалам входить в зону

и покидать ее, запрашивать и освобождать полосу пропускания, обновлять данные о состоянии и т. п. Наконец, нужен протокол для непосредственной передачи данных. На этом участке работает RTP на основе UDP, и управляется он, как обычно, RTCP. Иерархия всех этих протоколов показана на илл. 7.37.

**Илл. 7.37.** Стек протоколов H.323

Чтобы понять, как эти протоколы взаимодействуют между собой, рассмотрим следующий пример. Допустим, ПК (терминал LAN с привратником) совершает вызов на удаленный телефон. Вначале компьютеру нужно найти привратника, поэтому он рассылает специальный широковещательный UDP-пакет через порт 1718. В ответ привратник сообщает свой IP-адрес. Теперь ПК должен у него зарегистрироваться путем отправки RAS-сообщения в пакете UDP. После регистрации компьютер отсылает привратнику RAS-сообщение допуска (запрос на резервирование полосы). Только после выделения этого ресурса можно устанавливать соединение. Предварительное резервирование пропускной способности позволяет привратнику ограничить число соединений на исходящей линии, что обеспечивает необходимое качество обслуживания.

Строго говоря, телефонные системы выполняют ту же работу. Когда вы поднимаете трубку, на местный абонентский пункт отсылается сигнал. Если на пункте достаточно мощности для обработки еще одного звонка, он генерирует непрерывный гудок. В ином случае вы ничего не услышите. На сегодняшний день размер системы настолько велик, что вы практически всегда слышите непрерывный гудок, но раньше, когда телефония только зарождалась, на это обычно требовалось несколько секунд. Так что если ваши внуки когда-нибудь спросят, зачем нужны непрерывные гудки до начала набора, теперь вы знаете, что им ответить. Хотя к тому времени стационарных телефонов, скорее всего, не останется.

Теперь ПК устанавливает TCP-соединение с привратником, чтобы осуществить телефонный звонок. Для этого используются существующие протоколы телефонной сети, ориентированные на установление соединения (поэтому и требуется TCP). Для сравнения, в телефонной системе нет RAS, которые позволяли бы телефонным аппаратам заявлять о своем присутствии. Разработчики

H.323 могли выбрать для передачи RAS-сообщений как UDP, так и TCP. Они остановились на UDP — протоколе с наименьшими издержками.

Когда терминалу уже выделена пропускная способность, он может отослать по TCP-соединению сообщение SETUP (стандарт Q.931). В нем указывается номер вызываемого абонента (или IP-адрес и порт, если вызывается удаленный компьютер). Привратник отвечает Q.931-сообщением CALL PROCEDING, подтверждая корректный прием запроса. Затем он пересылает сообщение SETUP на шлюз.

Шлюз (наполовину компьютер, наполовину — телефонный коммутатор) осуществляет стандартный звонок на обычный телефон. Оконечная телефонная станция вызываемого абонента выполняет привычную работу (у абонента раздается звонок), а также отсылает обратно Q.931-сообщение ALERT, извещая ПК о том, что началась серия звонков. Когда абонент поднимает трубку, оконечная телефонная станция отправляет сообщение CONNECT, сообщая компьютеру, что соединение установлено.

После установления соединения привратник перестает принимать участие в этом процессе, хотя шлюз, конечно, продолжает работать, обеспечивая двустороннюю связь. Пакеты идут в обход привратника и направляются напрямую по IP-адресу шлюза. Такую ситуацию можно сравнить с обычным каналом между двумя сторонами. Это просто соединение физического уровня, по которому передаются биты, не больше. Ни одна из сторон ничего не знает о другой.

Для переговоров о параметрах звонка применяется протокол H.245. Он использует всегда открытый управляющий канал H.245. Прежде всего, стороны сообщают о своих возможностях, например о поддержке видео (H.323 может это делать), конференц-связи, используемых кодеках и т. п. После этого создаются два однонаправленных канала, с которыми ассоциируются определенные кодеки и которым присваиваются заданные параметры. Поскольку на каждой из сторон может быть разное оборудование, возможна ситуация, когда каждый однонаправленный канал использует свой кодек. После достижения договоренности по всем вопросам можно начинать передачу данных (по протоколу RTP). Управление производится RTCP, контролирующим перегрузку. Если передаются видеоданные, RTCP занимается синхронизацией аудио- и видеоряда. На илл. 7.38 показаны различные виды логических каналов. Когда на одной из сторон вешают трубку, по каналу Q.931 передается сигнал окончания связи, чтобы высвободить ресурсы, которые больше не нужны после завершения звонка.

После разрыва соединения вызывающий ПК должен снова связаться с привратником и отправить ему RAS-сообщение с запросом освобождения зарезервированной пропускной способности. Впрочем, вместо этого он может осуществить новый звонок.

Мы до сих пор не касались качества обслуживания применительно к H.323, а ведь на самом деле это довольно важный фактор для успешной передачи конференций в реальном времени. Дело в том, что QoS не входит в область рассмотрения H.323. Если сеть способна обеспечить между ПК и шлюзом стабильное соединение без джиттера, значит, нам повезло и QoS во время звонка будет хорошим; в противном случае — плохим. Однако любой телефонный звонок будет лишен перебоев благодаря дизайну телефонной сети.

**Илл. 7.38.** Логические каналы между звонящим и вызываемым абонентами во время разговора

## *SIP — протокол установления сеанса*

Стандарт H.323 был разработан МСЭ. Многим представителям интернет-сообщества он показался типичным телекоммуникационным продуктом: громоздким, сложным и недостаточно гибким. Было решено организовать специальный комитет IETF для создания более простой и гибкой системы передачи речи поверх IP. Основным результатом деятельности комитета стал **протокол установления сеанса (Session Initiation Protocol, SIP)**, последняя версия которого описана в RFC 3261. Данный протокол определяет способы установления телефонного интернет-соединения, организации видеоконференций и создания других мультимедийных соединений. В отличие от стандарта H.323, представляющего собой целый набор протоколов, SIP — это единый модуль, способный взаимодействовать с разнообразными интернет-приложениями. Например, номера телефонов определяются в виде URL-адресов, то есть на веб-страницах можно размещать гиперссылки, щелкнув по которым пользователь может начать телефонный звонок (аналогично схема *mailto* позволяет написать e-mail и отправить его по указанному в ссылке адресу).

SIP позволяет устанавливать и двусторонние сеансы (обычные телефонные соединения), и многосторонние (когда каждый из участников может слушать собеседников и говорить), и широковещательные (когда один из участников говорит, а остальные могут только слушать). Во время сеанса связи может передаваться звук, видео или другие данные. Эта возможность используется, например, в сетевых многопользовательских играх в реальном времени. SIP занимается только установлением, управлением и разрывом соединений. Для передачи данных используются другие протоколы, например, RTP/RTCP. SIP — это протокол прикладного уровня, работающий поверх TCP или UDP.

SIP предоставляет разнообразные службы, включая поиск вызываемого абонента (который может быть далеко от своего домашнего компьютера), определение его возможностей, поддержку механизмов установления и разрыва телефонного соединения. В самом простом случае SIP устанавливает сеанс связи между компьютерами звонящего и вызываемого абонентов. Именно эту ситуацию мы сейчас и рассмотрим.

Телефонные номера в SIP представлены в виде URL с помощью *sip*-схемы. Например, sip:ilse@cs.university.edu свяжет вас с пользователем по имени Ilse, хост которого имеет DNS-имя cs.university.edu. SIP URL могут содержать также адреса формата IPv4, IPv6 или реальные номера телефонов.

Протокол SIP является текстовым и построен по модели HTTP. Одна из сторон отсылает ASCII-сообщение, в котором первая строка содержит имя метода, а ниже следуют дополнительные строки с заголовками для передачи параметров. Многие заголовки взяты из стандарта MIME, что позволяет SIP взаимодействовать с существующими интернет-приложениями. Шесть методов базовой спецификации перечислены на илл. 7.39.

| Метод | Описание |
|---|---|
| INVITE | Запрос установления сеанса связи |
| ACK | Подтверждение установления сеанса |
| BYE | Запрос окончания сеанса |
| OPTIONS | Опрос возможностей хоста |
| CANCEL | Отмена запроса |
| REGISTER | Информирование сервера переадресации о текущем местоположении пользователя |

**Илл. 7.39.** Методы SIP

Для установки сеанса связи звонящий должен либо создать TCP-соединение с вызываемым абонентом и передать по нему сообщение `INVITE`, либо отправить это же сообщение в UDP-пакете. В обоих случаях заголовки во второй и всех последующих строках описывают структуру тела сообщения, содержащего информацию о возможностях звонящего, типах мультимедиа и форматах. Если вызываемый абонент принимает звонок, он отсылает в качестве ответа трехзначный код результата, подобный HTTP (группы этих кодов перечислены на илл. 7.26, код 200 означает прием вызова). Следом за строкой с кодом результата вызываемый абонент может также сообщить данные о своих возможностях, типах мультимедиа и форматах.

Соединение устанавливается по протоколу «тройного рукопожатия»; звонящий высылает `ACK` для окончания работы протокола и подтверждения приема кода 200.

Любая из сторон может отправить запрос окончания сеанса связи, для этого используется метод `BYE`. Сеанс считается завершенным после получения подтверждения от противоположной стороны.

Метод `OPTIONS` применяется для опроса возможностей устройства. Обычно это делается перед запуском сеанса связи, чтобы определить, поддерживается ли тип сеанса, на который рассчитывает вызывающая сторона (например, передача голоса по IP).

Метод `REGISTER` связан со способностью протокола SIP находить пользователя и соединяться с ним, даже если его нет дома. Сообщение с данным методом отправляется на SIP-сервер определения местонахождения, который отслеживает, кто и где находится в данный момент. Позднее с помощью этого сервера можно попробовать найти абонента. Используемая при этом операция переадресации показана на илл. 7.40. Здесь мы видим, что звонящий отправляет сообщение `INVITE` на прокси-сервер. Это делает возможную переадресацию незаметной. Прокси пытается разыскать абонента и отсылает `INVITE` по найденному адресу. Далее он действует в качестве ретранслятора для последующих сообщений в «тройном рукопожатии». Сообщения `LOOKUP` и `REPLY` не входят в SIP; на этой стадии может использоваться любой подходящий протокол в зависимости от типа сервера определения местонахождения.

**Илл. 7.40.** Использование прокси-сервера и переадресации в протоколе SIP

SIP обладает множеством других функций, которые мы не стали описывать подробно. Среди них — ожидание вызова, отображение звонка, шифрование и аутентификация звонящего. Кроме того, SIP позволяет звонить с компьютера на обычный телефон, если есть доступ к соответствующему шлюзу между интернетом и телефонной системой.

### *Сравнительный анализ H.323 и SIP*

H.323 и SIP поддерживают как двустороннюю, так и многостороннюю связь. Оконечным оборудованием могут служить как компьютеры, так и обычные телефоны. И там и там стороны предварительно договариваются о параметрах; возможно использование шифрования данных и протоколов RTP/RTCP. Сводная информация о сходствах и различиях представлена на илл. 7.41.

Несмотря на схожий набор свойств и характеристик, протоколы разительно отличаются друг от друга концепцией и философией. H.323 — это типичный тяжеловесный стандарт, характерный для телефонной индустрии. Он описывает целый стек протоколов и очень точно указывает, что разрешено, а что запрещено. Такой подход дает хорошо определенные протоколы на каждом уровне, тем самым упрощается задача взаимодействия сетей. Однако платой за это оказывается

большой, сложный и жесткий стандарт, тяжело адаптируемый к приложениям, которые появятся в будущем.

| Аспект | H.323 | SIP |
|---|---|---|
| Разработчик | МСЭ | IETF |
| Совместимость с телефонной системой | Полная | В большой мере |
| Совместимость с интернетом | Присутствует, с течением времени | Присутствует |
| Архитектура | Монолитная | Модульная |
| Завершенность | Полный стек протоколов | SIP обеспечивает лишь установление соединения |
| Переговоры относительно параметров | Есть | Есть |
| Сигналы при вызове | Q.931 поверх TCP | SIP поверх TCP или UDP |
| Формат сообщений | Двоичный | ASCII |
| Передача мультимедийных данных | RTP/RTCP | RTP/RTCP |
| Многосторонняя связь | Есть | Есть |
| Мультимедийные конференции | Есть | Нет |
| Адресация | URL или номер телефона | URL |
| Разрыв связи | Явный или разрыв TCP-соединения | Явный или по тайм-ауту |
| Обмен мгновенными сообщениями | Нет | Есть |
| Шифрование | Есть | Есть |
| Объем описания стандарта | 1400 страниц | 250 страниц |
| Реализация | Громоздкая и сложная | Умеренно сложная, с отдельными проблемами |
| Статус | Широко распространен, особенно видео | Хорошая альтернатива, особенно для речи |

**Илл. 7.41.** Сравнение H.323 и SIP

SIP, напротив, представляет собой типичный интернет-протокол; его работа основана на обмене короткими текстовыми строками. Это небольшой модуль, который хорошо взаимодействует с другими протоколами интернета, но несколько хуже согласуется с существующими сигнальными протоколами телефонной системы. Поскольку модель системы передачи данных по IP, предложенная IETF, использует модульный принцип, она достаточно гибкая и может

легко адаптироваться к новым приложениям. Недостатком этого протокола являются проблемы совместимости, вызванные тем, что люди по-разному его интерпретируют.

## 7.5. ДОСТАВКА КОНТЕНТА

Когда-то интернет был исключительно средством двухточечной коммуникации подобно телефонной сети. Изначально он использовался в научной среде для того, чтобы подключаться по сети к удаленным компьютерам и выполнять на них определенные задачи. Для общения люди долгое время использовали электронную почту, сейчас к этому добавилась еще и видео- и голосовая IP-телефония. Однако по мере роста интернет становился более ориентированным на контент, чем на коммуникацию. Теперь пользователи чаще всего используют его для поиска информации и в огромных объемах скачивают музыку, видео и другие материалы. Смещение акцента на контент столь явное, что большая часть пропускной способности интернета сейчас используется для передачи сохраненного видео.

Задача распространения контента существенно отличается от задачи обеспечения двухточечной связи, предъявляя совершенно другие требования к сети. Например, если Салли хочет поговорить с Джоном, она может позвонить ему на мобильный телефон с помощью IP-телефонии. Нужно установить связь с конкретным устройством — нет смысла звонить на компьютер Пола. Но если Джон хочет посмотреть последний матч своей любимой команды по крикету, он будет рад получить это видео с любого устройства, которое его предоставит. Это может быть компьютер Салли, или Пола, или, что наиболее вероятно, неизвестный сервер в интернете. Таким образом, местоположение контента не имеет значения, кроме тех случаев, когда это затрагивает производительность (и законность).

Еще одно отличие заключается в том, что некоторые веб-узлы, предоставляющие контент, стали чрезвычайно популярными. Яркий пример — YouTube. Он позволяет пользователям делиться своими видео на любую тему, которую только можно вообразить. Многие хотят это сделать, а все остальные хотят его смотреть. Сегодня на потоковое видео приходится более 70 % интернет-трафика, причем подавляющая часть данных доставляется небольшим числом поставщиков контента.

Ни один сервер не может обеспечить достаточную мощность и надежность, чтобы управлять таким потрясающим уровнем спроса. Вместо этого YouTube, Netflix и другие крупные контент-провайдеры создают свои собственные сети распределения контента. Эти сети используют центры обработки данных по всему миру, чтобы поставлять контент гигантскому количеству пользователей, обеспечивая хорошую производительность и доступность.

Методы распределения контента со временем совершенствовались. На ранних этапах развития Всемирной паутины ее популярность почти ее уничтожила. Растущее число запросов к контенту приводило к тому, что серверы и сети часто были перегружены. Люди начали расшифровывать WWW как World Wide Wait (Всемирное ожидание). Чтобы снизить бесконечные задержки, исследователи

разработали ряд архитектур, позволяющих использовать пропускную способность для распределения контента.

Одной из наиболее распространенных архитектур является **сеть доставки контента (Content Delivery Network, CDN)**, иногда ее называют **сетью распространения контента (Content Distribution Network)**. CDN, по сути, представляет собой огромный распределенный набор кэшей, которые доставляют контент клиентам напрямую. Изначально CDN были прерогативой исключительно крупных контент-провайдеров. Провайдер популярного контента мог заплатить CDN (к примеру, Akamai) за распространение этого контента (то есть за предварительное заполнение им кэшей сети). Сегодня собственные CDN развертывают не только крупные поставщики контента (как Netflix или Google), но и многие интернет-провайдеры, предлагающие собственный контент (например, Comcast).

Еще один способ распространения контента сводится к использованию **одноранговой** сети **(Peer-to-Peer P2P)**, в которой компьютеры доставляют контент друг другу, обычно без специально предусмотренных отдельных серверов или какого-либо центрального пункта управления. Эта идея вдохновляет людей, поскольку много небольших участников, объединившись, способны произвести громадный эффект.

## 7.5.1. Контент и интернет-трафик

Чтобы проектировать и строить хорошие сети, нужно понимать, какой трафик они должны нести. К примеру, при смещении акцента на доступ к контенту серверы переместились из офисов компаний в центры хранения и обработки данных. Эти центры предоставляют множество компьютеров с превосходным подключением к сети. Даже если вам нужен небольшой сервер, сегодня легче и дешевле арендовать виртуальный сервер в дата-центре, чем работать с реальным компьютером с широкополосным интернет-доступом дома или в офисе.

Интернет-трафик асимметричен. Многие параметры, с которыми мы имеем дело, сгруппированы вокруг средней величины. Например, рост большинства взрослых людей близок к среднему. Есть некоторое количество высоких и низких людей, и совсем мало очень высоких или очень низких. Аналогично, романы в основном содержат несколько сотен страниц, и очень немногие — 20 или 10 000 страниц. Для таких свойств можно найти диапазон, который не слишком велик, но охватывает большую часть данных.

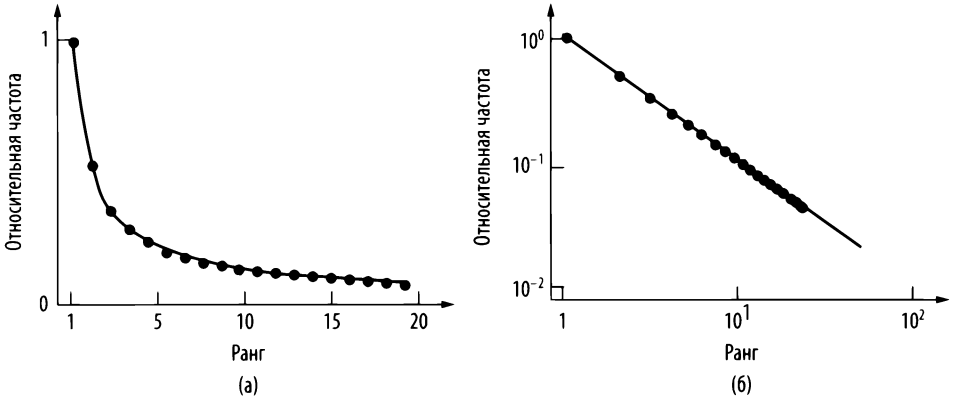
Интернет-трафик устроен иначе. Как известно, в интернете уже достаточно долго существует несколько сайтов с громадным трафиком (например, Google, YouTube, Facebook) и огромное количество сайтов с гораздо меньшими объемами трафика.

Опыт работы с пунктами видеопроката, библиотеками и другими подобными организациями показывает, что не все фильмы или книги одинаково популярны. Экспериментально доказано, что если в пункте проката есть $N$ фильмов, то доля заявок на конкретный фильм, стоящий на $k$-м месте в списке популярности, примерно равна $C/k$. Здесь $C$ — это число, дополняющее сумму долей до 1, а именно:

$$C = 1/(1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/N).$$

Таким образом, самый популярный фильм берут примерно в семь раз чаще, чем седьмой в списке популярности. Эта зависимость называется **законом Ципфа (Zipf's law)** (Zipf, 1949), в честь Джоржа Ципфа, профессора лингвистики в Гарвардском университете. Он заметил, что частота использования слова в большом тексте инверсионно пропорциональна его рангу. Например, сороковое в списке самых частотных слов используется в два раза чаще, чем восьмидесятое, и в три раза чаще, чем сто двадцатое.

Распределение Ципфа показано на илл. 7.42 (а). Здесь иллюстрируется утверждение о том, что существует небольшое количество популярных элементов и огромное — непопулярных. Чтобы выявлять распределения такого вида, удобно размещать данные в логарифмическом масштабе по обеим осям, как показано на илл. 7.42 (б). В результате должна получаться прямая.




**Илл. 7.42.** Распределение Ципфа. (а) Линейная шкала. (б) Логарифмическая шкала по обеим осям

Когда ученые впервые стали изучать популярность веб-страниц, оказалось, что она тоже приблизительно соответствует закону Ципфа (Бреслау и др., Breslau et al., 1999). Распределение Ципфа относится к семейству распределений, известных как **степенные законы (power laws)**. Эти законы проявляются во многих сферах человеческой деятельности, например в распределении городского населения и распределении богатства. Они также описывают ситуацию, когда имеются несколько крупных и множество более мелких игроков, и здесь снова получаем распределение вдоль прямой линии на графике с логарифмической шкалой по обеим осям. Вскоре было обнаружено, что топологию интернета можно условно описать с помощью степенных законов (Сиганос и др.; Siganos et al., 2003). Затем исследователи начали изображать все мыслимые свойства интернета на логарифмической шкале и, увидев прямую линию, восклицали: «Степенной закон!»

Впрочем, важна не прямая линия на логарифмической шкале сама по себе, а то, как эти распределения влияют на проектирование и использование сетей. Многие виды контента подчиняются закону Ципфа или другим степенным законам, поэтому принципиально важно, что популярность веб-узлов интернета

также описывается этими распределениями. Это означает, что понятие *среднего* сайта бесполезно. Сайты лучше делить на популярные и непопулярные; и те и другие важны. Популярные сайты, очевидно, являются значимым фактором, поскольку всего несколько таких сайтов генерируют огромную долю интернет-трафика. Возможно, это покажется удивительным, но непопулярные сайты тоже имеют значение. Их очень много, поэтому они вносят существенный вклад в общий трафик. Идея о том, что множество непопулярных решений вместе могут сыграть большую роль, изложена в книге «Длинный хвост» («The Long Tail») Криса Андерсона (Anderson, 2008a).

Чтобы эффективно работать в этом асимметричном мире, мы должны уметь строить оба вида веб-сайтов. Непопулярные сайты легки в управлении. С использованием DNS несколько различных сайтов могут фактически указывать на один и тот же компьютер в интернете, который ими управляет. С другой стороны, популярные сайты трудно поддерживать. С этим вряд ли справится один компьютер, каким бы мощным он ни был; кроме того, использование одного компьютера приведет к тому, что в случае его поломки (а это обязательно произойдет) сайт окажется недоступным для миллионов пользователей. Чтобы управлять этими сайтами, нужно построить системы распределения контента. Позже мы к этому перейдем.

## 7.5.2. Серверные фермы и веб-прокси

Сетевые проекты, которые мы рассматривали до сих пор, состояли из одного сервера, взаимодействующего с несколькими клиентскими компьютерами. Для создания больших веб-узлов с хорошими характеристиками мы можем увеличивать скорость обработки или на серверной, или на клиентской стороне. На серверной стороне более мощный веб-сервер может быть построен путем организации серверной фермы, в которой кластер компьютеров действует как единый сервер. На клиентской стороне лучшая производительность может быть достигнута с помощью более эффективных методов кэширования. В частности, кэширующий прокси-сервер обеспечивает большой общий кэш для группы клиентов.

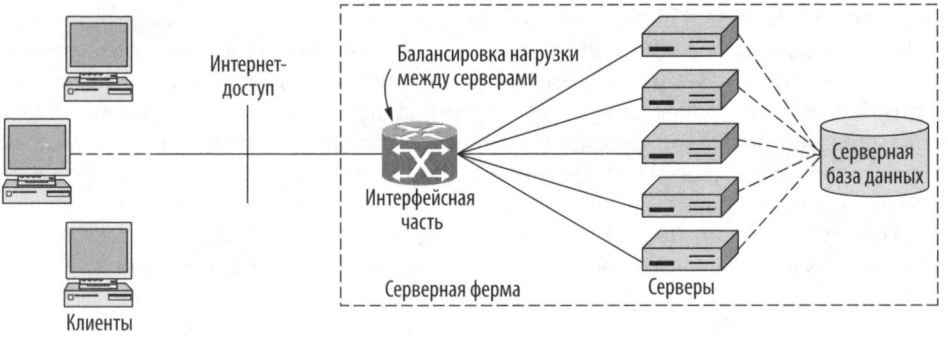
Мы опишем все эти методы по очереди. Однако заметим, что они не подходят для построения крупнейших популярных веб-сайтов. В этом случае нужны методы распределения контента, в которых задействованы компьютеры, расположенные в самых разных местах; эти методы мы обсудим в следующих разделах.

### *Серверные фермы*

Какой бы вычислительной мощностью и пропускной способностью ни обладал компьютер, он сможет обслужить лишь некоторое количество сетевых запросов, пока нагрузка не станет слишком большой. Решение в данной ситуации — использовать несколько компьютеров, чтобы сделать веб-сервер. В итоге мы получаем модель **серверной фермы (server farm)**, показанной на илл. 7.43.

Сложность этой, казалось бы, простой модели заключается в том, что набор компьютеров, образующих серверную ферму, должен выглядеть для клиентов

как единый логический веб-сайт. В противном случае мы просто будем иметь дело с несколькими веб-сайтами, работающими параллельно.




**Илл. 7.43.** Серверная ферма

Существует несколько возможных путей решения этой задачи. Все они предполагают, что каждый сервер может обработать запрос от любого клиента, а значит, он должен иметь копию веб-сайта. Для этого серверы соединяются с общей внутренней базой данных (это показано пунктирными линиями на илл. 7.43).

Пожалуй, наиболее распространенное решение сводится к распределению запросов по серверам фермы с помощью DNS. Когда выполняется DNS-запрос для получения URL-адреса домена, возвращаемый DNS-сервером ответ перенаправляет клиента к CDN-сервису (обычно используя NS-запись со ссылкой на авторитетный для этого домена сервер имен). CDN-сервис стремится вернуть клиенту IP-адрес ближайшей к клиенту реплики сервера. Если в качестве ответа возвращается несколько IP-адресов, обычно клиент пытается подключиться к первому из них. Таким образом, как и задумывалось, для посещения одного и того же сайта разные клиенты связываются с разными серверами, которые должны располагаться поблизости от них. Этот процесс иногда называют **сопоставлением клиентов (client mapping)**. Обратите внимание, что он опирается на сведения авторитетного сервера имен о топологическом или географическом положении клиента. Мы подробно обсудим сопоставление клиентов с использованием DNS после того, как рассмотрим сети доставки контента.

Еще одним популярным методом балансировки нагрузки сегодня является **произвольная IP-адресация (IP anycast)**. В двух словах, это процесс, при котором один IP-адрес может объявляться в различных точках подключения к сети (например, в сетях в Европе и в США). Если все проходит нормально, то клиент, желающий связаться с определенным IP-адресом, направляется к ближайшей конечной точке сети. Конечно, как мы знаем, междоменная маршрутизация в интернете не всегда выбирает кратчайший (или даже оптимальный) путь. Поэтому метод произвольной IP-адресации не такой детализированный и управляемый по сравнению с сопоставлением клиентов с помощью DNS. Несмотря на это, некоторые крупные CDN, такие как CloudFlare, все же используют комбинацию двух этих методов.

Другие, менее популярные подходы основаны на работе **интерфейсной части (front end)**, распределяющей входящие запросы по пулу серверов в серверной ферме, даже если для связи с этой фермой клиент использует один IP-адрес получателя. Интерфейсная часть обычно представляет собой сетевой коммутатор на канальном уровне или IP-маршрутизатор, то есть устройство, которое управляет фреймами или пакетами. Все решения основаны на том, что эти устройства (или серверы) просматривают заголовки сетевого, транспортного или прикладного уровня и используют их нестандартным образом. Веб-запрос и ответ передаются как TCP-соединение. Для корректной работы интерфейсная часть должна отправить все пакеты одного запроса к одному и тому же серверу.

Простая схема интерфейсной части — передавать все входящие запросы всем серверам. Каждый сервер отвечает только на часть запросов по предварительному соглашению. Допустим, 16 серверов проверяют IP-адрес источника и отвечают на запрос, только если последние 4 бита IP-адреса совпадают с их настроенными селекторами. Остальные пакеты отбрасываются. Хотя это и расточительно с точки зрения входящей пропускной способности, но поскольку ответы зачастую намного длиннее запросов, этот подход эффективнее, чем кажется.

В более общем варианте структуры интерфейсная часть может проверять IP-, TCP- и HTTP-заголовки пакетов и произвольным образом распределять их по серверам. Этот подход называют политикой **балансировки нагрузки (load balancing)**, так как его цель сводится к равномерному распределению рабочей нагрузки между серверами. Политика балансировки бывает простая и сложная. В первом случае серверы используются один за другим по очереди или по кругу циклически. При этом интерфейс должен помнить соответствие каждого запроса, чтобы последующие пакеты одного запроса были отправлены к конкретному серверу. Кроме того, чтобы сделать сайт надежнее, чем один сервер, интерфейс должен фиксировать отказы серверов и прекращать отсылать им запросы.

### *Веб-прокси*

Кэширование улучшает работу, уменьшая продолжительность времени ответа и загруженность сети. Если браузер сам определяет, что кэшированная страница актуальна, он может немедленно извлечь ее из кэша вообще без сетевого трафика. Но даже если он должен запросить сервер о подтверждении актуальности страницы, продолжительность времени ответа сокращается, как и сетевая нагрузка (особенно для больших страниц), поскольку отсылается только маленькое сообщение.

И все-таки лучшее, что может сделать браузер, — кэшировать все веб-страницы, которые посетил пользователь. Из нашего обсуждения популярности сайтов вы, вероятно, помните, что кроме нескольких популярных страниц, постоянно посещаемых многими людьми, существует огромное число непопулярных страниц. На практике это ограничивает эффективность кэширования браузером, потому что пользователи заходят на многие страницы однократно и их каждый раз нужно получать с сервера.

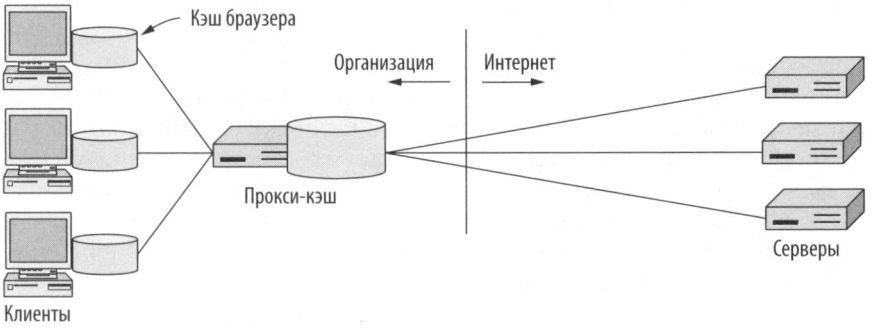
Единственный способ использовать кэш эффективнее — сделать его общим для нескольких пользователей. При этом страница, уже доставленная для одного

из них, может быть возвращена другому, когда тот сделает такой же запрос. Без кэширования в браузере обоим пользователям пришлось бы получать страницы с сервера. Конечно, общий доступ невозможен в случае зашифрованного трафика, страниц, требующих аутентификации, и возвращаемых программами некэшируемых страниц (например, с информацией о текущих курсах акций). Динамические страницы, особенно создаваемые программами, также часто не позволяют эффективно использовать кэширование. Однако есть множество веб-страниц, которые доступны многим и выглядят для всех одинаково (например, изображения).

Чтобы обеспечить пользователям общий доступ к кэшу, используется **веб-прокси (Web proxy)**. Прокси — это агент, который действует от чужого имени, например другого пользователя. Есть много видов прокси. Например, ARP-прокси отвечает на ARP-запросы от имени пользователя, который находится где-то в другом месте (и не может ответить сам). Веб-прокси выполняет веб-запросы от имени его пользователей. Обычно он кэширует веб-ответы, и так как он находится в совместном доступе, его кэш значительно больше, чем у браузера.

При использовании прокси типичный вариант для организации (компании или интернет-провайдера) — задействовать один веб-прокси для всех пользователей. В обоих случаях выгодно увеличить скорость веб-запросов для пользователей, а также сократить потребности в пропускной способности. Для домашних пользователей обычно устанавливается постоянная цена, независимо от потребления, но компании и провайдеры, как правило, платят в зависимости от используемой пропускной способности.

Конфигурация с использованием прокси показана на илл. 7.44. Каждый браузер настроен так, чтобы отправлять запросы к прокси, а не к реальному серверу страницы. Если у прокси есть эта страница, он сразу ее возвращает. В противном случае прокси берет страницу с сервера, добавляет ее к кэшу для будущего использования и возвращает клиенту то, что он запросил.




**Илл. 7.44.** Прокси-кэш между веб-браузерами и веб-серверами

Кроме отправки запросов к прокси, клиенты выполняют и собственное кэширование, используя кэш браузера. Обращение к прокси происходит только после того, как браузер попробовал удовлетворить запрос из своего кэша. Таким образом, прокси обеспечивает второй уровень кэширования.

Чтобы обеспечить дополнительные уровни кэширования, могут быть добавлены следующие прокси. Каждый прокси (или браузер) делает запрос к своему **вышестоящему прокси (upstream proxy)**. Каждый вышестоящий прокси производит кэширование для **нижестоящих прокси (downstream proxy)** или браузеров. В результате может получиться, что браузеры компании используют ее прокси, который, в свою очередь, использует прокси интернет-провайдера, а тот контактирует с веб-серверами напрямую. Но на практике, чтобы извлечь большую часть потенциальной выгоды, часто достаточно одного уровня прокси-кэширования (см. илл. 7.44). Проблемой снова оказывается «длинный хвост» популярности. Исследования веб-трафика показали, что кэш общего доступа особенно полезен, пока число пользователей не превосходит количество сотрудников небольшой компании (скажем, 100 человек). Когда это число увеличивается, преимущества использования общего кэша становятся незначительными, поскольку для кэширования непопулярных запросов не хватает места.

Веб-прокси часто устанавливают ради дополнительных преимуществ, например из-за возможности фильтровать контент. Системный администратор может создать на прокси черный список сайтов или фильтровать запросы. К примеру, многие администраторы не одобряют просмотр YouTube (или, того хуже, порнографии) на рабочем месте и ставят соответствующие фильтры. Еще одно преимущество — конфиденциальность (или анонимность): прокси скрывает личность пользователя от сервера.

## 7.5.3. Сети доставки контента

Серверные фермы и веб-прокси помогают создавать большие сайты и улучшать работу сети, но этих инструментов недостаточно для действительно популярных веб-сайтов, которые должны предоставлять контент в глобальном масштабе. К ним требуется другой подход.

**Сети доставки контента (Content Delivery Networks, CDN)** переворачивают идею традиционного веб-кэширования с ног на голову. Клиентам больше не нужно искать копии страниц в ближайшем кэше. Вместо этого провайдер размещает копии страницы в наборе узлов в разных местах и направляет клиента, чтобы тот использовал в качестве сервера узел, расположенный поблизости.

Первой применять DNS для распределения контента начала компания Akamai в 1998 году. Тогда, на раннем этапе своего развития, интернет едва справлялся с нагрузкой. Akamai была первой крупной CDN и очень быстро стала лидером отрасли. Она построила удачную бизнес-модель и систему стимулирования (возможно, еще более удачную, чем сама идея использования DNS для соединения клиентов с ближайшими узлами). Компании платят Akamai за доставку своего контента клиентам, чтобы получить удобные для пользователей веб-сайты с низким временем отклика. Узлы CDN должны располагаться там, где есть хорошая скорость подключения; изначально это подразумевало сети интернет-провайдера. На практике узел CDN представляет собой стандартную 19-дюймовую аппаратную стойку с компьютером и множеством дисков, из которой выходит оптоволоконный кабель для подключения к внутренней LAN интернет-провайдера.

ISP выгодно иметь узел CDN в своих сетях, поскольку это снижает величину необходимой им исходящей пропускной способности (за которую они платят). Кроме того, при наличии узла CDN контент доставляется пользователям с меньшей задержкой. Таким образом, выигрывают и интернет-провайдер, и клиенты, а CDN делает деньги. Начиная с 1998 года в этой сфере бизнеса появилось множество других компаний, включая Cloudflare, Limelight, Dyn и т. д., так что сейчас это конкурентная отрасль с большим количеством провайдеров. Как уже упоминалось, многие крупные провайдеры контента, такие как YouTube, Facebook и Netflix, используют собственные сети доставки контента.

Самые большие CDN включают в себя сотни тысяч серверов, расположенных в разных странах по всему миру. Столь большая емкость CDN часто помогает сайтам в защите от DDoS-атак. Если злоумышленнику удается обеспечить отправку сотен или тысяч запросов в секунду на сайт, использующий CDN, то CDN зачастую способна ответить на все эти запросы. Таким образом, атакованному сайту удается выдержать лавинообразное увеличение числа запросов. То есть CDN позволяет быстро повысить объем передаваемых сайтом данных. Некоторые CDN-провайдеры даже рекламируют свою способность справляться с крупными DDoS-атаками как отдельное преимущество для привлечения клиентов.

Показанные в нашем примере узлы CDN обычно являются кластерами компьютеров. Переадресация DNS происходит на двух уровнях: один — для сопоставления клиента с близко расположенной частью сети, второй — для распределения нагрузки между узлами этой части. При этом важно обеспечить и надежность, и высокую производительность. Чтобы иметь возможность перемещения клиента с одного компьютера в кластере на другой, ответы на втором уровне DNS предоставляются с коротким временем жизни, благодаря чему клиент повторно выполняет разрешение адреса спустя некоторое время. Наконец, до сих пор мы рассматривали распределение статических объектов (изображения и видео), но CDN также могут поддерживать динамически создаваемые страницы, потоковые медиаданные и многое другое. Кроме того, они широко используются для распространения видео.

### *Заполнение узлов кэша CDN*

На илл. 7.45 показан пример пути следования данных в случае их распространения CDN-сетью. Этот путь представляет собой дерево. Исходный сервер в CDN рассылает копию контента по другим узлам в сети, расположенным в Сиднее, Бостоне и Амстердаме (пунктирные линии). Затем клиенты забирают страницы с ближайших узлов CDN (сплошные линии). Таким образом, оба клиента в Сиднее берут копию страницы, расположенную в этом же городе; они не достают страницу с исходного сервера, который, возможно, находится в Европе.

Использование древовидной структуры имеет три преимущества. Во-первых, распространение контента может масштабироваться до любого необходимого числа клиентов при увеличении количества узлов CDN (и уровней дерева, когда распределение среди узлов CDN станет узким местом). Древовидная структура

эффективна независимо от количества клиентов. Исходный сервер не перегружен, ведь он взаимодействует с клиентами через дерево узлов CDN; ему не придется самому отвечать на каждый запрос страницы. Во-вторых, каждый клиент получает хорошую производительность за счет доставки страницы с ближайшего, а не отдаленного сервера. Это связано с тем, что соединение устанавливается быстрее, из-за чего медленный старт TCP ускоряется, при этом более короткий сетевой путь с меньшей вероятностью проходит через области перегрузки в интернете. Наконец, общая загруженность сети также сведена к минимуму. Если узлы CDN удачно расположены, то любая страница передается в каждом участке сети только один раз. Это важно, поскольку в конечном счете кто-то платит за полосу пропускания.

**Илл. 7.45.** Дерево распределения CDN

С ростом применения шифрования в интернете, в частности, использования протокола HTTPS для распространения веб-контента, доставка контента из CDN приобрела более сложный характер. Допустим, вам нужно загрузить главную страницу сайта https://nytimes.com/. DNS-поиск для этого домена выдает ссылку на сервер имен компании Dyn, например ns1.p24.dynect.net, который, в свою очередь, перенаправляет вас на IP-адрес, размещенный в CDN этой компании. Но теперь этот сервер должен доставить вам контент, аутентифицированный *New York Times*. Для этого он, вероятно, должен располагать закрытыми ключами шифрования для сайта *New York Times* или сертификатом для сайта nytimes.com (или и тем и другим). Таким образом, CDN придется доверить конфиденциальную информацию провайдера контента, а сервер потребуется настроить так, чтобы он фактически выступал в роли агента сайта nytimes.com. Альтернативный подход состоит в том, чтобы направлять все запросы клиента обратно на исходный сервер, который сможет доставлять сертификаты и контент по протоколу HTTPS, но это сведет на нет практически все преимущества CDN в плане производительности. Решение обычно представляет собой компромисс: CDN генерирует сертификат от имени контент-провайдера и доставляет

контент из CDN с помощью этого сертификата, выступая в роли организации. Это позволяет достичь основных целей: шифрования контента, идущего от CDN к пользователю, и его аутентификации для пользователя. Более сложные решения, которые требуют развертывания сертификатов на исходном сервере, также позволяют шифровать трафик между клиентом и узлами кэша. Компания Cloudflare предлагает хороший обзор таких решений на своем сайте по адресу https://cloudflare.com/ssl/.

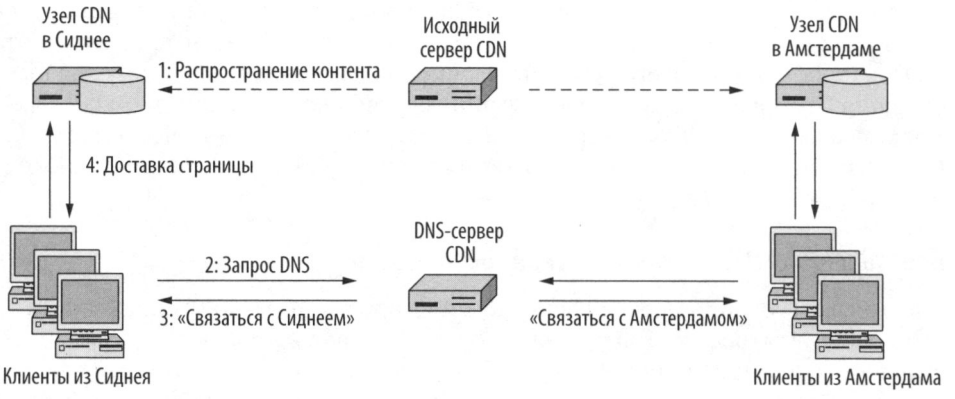
### *Переадресация DNS и сопоставление клиентов*

Идея использования дерева распределения проста. Гораздо сложнее понять, как сопоставлять клиентов с нужными узлами кэша в этом дереве. Как представляется, эту проблему можно решить с помощью прокси-серверов. Если бы каждый клиент на илл. 7.45 был настроен на использование одного из узлов CDN — в Сиднее, Бостоне или Амстердаме — в качестве кэширующего веб-прокси, распределение было бы древовидным.

Как упоминалось выше, самым распространенным способом сопоставления или направления клиентов к ближайшим узлам кэша CDN является **переадресация DNS (DNS redirection)**. Рассмотрим этот подход более подробно. Предположим, клиент хочет попасть на страницу с URL-адресом http://www.cdn.com/page.html. Чтобы ее загрузить, браузер использует DNS для получения IP-адреса, соответствующего имени www.cdn.com. Этот DNS-поиск осуществляется как обычно. Используя протокол DNS, браузер узнает IP-адрес сервера имен для домена cdn.com, после чего связывается с сервером имен и просит его разрешить имя www.cdn.com. Но поскольку сервер имен запускается CDN-сетью, вместо выдачи одного IP-адреса на все запросы он выясняет IP-адрес клиента и возвращает ответ в зависимости от его местоположения. Ответом будет IP-адрес ближайшего к клиенту узла CDN. Так, если клиент в Сиднее обращается к серверу имен CDN с запросом IP-адреса www.cdn.com, он получит IP-адрес сиднейского узла CDN, а на тот же запрос от клиента в Амстердаме будет выдан IP-адрес амстердамского узла.

Это вполне допустимая стратегия согласно семантике DNS. Мы уже видели, что серверы имен могут возвращать меняющиеся списки IP-адресов. После анализа имени клиент в Сиднее получает страницу от сиднейского узла CDN. Дальнейшие страницы с того же «сервера» будут также взяты непосредственно от этого узла благодаря кэшированию DNS. Весь процесс показан на илл. 7.46.

Сложный вопрос в этом процессе — что значит ближайший узел CDN и как его найти. Это задача **сопоставления клиентов (client mapping)**, которую мы уже упоминали выше. Здесь нужно принять во внимание минимум два фактора. Первый — сетевое расстояние. Сетевой путь клиента к узлу CDN должен быть коротким и иметь высокую пропускную способность (таким образом, загрузка ускоряется). Чтобы определить сетевое местоположение клиента по его IP-адресу, CDN используют заранее вычисленное соответствие. Расстояние до выбранного узла может не быть кратчайшим — значение имеет комбинация длины сетевого пути и его максимальной емкости.




**Илл. 7.46.** Направление клиентов к ближайшим узлам CDN с использованием DNS

Второй фактор — нагрузка, которая уже имеется на узле CDN. Если узлы перегружены, они будут отсылать ответы медленно (точно так же, как перегруженные веб-серверы), а этого мы стремимся избежать в первую очередь. Поэтому иногда нужно распределять нагрузку между узлами CDN, отправляя часть клиентов к узлам, которые находятся несколько дальше, но при этом не так загружены.

Способность авторитетного DNS-сервера CDN сопоставлять клиента с ближайшим узлом кэша CDN зависит от того, может ли он определить местоположение клиента. Как уже упоминалось в разделе, посвященном DNS, современные расширения протокола DNS (например, механизм клиентской подсети EDNS) позволяют авторитетному серверу имен выяснить IP-адрес клиента. Возможный переход на протокол DoH порождает новые сложности, поскольку IP-адрес локального рекурсивного распознавателя может находиться очень далеко от клиента. Если локальный рекурсивный DNS-распознаватель не передает IP-адрес клиента (что обычно и происходит, поскольку весь смысл состоит в защите его личных данных), то CDN-сетям, не выполняющим DNS-разрешение для своих клиентов, очень сложно осуществить сопоставление. С другой стороны, CDN, также использующие DoH-совместимый распознаватель (например, CloudFlare и Google), могут получить значительные преимущества, так как они смогут напрямую выяснять IP-адреса клиентов, отправивших DNS-запросы. При этом запрашиваемый контент зачастую будет находиться непосредственно в данной CDN. Такая централизация службы DNS может произвести еще одно коренное изменение в распределении контента в течение ближайших нескольких лет.

В данном разделе представлено упрощенное описание работы CDN. На практике этот процесс включает в себя множество других важных деталей. К примеру, рано или поздно диски узлов CDN полностью заполняются, поэтому их нужно регулярно очищать. Была проделана большая работа по определению того, когда и какие файлы следует удалять; например, см. книгу Басу и др. (Basu et al., 2018).

## 7.5.4. Одноранговые сети

Не каждый может установить CDN из 1000 узлов, расположенных в разных уголках планеты, чтобы распространять свой контент. (На самом деле арендовать 1000 виртуальных машин по всему миру не трудно — индустрия хостинга хорошо развита и конкурентна. Но это лишь первый шаг в установке CDN.) К счастью, имеется доступная альтернатива: она проста в использовании и может распространять огромное количество контента. Это **одноранговая**, или **пиринговая**, **сеть (Peer-to-Peer network, P2P)**.

P2P-сети внезапно обрели популярность в 1999 году. Поначалу они массово применялись для нарушения закона: 50 млн пользователей сети Napster обменивались защищенными авторским правом песнями без разрешения правообладателей. Позже Napster был закрыт в судебном порядке, что вызвало большой общественный резонанс. Однако технология равноправных узлов открывает множество полезных возможностей и для законного использования. Другие системы продолжали развиваться с таким большим интересом со стороны пользователей, что P2P-трафик быстро превзошел веб-трафик. На сегодняшний день самым популярным P2P-протоколом остается BitTorrent. Он широко применяется для распространения видео (лицензионного и общедоступного), а также другого объемного контента (например, дисковые образы дистрибутивов операционных систем), что составляет значительную долю общего интернет-трафика. Мы рассмотрим этот протокол чуть позже.

### *Общие сведения*

Основная идея файлообменных P2P-сетей состоит в том, что множество компьютеров объединяют свои ресурсы, формируя систему распределения контента. Часто это обычные домашние компьютеры; нет никакой необходимости в том, чтобы они располагались в интернет-центрах обработки данных. Эти отдельные компьютеры называются **пирами (peer — «равный»)**; каждый из них может выступать и в роли клиента другого пира, получая его контент, и в роли сервера, предоставляя контент другим пирам. Одноранговые системы интересны отсутствием специализированной инфраструктуры в отличие от CDN. В распространении контента участвуют все узлы, часто без централизованного контроля. У этой технологии существует много областей применения (Карагианнис и др.; Karagiannis et al., 2019).

Многие люди в восторге от технологии P2P, поскольку видят в ней расширение возможностей обычного пользователя. Причина не только в том, что для запуска CDN нужны усилия большой организации, в то время как любой обладатель компьютера может присоединиться к сети P2P. Сети P2P имеют колоссальную емкость для распространения контента, что может сравниться с крупнейшими веб-сайтами.

### *Первые одноранговые сети: Napster*

Как упоминалось ранее, в первых одноранговых сетях (таких, как Napster) использовалась централизованная служба каталогов. Пользователи устанавливали

клиентское ПО для поиска в своем локальном хранилище тех файлов, к которым предоставлялся общий доступ. Затем программа передавала централизованной службе каталогов метаданные, описывающие эти файлы (имя и размер файла, идентификатор пользователя, предоставляющего общий доступ к контенту, и т. д.). Пользователи, желающие загрузить файлы из сети Napster, производили поиск по централизованному серверу каталогов и выясняли, у каких пользователей находился нужный им файл. Сервер сообщал им IP-адрес пира, предоставляющего общий доступ к нужному файлу, после чего клиентское ПО пользователя соединялось с этим хостом напрямую и скачивало файл.

Однако у схемы с централизованным сервером каталогов Napster был побочный эффект. Посторонние пользователи могли достаточно легко узнать, кем были размещены конкретные файлы (то есть фактически сканировать всю сеть). В определенный момент стало очевидно, что значительная часть контента была защищена авторским правом. Это привело к судебному запрету и закрытию сервиса. Также было обнаружено еще одно следствие применения централизованного сервера каталогов: чтобы вывести из строя весь сервис, достаточно было лишь отключить сервер. Без него Napster стал практически непригодным для использования. В качестве ответной меры разработчики новых одноранговых сетей стали проектировать более устойчивые к отключению или сбою системы. Обычно это достигалось путем децентрализации каталога или процесса поиска. Этот подход был взят на вооружение в одноранговых системах следующего поколения, таких как Gnutella.

### *Децентрализация каталога: Gnutella*

Созданная в 2000 году, сеть Gnutella была призвана решить некоторые проблемы, свойственные централизованной службе каталогов Napster, с помощью полностью распределенной функции поиска. В сети Gnutella присоединившийся к сети пир ищет другие подключенные пиры с помощью специального процесса обнаружения. Сначала он связывается с несколькими общеизвестными пирами сети, которые он должен был обнаружить на этапе начальной загрузки. Один из механизмов здесь сводится к тому, чтобы само ПО предоставляло некоторый набор IP-адресов пиров сети Gnutella. Обнаружив этот набор, пир может отправить поисковые запросы этим соседним пирам, те, в свою очередь, передадут их своим соседям, и т. д. Этот общий метод поиска в одноранговой сети часто называют **сплетнями (gossip)**.

Хотя метод сплетен решил некоторые проблемы полуцентрализованных сервисов (таких, как Napster), он быстро столкнулся с рядом других трудностей. Во-первых, в сети Gnutella пиры постоянно присоединялись к сети и покидали ее, ведь это были просто компьютеры других пользователей, которые то и дело подключались и отключались от сети. По сути, у пользователей не было особых причин для того, чтобы оставаться в сети после получения нужных файлов. Такое поведение, **фрирайдинг (free-riding)**, было довольно распространено: около 70 % пользователей не предоставляли никакого собственного контента (Адар и Губерман; Adar and Huberman, 2000). Вторая проблема состояла в том, что лавинообразные методы, и в особенности протокол сплетен, трудно поддаются

масштабированию. Это остро проявилось, когда Gnutella стала популярной: увеличение числа участников сети сопровождалось экспоненциальным ростом количества сообщений-сплетен. В результате для пользователей с ограниченной пропускной способностью сеть была практически непригодна. Введение так называемых **ультрапиров (ultra-peers)** в какой-то мере смягчило проблему, но в целом Gnutella отличалась весьма нерациональным использованием доступных сетевых ресурсов. Трудности с масштабируемостью процесса поиска в сети Gnutella подтолкнули исследователей к использованию **распределенных хеш-таблиц (Distributed Hash Tables, DHT)**. Они направляют процесс поиска в соответствующую одноранговую сеть в зависимости от его хеш-значения. При этом каждый узел этой сети отвечает за предоставление информации, охватывающей лишь некоторое подмножество всего пространства поиска, а DHT-таблица должна направлять запрос к пиру, способному предоставить нужный результат. DHT-таблицы применяются во многих современных одноранговых сетях, включая eDonkey (где DHT-таблица используется для поиска) и BitTorrent (где DHT-таблица нужна для масштабирования процесса отслеживания пиров в сети, о котором мы поговорим в следующем разделе).

Наконец, еще одна проблема заключалась в том, что Gnutella не производила автоматическую проверку содержимого скачиваемых пользователями файлов; в результате в ней появилось много поддельного контента. Почему же так произошло?

Это можно объяснить целым рядом причин. Например, как и любой интернет-сервис, Gnutella может подвергнуться атаке типа «отказ в обслуживании», что и произошло. Одна из самых простых DDoS-атак против сети — **атака загрязнения (pollution attack)**. Такие атаки заполонили Gnutella поддельным контентом. В том, чтобы эти сети стали бесполезными, были крайне заинтересованы представители звукозаписывающей индустрии (в первую очередь Американская ассоциация компаний звукозаписи). Оказалось, что они засоряли подобные сети огромным количеством поддельных материалов, чтобы заставить пользователей отказаться от использования сетей для обмена авторским контентом.

Таким образом, одноранговые сети столкнулись с рядом сложностей: нужно было обеспечить надлежащее масштабирование, убедить пользователей оставаться в сети после скачивания контента и обеспечить его проверку. Как будет показано далее, дизайн сети BitTorrent обеспечил решение всех этих проблем.

### *Решение проблем масштабирования, мотивации пользователей и проверки контента: BitTorrent*

Протокол BitTorrent разработан Брэмом Коэном в 2001 году, чтобы позволить набору одноранговых узлов быстро и легко обеспечивать общий доступ к файлам. Существуют десятки бесплатных поддерживающих его клиентов, аналогично тому, как множество браузеров поддерживает протокол HTTP для взаимодействия с веб-сервером. Этот протокол доступен как открытый стандарт, его описание находится на сайте bittorrent.org.

В типичной одноранговой системе, например, организованной с помощью протокола BitTorrent, каждый пользователь обладает некоторой информацией,

которая может представлять для кого-то интерес (бесплатное ПО, музыка, видео, фотографии и т. д.) Чтобы предоставить контент в общий доступ, необходимо ответить на следующие три вопроса:

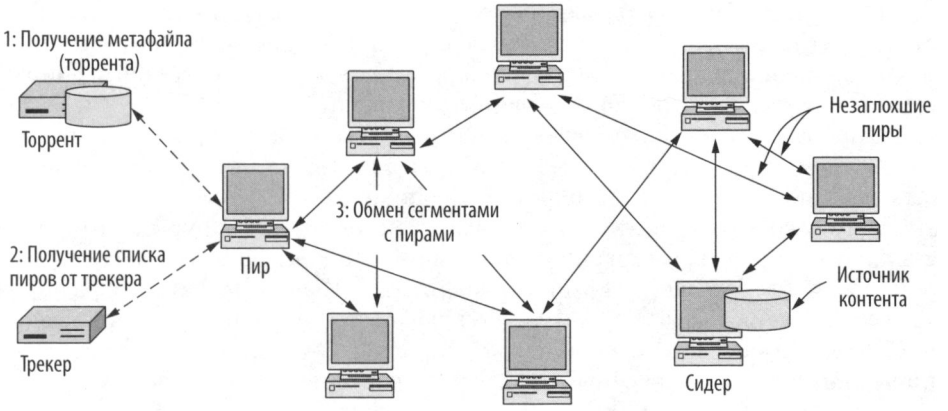
1. Как пир находит другие пиры с нужным ему контентом?
2. Как контент дублируется пирами, чтобы обеспечить быстрое скачивание для всех?
3. Как пиры поощряют друг друга к тому, чтобы наряду со скачиванием контента для себя они загружали его для других?

Наличие первого вопроса объясняется тем, что не все пиры содержат весь контент. Для решения этой проблемы в протоколе BitTorrent каждый контент-провайдер создает описание контента — **торрент (torrent)**. Торрент намного меньше, чем сам контент, и используется пиром, чтобы проверить целостность данных, которые он загружает с других пиров. Пользователи, которые хотят получить нужные им данные, должны сначала скачать торрент (допустим, найти его на веб-странице, рекламирующей контент).

Торрент — это просто файл в определенном формате, содержащий два ключевых вида информации. Прежде всего, это имя трекера — сервера, который направляет пиры к содержимому торрента. Вторым видом информации является список фрагментов одинакового размера, или **сегментов (chunks)**, из которых состоит контент. В первых версиях BitTorrent трекер был централизованным сервером. Как и в случае Napster, централизация делала трекер единственным уязвимым звеном сети. В современных версиях BitTorrent функциональность трекера в большинстве случаев децентрализуется с помощью DHT-таблицы. Для разных торрентов могут использоваться различные размеры сегментов, обычно в диапазоне от 64 до 512 Кбайт. Файл торрента содержит имя каждого сегмента, предоставленного как 160-битный SHA-1 хеш сегмента. Мы рассмотрим криптографические хеши, такие как SHA-1, в главе 8. Пока вы можете рассматривать хеш как более длинную и безопасную контрольную сумму. Торрент-файл, содержащий размер сегментов и хеши, как минимум на три порядка меньше, чем контент, поэтому он может быть передан быстро.

Чтобы загрузить контент, описанный в торренте, пир сначала контактирует с его трекером. **Трекер (tracker)** — это сервер (или группа серверов, организованная с помощью DHT), который поддерживает список всех остальных пиров, активно производящих скачивание и загрузку контента. Этот набор пиров называется **роем (swarm)**. Члены роя постоянно контактируют с трекером, чтобы сообщать, что они все еще активны, либо о том, что они покидают рой. Когда новый пир связывается с трекером, чтобы присоединиться к рою, трекер сообщает ему об остальных пирах в рое. Получение торрента и контакт с трекером — это первые два шага для загрузки контента (илл. 7.47).

Второй вопрос: каким образом делиться контентом, чтобы обеспечивать при этом быструю загрузку? Когда формируется начальный рой, некоторые пиры должны иметь все сегменты, составляющие контент. Эти пиры называются **сидерами (seeders — «сеятели»)**. Другие пиры, которые присоединяются к рою, не будут иметь никаких сегментов; это пиры, скачивающие контент.




**Илл. 7.47.** BitTorrent

В то время как пир участвует в рое, он одновременно скачивает отсутствующие сегменты с других пиров и загружает имеющиеся у него сегменты тем пирам, которым они нужны. Этот обмен показан как последний шаг распределения контента на илл. 7.47. Какое-то время пир собирает сегменты, пока не загрузит весь контент. Пир может покинуть рой (и вернуться) в любое время. Обычно он остается в течение некоторого короткого периода после окончания своей собственной загрузки. Из-за появляющихся и исчезающих пиров «текучесть» в рое может быть довольно большой.

Чтобы данный метод работал корректно, каждый сегмент должен быть доступен у большого числа пиров. Если бы все получали сегменты в одинаковом порядке, многие пиры зависели бы от сидеров следующего сегмента. Это создало бы узкое место. Вместо этого пиры обмениваются списками имеющихся у них сегментов. Затем они выбирают для загрузки преимущественно редкие сегменты, которые трудно найти. Идея в том, что скачивание редкого сегмента приведет к созданию еще одной его копии, что, в свою очередь, облегчит его поиск и скачивание для других пиров. Если так будут поступать все пиры, то спустя некоторое время все сегменты станут широко доступны.

Третий вопрос касается мотивации пользователей. Узлы CDN-сетей предназначены лишь для того, чтобы предоставлять контент пользователям, но P2P-узлы работают иначе. Это компьютеры пользователей, а люди часто больше заинтересованы в том, чтобы получить нужный им фильм, чем в том, чтобы помочь в скачивании остальным. То есть у пользователей есть мотивация к тому, чтобы обмануть систему. Узлы, берущие ресурсы из системы без какого-либо ответного вклада, называют **фрирайдерами (free-riders)**, или **личерами (leechers — «пиявки»)**. Если их слишком много, система перестает нормально функционировать. Первые P2P-системы известны наличием таких узлов (Сарою и др.; Saroiu et al., 2003), поэтому протокол BitTorrent был призван минимизировать их количество.

BitTorrent пытается решить эту проблему путем вознаграждения пиров, которые производят достаточную исходящую загрузку. Каждый пир беспорядочно обращается к другим пирам, получает сегменты от них и в то же время пересылает

сегменты к ним. Пир продолжает обмен сегментами только с небольшим количеством других пиров, что дает самую высокую производительность скачивания, и одновременно в случайном порядке пробует подключиться к другим пирам в поиске хороших партнеров. Случайные обращения к пирам также позволяют вновь прибывшим получить начальные сегменты, которыми они могут обменяться с другими пирами. Пиры, с которыми узел в данный момент обменивается сегментами, называют **незаглохшими (unchoked)**.

Через какое-то время этот алгоритм должен сопоставить друг с другом пиры с похожей скоростью загрузки и скачивания. Чем активнее пир содействует остальным, тем больше он может ожидать в ответ. Использование набора пиров также помогает насыщать пропускную способность загрузки пиров для высокой производительности. И наоборот, если пир не пересылает сегменты на другие пиры или делает это очень медленно, рано или поздно он окажется отрезанным, или **заглохшим (choked)**. Эта стратегия не позволяет использовать рой, не давая ничего взамен.

Этот алгоритм заглушения иногда описывается как реализация стратегии **«око за око» (tit-for-tat)**, поощряющей сотрудничество в повторяющихся взаимодействиях. Это широко известная стратегия из теории игр, при которой: 1) у игроков нет мотивации к обману, если они постоянно играют друг с другом (как происходит в сети BitTorrent, где пиры должны периодически обмениваться сегментами) и 2) игроки наказываются при отказе от сотрудничества (как в случае заглохших пиров). Несмотря на эту схему, клиенты BitTorrent все же могут прибегать к различным способам обмана системы (Пиатек и др.; Piatek et al., 2007). Например, поскольку алгоритм сети BitTorrent поощряет выбор редких сегментов, пиры могут предоставить ложную информацию о том, какими сегментами файла они располагают (например, они могут сообщить о наличии редких сегментов) (Лиокас и др.; Liogkas et al., 2006). Кроме того, существуют программы, позволяющие клиентам сообщать трекеру ложную информацию о соотношении между объемом скачанной и переданной информации (то есть сообщать о том, что они раздавали данные другим пирам, хотя в действительности этого не было). По этим причинам критически важно, чтобы пиры проверяли сегменты, получаемые от остальных. Это можно делать путем сравнения хеша SHA-1 каждого сегмента, указанного в торрент-файле, с соответствующим вычисленным хешем SHA-1 каждого скачиваемого сегмента.

Еще одной проблемой является мотивация пиров к тому, чтобы оставаться в рое сети BitTorrent в качестве сидера даже после скачивания нужного файла. Если они не будут этого делать, может возникнуть ситуация, когда ни у кого в рое не будет целого файла или (что еще хуже) когда в рое вообще не будет определенных сегментов, что сделает невозможным скачивание файла полностью. Это особенно остро проявляется в случае непопулярных файлов (Менаше и др.; Menasche et al., 2013). Для решения проблемы мотивации были разработаны различные методы (Рамачандран и др.; Ramachandran et al., 2007).

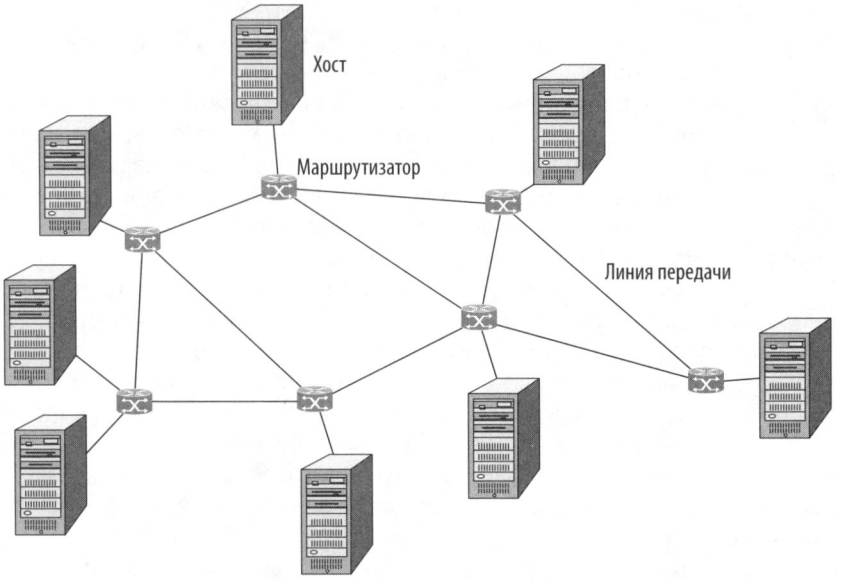
В ходе этого раздела вы убедились, что к теме протокола BitTorrent прилагается обширная терминология: торренты, рои, личеры, сидеры, трекеры и т. д. Чтобы узнать больше о BitTorrent, ознакомьтесь с короткой статьей Коэна (Cohen, 2003).

## 7.5.5. Эволюция интернета

Как было упомянуто в главе 1, интернет прошел довольно причудливый исторический путь, возникнув в ходе осуществления научно-исследовательского проекта, над которым работали несколько десятков американских университетов по договору с агентством ARPA. Трудно даже определить, когда именно он появился. Возможно, это произошло 21 ноября 1969 года, когда была установлена связь между двумя узлами ARPANET, расположенными в Калифорнийском университете в Лос-Анджелесе (UCLA) и в Исследовательском институте Стэнфорда (SRI)? Или это случилось 17 декабря 1972 года, когда к ARPANET подключилась гавайская сеть AlohaNet, образовав тем самым интерсеть? Или, быть может, интернет появился 1 января 1983 года, когда ARPA официально утвердила протокол TCP/IP? А возможно, он возник в 1989 году, когда Тим Бернерс-Ли предложил создать то, что впоследствии стало Всемирной паутиной? Сложно сказать. Однако ни у кого нет сомнений, что со времен становления сети ARPANET и первых зачатков Всемирной паутины интернет подвергся огромным изменениям, значительная часть которых была обусловлена повсеместным распространением CDN и облачных вычислений. Ниже мы кратко рассмотрим этот процесс.

На илл. 7.48 представлена базовая модель сети ARPANET и первой версии интернета. Она включает в себя три типа компонентов:

1. Хосты (компьютеры, выполняющие определенную работу для пользователей).
2. Маршрутизаторы (которые в ARPANET обозначались как «IMP»), производящие коммутацию пакетов.




**Илл. 7.48.** В первой версии интернета использовались главным образом двухточечные соединения

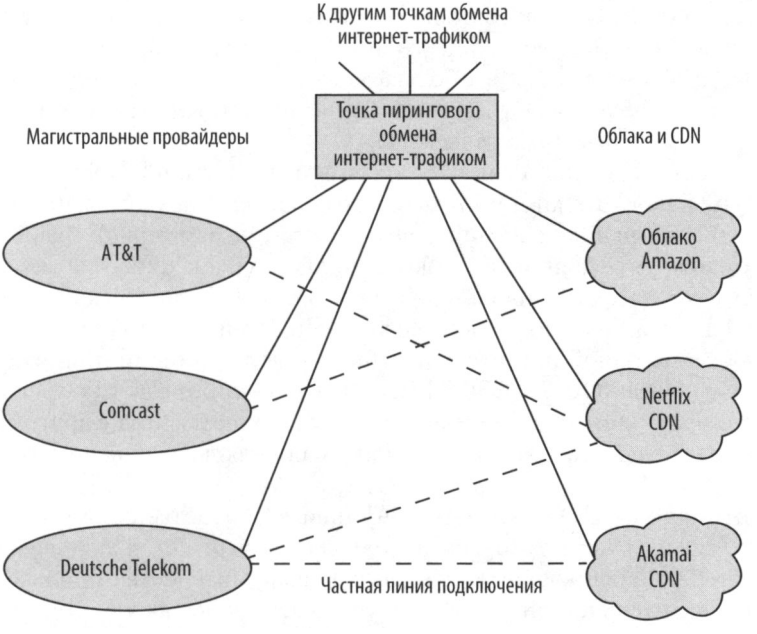
3. Линии передачи (изначально выделенные телефонные линии со скоростью 56 Кбит/с). Каждый маршрутизатор был подключен к одному или нескольким компьютерам.

Концептуальная модель исходной архитектуры интернета строилась главным образом на базовой идее двухточечной коммуникации. Все компьютеры (хосты) рассматривались как равноправные узлы (хотя некоторые были намного мощнее других), и любой из них мог отправлять пакеты любому другому, поскольку каждый компьютер имел уникальный адрес. С появлением протокола TCP/IP длина этого адреса стала составлять целых 32 бита, что в то время казалось бесконечно большим размером. Сегодня он кажется ничтожно малым. В качестве модели передачи использовалась простая дейтаграммная система без сохранения информации о состоянии клиента, где каждый пакет содержит свой адрес назначения. После прохождения через маршрутизатор пакет уже не отслеживался. Маршрутизация производилась переходами. Путь пакета определялся в зависимости от указанного в нем адреса назначения и того, какую линию передачи следовало использовать для этого адреса согласно таблицам маршрутизатора.

Положение стало меняться, когда интернет вышел за рамки академической среды и приобрел коммерческий характер. Это привело к созданию магистральных сетей со сверхскоростными каналами, администрируемых такими крупными телекоммуникационными компаниями, как AT&T и Verizon. Каждая компания использовала собственную магистральную сеть, однако они связывались друг с другом через точки пирингового обмена. Появились интернет-провайдеры, взявшие на себя задачу подключения к интернету домашних пользователей и предприятий, а также региональные сети, соединявшие провайдеров с магистральными сетями (см. илл. 1.17). Следующим шагом стало появление общенациональных интернет-провайдеров и CDN (см. илл. 1.18).

В главе 1 мы отмечали, что появление облачных вычислений и сверхбольших CDN привело к новым изменениям в структуре интернета. Такие компании, как Amazon и Microsoft, сегодня используют облачные центры обработки данных, включающие сотни тысяч размещенных в одном здании компьютеров, что позволяет их пользователям (обычно это крупные компании) буквально за считаные секунды выделить для своих задач 100, 1000 или 10 000 компьютеров. Во время большой распродажи в так называемый киберпонедельник (понедельник после Дня благодарения) компания Walmart может выделить 10 000 компьютеров для обработки возросшей нагрузки, просто автоматически запросив их у своего облачного провайдера; они будут предоставлены буквально через несколько секунд. После возвращения к нормальному режиму во вторник эти мощности можно будет вернуть назад. Почти все крупные компании, имеющие дело с миллионами пользователей, используют облачные сервисы, чтобы практически мгновенно увеличивать или уменьшать свои вычислительные мощности по мере необходимости. Как мы уже говорили, дополнительным преимуществом облачных сервисов является достаточно хорошая защита от DDoS-атак. В силу огромных размеров облако может принять тысячи запросов за секунду, ответить на них и продолжить нормальное функционирование, не позволяя DDoS-атаке достигнуть своей цели.

CDN имеют иерархическую структуру, включающую основной сайт (который может быть продублирован два или три раза для надежности) и множество размещенных по всему миру кэшей для контента. Когда пользователь запрашивает контент, он доставляется из ближайшего кэша. Это позволяет снизить величину задержки и распределить рабочую нагрузку. Первая крупная коммерческая CDN от компании Akamai содержит более 200 000 узлов кэша в более чем 1500 сетях, охватывающих более 120 стран. Аналогично CDN от компании CloudFlare сегодня включает в себя кэширующее оборудование в более чем 90 странах. Поскольку обычно узлы кэша CDN находятся рядом с офисом интернет-провайдера, пересылка данных между CDN и провайдером осуществляется по сверхбыстрому оптоволоконному каналу, длина которого может составлять всего 5 м. В силу новых реалий архитектура интернета приобрела вид, показанный на илл. 7.49: основная часть трафика — это обмен данными между сетями доступа (например, региональными) и распределенной облачной инфраструктурой (CDN или облачными сервисами).




**Илл. 7.49.** Большую часть интернет-трафика сегодня составляет трафик облачных сервисов и CDN; при этом сети доступа и интернет-провайдеры активно обмениваются трафиком через частные линии подключения

Пользователи отправляют крупным серверам запросы на выполнение определенных действий, после чего серверы выполняют эти запросы, создавая веб-страницу, отражающую результаты их действий. Вот некоторые примеры таких запросов:

1. Купить продукт в интернет-магазине.
2. Получить электронное письмо от почтового сервиса.

3. Выдать банку платежное поручение.
4. Произвести потоковую передачу песни или фильма на устройство пользователя.
5. Обновить страницу в Facebook.
6. Отобразить статью онлайн-версии газеты.

Этой модели соответствует почти весь трафик современного интернета. Стремительное распространение облачных сервисов и CDN полностью перевернуло традиционную клиент-серверную модель интернет-трафика, при которой клиент получал контент или обменивался им с одним сервером. Сегодня подавляющее большинство контента и коммуникаций обслуживается распределенными облачными сервисами; многие интернет-провайдеры направляют им большую часть своего трафика. В наиболее развитых регионах просто нет необходимости в том, чтобы пользователи получали доступ к огромным объемам контента, используя транзитную инфраструктуру большой протяженности. CDN разместили популярный контент ближе к пользователям. Часто это означает и малую географическую удаленность, и доступ через прямую линию подключения от интернет-провайдера. Таким образом, все больше контента доставляется по CDN, которые либо напрямую подключены к сетям доступа по частным соединениям, либо даже имеют в них собственные узлы кэша.

Магистральные сети позволяют многим облачным сервисам и CDN связываться друг с другом через точки пирингового обмена, если между ними нет частной выделенной линии подключения. Так, к точке обмена интернет-трафиком DE-CIX в Франкфурте подключено около 2000 сетей, а к точкам обмена AMS-IX в Амстердаме и LINX в Лондоне — примерно по 1000. К каждой крупной точке обмена в США подключено по несколько сотен сетей. Эти точки также соединены между собой одной или несколькими оптоволоконными линиями OC-192 (9,6 Гбит/с) и/или OC-768 (38,5 Гбит/с). Точки пирингового обмена и крупные коммуникационные сети, которые в них соединяются друг с другом, образуют магистраль интернета, к которой напрямую подключено большинство облаков и CDN.

Как уже упоминалось, CDN компании Akamai включает в себя более 200 000 серверов, большая часть которых расположена внутри сетей интернет-провайдеров. Эта тенденция продолжит менять ландшафт интернета в ближайшие годы. Все больше распространяются и другие CDN, такие как CloudFlare. Наконец, свои CDN начинают развертывать провайдеры контента и сервисов. К примеру, компания Netflix развернула CDN под названием Open Connect, которая размещает контент Netflix в узлах кэша, либо в точках обмена интернет-трафиком, либо непосредственно внутри сети доступа интернет-провайдера. Степень, в которой интернет-путь пройдет через отдельную магистральную сеть или точки обмена, зависит от множества факторов, таких как затраты, доступные возможности подключения в регионе и экономия за счет масштабирования. Если в Европе и ряде других частей мира очень популярны точки обмена интернет-трафиком, то в США чаще применяется прямое подключение посредством частных межсетевых соединений.

## 7.6. РЕЗЮМЕ

Именование в ARPANET изначально было очень простым: в текстовом файле перечислялись имена всех хостов и соответствующие им IP-адреса. Каждую ночь этот файл подгружался на все компьютеры. Но когда ARPANET перерос в интернет и его объем невероятно увеличился, потребовалась гораздо более изысканная и динамичная схема именования. Сегодня именование доменов в интернете реализуется при помощи иерархического подхода — системы доменных имен, DNS. Эта система организует все подключенные к интернету компьютеры в набор деревьев. На верхнем уровне находятся общеизвестные родовые домены, включая com, edu и около 200 национальных доменов. DNS реализована в виде распределенной базы данных, серверы которой расположены по всему миру. Обратившись к DNS-серверу, процесс может преобразовать доменное веб-имя в IP-адрес, используемый для общения с компьютером в этом домене. DNS применяется для различных целей; в последнее время в силу выявленных проблем конфиденциальности возникла тенденция к шифрованию DNS-трафика с помощью протокола TLS или HTTPS. В итоге это может привести к централизации службы DNS, что, в свою очередь, изменит фундаментальные аспекты архитектуры интернета.

Электронная почта — это одно из самых ранних и популярных приложений интернета. Ею до сих пор пользуются все, от мала до велика. Большинство существующих сегодня систем электронной почты соответствует спецификациям RFC 5321 и 5322. Сообщения содержат ASCII-заголовки и данные разных типов, прописанных в MIME-заголовках. Почта отдается агентам передачи для доставки, а затем извлекается и отображается во многих пользовательских агентах, включая веб-приложения. Отправленная почта доставляется по протоколу SMTP, устанавливающему TCP-соединение между хостом-источником и хостом-получателем.

Всемирная паутина является приложением, которое многие считают интернетом. Изначально это была система для обработки страниц, написанных на HTML и содержащих гиперссылки на другие страницы, которые загружались при помощи TCP-соединения браузера с сервером. Сегодня большая часть контента является динамической, или на стороне сервера (например, с PHP), или на стороне браузера (например, с JavaScript). Динамические страницы на сервере в сочетании с внутренними базами данных позволяют использовать системы электронной торговли и поиска. Динамические страницы браузера превращаются в полнофункциональные приложения, такие как электронная почта, которая работает внутри браузера и использует веб-протоколы для общения с удаленными серверами. С ростом рекламной индустрии в интернете стало практически повсеместно использоваться отслеживание, реализуемое с помощью самых разных методов, от использования файлов cookie до снятия цифрового отпечатка пользователя.

Хотя существуют способы блокировки некоторых типов отслеживающих механизмов (например, файлов cookie), иногда такая блокировка плохо сказывается на функциональности веб-сайта. Кроме того, некоторые механизмы

отслеживания чрезвычайно трудно поддаются блокировке (к примеру, метод снятия цифрового отпечатка).

Цифровое аудио и видео стали ключевыми факторами развития интернета с 2000 года. Сегодня передача видеоданных составляет большую часть интернет-трафика. Значительная его доля передается с веб-сайтов потоковым методом с помощью определенного набора протоколов; также широко используется TCP. При этом медиаданные передаются в режиме реального времени множеству пользователей. Так работают радио- и телестанции, транслирующие самые разнообразные программы в интернете. Аудио и видео также используются для конференций в реальном времени. Для многих звонков применяется IP-телефония или приложения видеоконференций, а не традиционные телефонные сети.

Существует небольшое количество крайне популярных веб-сайтов и огромное количество не слишком популярных. Для обслуживания популярных сайтов были созданы сети распределения контента, CDN. Они используют DNS, чтобы направлять клиентов на ближайший сервер. Серверы расположены в центрах обработки данных по всему миру. В качестве альтернативы P2P-сети позволяют нескольким компьютерам совместно использовать контент (например, видео). Они обеспечивают емкость распределения контента, которая растет с увеличением числа компьютеров-участников, так что подобные сети могут соперничать с крупнейшими сайтами.

## ВОПРОСЫ И ЗАДАЧИ

1. На илл. 7.5 после слова laserjet не поставлена точка. Почему?
2. Приведите пример, аналогичный показанному на илл. 7.7, в котором распознаватель производил бы поиск доменного имени course-info.cs.uchicago.edu за восемь шагов. При каком сценарии это произойдет на практике?
3. Какая DNS-запись проверяет ключ, который используется для подписи DNS-записей для авторитетного сервера имен?
4. Какая DNS-запись проверяет подпись DNS-записей для авторитетного сервера имен?
5. Опишите процесс сопоставления клиентов, при котором часть инфраструктуры DNS выясняет, какой сервер контента расположен ближе других к клиенту, отправившему DNS-запрос. На какие допущения нужно будет опереться при определении местоположения клиента?
6. Рассмотрите ситуацию, при которой кибертеррорист заставляет «упасть» все DNS-серверы в мире. Как это повлияет на доступ в интернет?
7. Объясните, какие преимущества и недостатки дает использование протокола TCP вместо UDP для запросов и ответов службы DNS.
8. Предполагается, что кэширование DNS-поиска производится в обычном порядке, и DNS-трафик не шифруется. Какие сторонние трекеры могут отслеживать весь DNS-поиск, производимый с вашего локального устройства? Кто может отслеживать DNS-поиск, если DNS-трафик шифруется с помощью протокола DoH или DoT?

9. Джон хочет получить оригинальное доменное имя и, используя специальную программу, случайным образом сгенерировать для него вторичное доменное имя. Он хочет зарегистрировать это имя в родовом домене com. Сгенерированное доменное имя насчитывает 253 знака. Разрешит ли регистратор домена com регистрацию этого имени?
10. Может ли компьютер иметь одно имя DNS и несколько IP-адресов? Как это может произойти?
11. Может ли компьютер иметь два имени DNS, относящихся к разным доменам верхнего уровня? Если да, приведите правдоподобный пример. Если нет, обоснуйте почему.
12. Некоторые системы электронной почты поддерживают поле заголовка `Content-Return:`. В нем указывается, нужно ли возвращать содержимое письма, если оно не будет доставлено получателю. Куда входит это поле — в состав конверта или в состав заголовка письма?
13. Вы получили электронное письмо и подозреваете, что его авторы преследуют неблаговидные цели. Однако содержимое поля `FROM` указывает, что оно было отправлено человеком, которого вы хорошо знаете. Можно ли доверять содержимому этого письма? Как можно проверить его подлинность?
14. Системы электронной почты хранят адресные книги e-mail, с помощью которых пользователь может найти нужный адрес. Для поиска по таким книгам имена адресатов должны быть разбиты на стандартные компоненты (например, имя и фамилия). Какие проблемы нужно решить, чтобы разработать соответствующий международный стандарт?
15. Крупная юридическая фирма предоставляет каждому из своих многочисленных сотрудников отдельный адрес электронной почты. Адрес состоит из логина, знака @, названия фирмы и домена com. Однако компания не определила точный формат логина. Так что одни сотрудники используют свои имена, другие — фамилии, а некоторые — инициалы. Теперь фирма хочет задать фиксированный формат вида имя.фамилия@название_фирмы.com, который можно было бы использовать для адресов всех ее сотрудников. Как это сделать и обойтись при этом без особых проблем?
16. 100-байтная ASCII-строка кодируется с использованием кодировки base64. Каким будет размер результирующей строки?
17. С помощью base64 ваш сокурсник кодировал ASCII-строку `«ascii»` и получил в результате строку `«YXNjaWJ»`. Объясните, что пошло не так в ходе кодирования, и предложите правильный способ кодирования строки.
18. В качестве лабораторной работы по компьютерным сетям вы создаете приложение для обмена мгновенными сообщениями. Оно должно передавать ASCII-текст и двоичные файлы. К сожалению, другой студент из вашей команды уже сдал на проверку серверный код, забыв реализовать функцию передачи двоичных файлов. Можно ли реализовать эту функцию, изменив только клиентский код?
19. В любом стандарте, таком как RFC 5322, должно быть описание точной грамматики — это требуется для взаимодействия различных реализаций.

Даже самые простые элементы должны быть четко определены. Например, в заголовках SMTP допустимы пробелы между символами. Приведите *два* убедительных альтернативных определения этих пробелов.

20. Назовите пять типов MIME, не указанных в тексте. Информацию можно взять из настроек браузера или из интернета.
21. Предположим, вы хотите переслать другу MP3-файл, но его провайдер ограничивает максимальный размер входящей почты до 1 Мбайт, а файл занимает 4 Мбайт. Можно ли решить поставленную задачу, используя RFC 5322 и MIME?
22. IMAP позволяет пользователям запрашивать и загружать почту из удаленного почтового ящика. Означает ли это, что внутренний формат почтовых ящиков должен быть стандартизован, чтобы любые клиентские программы, использующие IMAP, могли обратиться к почтовому ящику на любом сервере? Поясните свой ответ.
23. Стандартный URL-адрес https подразумевает, что веб-сервер прослушивает порт 443. Однако он может прослушивать и другой порт. Придумайте практичный синтаксис для URL-адреса, обеспечивающего доступ к файлу с использованием нестандартного порта.
24. Представьте, что сотрудник факультета математики Стэнфордского университета написал новый документ, который он хочет распространить по FTP, чтобы его коллеги оставили отзывы. Он помещает документ в каталог ftp/pub/forReview/newProof.pdf. Как будет выглядеть URL этого документа?
25. Имеется веб-страница, загрузка которой по HTTP занимает 3 с при использовании постоянного соединения и последовательных запросов. Из этих 3 с 150 мс уходит на установку соединения и получение первого ответа. Загрузка той же страницы с использованием конвейеризованных запросов занимает 200 мс. Предполагается, что отправка запроса осуществляется мгновенно и что интервал между поступлением запроса и выдачей ответа одинаковый для всех запросов. Сколько запросов выполняется при загрузке этой веб-страницы?
26. В качестве лабораторной работы по компьютерным сетям вы создаете сетевое приложение. Другой студент из вашей команды говорит, что поскольку ваша система использует для связи протокол HTTP поверх TCP, в ней не нужно учитывать вероятность разрыва связи между хостами. Что вы ему ответите?
27. Для каждого из перечисленных случаев укажите: (1) возможно ли и (2) лучше ли использовать PHP-скрипт или JavaScript и почему:

    а) Отображение календаря на любой месяц, начиная с сентября 1752 года.

    б) Отображение расписания рейсов из Амстердама в Нью-Йорк.

    в) Вывод полинома с коэффициентами, введенными пользователем.

28. Заголовок `If-Modified-Since` может использоваться для проверки актуальности кэшированной страницы. Соответствующие запросы могут отсылаться на страницы, содержащие изображения, звуки, видео и т. д., а также на обычные

страницы на HTML. Как вы думаете, эффективность этого метода будет выше для изображений JPEG или для страниц HTML? Хорошенько подумайте над значением слова «эффективность» и после этого объясните свой ответ.

29. Вы запрашиваете у сервера веб-страницу. Ответ сервера содержит заголовок `Expires` с датой истечения срока действия, в нем указан следующий день. Через пять минут вы запрашиваете ту же страницу у того же сервера. Может ли он отправить вам более новую версию страницы? Объясните свой ответ.
30. Имеет ли смысл отдельному провайдеру функционировать в качестве CDN? Если да, то как должна работать система? Если нет, то чем плоха такая идея?
31. При кодировании звуковых компакт-дисков используется частота дискретизации 44 100 Гц и 16-битные сэмплы. Каков при этом битрейт несжатых данных? Сколько байтов потребуется для звуковых данных длительностью в один час? На компакт-диске можно разместить 700 Мбайт данных. Для чего нужна неиспользованная часть этого объема?
32. При кодировании звуковых компакт-дисков используется частота дискретизации 44 100 Гц и 16-битные сэмплы. Насколько целесообразным будет повышение качества звука путем использования частоты дискретизации 88 000 Гц в сочетании с 16-битными сэмплами? А что можно сказать об использовании частоты дискретизации 44 100 Гц и 24-битных сэмплов?
33. Предположим, что для звукового компакт-диска не используется сжатие. Сколько мегабайтов данных должен содержать диск, чтобы проигрывать музыку в течение двух часов?
34. Сервер, на котором расположен популярный чат, отправляет данные своим клиентам со скоростью 32 Кбит/с. Какой размер пакета он использует, если эти данные поступают клиентам с интервалом в 100 мс? Каким будет размер пакета, если клиенты будут получать данные с интервалом в 1 с?
35. Сервер аудиопотока расположен на удалении от клиентского компьютера с установленным плеером. Из-за расстояния возникает задержка в 100 мс в одном направлении. Сервер отдает данные со скоростью 1 Мбит/с. Если проигрыватель содержит буфер объемом 2 Мбайт, что можно сказать о расположении нижнего и верхнего пределов заполнения этого буфера?
36. Вы просматриваете потоковое видео, но за 10 с до его конца происходит разрыв вашего интернет-соединения. Видео имеет разрешение 2048 × 1024 пикселей, использует 16 бит для каждого пикселя и воспроизводится с частотой 60 кадров/с. На вашем компьютере буферизовано 64 Мбайт (67 108 864 байта) закодированных данных. Коэффициент сжатия равен 32:1. Удастся ли вам просмотреть видео до конца?
37. Предположим, что в беспроводной среде передачи теряется много пакетов. Как следует пересылать несжатые звуковые данные CD-качества, чтобы потеря пакетов могла вызывать ухудшение качества звука, но не разрывы в звучании?
38. В данной главе мы обсудили схему буферизации видео, представленную на илл. 7.34. Будет ли эта схема работать и для только лишь звуковых данных? Ответ поясните.

39. Потоковая передача аудио- и видеоданных в реальном времени должна быть плавной. При этом на пользовательский опыт могут влиять два фактора — случайная задержка и джиттер пакетов. Не являются ли они, по сути, одним и тем же? При каких условиях возникает каждый из них? Можно ли с ними успешно бороться, и если да, то как?
40. Какая скорость требуется для передачи несжатого цветного изображения размером 1920 × 1080 пикселей и 16 битами на пиксель при 60 кадрах/с?
41. Какой коэффициент сжатия нужно обеспечить для передачи видео в формате 4K по каналу с пропускной способностью 80 Мбит/с? Предполагается, что видео воспроизводится со скоростью 60 кадров/с и для значений каждого пикселя используется по 3 байта.
42. Допустим, DASH-система с частотой 50 кадров/с разбивает видео на 10-секундные сегменты, каждый из которых содержит ровно 500 кадров. Вызовет ли это какие-либо проблемы? (*Подсказка:* подумайте, какие виды кадров используются в MPEG.) Если проблемы возникнут, как их устранить?
43. Представьте, что сервис потоковой передачи видео решил использовать протокол UDP вместо TCP. UDP-пакеты могут прибывать в порядке, отличном от того, в котором они были отправлены. Какую проблему это порождает и как с ней справиться? Не внесет ли ваше решение какие-либо осложнения?
44. Сотрудник компании по разработке облачных игр предлагает создать новый протокол транспортного уровня, который будет лишен недостатков протоколов TCP и UDP и обеспечит низкий уровень задержки и джиттера для мультимедийных приложений. Объясните, почему это не сработает.
45. Рассмотрим пример видеосервера, обслуживающего 50 000 клиентов. Каждый клиент смотрит три фильма в месяц. Предположим, что две трети всех фильмов начинают просматривать в 21:00. Сколько фильмов одновременно должен передавать сервер в этот период? Если для передачи каждого фильма требуется 4 Мбит/с, сколько соединений типа OC-12 нужно для соединения сервера с сетью?
46. При каких условиях лучше не использовать CDN?
47. На популярном веб-сайте размещены 2 миллиарда видеозаписей. Если их популярность распределена по закону Ципфа, то какая доля просмотров приходится на 10 самых популярных видео?
48. Одним из преимуществ одноранговых систем является то, что у них часто нет централизованного пункта управления, что повышает их отказоустойчивость. Объясните, почему протокол BitTorrent не является полностью децентрализованным.
49. Расскажите, в силу каких причин и каким образом BitTorrent-клиент может обманывать других участников BitTorrent-сети.

# ГЛАВА 8
# Сетевая безопасность

В первые десятилетия своего существования компьютерные сети в основном использовались научными сотрудниками в университетах для отправки электронных писем и офисными работниками для общего доступа к принтеру. В таких условиях вопросам безопасности не уделялось большого внимания. Но теперь, когда миллионы людей совершают покупки, управляют банковскими счетами и заполняют налоговые декларации в интернете, проблема безопасности воспринимается очень серьезно (учитывая, что за это время были обнаружены многочисленные уязвимости). Мы рассмотрим эти вопросы с разных точек зрения, укажем на подводные камни и обсудим алгоритмы и протоколы, повышающие сетевую безопасность.

Исторически взлом сетей существовал задолго до появления интернета. Правда, атакам тогда подвергалась телефонная сеть, а вмешательство злоумышленников в работу сигнального протокола называлось **телефонным фрикингом (phone phreaking)**. Это явление возникло в конце 1950-х годов и получило широкое распространение в 1960-е и 1970-е годы. В те дни авторизация и маршрутизация звонков еще были «внутриполосными»: для управления коммутаторами телефонная компания передавала звуки определенной частоты в том же канале, в котором осуществлялась голосовая связь.

Джон Дрейпер (John Draper), один из самых известных телефонных фрикеров и весьма неоднозначная личность, в конце 1960-х годов обнаружил, что игрушечный свисток из коробки кукурузных хлопьев Cap'n Crunch издавал звук с частотой 2600 Гц, которая, как оказалось, также использовалась компанией AT&T для авторизации междугородних звонков. С помощью свистка Дрейпер мог звонить по межгороду бесплатно. Позже он получил прозвище Капитан Кранч (Captain Kranch). На основе свистка Дрейпер создал так называемую синюю коробку — устройство для взлома телефонной системы. В 1974 году Дрейпер был арестован за неуплату услуг телефонной связи и отправился в тюрьму. Однако его пример все же успел побудить еще двух пионеров информационных технологий из окрестностей Сан-Франциско, Стива Возняка (Steve Wozniak) и Стива Джобса (Steve Jobs), к тому, чтобы тоже заняться фрикингом и создать собственные «синие коробки». Позже они

разработали компьютер *Apple*. По словам Возняка, если бы не Капитан Кранч, Apple никогда бы не появился на свет[1].

Безопасность — весьма обширная тема, включающая множество вопросов, связанных с человеческими пороками. В простейшем случае ее обеспечение сводится к тому, чтобы не позволять чрезмерно любопытным людям читать или, что еще хуже, незаметно менять чужие сообщения. Системы безопасности следят, чтобы злоумышленники не могли нарушить работу базовых элементов сети (BGP или DNS), сделать недоступными ссылки или сетевые службы или получить несанкционированный доступ к удаленным службам. Также они выясняют, что, к примеру, сообщение с требованием оплатить все счета до пятницы действительно отправила налоговая служба, а не мошенники. Системы безопасности имеют дело с проблемами перехвата и имитации легитимных сообщений, а также с пользователями, отрицающими факт отправки ими подобных писем.

В основном проблемы безопасности создают злоумышленники, стремящиеся извлечь выгоду, привлечь к себе внимание или причинить кому-то вред. Несколько наиболее распространенных типов взломщиков перечислены на илл. 8.1. Из этого списка должно быть ясно, что задача обеспечения сетевой безопасности включает в себя значительно больше, нежели просто устранение программных ошибок. Часто стоит задача перехитрить умного, убежденного и иногда хорошо финансируемого противника. Меры, способные остановить случайного взломщика, мало повлияют на серьезного преступника.

В своей статье для журнала *;Login:* ассоциации USENIX Джеймс Микенс (James Mickens) из компании Microsoft (впоследствии ставший профессором Гарвардского университета) высказал мысль о том, что следует различать обычных хакеров и, к примеру, опытных агентов спецслужб. Для защиты от заурядных злоумышленников достаточно использовать базовые меры безопасности, продиктованные обыкновенным здравым смыслом. Микенс наглядно демонстрирует это различие:

> *Если вы имеете дело с Моссадом, вы умрете, и ничто вас не спасет. Тот факт, что вы используете префикс https://, его не остановит. Если Моссаду понадобятся ваши данные, ваш мобильный телефон с помощью дрона заменят на кусок урана в форме телефона. А когда вы умрете от рака, они соберут пресс-конференцию и заявят: «Это были не мы», в то время как на их футболках будет написано «ЭТО ТОЧНО БЫЛИ МЫ». Затем они скупят все ваши вещи и смогут непосредственно взглянуть на фотографии вашего отпуска, вместо того чтобы читать ваши скучные электронные письма о нем.*

Этим Микенс хотел сказать, что продвинутые злоумышленники применяют намного более совершенные методы взлома и их очень сложно остановить. Кроме того, как показывает статистика преступлений, наиболее разрушительные атаки

---

[1] Джон Дрейпер стал одним из первых сотрудников Apple, где разрабатывал программное и аппаратное обеспечение. Является создателем текстового процессора Easy Writer. Сегодня Дрейпер — один из самых известных специалистов в сфере информационной безопасности. — *Примеч. ред.*

часто проводятся затаившими обиду инсайдерами. Следовательно, системы безопасности должны учитывать и этот фактор.

| Взломщик | Цель |
|---|---|
| Студент | Прочитать чужие письма из любопытства |
| Хакер | Проверить на прочность чужую систему безопасности; украсть данные |
| Торговый агент | Притвориться представителем всей Европы, а не только Андорры |
| Корпорация | Разведать стратегические маркетинговые планы конкурента |
| Уволенный сотрудник | Отомстить фирме за увольнение |
| Бухгалтер | Украсть деньги компании |
| Биржевой брокер | Отказаться от обещания, данного клиенту по электронной почте |
| Аферист | Украсть номера кредитных карт для продажи |
| Шпион | Узнать военные или производственные секреты противника |
| Террорист | Украсть секреты производства бактериологического оружия |

**Илл. 8.1.** Типы взломщиков и цели их действий

# 8.1. ОСНОВЫ СЕТЕВОЙ БЕЗОПАСНОСТИ

При решении проблем сетевой безопасности традиционно выделяют три основных параметра безопасности: конфиденциальность, целостность и доступность. **Конфиденциальность** подразумевает недоступность информации для неавторизованных пользователей. Именно это сразу приходит в голову при упоминании сетевой безопасности. **Целостность** означает, что полученная информация является в точности той, которая была отправлена, без изменений со стороны злоумышленников. **Доступность** подразумевает недопущение выхода из строя систем и сервисов вследствие сбоев, перегрузок или специально внесенных ошибок конфигурации. Хороший пример попытки поставить доступность под угрозу — это DoS-атака. Такие атаки часто бьют по особо важным целям: банкам, авиакомпаниям или средним школам во время экзаменов. Кроме классического триумвирата конфиденциальности, целостности и доступности, которые играют доминирующую роль в сфере безопасности, важны и некоторые другие аспекты. В частности, **аутентификация** позволяет определить, с кем именно вы имеете дело, прежде чем предоставить ему доступ к конфиденциальной информации или заключить коммерческую сделку. Наконец, **неотказуемость (nonrepudiation)** связана с подписями. Как доказать, что ваш клиент действительно оформил электронный заказ на десять миллионов винтиков с левосторонней резьбой по 89 центов за штуку, если теперь он утверждает, что цена была 69 центов? А еще он может заявить, что вообще ничего у вас не заказывал, если увидит, что китайские фирмы наводнили рынок винтиками с левосторонней резьбой за 49 центов.

Все эти параметры касаются и традиционных систем, но с некоторыми существенными отличиями. Конфиденциальность и целостность обеспечиваются с помощью заказных писем и хранения документов в несгораемых сейфах. Сегодня ограбить почтовый поезд значительно сложнее, чем во времена Джесси Джеймса[1]. Кроме того, людям обычно несложно на глаз отличить оригинальный бумажный документ от копии. В качестве проверки попробуйте сделать фотокопию настоящего банковского чека. В понедельник попытайтесь обналичить настоящий чек в вашем банке. А во вторник попробуйте сделать то же самое с фотокопией. Проследите за разницей в поведении банковского служащего.

Что касается аутентификации, ее можно провести самыми разными способами — по лицу, голосу или почерку. В случае бумажных документов подтверждение подлинности обеспечивается с помощью подписей на печатных бланках, печатей, рельефных знаков и т. д. Подделка может быть обнаружена специалистами по почерку, бумаге и чернилам. При работе с электронными документами все это недоступно. Очевидно, требуются другие решения.

Прежде чем перейти к обсуждению самих решений, имеет смысл потратить несколько минут и попытаться определить, к какому уровню стека протоколов относится система сетевой безопасности. Скорее всего, какой-то один уровень для нее определить сложно, каждый из них должен внести свой вклад. На физическом уровне с подслушиванием можно бороться за счет размещения передающих кабелей (или, что еще лучше, оптического волокна) в герметичных металлических трубах, наполненных инертным газом под высоким давлением. Любая попытка просверлить трубу приведет к утечке части газа из трубы, давление снизится, и это послужит сигналом тревоги. Этот метод применяется в некоторых военных системах.

На канальном уровне пакеты, передаваемые по двухточечному каналу, могут зашифровываться при передаче в канал и расшифровываться при получении. При этом все детали известны лишь канальному уровню, а более высокие уровни могут даже не догадываться о том, что происходит. Однако такое решение перестает работать, если пакету нужно преодолеть несколько маршрутизаторов, поскольку его придется расшифровывать на каждом из них, что сделает его беззащитным перед атаками внутри маршрутизатора. Кроме того, при таком подходе нельзя обеспечивать безопасность отдельных сеансов (например, заказов в интернет-магазине) и при этом не защищать остальные. Тем не менее этот метод, называемый **канальным шифрованием (link encryption)**, легко может быть введен в любой сети и часто бывает полезен.

На сетевом уровне можно развернуть брандмауэры, которые не позволят вредоносному трафику поступать в сеть или покидать ее. Также на этом уровне можно использовать протокол IPSec, защищающий IP-пакеты путем шифрования пользовательских данных. На транспортном уровне можно использовать сквозное шифрование всего соединения (то есть на всем пути от процесса к процессу). Аутентификация пользователей и неотказуемость обычно обеспечиваются на прикладном уровне, хотя иногда (например, в беспроводных сетях)

[1] Американский преступник XIX века, грабитель банков и поездов. — *Примеч. ред.*

аутентификация может происходить на более низких уровнях. Поскольку проблема безопасности затрагивает все уровни стека сетевых протоколов, в данной книге ей отводится целая глава.

## 8.1.1. Базовые принципы безопасности

Хотя обеспечение безопасности на всех уровнях сетевого стека является очевидной необходимостью, порой очень трудно определить, насколько эффективно и всесторонне решены связанные с ней проблемы. Другими словами, в большинстве случаев сложно *гарантировать* безопасность. Вместо этого мы пытаемся максимально повысить уровень безопасности, последовательно применяя ряд принципов. Классические принципы безопасности были сформулированы еще в 1975 году Джеромом Зальцером (Jerome Saltzer) и Майклом Шредером (Michael Schroeder):

1. **Минимизация количества используемых механизмов**. Его также иногда называют принципом простоты. Сложные системы обычно содержат больше ошибок, чем простые. Кроме того, пользователи могут плохо понимать принцип их действия и использовать их неправильно или небезопасно. Простые системы — хороший выбор. Например, хотя система PGP (см. раздел 8.11) предоставляет мощные возможности защиты электронной почты, она не получила широкого распространения, поскольку многие пользователи находят ее громоздкой. Простота также помогает минимизировать **поверхность атаки (attack surface)** — все точки, в которых злоумышленник может атаковать систему. Если ненадежному пользователю предлагается большой набор функций, каждая из которых реализована с помощью множества строк кода, такая система имеет большую поверхность атаки. И если в определенной функции нет особой необходимости, лучше обойтись без нее.
2. **Отказоустойчивые настройки по умолчанию**. Допустим, вам нужно организовать доступ к ресурсу. Лучше всего задать четкие правила относительно того, в каких случаях пользователь может получить доступ, вместо того чтобы пытаться определить условия отказа в доступе. Иными словами, отсутствие права доступа по умолчанию — более безопасный подход.
3. **Полная опосредованность**. При предоставлении доступа к любому ресурсу необходимо выполнять проверку на наличие полномочий. То есть мы должны каким-то способом определять источник запроса.
4. **Минимизация полномочий**. Согласно этому принципу любая система или подсистема должна обладать только теми полномочиями, которые являются достаточными для ее работы. Взломав такую систему, злоумышленники смогут получить минимальный объем полномочий.
5. **Разделение задач**. Данный принцип тесно связан с предыдущим: лучше разделить систему на несколько компонентов с минимально возможными полномочиями, вместо того чтобы наделять один компонент всеми правами. Это, опять же, ограничивает возможности злоумышленников в случае взлома ими отдельного компонента.

6. **Минимизация количества общих механизмов**. Этот немного более сложный принцип гласит, что нужно минимизировать число механизмов, совместно используемых несколькими пользователями. Если мы выбираем между реализацией сетевой процедуры в операционной системе (где ее глобальные переменные являются общими) и в библиотеке пространства пользователя (где к ней, по сути, имеет доступ только пользовательский процесс), предпочтительнее второй вариант. Общие данные в операционной системе вполне могут послужить информационным путем между пользователями. Пример такой атаки продемонстрирован в разделе, посвященном перехвату TCP-соединений.
7. **Открытый дизайн**. Данный принцип предельно прост: дизайн системы не должен быть секретным. Это обобщение используемого в криптографии принципа Керкгоффса. В 1883 году голландец Огюст Керкгоффс (Auguste Kerckhoffs) опубликовал две статьи по теме военной криптографии, в которых утверждалось, что система шифрования должна оставаться надежной, даже если все ее устройство, за исключением ключа шифрования, станет общеизвестным. Другими словами, не стоит надеяться на «безопасность через неясность», вместо этого нужно исходить из предположения, что злоумышленник сразу поймет устройство системы и принцип действия алгоритмов шифрования и дешифрования.
8. **Психологическая приемлемость**. Этот принцип носит совершенно нетехнический характер. Правила и механизмы безопасности должны быть легкими в использовании и понимании. Здесь снова уместно вспомнить систему PGP для защиты электронной почты, многие реализации которой не соответствуют данному принципу. Однако понятие приемлемости этим не ограничивается. Помимо удобства в использовании важно позаботиться о том, чтобы смысл применения правил и механизмов был очевиден.

Важную роль в обеспечении безопасности играет **изоляция**. Она гарантирует разделение компонентов (программ, компьютерных систем или даже целых сетей), относящихся к разным защищаемым зонам или обладающих различными правами доступа. Все взаимодействия между разными компонентами происходят с надлежащим контролем полномочий. Изоляция, минимизация полномочий и строгий контроль движения информации между компонентами позволяют создавать системы с высокой степенью разделения.

Помимо технических вопросов, касающихся устройства систем, тема сетевой безопасности затрагивает проблемы, связанные с математическими основами и криптографией. Ярким примером проблемы первого типа является классический **«пинг смерти» (ping of death)**. С его помощью злоумышленники вывели из строя множество хостов по всему интернету. Используя параметры фрагментации IP-пакетов, они создавали эхо-запросы ICMP, превышающие максимально допустимый размер IP-пакета. Поскольку сторона получателя не рассчитывала на прием пакетов столь большого размера, в зарезервированном ею буфере не хватало места, и лишние байты перезаписывали данные, расположенные в памяти после буфера. Конечно, это вызывало программный сбой — переполнение

буфера. Примером криптографической проблемы является использование 40-битного ключа в исходной версии алгоритма WEP-шифрования для сетей Wi-Fi. Обладая достаточной вычислительной мощностью, злоумышленники могли легко подобрать такой ключ методом простого перебора.

## 8.1.2. Базовые принципы проведения атак

Простейший способ рассмотрения вопросов безопасности системы сводится к тому, чтобы поставить себя на место злоумышленника. Поэтому теперь, ознакомившись с базовыми принципами безопасности, давайте рассмотрим **базовые принципы проведения атак**.

С точки зрения злоумышленника, безопасность системы представляет собой ряд задач, которые он должен решить для достижения своих целей. Существует множество способов нарушения конфиденциальности, целостности, доступности и других параметров безопасности. Например, чтобы нарушить конфиденциальность сетевого трафика, злоумышленник может взломать систему и считать данные напрямую, обманом заставить участников коммуникации отправлять данные без шифрования и перехватить их либо, в более амбициозном сценарии, взломать шифр. Каждый из этих способов используется на практике и включает в себя несколько шагов. Мы поговорим об этом подробнее в разделе 8.2, а пока лишь кратко перечислим эти шаги и возможные способы их реализации.

1. **Разведка (reconnaissance)**. Александр Грэм Белл (Alexander Graham Bell)[1] как-то сказал: «Подготовка — ключ к успеху». Проведение атак в компьютерных сетях здесь не является исключением. Первое, что должен сделать злоумышленник, — это собрать как можно больше сведений об объекте атаки. Если вы планируете атаковать кого-нибудь, используя спам или социальную инженерию, рекомендуем вам потратить некоторое время на ознакомление с профилями пользователей, у которых вы хотите выманить информацию, и даже изучить содержимое их мусорных корзин. Однако в этой главе мы ограничимся лишь техническими аспектами атак и соответствующих защитных мер. В сетевой безопасности этап разведки сводится к сбору информации, полезной для злоумышленника. С какими устройствами мы взаимодействуем? Какие протоколы при этом задействованы? Какова топология сети? На каких компьютерах работают те или иные службы? И так далее... Мы обсудим разведку в разделе 8.2.1.
2. **Прослушивание (sniffing) и перехват (snooping)**. Важный этап многих сетевых атак — перехват сетевых пакетов. Очевидно, что возможность перехвата сетевого трафика полезна для злоумышленника, если конфиденциальная информация передается в незашифрованном виде. Однако даже из зашифрованного трафика часто можно извлечь полезные данные — MAC-адреса

[1] Американский и канадский ученый, один из основоположников телефонии, основатель компании AT&T. — *Примеч. ред.*

участников коммуникации, информацию о том, кто с кем и когда связывался, и т. д. Кроме того, злоумышленник может перехватывать такой трафик с целью взломать шифр. Если злоумышленник имеет доступ к сетевому трафику других пользователей и может его «прослушивать», это говорит о недостаточном соблюдении по крайней мере двух принципов безопасности — минимизации полномочий и полной опосредованности. Прослушивание трафика не представляет затруднений в широковещательных сетях (например, сети Wi-Fi), но как перехватить трафик, если вы даже не можете подключиться к каналу, по которому он передается? Тема прослушивания будет рассмотрена подробнее в разделе 8.2.2.

3. **Подмена данных, спуфинг (spoofing)**. Еще одним базовым инструментом злоумышленников является выдача себя за кого-то другого. При этом подложный сетевой трафик имитирует трафик другого компьютера. Например, мы можем легко заменить адрес отправителя во фрейме Ethernet или IP-пакете для обхода защиты или проведения DoS-атаки, поскольку эти протоколы очень просты. А как насчет таких сложных протоколов, как TCP? Ведь при отправке TCP-сегмента `SYN` для установления соединения с сервером с использованием подложного IP-адреса сервер ответит на этот IP-адрес сегментом `SYN/ACK` (второй этап установки соединения), и если злоумышленник не находится в том же сегменте сети, то он не увидит этот ответ. Это значит, что он не узнает, какой порядковый номер использует сервер, и не сможет установить соединение. Подмена данных нарушает принцип полной опосредованности: не имея возможности определить источник запроса, мы не можем нормально осуществить для него посредничество. В разделе 8.2.3 мы поговорим о подмене данных более подробно.
4. **Нарушение работы (disruption)**. Третий компонент из тройки базовых параметров безопасности, доступность, приобретает все большую важность для злоумышленников, учитывая то, насколько разрушительными могут быть атаки типа **«отказ в обслуживании» (Denial of Service, DoS)**. Более того, в ответ на появление новых средств защиты эти атаки становятся все более изощренными. Можно утверждать, что DoS-атака происходит из-за нарушения принципа минимизации количества общих механизмов (то есть недостаточной изоляции компонентов). В разделе 8.2.4 мы подробно поговорим об эволюции этого вида атак.

Используя различные комбинации этих базовых компонентов, злоумышленники могут проводить целый ряд атак. Например, проведя разведку и прослушивание, они могут узнать адрес компьютера потенциальной жертвы и выяснить, что этот пользователь настолько доверяет серверу, что автоматически принимает от него любой запрос. С помощью DoS-атаки (направленной на нарушение работы) злоумышленники могут вывести из строя настоящий сервер, чтобы он больше не отвечал жертве, а затем отправить подложные запросы, имитирующие запросы сервера. Именно так была осуществлена одна из самых известных атак в истории интернета, которой подвергся Суперкомпьютерный центр Сан-Диего. Чуть позже мы обсудим эту атаку подробнее.

### 8.1.3. Методы борьбы с угрозами

Теперь, ознакомившись с методами злоумышленников, давайте обсудим, как с ними можно бороться. Поскольку большинство атак осуществляется через сеть, сообщество специалистов по безопасности быстро пришло к мысли о том, что сеть также может быть подходящим местом для отслеживания атак. В разделе 8.3 мы подробно поговорим о брандмауэрах, системах обнаружения вторжений и других подобных средствах защиты.

Разделы 8.2 и 8.3 посвящены системным вопросам относительно попыток злоумышленников получить конфиденциальную информацию, а в разделах 8.4–8.9 мы перейдем к более формальным аспектам сетевой безопасности и поговорим о **криптографии** и **аутентификации**. В компьютерных системах реализуется ряд криптографических примитивов, основанных на математической теории. Поэтому даже если сетевой трафик попадет в руки взломщиков, это не приведет к каким-либо тяжелым последствиям. К примеру, злоумышленники все равно не смогут нарушить конфиденциальность, модифицировать контент или успешно воспроизвести передаваемые по сети сообщения. Тема криптографии достаточно обширна, поскольку существуют различные типы примитивов самого разного назначения (для подтверждения подлинности, для шифрования с использованием открытых или симметричных ключей и т. д.). При этом у каждого типа, как правило, есть несколько реализаций. В разделе 8.4 представлены ключевые концепции криптографии, а в разделах 8.5 и 8.6 рассмотрены методы шифрования с симметричным и открытым ключами соответственно. Раздел 8.7 посвящен цифровым подписям, а раздел 8.8 — управлению ключами.

В разделе 8.9 поднимается важная проблема обеспечения безопасной аутентификации. Аутентификация призвана полностью устранить возможность подмены данных, позволяя процессу убедиться в том, что другой участник коммуникации является именно тем, за кого себя выдает. По мере того как безопасность приобретала все большую важность, специалисты разрабатывали различные протоколы аутентификации. Как мы увидим далее, такие протоколы обычно основаны на криптографии.

От аутентификации мы перейдем к конкретным примерам решений (как правило, криптографических) по обеспечению безопасности. В разделе 8.10 мы рассмотрим такие средства защиты сетевой коммуникации, как IPsec и VPN, и поговорим об обеспечении безопасности в беспроводных сетях. В разделе 8.11 обсудим средства защиты электронной почты, включая PGP и S/MIME. В разделе 8.12 речь пойдет о веб-безопасности в более широком смысле, и в частности о защищенном протоколе DNS (DNSSEC), исполняемых в браузерах скриптах и протоколе SSL. Мы убедимся, что в этих технологиях используются многие идеи, изложенные в предыдущих разделах.

Наконец, раздел 8.13 посвящен социальным аспектам. Каким образом сетевая безопасность затрагивает такие важные вопросы, как неприкосновенность частной жизни и свобода слова? А что насчет авторских прав и защиты интеллектуальной собственности? Поскольку тема безопасности крайне важна, мы должны тщательно рассмотреть все эти вопросы.

Перед тем как мы перейдем к подробному рассмотрению этой темы, следует еще раз напомнить, что в целом обеспечение безопасности является весьма обширной областью изучения. В этой главе мы сфокусируемся лишь на безопасности сетей и коммуникации, оставив без внимания вопросы, связанные с аппаратным обеспечением, операционными системами, приложениями или пользователями. Это значит, что мы не будем подробно рассматривать программные ошибки и касаться аутентификации пользователей с использованием биометрических данных, защиты с помощью паролей, атак переполнения буфера, применения троянских программ и вирусов, подмены входа в систему и изоляции процессов. Эти темы детально обсуждаются в главе 9 книги «Modern Operating Systems»[1] Таненбаума и Боса (Tanenbaum and Bos, 2015). Все желающие узнать о вопросах обеспечения безопасности на уровне системы могут обратиться к этой книге. Итак, начнем.

## 8.2. ОСНОВНЫЕ КОМПОНЕНТЫ АТАКИ

Прежде всего мы рассмотрим базовые компоненты, из которых состоит сетевая атака. Практически все они проводятся по определенному «рецепту», где эти «ингредиенты» ловко смешиваются в разных вариациях.

### 8.2.1. Разведка

Представьте, что вы — хакер, и в одно прекрасное утро вы решили взломать компьютерную систему организации X. С чего начать? Вы мало что знаете об этой организации и физически находитесь далеко от нее, что исключает изучение содержимого мусорных корзин или подсматривание через плечо. Конечно, вы всегда можете применить методы **социальной инженерии** и попытаться выманить конфиденциальную информацию у сотрудников, отправляя им электронные письма (спам) и общаясь с ними по телефону или в социальных сетях, но нас больше интересуют технические аспекты, связанные с компьютерными сетями. Например, сможете ли вы узнать, сколько компьютеров использует организация, как они подключены к сети и какие службы на них запущены?

Для начала предположим, что злоумышленник располагает IP-адресами нескольких компьютеров компании, включая веб-серверы, серверы имен, серверы авторизации и прочие устройства, взаимодействующие с внешним миром. Сначала злоумышленник должен изучить сервер. Какие TCP- и UDP-порты открыты? Чтобы это выяснить, нужно просто попытаться установить TCP-соединение, используя все возможные номера портов. Установление соединения показывает, что порт прослушивается соответствующей службой. Например, если компьютер реагирует на использование порта 25, это говорит о наличии SMTP-сервера, если удается установить соединение с портом 80, значит, имеется веб-сервер, и т. д. Это справедливо и для UDP-портов. К примеру, если исследуемый компьютер

[1] Таненбаум Э., Бос Х. «Современные операционные системы». СПб., издательство «Питер».

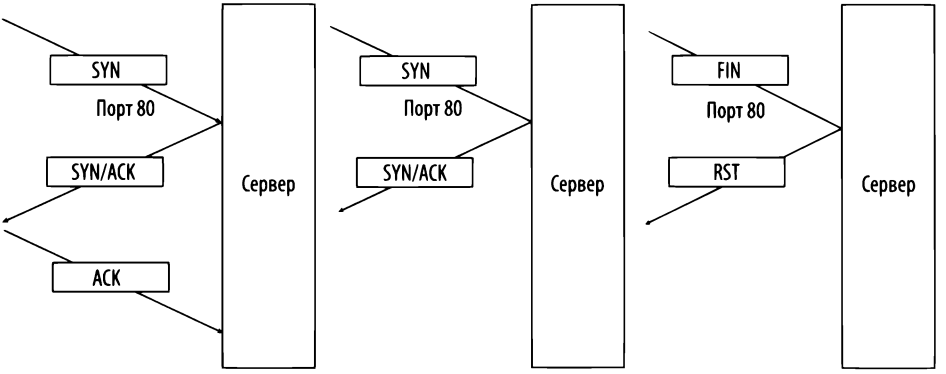
реагирует на использование UDP-порта 53, мы понимаем, что на нем запущена служба DNS, поскольку этот порт зарезервирован для нее.

### Сканирование портов

Исследование компьютера на предмет активности портов называется **сканированием портов (port scanning)** и может оказаться довольно сложным. Описанный выше метод **сканирования подключением (connect scan)** или **открытого сканирования (open scan)**, при котором злоумышленник пытается установить полное TCP-соединение с нужным компьютером, напротив, предельно прост. Однако, будучи вполне эффективным, он имеет крупный недостаток — он слишком заметен для службы безопасности. Многие серверы регистрируют все успешные TCP-соединения в журнале, и попадание в такой журнал на этапе разведки отнюдь не входит в планы злоумышленника. Чтобы избежать этого, он может предпринимать заведомо неудачные попытки соединения, используя метод **полуоткрытого сканирования (half-open scan)**. При таком сканировании злоумышленник лишь делает вид, что хочет установить соединение: он отправляет TCP-пакеты с флагом `SYN`, содержащим интересующие его номера портов, и в ответ получает от сервера соответствующие сегменты `SYN/ACK` для открытых портов, но после этого так и не завершает процесс «тройного рукопожатия». Большинство серверов не регистрирует в журнале неудачные попытки подключения.

Если метод полуоткрытого сканирования эффективнее, зачем мы вообще обсуждаем метод сканирования подключением? Дело в том, что применить полуоткрытое сканирование может лишь продвинутый хакер. Установить полное соединение с TCP-портом, как правило, можно почти с любого компьютера с помощью простых инструментов (например, утилиты telnet), часто доступных для непривилегированных пользователей. Но в случае полуоткрытого сканирования злоумышленник должен четко определить, какие пакеты нужно передавать, а какие нет. В большинстве систем нет стандартных инструментов, которые позволяют обычным пользователям сделать это, поэтому выполнить полуоткрытое сканирование могут только пользователи с правами администратора.

Как при сканировании подключением, так и при полуоткрытом сканировании предполагается, что можно инициировать TCP-соединение с любого компьютера, расположенного за пределами сети, в которой находится цель атаки. Но на практике брандмауэр может не позволить установить соединение с компьютера злоумышленника. В частности, он может блокировать все сегменты `SYN`. В этом случае хакеру придется прибегнуть к более экзотическим методам сканирования. Например, метод **FIN-сканирования (FIN scan)** сводится к тому, чтобы вместо сегмента `SYN` передавать TCP-сегмент `FIN`, который обычно применяется для закрытия соединения. На первый взгляд это не имеет смысла, ведь у нас нет ни одного соединения. Тем не менее ответ на пакет `FIN` может отличаться в зависимости от того, открыт ли порт (и прослушивается ли соответствующей службой) или закрыт. Так, многие реализации TCP отправляют TCP-пакет `RST`, если порт закрыт, и молчат, если он открыт. Три описанных базовых метода сканирования показаны на илл. 8.2.




(а) Сканирование подключением: установление соединения означает, что порт открыт

(б) Полуоткрытое сканирование: ответ с SYN/ACK означает, что порт открыт

(в) FIN-сканирование: ответ с RST означает, что порт закрыт

**Илл. 8.2.** Базовые методы сканирования портов. (а) Сканирование подключением. (б) Полуоткрытое сканирование. (в) FIN-сканирование

Сейчас вы, вероятно, задаетесь вопросом: «Если мы можем сканировать компьютер с помощью флагов `SYN` и `FIN`, можно ли применять для этого и другие флаги?» Вы абсолютно правы. Можно использовать любую конфигурацию флагов, если она приводит к разным ответам для открытых и закрытых портов. Еще один широко известный вариант сводится к одновременной установке сразу нескольких флагов: `FIN`, `PSH` и `URG`. Этот метод называется **Xmas-сканированием (Xmas scan)**[1], потому что пакет при этом сверкает, как рождественская елка.

На илл. 8.2 (а) установление соединения означает, что порт открыт. На илл. 8.2 (б) об этом говорит ответ `SYN/ACK`. Наконец, на илл. 8.2 (в) ответ `RST` сообщает, что порт закрыт.

Поиск открытых портов является лишь первым шагом. После этого нужно выяснить, какой именно сервер работает на конкретном порте, с каким программным обеспечением (и какой версии) и под какой операционной системой. Допустим, мы узнали, что открыт порт 8080. По всей вероятности, его использует веб-сервер, но у нас нет в этом уверенности. Даже если это веб-сервер, то какой именно — Nginx, Lighttpd или Apache? Хакеру могут быть известны только, к примеру, уязвимости сервера Apache в версии 2.4.37 для ОС Windows. Поэтому выяснение всех этих деталей, то есть **снятие цифрового отпечатка (fingerprinting)**, играет очень важную роль. Так же как и при сканировании портов, получить эти сведения можно благодаря различиям (зачастую незначительным) в ответной реакции для разных серверов и операционных систем. Если вам это кажется очень сложным, не пугайтесь. Как и в случае многих других сложных вещей в сфере компьютерных сетей, уже нашлась добрая душа, объединившая для вас все эти методы сканирования и снятия цифровых отпечатков в удобные и многофункциональные программы, например **netmap** и **zmap**.

[1] От англ. «Xmas» — «Christmas», Рождество. — *Примеч. пер.*

### Утилита traceroute

Итак, хакер может выяснить, какие службы активны на одном компьютере, но что насчет остальных устройств в сети? Отталкиваясь от первого IP-адреса, злоумышленник может «прощупать» окружение этого компьютера и посмотреть, какие еще компьютеры доступны. Например, если первый компьютер использует IP-адрес 130.37.193.191, можно также проверить адреса 130.37.193.192, 130.37.193.193 и все другие возможные адреса в этой локальной сети. Злоумышленник также может воспользоваться программой **traceroute**, чтобы выяснить, по какому пути данные идут к исходному IP-адресу. Утилита traceroute последовательно отправляет целевому компьютеру небольшие партии UDP-пакетов с временем жизни (TTL), равным единице, двум, трем и т. д. Первый маршрутизатор уменьшает значение TTL и сразу же удаляет первую партию пакетов (потому что TTL становится равным нулю), после чего отправляет назад ICMP-сообщение об ошибке, в котором говорится, что время жизни пакетов истекло. Второй маршрутизатор делает то же самое для следующей партии пакетов, третий — для третьей, и т. д., пока, наконец, определенные UDP-пакеты не дойдут до нужного компьютера. Собрав воедино эти пакеты ICMP-сообщений и соответствующие IP-адреса источников, утилита traceroute может определить общий маршрут. На основе полученных результатов хакер может попытаться выявить дополнительные мишени, проверив диапазоны адресов маршрутизаторов, расположенных поблизости от целевого компьютера, и тем самым получив общее представление о топологии сети.

## 8.2.2. Прослушивание и перехват (и немного подмены данных)

Многие сетевые атаки начинаются с перехвата сетевого трафика. Использование этого компонента предполагает, что хакер имеет доступ к сети жертвы. Он может, например, принести ноутбук в зону действия ее Wi-Fi-сети или получить доступ к ПК в проводной сети. Прослушивание в таких широковещательных сетях, как сети Wi-Fi или первая версия сети Ethernet, не представляет большого труда: нужно просто подключиться к каналу в подходящем для этого месте, а затем прослушивать поступающий поток битов. Для этого злоумышленники переводят свои сетевые интерфейсы в **неизбирательный режим (promiscuous mode)**, что позволяет принимать все идущие через канал пакеты, включая предназначенные для других хостов, и перехватывать трафик с помощью таких инструментов, как **tcpdump** и **Wireshark**.

### Прослушивание в коммутируемых сетях

Однако в сетях других типов дело часто обстоит далеко не так гладко — взять хотя бы сеть Ethernet. В отличие от первых реализаций, современная версия Ethernet уже не является сетью с разделяемой средой. Связь носит коммутируемый характер, и даже в случае подключения злоумышленников к какому-то сегменту сети они никогда не получат Ethernet-фреймы, предназначенные

для других хостов этого сегмента. В частности, как вы помните, коммутаторы Ethernet способны самообучаться и быстро создают таблицу пересылки. Принцип самообучения прост и эффективен: при поступлении фрейма Ethernet от хоста *A* на порт 1 коммутатор фиксирует, что трафик для хоста *A* должен направляться именно на этот порт. Теперь он знает, что весь трафик, содержащий MAC-адрес хоста *A* в поле назначения заголовка Ethernet, должен пересылаться на порт 1. Аналогично он будет передавать трафик для хоста *B* на порт 2, и т. д. После составления полной таблицы пересылки коммутатор уже не будет передавать трафик, явно адресованный хосту *B*, на какой-то другой порт помимо порта 2. А чтобы прослушать трафик, злоумышленники должны каким-то образом этого добиться.

Преодолеть проблему коммутации можно несколькими способами. Строго говоря, все они подразумевают подмену данных, но мы все же рассмотрим их здесь, поскольку их единственная цель состоит в прослушивании трафика.

Первый метод, **клонирование MAC-адреса (MAC cloning)**, сводится к тому, чтобы продублировать MAC-адрес хоста, трафик которого вам нужно прослушать. Если вы сообщите, что обладаете нужным MAC-адресом (отправляя фреймы Ethernet, в которых он указан), то коммутатор исправно запишет это в свою таблицу, после чего будет отсылать на ваш компьютер весь трафик, адресованный жертве. Конечно, сначала нужно узнать этот адрес. Его можно извлечь из ARP-запросов, которые отправляет интересующий вас компьютер (поскольку они рассылаются всем хостам в сегменте сети). Другая сложность заключается в том, что сопоставление с вашим компьютером будет удалено коммутатором, как только настоящий владелец MAC-адреса возобновит коммуникацию. Таким образом, **отравление таблицы коммутатора (switch table poisoning)** придется повторять постоянно.

Существует альтернативный, но схожий подход, при котором используется одна особенность: таблица коммутатора имеет ограниченный размер. Злоумышленник «заваливает» коммутатор фреймами Ethernet с поддельными адресами отправителей. Не зная, что они фальшивые, коммутатор фиксирует их, пока таблица не заполнится и для размещения новых записей не потребуется удаление старых. Поскольку теперь у коммутатора нет записи для целевого хоста, предназначенный для этого хоста трафик начинает рассылаться по всей сети. **Лавинная рассылка MAC-адресов (MAC flooding)** снова превращает сеть Ethernet в широковещательную среду, какой она была в далеком 1979 году.

Вместо того чтобы вводить коммутатор в заблуждение, злоумышленник может напрямую атаковать хост с помощью метода **подмены ARP (ARP spoofing)** или **отравления ARP (ARP poisoning)**. Как упоминалось в главе 5, протокол ARP помогает компьютеру определить, какой MAC-адрес соответствует тому или иному IP-адресу. Для этого реализация ARP на компьютере ведет таблицу сопоставления IP-адресов с MAC-адресами для всех хостов, которые связывались с данным компьютером, — **таблицу ARP (ARP table)**. Время жизни каждой записи в ней обычно составляет несколько десятков минут. По истечении этого срока MAC-адрес удаленного абонента забывается, исходя из предположения, что абоненты больше не обмениваются данными (в этом случае значение TTL

сбрасывается), и для последующих коммуникаций сначала нужно выполнить ARP-поиск. ARP-поиск представляет собой обычную широковещательную рассылку сообщения примерно следующего содержания: «Мне нужен MAC-адрес хоста с IP-адресом 192.168.2.24. Если это ваш адрес, пожалуйста, сообщите об этом». Этот поисковый запрос включает MAC-адрес отправителя, чтобы хост 192.168.2.24 знал, кому отвечать, и его IP-адрес, чтобы хост 192.168.2.24 добавил в свою ARP-таблицу сопоставление IP- и MAC-адресов отправителя.

Каждый раз, когда хакер видит ARP-запрос к хосту 192.168.2.24, он может попытаться первым предоставить источнику запроса свой собственный MAC-адрес. В этом случае весь трафик, предназначенный для хоста 192.168.2.24, будет направляться на компьютер злоумышленника. На самом деле, поскольку реализации ARP обычно просты и не отслеживают состояние, взломщик зачастую может отправлять ARP-ответы, даже не дожидаясь появления запросов: реализация ARP беспрекословно примет все эти ответы и сохранит в ARP-таблице соответствующие сопоставления.

Применив этот трюк к обоим участникам коммуникации, злоумышленник сможет принимать весь трафик, которым они обмениваются. Обеспечив пересылку этих фреймов на правильный MAC-адрес, он тем самым установит скрытый шлюз атаки **«человек посередине» (Man-in-the-Middle, MITM)**, способный перехватывать весь трафик между двумя хостами.

## 8.2.3. Подмена данных (помимо ARP)

Как правило, подмена данных подразумевает передачу байтов по сети с использованием поддельного адреса отправителя. Злоумышленники могут подделывать не только ARP-пакеты, но и любой другой сетевой трафик. В качестве примера можно вспомнить SMTP — удобный текстовый протокол, который сегодня повсеместно используется для отправки электронной почты. В нем применяется заголовок `Mail From:` для указания адреса отправителя письма, но по умолчанию указанный адрес не проверяется. То есть при желании вы можете поместить в этот заголовок что угодно, и все ответы будут отправляться на этот адрес. При этом получатель даже не увидит содержимого `Mail From:` — вместо него почтовый клиент отобразит содержимое отдельного заголовка `From:`. Но и это поле SMTP не проверяет, что позволяет подменить его содержимое. Например, вы можете отправить своим однокурсникам письмо с сообщением о том, что они провалили экзамен, и в качестве отправителя указать преподавателя. Если вы также укажете свой электронный адрес в заголовке `Mail From:`, то все ответы паникующих студентов окажутся в вашем почтовом ящике. Вот это веселье! Однако когда злоумышленники подделывают адрес для рассылки фишинговых писем, якобы приходящих из надежного источника, это уже далеко не так невинно. Например, письмо от «вашего доктора» может содержать ссылку на срочную информацию о результатах медосмотра, щелкнув по которой вы попадете на сайт, сообщающий, что ваше здоровье в полном порядке (без упоминания о том, что на ваш компьютер загружен вирус). А письмо от «вашего банка» может нести угрозу вашим финансам.

Хотя подмена ARP осуществляется на канальном уровне, а подмена SMTP — на прикладном, в принципе, методы подмены могут применяться на любом уровне стека протоколов. Иногда это достаточно просто сделать. Например, любой, кто умеет создавать пользовательские пакеты, может легко подделать фрейм Ethernet, IP-дейтаграмму или UDP-пакет. Достаточно лишь изменить адрес отправителя, и эти протоколы никак не смогут выявить обман. Многие другие протоколы гораздо труднее поддаются обману. Например, в случае TCP-соединений конечные точки сохраняют такие данные о состоянии, как порядковые номера и номера подтверждения, что сильно усложняет подмену данных. Если злоумышленнику не удастся получить порядковые номера путем прослушивания или угадать их, то получатель отбросит поддельные TCP-сегменты как выходящие за рамки установленного окна. Как мы увидим далее, помимо этого имеется ряд других существенных сложностей.

Даже простые протоколы позволяют злоумышленникам наносить весьма значительный ущерб. Вскоре мы узнаем, насколько разрушительными могут быть DoS-атаки, вызванные подменой UDP-пакетов. Но сначала мы обсудим, как хакеры с помощью подмены UDP-дейтаграмм службы DNS перехватывают информацию, отправляемую клиентами на сервер.

### *Спуфинг DNS*

Поскольку служба DNS использует протокол UDP для пересылки своих запросов и ответов, подмена данных здесь не представляет большого труда. Например, как и в случае подмены ARP, можно подождать, пока клиент не отправит поисковый запрос в отношении домена trusted-services.com, а затем, опередив настоящую службу DNS, предоставить клиенту поддельный ответ. В нем будет указано, что для этого домена следует использовать принадлежащий вам IP-адрес. Это легко сделать, если вы можете прослушать поступающий от клиента трафик (и таким образом увидеть содержимое запроса на DNS-поиск, на который нужно ответить). Но что, если вы не можете увидеть этот запрос? В конце концов, если вы уже и так прослушиваете линию, то перехват этого трафика путем спуфинга DNS не имеет особого смысла. Кроме того, что, если вам нужно перехватить трафик не одного, а сразу нескольких пользователей?

Если злоумышленник использует тот же локальный сервер имен, что и жертва, он просто отправляет собственный запрос, скажем, для домена trusted-services.com. После этого сервер начинает искать этот IP-адрес и пытается связаться со следующим сервером имен в процессе поиска. На этот запрос локального сервера имен злоумышленник немедленно выдает поддельный ответ, якобы от следующего сервера. В результате локальный сервер имен сохраняет в своем кэше сфальсифицированное сопоставление и предоставляет его, когда жертва (или кто-то еще), наконец, запрашивает данные по домену trusted-services.com. Обратите внимание, что этот метод может сработать и в том случае, если взломщик не использует тот же локальный сервер имен, что и цель атаки. Для этого нужно как-то склонить жертву к отправке поискового запроса с доменным именем, предоставленным злоумышленником. Например, он может отправить письмо со ссылкой, при активации которой браузер выполнит поиск по нужному имени.

После отравления данных сопоставления для домена trusted-services.com все последующие запросы по нему будут возвращать сфальсифицированные данные.

Проницательный читатель может возразить, что это далеко не так просто. Ведь у каждого DNS-запроса есть 16-битный идентификатор, и ответ принимается, только если он содержит такой же идентификатор. Но если хакер не видит запрос, то ему придется подбирать этот идентификатор наугад. Вероятность успеха подбора для отдельного ответа составляет 1 : 65 536. В среднем хакеру придется отправить десятки тысяч DNS-ответов за очень короткий промежуток времени, чтобы сфальсифицировать лишь одно сопоставление на локальном сервере имен, не имея при этом никакой обратной связи. Это нельзя назвать простой задачей.

### *Атака «дней рождения»*

Существует и более простой метод, который называют **атакой «дней рождения» (birthday attack)**, или **парадоксом «дней рождения» (birthday paradox)**; хотя, строго говоря, это вовсе не парадокс. Идея этого метода заимствована из задачи, часто используемой профессорами математики при изложении теории вероятности. Вопрос: сколько студентов должно быть в классе, чтобы вероятность того, что двое из них родились в один день, превысила 50 %? На первый взгляд кажется, что ответом будет число, значительно превышающее 100. Однако согласно теории вероятности это число на самом деле равно 23. Для 23 людей вероятность того, что *ни у кого* из них не совпадает день рождения, равна:

$$\frac{365}{365}\times\frac{364}{365}\times\frac{363}{365}\times\cdots\times\frac{343}{365}=0{,}497203.$$

То есть вероятность того, что два студента отмечают день рождения в один день, превышает 50 %.

Вообще, если имеется сопоставление $n$ входных значений (людей, идентификаторов и т. д.) и $k$ возможных выходных значений (дней рождения, идентификаторов и т. д.), то мы имеем $n(n-1)/2$ входных пар. При $n(n-1)/2 > k$ вероятность того, что будет получено хотя бы одно совпадение, довольно значительна. Таким образом, оно вероятно уже примерно при $n > \sqrt{2k}$. Ключевой момент состоит в том, что мы не ищем совпадение с днем рождения одного конкретного человека, а сравниваем даты рождения всех студентов и считаем успехом любое найденное соответствие.

Учитывая вышесказанное, хакеры сначала отправляют несколько сотен DNS-запросов в отношении того домена, для которого им нужно фальсифицировать сопоставление. Локальный сервер имен пытается поочередно разрешить каждый из этих запросов путем отправки запроса серверу имен следующего уровня. Вероятно, это не слишком разумно — какой смысл многократно запрашивать данные по одному и тому же домену? Но никто и не утверждает, что серверы имен отличаются большой сообразительностью, и именно так, к примеру, работает популярный сервер имен BIND. Как бы там ни было, сразу после отправки запросов злоумышленники также отсылают сотни поддельных ответов на поисковые запросы, якобы от сервера имен следующего уровня. Эти ответы содержат разные варианты идентификатора запроса. При этом локальный сервер имен

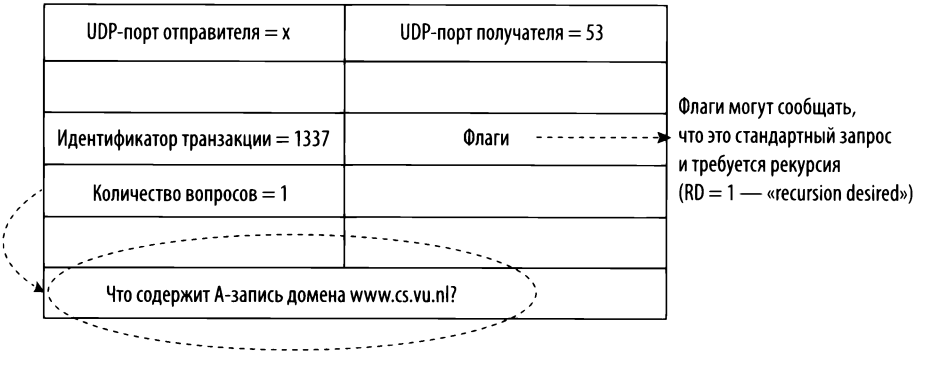
косвенно производит сравнение по принципу «многие ко многим», поскольку будет принят любой ответ, идентификатор которого совпадает с идентификатором запроса от локального сервера имен. Обратите внимание, что, как и в случае с днями рождения студентов, сервер имен сравнивает все запросы от локального сервера имен со всеми поддельными ответами.

Произведя отравление локального сервера имен какого-нибудь веб-сайта, взломщики могут, к примеру, получить доступ к трафику, который передают на этот сайт все клиенты сервера имен. Настроив свои соединения на получение данных от сайта и обеспечив ретрансляцию всего трафика между клиентами и сервером, они реализуют атаку типа «человек посередине».

### *Атака Каминского*

Ситуация может принять еще более серьезный оборот, если злоумышленники не будут ограничиваться отдельным веб-сайтом и решат отравить данные сопоставления для целой зоны. Такое нарушение получило известность как DNS-атака Дэна Каминского; в свое время она повергла в шок многих специалистов по информационной безопасности и сетевых администраторов по всему миру. Чтобы понять, чем они были так напуганы, необходимо детально разобраться в том, как происходит DNS-поиск.

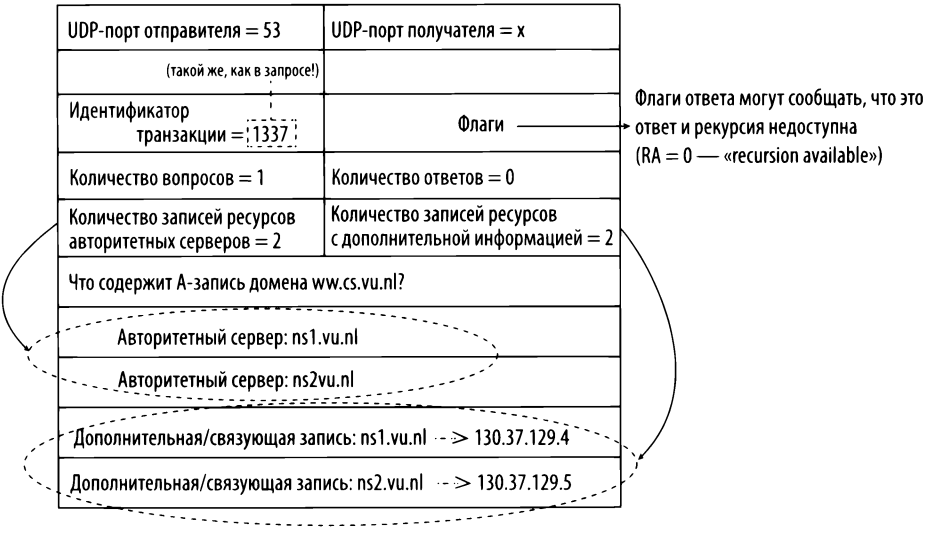
В качестве примера рассмотрим процесс обработки DNS-запроса на поиск IP-адреса домена www.cs.vu.nl. После получения этого запроса локальный сервер имен, в свою очередь, отправляет запрос либо на корневой сервер имен, либо на сервер имен домена верхнего уровня (ДВУ) по отношению к домену .nl. Второй сценарий происходит чаще по той причине, что IP-адрес сервера имен ДВУ обычно уже имеется в кэше локального сервера имен. На илл. 8.3 показан такой запрос локального сервера имен (запрашивающий *A*-запись домена) при выполнении рекурсивного поиска с идентификатором 1337.




**Илл. 8.3.** DNS-запрос в отношении домена www.cs.vu.nl

Серверу имен ДВУ не известно точное сопоставление, однако он знает имена DNS-серверов Университета Врийе и возвращает их в ответе, поскольку не производит рекурсивный поиск. На илл. 8.4 показан ответ, несколько

полей которого заслуживают особого внимания. Во-первых, мы сразу видим, что флаги напрямую сообщают о нежелании сервера выполнять рекурсивный поиск, и потому дальнейший поиск будет итеративным. Во-вторых, в ответе содержится идентификатор 1337, который был использован в поисковом запросе. В-третьих, ответ содержит символьные наименования серверов имен университета ns1.vu.nl и ns2.vu.nl в виде записей NS. Эти данные являются авторитетными и, в принципе, достаточными для того, чтобы локальный сервер имен мог завершить обработку запроса. Ему нужно лишь свериться с *A*-записью одного из этих серверов имен, связаться с ним и запросить IP-адрес домена www.cs.vu.nl. Но чтобы это сделать, он должен еще раз связаться с тем же сервером имен ДВУ, на этот раз чтобы запросить IP-адрес сервера имен университета. Поскольку это вводит дополнительную задержку в пути туда и обратно, данный подход не слишком эффективен. Чтобы исключить необходимость в этой дополнительной операции поиска, сервер имен ДВУ услужливо предоставляет в своем ответе IP-адреса двух серверов имен университета в виде дополнительных записей с коротким временем жизни. Эти **связующие DNS-записи (DNS glue records)** как раз и являются ключевым элементом атаки Каминского.




**Илл. 8.4.** DNS-ответ, отправляемый сервером имен ДВУ

Злоумышленники действуют так. Сначала они отправляют запросы на поиск данных о несуществующем поддомене в домене университета, таком как ohdeardankaminsky.vu.nl. Поскольку этого поддомена нет, ни один сервер имен не может предоставить данные сопоставления из своего кэша. Вместо этого локальный сервер должен связаться с сервером имен ДВУ. Сразу после отправки запросов взломщики отсылают множество поддельных ответов, якобы от сервера имен ДВУ, как и в случае обычного спуфинга DNS. Однако

на этот раз в ответе говорится, что сервер имен ДВУ не располагает нужными данными (то есть он не предоставляет *A*-запись), не выполняет рекурсивный поиск и рекомендует локальному серверу имен завершить поиск путем обращения к одному из серверов имен университета. Он даже может предоставить реальные имена этих серверов. Единственное, что при этом фальсифицируется, это связующие записи: вместо них злоумышленники дают подконтрольные им IP-адреса. В результате при запросе данных о любом поддомене в домене .vu.nl производится обращение к серверу имен злоумышленников, которые могут предоставить в ответ какой угодно IP-адрес. Таким образом, они способны провести атаку типа «человек посередине» против любого сайта в домене университета!

Очень многие (хотя и не все) реализации серверов имен были уязвимы к этой атаке. Очевидно, у интернета возникла проблема. В связи с этим в штаб-квартире компании Microsoft в Редмонде было собрано экстренное совещание. Как позже вспоминал Каминский, оно было окутано столь строгой атмосферой секретности, что «многие из прилетевших в Microsoft людей даже не знали, о какой программной ошибке пойдет речь».

Каким же образом все эти умные люди справились с проблемой? По правде говоря, решить ее полностью им не удалось, хотя они и сумели существенно усложнить хакерам задачу. Напомним, что корневая проблема, из-за которой спуфинг DNS возможен, заключается в том, что длина идентификатора запроса составляет лишь 16 бит. Это позволяет получить его либо путем прямого перебора, либо с помощью атаки «дней рождения». Увеличение длины идентификатора запроса существенно снижает вероятность успешного проведения такой атаки. Однако просто изменить формат сообщения протокола DNS затруднительно; кроме того, это нарушило бы работу многих существующих систем.

Было принято решение дополнить длину произвольного идентификатора путем введения элемента случайности и в UDP-порт отправителя (не прибегая к реальному увеличению идентификатора запроса). К примеру, при отправке DNS-запроса на сервер имен ДВУ модифицированный сервер имен будет случайным образом выбирать номер порта из тысяч доступных номеров и использовать этот порт в качестве UDP-порта отправителя. Теперь злоумышленнику нужно подобрать не только идентификатор запроса, но и номер порта, успев сделать это до поступления реального ответа. Кодировка 0x20, о которой мы говорили в главе 7, дополняет идентификатор транзакции еще несколькими битами. При этом учитывается, что DNS-запросы нечувствительны к регистру.

К счастью, протокол **DNSSEC** предоставляет более надежную защиту от спуфинга DNS. Он содержит ряд расширений протокола DNS, призванных обеспечить одновременно и целостность, и аутентификацию источника DNS-данных для клиентов службы DNS. Однако процесс внедрения DNSSEC идет чрезвычайно медленно. Хотя первые работы над ним проводились еще в начале 1990-х годов, а первая спецификация RFC для него была опубликована комитетом IETF в 1997 году, только сейчас DNSSEC начинает находить более-менее широкое применение. Чуть позже мы поговорим об этом подробнее.

### *Подмена TCP*

В TCP произвести подмену данных гораздо сложнее, чем в протоколах, рассмотренных выше. Чтобы создать впечатление, что TCP-сегмент пришел от какого-то компьютера в интернете, злоумышленникам нужно подобрать не только номер порта, но и правильные порядковые номера. Кроме того, при подмене TCP-сегментов чрезвычайно сложно поддерживать нормальное TCP-соединение. Есть две разновидности такой атаки:

1. **Подмена соединения (connection spoofing).** Злоумышленник устанавливает *новое* соединение, выдавая себя за пользователя, использующего другой компьютер.
2. **Захват соединения (connection hijacking).** Злоумышленник внедряет данные в рамках уже существующего соединения между двумя пользователями, выдавая себя за одного из них.

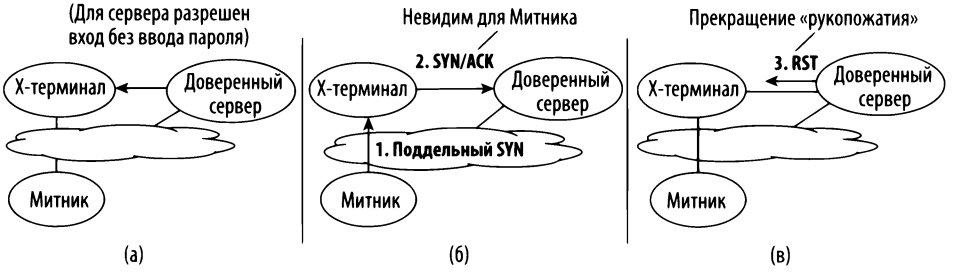
Яркий пример **подмены TCP-соединения (TCP connection spoofing)** — атака на Суперкомпьютерный центр Сан-Диего, проведенная Кевином Митником (Kevin Mitnick) 25 декабря 1994 года. Это одна из самых известных хакерских атак в истории, ставшая темой нескольких книг и фильмов, включая высокобюджетный триллер «Взлом» («Takedown»), снятый по книге системного администратора Суперкомпьютерного центра. (Неудивительно, что он изображен в фильме как очень крутой парень.) Мы обсудим эту историю, поскольку она хорошо демонстрирует, насколько сложно произвести подмену TCP.

К тому моменту, когда Кевин Митник обратил свой взор на Суперкомпьютерный центр Сан-Диего, он уже достаточно давно занимался «хулиганством» в интернете. Надо сказать, что идея проводить атаки в рождественские дни вполне разумна, ведь в праздники пользователей обычно меньше, как и контроля со стороны администраторов. Проведя предварительную разведку, Митник обнаружил, что один из компьютеров центра (X-терминал) доверял другому компьютеру этого центра (серверу).

Эта конфигурация показана на илл. 8.5 (а). Серверу оказывалось безоговорочное доверие, и любой его пользователь мог войти в X-терминал как администратор с помощью удаленной оболочки (*rsh*) без пароля. План Митника состоял в том, чтобы установить TCP-соединение с X-терминалом, выдавая себя за сервер, а затем с его помощью полностью отключить парольную защиту — в те дни это можно было сделать, записав «+ +» в файле `.rhosts`.

Однако сделать это было не так просто. Если бы Митник отправил X-терминалу поддельный запрос на установление TCP-соединения (сегмент `SYN`) с IP-адресом сервера (шаг 1 на илл. 8.5 (б)), то X-терминал отправил бы ответ `SYN/ACK` реальному серверу, и Митник бы этого не увидел (шаг 2 на илл. 8.5 (б)). В результате он бы не узнал изначальный порядковый номер X-терминала (ISN). Это практически случайный номер, который был нужен Митнику для проведения третьего этапа «рукопожатия» TCP (на котором, как мы видели ранее, передается первый сегмент с данными). Что еще хуже, получив сегмент `SYN/ACK`, сервер сразу же ответил бы на это сегментом `RST`, чтобы прекратить

установление соединения (шаг 3 на илл. 8.5 (в)). Поступление SYN/ACK говорит о проблеме, ведь сервер не отправлял перед этим SYN.



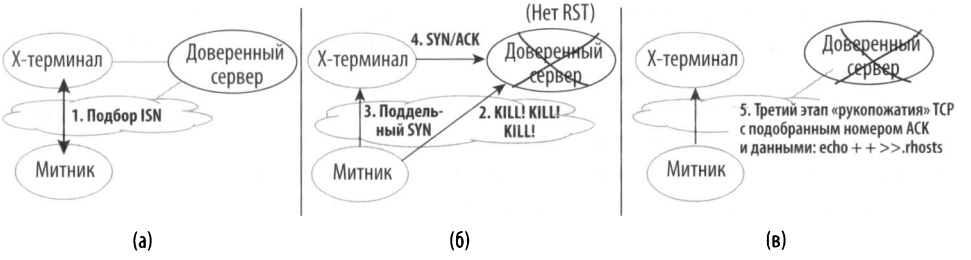

**Илл. 8.5.** Проблемы, с которыми столкнулся Кевин Митник при атаке на Суперкомпьютерный центр Сан-Диего

Тот факт, что Митник не видел сегмент SYN/ACK и поэтому не мог получить ISN, не стало бы проблемой, если бы этот номер был предсказуемым (например, начинался с 0 для каждого нового соединения). Но поскольку ISN выбирался достаточно случайно для каждого соединения, нужно было выяснить, как он генерировался, чтобы *предсказать*, какой номер использует X-терминал в невидимом сегменте SYN/ACK, который он отправит серверу.

Чтобы решить все эти проблемы, Митник провел свою атаку в несколько этапов. Он начал с активного взаимодействия с X-терминалом с использованием неподдельных сообщений SYN (шаг 1 на илл. 8.6 (а)). Хотя эти попытки не привели к установлению TCP-соединения с X-терминалом, они позволили получить некоторую последовательность ISN. К счастью для Кевина, номера выбирались не *таким уж* случайным образом. Понаблюдав за ними некоторое время, он смог выявить закономерность. Теперь он был уверен, что, получив один ISN, он сможет предсказать следующий. Затем он позаботился о том, чтобы доверенный сервер не мог сбросить его попытки подключения. Для этого он запустил DoS-атаку, которая сделала сервер недоступным (шаг 2 на илл. 8.6 (б)). После этого уже можно было приступить к проведению реальной атаки.

После отправки поддельного пакета SYN (шаг 3 на илл. 8.6 (б)) он определил, какой ISN будет использован X-терминалом в отсылаемом на сервер ответе SYN/ACK (шаг 4 на илл. 8.6 (б)). Он указал этот номер на третьем и последнем шаге, где отправил команду `echo «+ +» >> .rhosts` в качестве данных на порт, используемый демоном удаленной оболочки (шаг 5 на илл. 8.6 (в)). После этого он мог войти в систему с любого компьютера без пароля.

Поскольку атака Митника стала возможной главным образом из-за предсказуемости номеров ISN протокола TCP, впоследствии разработчики сетевых стеков приложили немало усилий к тому, чтобы повысить степень случайности при выборе протоколом TCP этих важных для безопасности номеров. Поэтому теперь реализовать такую атаку уже невозможно. Сегодня злоумышленники должны подбирать ISN иначе, например так, как это делается в случае захвата соединения. Обсудим эту разновидность атак более подробно.




**Илл. 8.6.** Атака Митника

### *Захват TCP-соединения*

Для захвата соединения необходимо преодолеть еще больше препятствий, чем для его подмены. Прежде всего предположим, что злоумышленники могут прослушивать существующее соединение между двумя пользователями (поскольку находятся в том же сегменте сети), и, соответственно, располагают точными порядковыми номерами и всей остальной необходимой информацией об этом соединении. Цель этого вида атак состоит в том, чтобы захватить существующее соединение путем внедрения данных в поток.

Для большей конкретности предположим, что злоумышленник хочет внедрить некоторые данные в TCP-соединение клиента, который вошел в веб-приложение на сервере, с тем чтобы либо клиент, либо сервер получил внедренные злоумышленником байты. Пусть для последних байтов, отправляемых клиентом и сервером, используются порядковые номера 1000 и 12500 соответственно. Допустим, все полученные до этого данные были подтверждены, и ни клиент ни сервер сейчас ничего не отправляют. В этот момент хакер внедряет, скажем, 100 байт в идущий к серверу TCP-поток путем отправки поддельного пакета, который содержит IP-адрес и порт источника клиента, а также IP-адрес и порт источника сервера. Этих четырех значений достаточно, чтобы сетевой стек демультиплексировал данные в нужный сокет. Помимо этого, злоумышленник предоставляет надлежащий порядковый номер (1001) и номер подтверждения (12501), и, таким образом, TCP передаст на веб-сервер 100 байт пользовательских данных.

Однако здесь возникает небольшая проблема. После передачи внедренных байтов приложению сервер отправит клиенту сообщение, подтверждающее их получение: «Спасибо за байты, готов получить байт номер 1101». Это сообщение оказывается неожиданным для клиента, и он думает, что сервер допустил ошибку. Ведь он не отправлял никаких данных и только собирается отправить байт 1001. Он быстро сообщает об этом серверу, отправив пустой сегмент с порядковым номером 1001 и номером подтверждения 12501. «Вот это да! — говорит сервер. — Спасибо, но это выглядит как старое подтверждение. А я уже получил следующие 100 байт. Лучше сообщить об этом удаленной стороне». Он снова отправляет подтверждение с номерами 1101 и 12501, на что клиент отвечает еще одним подтверждением, и т. д. Такая ситуация называется **штормом подтверждений (ACK storm)**. Этот цикл прекратится лишь в том случае, если одно

из подтверждений будет потеряно (в силу того, что протокол TCP не производит повторную передачу подтверждений, не содержащих какие-либо данные).

Каким же образом злоумышленник может утихомирить шторм подтверждений? Для этого существует несколько способов, которые мы сейчас обсудим. Самый простой подход сводится к тому, чтобы напрямую разорвать соединение, отправив RST-сегмент участникам коммуникации. В качестве альтернативы можно произвести отравление ARP, чтобы одно из подтверждений ушло на несуществующий адрес и потерялось. Еще одна стратегия состоит в том, чтобы внести настолько большое рассогласование между сервером и клиентом, чтобы сервер начал игнорировать все данные от клиента, и наоборот. Это достаточно сложно сделать путем отправки большого количества данных, но хакер может легко реализовать это на этапе установления соединения. Идея в следующем. Злоумышленник ждет, пока клиент не установит соединение с сервером. После отправки сервером ответного пакета SYN/ACK взломщик отсылает ему пакет RST для завершения соединения, а сразу за ним — пакет SYN с теми же IP-адресом и TCP-портом источника, которые изначально использовал клиент, но с другим порядковым номером клиентской стороны. После отправки сервером следующего пакета SYN/ACK как сервер, так и клиент будут находиться в состоянии установленного соединения, но не смогут взаимодействовать друг с другом. Ведь разница между их порядковыми номерами настолько большая, что они всегда будут находиться за пределами допустимого окна. Вместо этого злоумышленник, выступающий в роли «человека посередине», будет ретранслировать трафик между сторонами и при желании внедрять нужные ему данные.

### *Внемаршрутный взлом TCP*

Некоторые разновидности атак настолько сложны, что трудно понять даже сам принцип их действия, не говоря уже о том, чтобы как-то от них защититься. В данном разделе мы рассмотрим одну из них. Обычно злоумышленники не находятся в том же сегменте сети, что и участники коммуникации, и не могут прослушивать трафик между ними. Атаки, проводимые в таких условиях, называются **внемаршрутным взломом TCP (off-path TCP exploit)** и очень трудны в реализации. Даже если игнорировать возможный шторм подтверждений, чтобы внедрить данные в существующее соединение, злоумышленник должен собрать довольно много информации:

1. Перед тем как приступать к реальной атаке, необходимо выявить наличие соединения между двумя пользователями в интернете.
2. Затем нужно выяснить, какие номера портов использовать.
3. Наконец, необходимо узнать порядковые номера.

Это довольно сложная задача для того, кто находится на другом конце интернета, но все-таки ее можно выполнить. Спустя десятилетия после атаки Митника на Суперкомпьютерный центр Сан-Диего исследователи в области информационной безопасности выявили новую уязвимость, позволившую им реализовать внемаршрутный взлом TCP на популярных системах Linux. Они описали свою

атаку в статье «Внемаршрутный взлом TCP: глобальное ограничение скорости опасно» (весьма удачное название, как мы увидим далее). Мы рассмотрим этот случай, поскольку он хорошо демонстрирует, что утечка секретной информации может произойти косвенным образом.

По иронии судьбы проведение этой атаки стало возможным благодаря новой функции, которая должна была, наоборот, повысить, а не снизить степень защиты системы. Как уже упоминалось, внемаршрутное внедрение данных было очень сложной задачей. Злоумышленник должен был угадать номера портов и порядковые номера, а это трудно сделать методом прямого подбора. Тем не менее это возможно, учитывая, что точный порядковый номер не требовался — достаточно было обеспечить отправку данных в пределах допустимого окна. То есть с некоторой (очень малой) долей вероятности злоумышленники могли сбросить соединение или внедрить данные в существующие соединения. В августе 2010 года для решения этой проблемы было выпущено новое расширение протокола TCP (RFC 5961).

Спецификация RFC 5961 изменила то, как TCP принимал сегменты `SYN`, `RST` и обычные сегменты данных. Описанная уязвимость имелась только в Linux, поскольку лишь в этой системе RFC 5961 была реализована должным образом. Чтобы понять, как изменился протокол TCP с новым расширением, нужно разобраться, как он работал до этого. Сначала выясним, как TCP принимал сегменты `SYN`. До выхода RFC 5961 при получении `SYN` для уже существующего соединения TCP отбрасывал этот пакет, если он находился за пределами допустимого окна, и сбрасывал соединение, если он находился в его пределах. Это объяснялось следующим образом. При получении `SYN` TCP предполагал, что другой абонент произвел перезапуск, а значит, существующее соединение утратило актуальность. Такое поведение было не слишком разумным, ведь чтобы сбросить соединение, злоумышленнику достаточно было получить лишь один сегмент `SYN` с порядковым номером, попадающим в окно получателя. Вместо этого RFC 5961 предложила не сбрасывать соединение сразу, а сначала отправлять **подтверждение запроса (challenge ACK)** очевидному источнику сегмента `SYN`. Если пакет пришел от реального удаленного узла, значит, этот узел действительно потерял предыдущее соединение и теперь устанавливает новое. В этом случае при получении подтверждения запроса он отправит пакет `RST` с корректным порядковым номером. Этого не смогут сделать злоумышленники, поскольку они не получат подтверждение запроса.

То же самое относится и к сегментам `RST`. В исходной версии TCP хосты удаляли `RST`-пакеты, если они не попадали в допустимое окно, в противном случае они сбрасывали соединение. Чтобы усложнить сброс чужого соединения, в RFC 5961 было предложено немедленно прерывать его, только если порядковый номер в `RST` точно совпадает с номером, с которого начинается окно получателя (то есть со следующим ожидаемым порядковым номером). Когда этот номер не совпадает с ожидаемым, но находится в пределах допустимого окна, хост не разрывает соединение и отсылает подтверждение запроса. Если мы имеем дело с реальным отправителем, то он передаст `RST`-пакет с корректным порядковым номером.

Наконец, для сегментов данных протокол TCP старого образца проводил две проверки. Сначала он проверял порядковый номер, и если тот находился в пределах допустимого окна, он также проверял номер подтверждения, который считался действительным при попадании в определенный (огромный) интервал. Мы обозначим порядковый номер первого неподтвержденного байта как *FUB*, а номер следующего байта, подлежащего отправке, как *NEXT*. В этом случае действительны все пакеты с номерами подтверждения в диапазоне [*FUB* – 2*GB*, *NEXT*], или половина всего пространства номеров подтверждения. Злоумышленники просто не могут не воспользоваться такой ситуацией! Более того, если номер подтверждения также находился в пределах окна, прежний TCP обрабатывал данные и производил стандартное продвижение окна. Вместо этого спецификация RFC 5961 предписывает принимать те пакеты, у которых номер подтверждения находится внутри окна (приблизительно), и отправлять подтверждение запроса в случае пакетов, попадающих в окно [*FUB* – 2*GB*, *FUB* – *MAXWIN*], где *MAXWIN* — самое большое из когда-либо объявленных узлом окон.

Разработчики RFC 5961 быстро поняли, что это расширение может порождать очень много подтверждений запроса, и предложили ограничить количество подтверждений. В реализации для Linux это означало отправку не более 100 подтверждений запроса в секунду в рамках всех соединений. То есть некая глобальная переменная, общая для всех соединений, должна была отслеживать количество переданных подтверждений. Когда оно достигало 100, отправка прекращалась до окончания 1-секундного интервала (что бы при этом ни происходило).

Как бы хорошо это ни звучало, здесь кроется одна проблема. Единственная глобальная переменная здесь отражает общее состояние, что может стать побочным каналом для хорошо продуманной атаки. Рассмотрим первое препятствие, стоящее перед злоумышленниками, — вопрос о наличии соединения между двумя пользователями. Как вы помните, подтверждение запроса высылается в трех случаях:

1. В сегменте `SYN` указаны правильные IP-адреса отправителя и получателя и номера портов; при этом неважно, каким будет порядковый номер.
2. Порядковый номер сегмента `RST` находится в пределах допустимого окна.
3. В сегменте данных номер подтверждения также находится в пределах окна для запросов.

Допустим, злоумышленники хотят выяснить, имеется ли соединение между пользователем с адресом 130.37.20.7 и веб-сервером (с портом получателя 80) с адресом 37.60.194.64. Поскольку взломщикам не нужен точный порядковый номер, им достаточно лишь подобрать номер порта отправителя. Для этого они подключаются к веб-серверу и отправляют 100 пакетов `RST` подряд, в ответ на что сервер отправляет 100 подтверждений запроса (либо меньше, если часть таких подтверждений уже была отправлена до этого, что, впрочем, маловероятно). После 100 пакетов `RST` злоумышленники отправляют поддельный сегмент `SYN`, выдавая себя за клиента с адресом 130.37.20.7 с помощью подобранного номера порта. Если номер подобран неверно, то ничего не происходит, и взломщикам

все равно приходит 100 подтверждений. Но если он подобран правильно, мы получаем первый сценарий, при котором сервер отсылает подтверждение запроса реальному клиенту. Поскольку сервер может отправлять только 100 подтверждений запроса в секунду, из них злоумышленники получат только 99. То есть, посчитав полученные подтверждения, они могут не только выявить наличие соединения между двумя хостами, но и определить используемый клиентом номер порта источника (скрытый). Конечно, для этого потребуется довольно много попыток, но это вполне осуществимо. Кроме того, существуют методы, позволяющие повысить эффективность этого процесса.

Получив номер порта, злоумышленники могут перейти к следующему этапу атаки: подбору порядкового номера и номера подтверждения. Это делается аналогичным образом. В случае порядкового номера хакеры снова отсылают 100 подлинных пакетов `RST` (побуждая сервер к отправке подтверждений запросов) и дополнительный поддельный `RST` с правильными IP-адресами и уже известными номерами портов, а также подобранным порядковым номером. Если подобранный номер попадает в окно, мы получаем второй сценарий. То есть злоумышленники могут определить, правильно ли они подобрали номер, подсчитав количество полученных подтверждений запроса.

Наконец, в случае номера подтверждения они повторяют тот же трюк, отправляя в дополнение к 100 `RST` пакет данных, у которого все поля содержат корректные данные, за исключением подобранного номера подтверждения. После этого у злоумышленников имеется вся информация, необходимая для сброса соединения или внедрения данных.

Внемаршрутный взлом TCP хорошо демонстрирует три момента. Во-первых, мы понимаем, насколько сложными могут быть сетевые атаки. Во-вторых, это прекрасный пример сетевой **атаки по побочным каналам (side-channel attack)**. С ее помощью хакеры получают важную информацию по обходному каналу. В данном случае они узнают все нужные им параметры подключения путем подсчета, казалось бы, никак не связанных с этим вещей. В-третьих, внемаршрутный взлом TCP показывает, что главная причина, из-за которой атаки по побочным каналам стали возможными, — хранение общего состояния в глобальной переменной. Уязвимости побочных каналов очень часто встречаются и в программном, и в аппаратном обеспечении, и всегда коренным источником проблем является совместное использование какого-нибудь важного ресурса. Конечно, мы и так это знали, ведь такое разделение ресурсов идет вразрез с базовой идеей Зальцера и Шредера о необходимости минимизировать количество общих механизмов, о которой мы говорили в начале главы. В сфере безопасности представление о том, что с другими нужно делиться, не соответствует действительности!

Прежде чем перейти к следующей теме (нарушение работы и отказ в обслуживании), следует отметить, что внедрение данных не относится к числу методов, представляющих исключительно теоретический интерес; он также активно используется на практике. После разоблачений, сделанных Эдвардом Сноуденом (Edward Snowden) в 2013 году, стало известно, что американское Агентство национальной безопасности (National Security Agency, NSA) проводило массовую слежку за гражданами. Одним из направлений этой деятельности было

проведение сложной сетевой атаки Quantum. Когда определенные пользователи подключались к популярным сервисам (*Twitter*, *Gmail* или *Facebook*), их перенаправляли на специальные серверы путем внедрения пакетов. Затем эти серверы взламывали компьютеры жертв для получения полного доступа к хранящейся на них информации. Конечно, NSA полностью отрицает это. Оно готово отрицать сам факт своего существования. Иногда «NSA» в шутку расшифровывают как «No Such Agency» («нет такого агентства»).

## 8.2.4. Нарушение работы

Попытки сделать систему недоступной называются атаками типа «отказ в обслуживании». При этом жертве отправляется такой объем данных, с которым она не справляется и перестает отвечать на запросы. Существует несколько причин, по которым компьютер может прекратить реагировать:

1. **Аварийный отказ.** Отправленный злоумышленником контент вызывает аварийный отказ или зависание компьютера жертвы. Примером такой атаки является «пинг смерти», упомянутый ранее.
2. **Алгоритмическая сложность.** Злоумышленник отправляет такие данные, обработка которых требует больших вычислительных затрат (на уровне алгоритмов). Предположим, сервер позволяет клиентам отсылать расширенные поисковые запросы. В этом случае атака за счет алгоритмической сложности может сводиться к отправке ряда сложных регулярных выражений, что приведет к ситуации с наихудшим временем поиска.
3. **Переполнение данными.** Злоумышленник закидывает жертву таким гигантским количеством запросов или ответов, что система не справляется с ними. Часто, хотя и не всегда, это приводит к аварийному отказу компьютера жертвы.

Атаки переполнения данными стали серьезной головной болью многих организаций, ведь на сегодняшний день провести крупномасштабную DoS-атаку очень просто и дешево. За небольшую денежную сумму можно арендовать сеть ботов с тысячами компьютеров и атаковать любой адрес. Если данные отправляются с большого количества компьютеров из разных уголков мира, то это **распределенная атака типа «отказ в обслуживании»**, или **DDoS-атака (Distributed Denial-of-Service)**. Устроить DDoS-атаку может даже далекий от техники человек с помощью существующих в интернете **«стрессоров»** или **«бутеров»** — специализированных сервисов, предлагающих для этой цели удобный интерфейс.

### *Переполнение SYN-пакетами*

В старые добрые времена DDoS-атаки были достаточно простыми. Например, можно было использовать множество взломанных компьютеров, чтобы обеспечить переполнение `SYN`-пакетами. Все эти компьютеры отправляли на сервер

TCP-сегменты SYN, в которых часто были подделаны данные об отправителе. Пока сервер отправлял ответы SYN/ACK, никакой другой компьютер не мог завершить «рукопожатие» TCP. Таким образом, сервер оставался в зависшем состоянии, а это довольно затратно. Каждый хост может поддерживать лишь ограниченное количество полуоткрытых TCP-соединений и уже не принимает новые соединения по достижении этого предела.

Существует много способов защиты от переполнения SYN-пакетами. Например, можно просто отбрасывать полуоткрытые соединения после достижения предела, отдавая предпочтение новым, или уменьшить время жизни принятых SYN-пакетов. Очень элегантное и простое решение, которое сегодня поддерживает большинство существующих систем, сводится к использованию **файлов SYN cookie**, уже упоминавшихся в главе 6. Специальный алгоритм устанавливает изначальный порядковый номер таким образом, чтобы серверу не нужно было запоминать *никакие* параметры подключения до получения третьего пакета в «тройном рукопожатия». Как вы помните, размер порядкового номера составляет 32 бита. При использовании файлов SYN cookie сервер выбирает изначальный порядковый номер так:

1. Первые 5 бит отводятся под результат деления $t$ по модулю на 32, где $t$ — значение медленно возрастающего таймера (например, его значение увеличивается с интервалом в 64 с).
2. В следующих 3 битах кодируется значение MSS (максимальный размер сегмента), что дает восемь возможных значений.
3. Оставшиеся 24 бита отводятся под криптографический хеш, вычисляемый на основе временной метки $t$, IP-адресов отправителя и получателя и номеров портов.

Преимуществом такого порядкового номера является то, что сервер может просто вставить его в пакет SYN/ACK и забыть об этом. Если процесс «рукопожатия» не завершится, это не приведет к каким-либо тяжелым последствиям. Если же он будет завершен, то наличие собственного порядкового номера в дополнение к номеру в подтверждении позволит серверу полностью воспроизвести состояние, необходимое ему для установления соединения. Сначала сервер проверяет, совпадает ли криптографический хеш с последним значением $t$, а затем быстро восстанавливает запись в SYN-очередь, используя 3-битное значение MSS. Хотя файлы SYN cookie позволяют использовать только восемь вариантов размера сегмента и несколько ускоряют рост порядковых номеров, это практически не дает какого-либо негативного эффекта. Особенно приятно то, что данная схема совместима с обычным протоколом TCP и не требует, чтобы клиент поддерживал такое же расширение.

Конечно, наличие файлов SYN cookie не исключает вероятность того, что хакеры проведут DDoS-атаку, завершив процесс «рукопожатия». Но это обойдется им гораздо дороже (поскольку их компьютеры тоже имеют ограничение на количество открытых TCP-соединений), и что еще важнее, они не смогут провести атаку на TCP с использованием поддельных IP-адресов.

### *Отражение и усиление DDoS-атак*

Между тем DDoS-атаки на основе TCP не являются единственным возможным вариантом. В последние годы все больше крупномасштабных DDoS-атак проводится с UDP в качестве транспортного протокола. Подмена UDP-пакетов обычно не представляет больших проблем. Более того, использование UDP позволяет заставить реальные серверы в интернете провести против жертвы **атаку с отражением (reflection attack)**. В этом случае хакер отправляет запрос с поддельным адресом отправителя реальному UDP-сервису (например, серверу имен). После этого сервер отвечает на поддельный адрес. Если отослать такие запросы большому количеству серверов, то лавина ответных UDP-пакетов, скорее всего, выведет компьютер жертвы из строя. Атаки с отражением имеют два главных преимущества:

1. Добавив дополнительный уровень косвенности, злоумышленник усложняет для жертвы задачу блокировки отправителей (ведь это реальные серверы).
2. Часто сервисы *усиливают* атаку за счет отправки больших ответов на небольшие запросы.

Именно к данному виду атак с усилением относятся некоторые известные DDoS-атаки с рекордным объемом трафика, измерявшимся терабитами в секунду. Для успешного осуществления такой атаки злоумышленник должен найти общедоступные сервисы с большим коэффициентом усиления, где один небольшой пакет запроса может повлечь большой ответный пакет (а лучше несколько). При этом байтовый коэффициент усиления будет отражать относительное усиление в байтах, а пакетный — относительное усиление, выраженное в количестве пакетов. На илл. 8.7 представлены коэффициенты усиления нескольких популярных протоколов. Как бы внушительно ни выглядели эти цифры, важно помнить, что это средние значения, и у некоторых серверов эти показатели могут быть еще выше. Что интересно, у протокола DNSSEC, созданного для устранения проблем безопасности DNS, коэффициент усиления намного выше, чем у обычного протокола DNS, и в случае отдельных серверов может превышать 100. Не отстают в этом плане и некорректно настроенные *memcached*-серверы (быстрые базы данных в оперативной памяти) — в ходе крупномасштабной атаки с усилением, проведенной с их помощью в 2018 году, был достигнут объем трафика в 1,7 Тбит/с и коэффициент усиления, намного превышающий 50 000.

| Протокол | Байтовый коэффициент усиления | Пакетный коэффициент усиления |
|---|---|---|
| NTP | 556,9 | 3,8 |
| DNS | 54,6 | 2,1 |
| BitTorrent | 3,8 | 1,6 |

**Илл. 8.7.** Коэффициенты усиления популярных протоколов

### Защита от DDoS-атак

Хотя защититься от столь огромных потоков трафика нелегко, все же существует несколько способов. Один из самых простых состоит в том, чтобы заблокировать трафик в непосредственной близости от его источника. Как правило, для этого используется **выходная фильтрация (egress filtering)**, при которой сетевое устройство, брандмауэр, блокирует все исходящие пакеты, у которых IP-адрес отправителя не входит в число адресов его сети. Конечно, при этом брандмауэр должен знать, какие пакеты могут прийти с определенным IP-адресом источника; как правило, это возможно только в пограничном сегменте сети. Например, брандмауэр может знать все диапазоны IP-адресов университетской сети и блокировать трафик от любого постороннего IP-адреса. Противоположностью выходной фильтрации является **входная фильтрация (ingress filtering)**, при которой сетевое устройство отбрасывает весь входящий трафик с внутренними IP-адресами.

Еще один способ защиты сводится к тому, чтобы попытаться «поглотить» DDoS-атаку с помощью резервной пропускной способности, что требует больших затрат, непозволительных для отдельной организации (за исключением самых крупных компаний). К счастью, это необязательно делать в одиночку. Если организации объединят свои ресурсы, то даже очень небольшие компании смогут позволить себе такую защиту от DDoS-атак. Как и в случае страхования, предполагается, что хакеры не атакуют всех сразу.

Так какую «страховку» мы при этом получаем? Существуют компании, предлагающие сайтам **облачную защиту от DDoS-атак**: при необходимости емкость увеличивается за счет облака. При этом фактически обеспечивается облачное экранирование или даже сокрытие IP-адреса реального сервера. Все запросы передаются расположенным в облаке прокси-серверам, которые по возможности отсеивают вредоносный трафик (в случае продвинутых атак это очень непросто) и направляют безопасные запросы на реальный сервер. В случае роста числа запросов или объема трафика, идущего на конкретный сервер, облако выделяет дополнительные ресурсы для обработки пакетов. То есть оно «поглощает» возрастающий поток данных. Обычно облако также выступает и в качестве **скруббера** («щетки») для очистки данных. Например, оно может удалять дублирующие друг друга TCP-сегменты или недопустимые комбинации TCP-флагов и в целом играть роль **брандмауэра веб-приложений (Web Application Firewall, WAF)**.

Существует несколько разных по стоимости вариантов ретрансляции трафика через облачные прокси-серверы. Если финансы позволяют, можно воспользоваться методом **создания черной дыры с помощью BGP (BGP blackholing)**. В этом случае предполагается, что владелец сайта целиком контролирует блок /24 из 16 777 216 адресов. Идея в том, чтобы владелец сайта просто удалил объявления BGP для этого блока в своих маршрутизаторах. Вместо этого поставщик облачной защиты начинает объявлять эти IP-адреса из *собственной* сети, чтобы весь идущий на сервер трафик сначала проходил через облако. Но далеко не каждый имеет в своем распоряжении целый блок сети и может оплатить перенаправление BGP. Есть более экономный вариант — **перенаправление DNS**

**(DNS rerouting)**. Администраторы сайта должны изменить DNS-сопоставления на своих серверах имен так, чтобы они указывали не на реальный сервер, а на серверы в облаке. В обоих случаях пакеты посетителей сайта сначала попадают на прокси-сервер поставщика облачной защиты, а затем перенаправляются на реальный сервер.

Хотя перенаправление DNS проще в реализации, такая защита будет надежной, только если вам удастся сохранять в тайне реальный IP-адрес сервера. Если злоумышленники узнают этот адрес, они смогут атаковать сервер напрямую, действуя в обход облака. К сожалению, существует целый ряд каналов, по которым может произойти утечка сведений об IP-адресе. Подобно протоколу FTP, некоторые веб-приложения передают IP-адрес удаленному абоненту путем внутриполосной передачи, что является практически неразрешимой проблемой. Еще одним каналом утечки являются архивные данные службы DNS, которые позволяют злоумышленникам увидеть, какие IP-адреса сервер использовал в прошлом. Некоторые компании занимаются тем, что собирают и продают такие архивы.

## 8.3. БРАНДМАУЭРЫ И СИСТЕМЫ ОБНАРУЖЕНИЯ ВТОРЖЕНИЙ

Возможность подключать любые компьютеры по всему миру друг к другу имеет как положительные, так и отрицательные стороны. Отдельным домашним пользователям интернет принес множество развлечений. А вот для специалистов по безопасности в корпорациях это настоящий кошмар. Большинство компаний располагает огромными объемами конфиденциальной информации, размещенной на подключенных к сети компьютерах: коммерческие тайны, планы развития производства, рыночные стратегии, аналитические финансовые отчеты, история налоговых платежей и т. д. Раскрытие этих сведений конкурентам может иметь ужасные последствия.

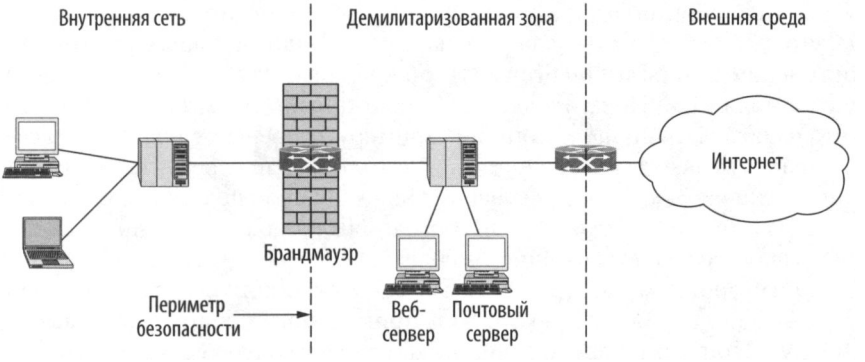
Помимо опасности утечки информации, существуют риски проникновения вредоносных программ — вирусов, червей и прочей цифровой заразы, способной красть секреты и уничтожать ценные данные. На борьбу с ней у системных администраторов уходит масса времени. Часто эту «инфекцию» заносят беззаботные сотрудники, желающие поиграть в новую модную компьютерную игру.

Следовательно, нам требуются механизмы, позволяющие оставлять только «хорошие» биты, устраняя все «плохие». Один из способов состоит в использовании шифрования для защиты данных во время их пересылки между безопасными сайтами. Однако шифрование не спасает от вирусов и хакеров, способных проникнуть в LAN компании. Чтобы защитить сеть, нам понадобится межсетевой экран.

### 8.3.1. Брандмауэры

**Брандмауэры (firewalls)** — это современная реализация средневекового принципа обеспечения безопасности. Они напоминают ров, вырытый вокруг замка. Суть конструкции в том, что все входящие и выходящие из замка должны

проходить по одному подъемному мосту, где полиция ввода/вывода проверяет их личность. Тот же принцип может применяться и в сетях: у компании может быть несколько LAN, соединенных произвольным образом, но весь внешний трафик должен проходить через электронный «подъемный мост» (межсетевой экран), как показано на илл. 8.8. Никакого другого пути попасть в сеть не существует.




**Илл. 8.8.** Брандмауэр, защищающий внутреннюю сеть

Брандмауэр работает как **пакетный фильтр (packet filter)**. Он исследует каждый входящий и исходящий пакет. Пакеты, удовлетворяющие определенным критериям, пропускаются. Не сумевшие пройти проверку пакеты удаляются.

Критерии фильтрации обычно устанавливаются правилами или таблицами, где перечислены допустимые и блокируемые отправители и получатели, а также стандартные правила, описывающие, что делать с исходящими и входящими пакетами. В общем случае настроек TCP/IP информация о получателе или отправителе состоит из IP-адреса и номера порта. Номер порта соответствует нужной службе. Например, порт 25 используется для почты, порт 80 — для HTTP. Некоторые порты просто могут быть заблокированы. Например, компания может блокировать поступление входящих пакетов для всех IP-адресов, привязанных к TCP-порту 79. Раньше этот порт широко применялся для получения адресов электронной почты с помощью службы Finger, но сегодня практически не используется, поскольку сыграл не последнюю роль в крупной интернет-атаке 1988 года.

Остальные порты блокируются не так просто. Сложность в том, что сетевые администраторы мечтают об идеальной защите, но не могут блокировать коммуникацию с внешним миром. Второй вариант был бы проще и безопаснее, однако поток жалоб от пользователей шел бы непрерывно. Здесь могут помочь такие решения, как **демилитаризованная зона (DeMilitarized Zone, DMZ)**, показанная на илл. 8.8. DMZ — это часть сети компании, которая находится за пределами периметра безопасности. Сюда допускаются все. Если разместить веб-сервер в DMZ, то сторонние компьютеры будут подключаться к нему, чтобы зайти на сайт компании. При этом брандмауэр можно настроить так, чтобы блокировать входящий TCP-трафик к порту 80, так что компьютеры извне не

смогут использовать этот порт для атаки компьютеров внутренней сети. Чтобы управлять веб-сервером, брандмауэр разрешает соединения между ним и компьютерами в сети.

По мере продолжения «гонки вооружений» между разработчиками средств защиты и хакерами брандмауэры становились все более сложными. Изначально они применяли набор правил независимо для каждого пакета, но составлять правила, предоставляющие полезную функциональность и при этом блокирующие весь нежелательный трафик, — задача не из легких. **Брандмауэры с контролем состояния соединений (statefull firewalls)** сопоставляют пакеты с соединениями и используют поля TCP/IP-заголовков для отслеживания этих соединений. Это позволяет создавать правила, которые, к примеру, разрешают внешнему веб-серверу отправлять пакеты внутреннему хосту, только если перед этим этот хост установил соединение с внешним сервером. Такое правило нельзя было создать при использовании схемы без контроля состояния: она может либо пропускать, либо отбрасывать все пакеты внешнего сервера.

Следующим уровнем сложности по сравнению с контролем состояния является реализация брандмауэром **шлюза прикладного уровня (application-level gateway)**. При этом брандмауэр просматривает содержимое пакетов, не ограничиваясь лишь TCP-заголовками, чтобы узнать, что делает приложение. Это позволяет отделить HTTP-трафик для серфинга в интернете от HTTP-трафика для обмена файлами между пирами. Администратор может написать правила, запрещающие сотрудникам компании обмениваться файлами, но позволяющие заходить на сайты, необходимые для работы. С помощью этих методов исходящий трафик может исследоваться примерно так же, как входящий (например, чтобы важные документы не уходили по электронной почте за пределы компании).

Из всего вышесказанного очевидно, что брандмауэры нарушают стандартную иерархию протоколов. Они являются устройствами сетевого уровня, но также контролируют транспортный и прикладной уровни, чтобы фильтровать данные. Это делает их уязвимыми. Например, брандмауэры ориентируются на стандартную нумерацию портов, чтобы определить, какой трафик находится в пакете. Хотя стандартные порты применяются часто, их используют далеко не все компьютеры и приложения. Некоторые пиринговые приложения выбирают порт динамически, чтобы его было труднее отследить (и заблокировать). Кроме того, высокоуровневая информация скрыта от брандмауэра шифрованием. Наконец, брандмауэр не может открыто сообщить компьютерам, которые пытаются связаться друг с другом с его помощью, почему соединение не удалось установить. Он должен сделать вид, что дело в отрезанном кабеле. В силу этих причин сетевые пуристы считают брандмауэры недостатком архитектуры интернета. Тем не менее интернет может представлять опасность для компьютера. Брандмауэры помогают решать эту проблему, так что они, скорее всего, никуда не исчезнут.

Даже в случае идеально настроенного брандмауэра остается множество вопросов, связанных с безопасностью. Например, если входящие пакеты пропускаются только со стороны конкретных сетей (например, LAN дочернего предприятия компании), взломщик, находящийся вне зоны действия брандмауэра, может

просто фальсифицировать адрес отправителя и тем самым преодолеть барьер. Если же нечестный сотрудник захочет переслать секретные документы, он может их зашифровать или просто сфотографировать и передать в виде JPEG-файлов, и тогда они смогут проникнуть через любые почтовые анализаторы. И мы еще не обсудили тот факт, что хотя 75 % атак происходит извне, атаки в зоне действия брандмауэра (например, недовольными сотрудниками) зачастую гораздо опаснее (см. отчет компании Verizon, 2009).

Еще один недостаток брандмауэров состоит в том, что они обеспечивают единый периметр или единую защиту. Если эта защита ломается — все пропало. По этой причине их используют для многоуровневой защиты. Например, брандмауэр может охранять вход во внутреннюю сеть, а на каждом компьютере также может быть свой брандмауэр. Если вам кажется достаточным наличие одного контрольно-пропускного пункта, видимо, вы давно не летали международными авиалиниями. Сегодня многие сети имеют несколько уровней защиты, вплоть до отдельных брандмауэров для каждого хоста. Это самый простой пример **эшелонированной защиты (defense in depth)**. Следует заметить, что как в аэропорту, так и в компьютерной сети наличие множества независимых механизмов защиты существенно усложняет злоумышленникам задачу взлома всей системы.

## 8.3.2. Обнаружение и предотвращение вторжений

Помимо брандмауэров и скрубберов, администраторы сетей могут развернуть ряд других средств защиты, включая системы обнаружения и предотвращения вторжений, которые мы сейчас кратко обсудим. Как можно понять из названия, **системы обнаружения вторжений (Intrusion Detection System, IDS)** служат для обнаружения атак, в идеале еще до того, как атаки успеют причинить какой-либо урон. Например, они могут генерировать предупреждения при выявлении ранних проявлений атаки, таких как сканирование портов, попытки войти в систему путем **прямого перебора паролей от SSH** (когда атакующий просто перебирает различные популярные пароли) или обнаружение сигнатуры новейшего и мощного эксплойта в TCP-соединении. В то же время иногда атаку удается выявить лишь на позднем ее этапе, когда система уже взломана и необычно себя ведет.

IDS можно классифицировать по тому, *где* и *как* они работают. **Хостовые системы обнаружения вторжений (Host-based IDS, HIDS)** работают непосредственно на конечной точке (ноутбуке или сервере) и отслеживают поведение программного обеспечения или входящий и исходящий трафик веб-сервера только в пределах данного компьютера. **Сетевые системы обнаружения вторжений (Network IDS, NIDS)**, напротив, проверяют сетевой трафик некоторой группы компьютеров. Обе эти разновидности систем имеют свои преимущества и недостатки.

Сетевые IDS привлекательны тем, что защищают большое количество компьютеров (и благодаря этому могут находить корреляции между событиями на разных хостах) и не используют их ресурсы. Таким образом, IDS не влияют на производительность компьютеров в защищаемой ими области сети. С другой стороны, при этом трудно устранять проблемы, касающиеся

отдельных компьютеров. Допустим, что в трафике TCP-соединения имеются TCP-сегменты, частично накладывающиеся друг на друга: пакет A содержит байты 1–200, а пакет B — байты 100–300. Очевидно, что в данном случае имеется наложение байтов пользовательских данных. Предположим, что байты в накладывающейся области несколько различаются. Что в таком случае должна сделать IDS?

На самом деле вопрос заключается в следующем: какие байты будет при этом использовать хост-получатель? Если ему нужны байты из пакета A, то IDS должна проверить их на отсутствие вредоносного контента и проигнорировать пакет B. Но что, если этот хост использует байты из пакета B? А если одна часть хостов в сети принимает байты из пакета A, а другая — из пакета B? Затруднения могут возникнуть, даже если все хосты ведут себя одинаково и IDS знает, как они производят повторную сборку данных из TCP-потока. Даже если все хосты обычно принимают байты из пакета A, IDS может обнаружить, что этот пакет все равно некорректен, поскольку до адресата он должен пройти еще два-три транзитных участка, а его значение TTL равно единице (то есть он не сможет достичь пункта назначения). Такие трюки со значением TTL и накладывающимися диапазонами байтов в IP-фрагментах или TCP-сегментах называют методами **обхода IDS (IDS evasion)**.

Еще одной проблемой при использовании сетевых IDS является шифрование. Если сетевые байты уже невозможно дешифровать, то системе IDS очень сложно проверить, являются ли они вредоносными. Это еще один пример того, как одно средство защиты (шифрование) может снизить эффективность другого средства (IDS). В качестве обходного маневра администраторы отдельных систем могут предоставить IDS ключи шифрования для сетевой IDS. Хотя этот подход работает, все же он не идеален, ведь тогда придется дополнительно заботиться об управлении ключами. При этом IDS видит *весь* сетевой трафик и, как правило, сама содержит множество строк кода. Из-за этого она становится привлекательной мишенью для атаки. Взломав ее, злоумышленники получат доступ ко всему сетевому трафику!

Недостатком хостовых IDS является то, что они используют ресурсы тех отдельных компьютеров, на которых работают, и видят лишь незначительную часть происходящих в сети событий. С другой стороны, им гораздо легче бороться с методами обхода IDS, поскольку они могут проверять поступающие данные после их повторной сборки сетевым стеком защищаемого компьютера. Кроме того, при использовании, к примеру, IPsec (когда шифрование и дешифрование пакетов происходит на сетевом уровне) хостовая IDS может проверять данные после расшифровки.

Как уже упоминалось, IDS могут различаться не только по расположению, но и по тому, *как* они выявляют угрозы. Здесь выделяют две основные категории. **Сигнатурные системы обнаружения вторжений (Signature-based IDS)** используют паттерны байтов или последовательностей пакетов, представляющие собой признаки известных разновидностей атак. Если вы знаете, что поступление в порт 53 UDP-пакета, содержащего 10 определенных байтов в начале пользовательских данных, говорит о применении эксплойта *E*, то IDS может проверять сетевой трафик на отсутствие этого паттерна и выдавать оповещение

при его обнаружении. Это оповещение будет носить конкретный характер («Я обнаружил *E*») и обладать высокой степенью уверенности («Я точно знаю, что это *E*»). В то же время IDS на основе сигнатур могут выявлять только известные разновидности угроз, для которых доступны сигнатуры. Еще один подход сводится к тому, чтобы IDS выдавала оповещение при обнаружении *необычного* поведения. Например, если компьютер, который обычно обменивается SMTP- и DNS-трафиком только с несколькими IP-адресами, вдруг начинает отправлять HTTP-трафик на множество совершенно незнакомых IP-адресов за пределами LAN, IDS может посчитать это подозрительным. Поскольку такие **системы обнаружения вторжений на основе аномалий (Anomaly-based IDS)**, или просто **системы обнаружения аномалий**, реагируют на любое нетипичное поведение, они могут выявлять и старые, и новые разновидности атак. Их недостатком является малоинформативность оповещений. Так, сообщение «в сети произошло нечто необычное» гораздо менее конкретное и полезное, чем сообщение «камера видеонаблюдения на воротах атакована вредоносной программой Hajime».

**Система предотвращения вторжений (Intrusion Prevention System, IPS)** должна не только выявлять, но и останавливать атаки. С этой точки зрения она действует так же, как уже известный нам брандмауэр. Например, обнаружив пакет с сигнатурой Hajime, IPS может отбросить его, не позволив ему дойти до камеры видеонаблюдения. Чтобы это сделать, IPS должна находиться на пути к объекту атаки и принимать решение о приеме или отбрасывании трафика на лету. IDS, напротив, может располагаться в другой части сети, при условии, что мы зеркалируем весь трафик так, чтобы IDS могла его видеть. Вы спросите: к чему все эти лишние сложности? Почему нельзя просто развернуть IPS и полностью избавиться от возможных угроз? Отчасти потому, что при этом встает вопрос производительности: скорость передачи данных будет определяться скоростью обработки данных в IPS. Если на обработку отведено слишком мало времени, то данные будут проверяться не слишком тщательно. Что, если будет допущена ошибка? Точнее, что, если IPS посчитает трафик какого-нибудь соединения вредоносным и отбросит эти данные, хотя на самом деле они не представляют никакой угрозы? Это весьма нежелательно, особенно если речь идет о важном соединении, от которого зависят бизнес-процессы. Возможно, лучше выдать оповещение и позволить кому-то другому изучить подозрительные данные, чтобы выяснить, действительно ли они представляют угрозу.

При этом очень важно знать, насколько часто подозрения IDS или IPS оказываются верными. Если система слишком часто поднимает ложную тревогу, то есть выдает много **ложноположительных результатов (false positives, FP)**, это закончится тем, что вы потратите много времени и денег на поиски черной кошки в темной комнате. Если же, наоборот, IDS или IPS ведет себя слишком сдержанно, часто не поднимая тревогу при атаке, то есть выдает много **ложноотрицательных результатов (false negatives, FN)**, то злоумышленники смогут легко взломать вашу систему. Эффективность системы защиты определяется тем, каким будет число ложноположительных и ложноотрицательных результатов по сравнению с количеством **истинно положительных (true positive, TP)** и **истинно**

**отрицательных (true negative, TN)** результатов. Свойства системы в этом отношении принято выражать в виде таких показателей, как **точность (precision)** и **полнота (recall)**. Точность показывает, какая доля сигналов тревоги была оправданной, что выражается следующей формулой: $P = TP/(TP + FP)$. Полнота показывает, какую долю реальных атак удалось выявить: $R = TP/(TP + FN)$. Можно применять комбинацию двух этих значений, **F-меру (F-measure)**: $F = 2PR/(P + R)$. Наконец, иногда нам нужно выяснить только то, как часто IDS или IPS выдает правильный результат, для этого используется показатель **доля верных результатов (accuracy)**: $A = (TP + TN)/$*общее количество*.

Что касается показателей полноты и точности, чем выше их значение, тем лучше. При этом между количеством ложноотрицательных и ложноположительных результатов обычно существует обратная зависимость: при уменьшении первого параметра растет второй, и наоборот. Диапазон допустимых значений для этих параметров зависит от конкретной ситуации. Если система защиты создается для Пентагона, то на первый план выходит исключение вероятности взлома. В этом случае лучше получать чуть больше ложноположительных результатов при минимальном количестве ложноотрицательных. Однако если система защиты создается для учебного заведения, то к ней предъявляются уже не столь жесткие требования. Вероятно, здесь можно обойтись без системного администратора, который проводит почти все свое рабочее время, анализируя ложные срабатывания системы.

Нам осталось рассмотреть еще одну особенность упомянутых показателей, которая показывает важность ложноположительных результатов. При этом мы используем слегка модифицированную версию примера, приведенного Стефаном Аксельсоном (Stefan Axelsson) в его известной статье о том, почему обнаружение вторжений является сложным процессом (Axelsson, 1999). Допустим, реальное количество заболевших определенной болезнью составляет 1 из 100 000 человек. Любой человек, у которого обнаруживается эта болезнь, умирает в течение месяца. К счастью, имеется тест, позволяющий эффективно выявлять наличие болезни. У этого теста доля верных результатов составляет 99 %: если пациент болен ($S$), то в 99 % случаев тест выдаст положительный результат (что в медицине является не хорошим, а плохим признаком!), а если пациент здоров ($H$), то в 99 % случаев тест выдаст отрицательный результат ($Neg$). Решив пройти этот тест, вы получаете положительный результат ($Pos$). Вопрос на миллион долларов: насколько это плохой признак? Или скажем иначе: стоит ли вам попрощаться с друзьями и семьей, распродать имущество и пуститься во все тяжкие в оставшиеся вам тридцать с небольшим дней или все же не стоит этого делать?

Чтобы ответить на этот вопрос, обратимся к математике. Мы должны определить вероятность наличия у вас заболевания при условии, что тест выдал положительный результат: $P(S|Pos)$. Нам известно следующее:

$$P(Pos|S) = 0{,}99$$
$$P(Neg|H) = 0{,}99$$
$$P(S) = 0{,}00001.$$

Чтобы вычислить $P(S|Pos)$, воспользуемся известной формулой Байеса:

$$P(S|Pos) = \frac{P(S)P(Pos|S)}{P(Pos)}.$$

В данном случае тест может выдавать только два варианта ответа — вы либо заболели, либо нет. То есть

$$P(Pos) = P(S)P(Pos|S) + P(H)P(Pos|H),$$

где $P(H) = 1 - P(S)$;

$P(Pos|H) = 1 - P(Neg|H)$, следовательно:

$$\begin{aligned} P(Pos) &= P(S)P(Pos|S) + (1 - P(S))(1 - P(Neg|H)) \\ &= 0{,}00001 * 0{,}99 + 0{,}99999 * 0{,}01. \end{aligned}$$

Таким образом

$$\begin{aligned} P(S|Pos) &= \frac{0{,}00001 * 0{,}99}{0{,}00001 * 0{,}99 + 0{,}99999 * 0{,}01}. \\ &= 0{,}00098 \end{aligned}$$

Итак, вероятность наличия у вас этого заболевания составляет менее 0,1 %, и вам пока не стоит паниковать (если, конечно, вы еще не успели распродать все свое имущество).

Как мы видим, окончательное значение вероятности здесь в значительной мере зависит от доли ложноположительных результатов $P(Pos|H) = 1 - P(Neg|H) = 0{,}01$. Это объясняется тем, что количество рассматриваемых происшествий является настолько малым (0,00001), что мы можем пренебречь влиянием всех остальных членов уравнения. Данную проблему называют **ошибкой базовой оценки (Base Rate Fallacy)**. Если мы заменим здесь слова «болезнь» и «положительный результат» словами «атака» и «оповещение об атаке», то увидим, что ошибка базовой оценки играет крайне важную роль при использовании любой IDS или IPS. Поэтому нужно стремиться к тому, чтобы количество ложных срабатываний было как можно меньше.

Кроме базовых принципов безопасности Зальцера и Шредера, многие исследователи предлагали использовать дополнительные, часто очень практичные принципы. В частности, здесь уместно вспомнить прагматичный **принцип эшелонированной защиты (principle of defense in depth)**. Часто для защиты системы целесообразно использовать несколько взаимно дополняющих методов. Например, чтобы остановить атаку, можно применять брандмауэр *в сочетании* с системой обнаружения вторжений *и* антивирусным сканером. Хотя эти средства не дают стопроцентной защиты по отдельности, их гораздо труднее обойти, если они используются одновременно.

## 8.4. КРИПТОГРАФИЯ

Слово **криптография (cryptography)** происходит от греческого «тайнопись». У криптографии долгая и яркая история, насчитывающая несколько тысяч лет. В данном разделе мы всего лишь кратко упомянем некоторые отдельные моменты в качестве введения к последующей информации. Желающим ознакомиться с полной историей криптографии рекомендуется книга Кана (Kahn, 1995). Для получения всестороннего представления о текущем положении дел см. работу Кауфмана и др. (Kaufman et al., 2002). Если вы хотите подробнее ознакомиться с математическими основами криптографии, см. книгу Крафта и Вашингтона (Kraft and Washington, 2018). Если вас, наоборот, не слишком интересуют математические основы, см. работу Эспозито (Esposito, 2018).

С профессиональной точки зрения понятия «шифр» и «код» отличаются друг от друга. **Шифр (cipher)** представляет собой посимвольное или побитовое преобразование, которое не зависит от лингвистической структуры сообщения. **Код (code)**, напротив, заменяет целое слово другим словом или символом. Коды в настоящее время не используются, хотя имеют богатую историю.

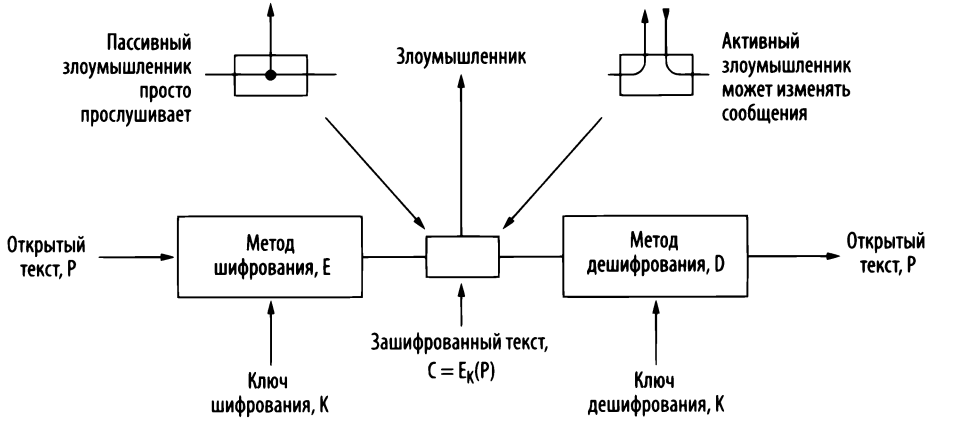
Наиболее успешным в истории считается код, использованный корпусом морской пехоты США в Тихом океане во время Второй мировой войны. Для ведения секретных переговоров применялись слова языка индейцев навахо, носители которого служили в американских войсках. Например, слово *чаи-да-гахи-найл-цайди* (что буквально означает «убийца черепах») означало противотанковое оружие. Язык навахо тоновый (для различения смысла используется повышение или понижение тона), весьма сложный, не имеет письменной формы. Но самое большое его достоинство заключалось в том, что ни один японец не имел о нем ни малейшего представления. В сентябре 1945 года в газете *San Diego Union* появилась статья, в которой были раскрыты сведения о том, насколько эффективным оказался опыт использования языка индейцев навахо во время военного противостояния с японцами. Японцы так и не смогли взломать этот код, и многие носители языка индейцев навахо были удостоены высоких воинских наград за отличную службу и смелость. Тот факт, что США смогли расшифровать японский код, а японцы так и не узнали язык навахо, сыграл важную роль в американской победе в Тихом океане.

### 8.4.1. Основы криптографии

Исторически искусство криптографии применяли и развивали представители четырех групп: военные, дипломаты, авторы дневников и влюбленные. Наиболее важный вклад в эту сферу на протяжении веков вносили военные. В военных организациях секретные сообщения традиционно отдавались для зашифровки и передачи низкоквалифицированным шифровальщикам с маленьким жалованьем. Объем сообщений не позволял выполнить эту работу с привлечением небольшого числа элитных специалистов.

До появления компьютеров одним из основных сдерживающих факторов в криптографии была способность шифровальщика выполнять необходимые преобразования, часто на поле боя, с помощью несложного оборудования. Кроме

того, достаточно тяжело было быстро переключаться с одного криптографического метода на другой, так как для этого требовалось переобучение множества людей. Тем не менее риск захвата шифровальщика в плен поставил на первый план задачу мгновенного перехода, в случае необходимости, к другому криптографическому методу. Эти противоречивые требования отражены в модели процесса шифрования-дешифрования (илл. 8.9).




**Илл. 8.9.** Модель шифрования (для шифра с симметричным ключом)

Сообщения, подлежащие зашифровке, **открытый текст (plaintext)**, преобразуются с помощью функции, параметром которой является **ключ (key)**. Результат шифрования, **зашифрованный текст (ciphertext)**, обычно передается по радио или через связного. Предполагается, что противник или злоумышленник (intruder) слышит и аккуратно копирует весь зашифрованный текст. Но в отличие от адресата, он не знает ключа дешифрования, и поэтому расшифровка сообщения представляет для него большие трудности, а порой просто невозможна. Иногда он не только прослушивает канал связи (пассивный злоумышленник), но также записывает сообщения и воспроизводит их позже, вставляет свои сообщения или модифицирует оригинал, прежде чем он достигнет получателя (активный злоумышленник). Искусство взлома шифров называется **криптоанализом (cryptanalysis)**. Искусство изобретать шифры (криптография) и искусство взламывать их (криптоанализ) вместе называются **криптологией (cryptology)**.

Для обозначения открытого/зашифрованного текста и ключей лучше всего применять специальную нотацию. Мы будем использовать формулу $C = E_K(P)$, обозначающую, что при зашифровке открытого текста $P$ с помощью ключа $K$ получается зашифрованный текст $C$. Аналогично формула $P = D_K(C)$ означает расшифровку зашифрованного текста $C$ для восстановления открытого текста. Из этих формул следует, что

$$D_K(E_K(P)) = P.$$

Эта нотация предполагает, что $E$ и $D$ — просто математические функции (чем они и являются). Единственная хитрость состоит в том, что каждая из них имеет два параметра, один из которых (ключ) мы записали не в виде аргумента, а в виде нижнего индекса, чтобы отличать его от сообщения.

Основное правило криптографии заключается в предположении, что криптоаналитику (взломщику шифра) известен используемый метод шифрования. Другими словами, он точно знает, как работают методы шифрования $E$ и дешифрования $D$ на илл. 8.9. На разработку, тестирование и внедрение нового метода тратятся огромные усилия. Поэтому каждый раз, когда старый метод оказывается скомпрометированным (или считается таковым), хранить алгоритм шифрования в секрете просто непрактично. А предположение, что метод является секретным, когда это уже не так, может принести еще больше вреда.

Здесь на помощь приходит ключ шифрования. Он состоит из относительно короткой строки, определяющей один из множества вариантов результата шифрования. В отличие от самого метода шифрования, который может меняться лишь раз в несколько лет, ключ можно менять столько, сколько нужно. Таким образом, наша базовая модель представляет собой постоянный и известный общий метод, в котором в качестве параметра используется секретный и легко изменяемый ключ. Идея, согласно которой криптоаналитику известен метод, а краеугольным камнем секретности является эксклюзивный ключ, называется **принципом Керкгоффса (Kerckhoff's principle)**. Его в 1883 году впервые высказал голландский военный криптограф Огюст Керкгоффс (Auguste Kerckhoff, 1883). Итак:

*Принцип Керкгоффса: все алгоритмы шифрования должны быть общедоступными; секретны только ключи.*

Не стоит придавать большое значение секретности алгоритма. Попытка сохранить алгоритм в тайне, называемая в отрасли **безопасностью за счет неясности (security by obscurity)**, обречена на провал. К тому же, опубликовав свой алгоритм, разработчик получает бесплатную консультацию от большого количества ученых-криптоаналитиков, горящих желанием взломать новую систему и тем самым продемонстрировать свой ум. Если никто не смог взломать алгоритм в течение долгого времени после его публикации, видимо, алгоритм достаточно прочен. (С другой стороны, исследователи иногда находят ошибки в системах защиты с открытым кодом наподобие OpenSSL спустя десять и более лет после их появления, поэтому принцип «чем больше глаз, тем меньше неполадок» далеко не всегда действует на практике.)

Поскольку только ключ является по-настоящему секретным, главный вопрос касается его длины. Рассмотрим простой кодовый замок. Его основной принцип состоит в том, что вы последовательно вводите цифры. Все это знают, но ключ хранится в секрете. Ключ длиной в две цифры образует 100 вариантов. Ключ длиной в три цифры означает 1000 вариантов, а при длине ключа в шесть цифр число комбинаций достигает миллиона. Чем длиннее ключ, тем выше **показатель трудозатрат (work factor)** взломщика шифра. При увеличении длины ключа показатель трудозатрат для взлома системы путем простого перебора значений

ключа растет экспоненциально. Секретность передаваемого сообщения обеспечивается мощным (но все же открытым) алгоритмом и длинным ключом. Чтобы не дать прочитать свою электронную почту младшему брату, достаточно ключа длиной в 64 бита. В коммерческих системах имеет смысл использовать ключи длиной 256 бит. Для защиты текстов от спецслужб развитых государств потребуются ключи минимум в 256 бит (и гораздо больше). При этом нужно отметить, что эти цифры относятся к симметричному шифрованию, когда для шифрования и дешифрования используются одинаковые ключи. Чуть позже мы подробно обсудим различия между симметричным и асимметричным шифрованием.

С точки зрения криптоаналитика, задача криптоанализа имеет три принципиальных варианта. В первом случае у криптоаналитика есть зашифрованный текст без соответствующего открытого текста; это **задача только с зашифрованным текстом (ciphertext-only problem)**. Криптограммы такого рода часто публикуются в газетах в разделе головоломок. Во втором случае криптоаналитик располагает зашифрованным текстом вместе с соответствующим открытым текстом; это **задача с известным открытым текстом (known plaintext problem)**. Наконец, в третьем случае криптоаналитик может зашифровать любой произвольно выбранный им сегмент открытого текста. Это называют **задачей с выбранным открытым текстом (chosen plaintext problem)**. Если бы криптоаналитикам было позволено задавать вопросы типа: «Как будет выглядеть зашифрованный результат для текста ABCDEFGHJKL?», криптограммы из газет решались бы очень легко.

Новички в криптографии часто полагают, что шифр достаточно надежен, если он может выдержать атаку первого типа (только зашифрованный текст). Такое предположение весьма наивно. Во многих случаях криптоаналитик может догадаться, как будут выглядеть определенные фрагменты зашифрованного текста. Например, после загрузки большинство компьютеров выводит на экран приглашение для ввода имени пользователя. Наличие нескольких пар сопоставленных друг с другом фрагментов зашифрованного и открытого текста может существенно упростить стоящую перед криптоаналитиком задачу. Для надлежащей защиты криптограф должен предусмотреть некоторый запас прочности и убедиться, что систему нельзя взломать, даже если его оппонент зашифрует произвольно выбранные фрагменты открытого текста.

Исторически методы шифрования разделились на две категории: метод подстановки и метод перестановки. Мы кратко рассмотрим их в качестве введения в современную криптографию.

## 8.4.2. Два базовых принципа криптографии

Хотя далее мы рассмотрим много различных криптографических систем, в основе каждой из них лежат два важнейших для понимания принципа. Обратите особое внимание — нарушать их очень опасно.

### *Избыточность*

Первый принцип гласит, что все зашифрованные сообщения должны содержать некую избыточность, то есть информацию, которая не требуется для понимания

сообщения. Приведем пример. Представьте компанию «Домосед», торгующую по почте товарами 60 000 наименований. Радуясь, что им удастся так экономно распорядиться ресурсами, программисты компании решили, что весь бланк заказа будет состоять из 16 байт для имени клиента, за которым последует поле товара из 3 байт (1 байт для обозначения количества и 2 байта для идентификатора товара). Последние 3 байта решено закодировать с помощью очень длинного ключа, известного только клиенту и компании «Домосед».

На первый взгляд эта схема кажется надежной, в частности потому, что пассивные злоумышленники не смогут расшифровать сообщения. К сожалению, в этой схеме имеется критический недостаток, полностью ее обесценивающий. Предположим, какой-нибудь недавно уволенный сотрудник захочет отомстить компании «Домосед» за увольнение. Перед уходом ему удается забрать с собой часть списка клиентов. За ночь он создает программу, отправляющую фиктивные заказы с настоящими именами клиентов. Поскольку списка ключей у него нет, он просто помещает в последние 3 байта случайные числа и отсылает сотни заказов компании «Домосед».

Получив эти сообщения, компьютер компании «Домосед» находит ключ для дешифрования по имени клиента. К несчастью для компании, почти все 3-байтные сообщения могут восприниматься как достоверные, поэтому компьютер начинает печатать заявки на доставку товаров. Может показаться странным, что клиент заказал 837 сидений для детских качелей или 540 песочниц, но вполне возможно, что он занимается строительством детских игровых площадок. Таким образом, активный злоумышленник (уволенный сотрудник) способен доставить очень много неприятностей, даже не вникая в смысл сообщений, отсылаемых его компьютером.

Эту проблему можно решить, добавив избыточную информацию к каждому сообщению. Если добавить к трем шифруемым байтам еще девять (например, нулевых), бывший сотрудник уже не сможет сформировать серьезный поток сообщений, которые достоверно выглядят. Мораль истории в том, что все сообщения должны содержать достаточное количество избыточной информации, чтобы активный злоумышленник не смог выдать случайный мусор за настоящие сообщения. Итак:

*Криптографический принцип номер 1: сообщения должны содержать избыточные данные.*

С другой стороны, наличие избыточной информации облегчает работу криптоаналитика по взлому шифра. Предположим, что конкуренция в бизнесе почтовых заказов чрезвычайно высока и что главный конкурент «Домоседа», фирма «Лежебока», очень хочет узнать объем продаж песочниц «Домоседа» в месяц. Для этого «лежебоки» подключились к телефонной линии «домоседов». В исходной схеме с 3-байтными номерами криптоанализ был почти невозможен, поскольку, предположив значение ключа, криптоаналитик не мог проверить правильность догадки, ведь почти все сообщения были технически корректны. С введением новой 12-байтной схемы криптоаналитик легко сможет отличить допустимое сообщение от недопустимого.

Другими словами, при расшифровке сообщения получатель должен иметь возможность проверить его подлинность путем анализа и, возможно, выполнения несложных вычислений. Избыточность требуется для того, чтобы можно было противостоять попыткам активных злоумышленников обмануть получателя фальшивыми сообщениями, содержащими мусор.

Вместе с тем добавление избыточной информации облегчает пассивным злоумышленникам задачу взлома системы, так что здесь есть определенное противоречие. Кроме того, избыточные данные никогда не должны принимать форму последовательности нулей в начале или в конце сообщения, так как при зашифровке таких сообщений некоторые алгоритмы дают более предсказуемые результаты, что облегчает работу криптоаналитиков. Многочлен циклического избыточного кода CRC (см. главу 3) намного больше подходит для этих целей, чем просто ряд нулей, ведь получатель может легко проверить его корректность, и это несколько усложняет задачу взломщика. Еще лучше использовать криптографическую хеш-функцию, речь о которой пойдет ниже. Пока просто запомните, что это более надежный вариант CRC.

### *Ограниченный срок годности*

Второй принцип криптографии состоит в следующем. Необходимо принять меры и удостовериться, что входящее сообщение является свежим, то есть было отправлено недавно. Этот принцип направлен на борьбу с активными злоумышленниками, которые воспроизводят перехваченные ими старые сообщения. Если этого не сделать, уволенный сотрудник может подключиться к телефонной линии компании «Домосед» и просто повторить отправленные ранее настоящие сообщения. Итак, сформулируем утверждение:

*Криптографический принцип номер 2: нужен способ борьбы с повторной отправкой старых сообщений.*

Один из способов — включить в каждое сообщение временную метку, которая действительна, скажем, только 60 с. Получатель просто хранит принятые сообщения в течение этого времени, отсеивая дубликаты. Сообщения старше 60 с игнорируются. Этот интервал не должен быть слишком коротким (например, 5 с), ведь не исключено, что системные часы отправителя и получателя работают с некоторым рассогласованием. Другие меры защиты от дубликатов представлены ниже.

## 8.4.3. Подстановочные шифры

При использовании **подстановочного шифра (substitution cipher)** каждый символ или группа символов заменяются другим символом или группой символов. Один из древнейших шифров такого рода — **шифр Цезаря (Caesar cipher)**, изобретение которого приписывают римскому императору. Этот шифр заменяет все буквы алфавита на другие с помощью циклического сдвига на три позиции. Так, буква *a* становится буквой *D*, *b* — *E*, *c* — *F*, ... и *z* — *C*. К примеру, слово *attack*

превращается в *DWWDFN*. Мы будем обозначать открытый текст строчными символами, а зашифрованный — прописными.

Некоторое обобщение шифра Цезаря представляет собой сдвиг алфавита не на три символа, а на произвольное число символов $k$. В этом случае $k$ становится ключом к общему методу циклически сдвигаемых алфавитов. Может, шифр Цезаря и обманул жителей Помпеи, но с тех пор ему уже никого не удалось ввести в заблуждение.

Следующее усовершенствование заключается в установлении соответствия между каждым символом в открытом тексте (для простоты, скажем, это 26 букв) и каким-либо другим. Например,

| | |
|---|---|
| открытый текст: | a b c d e f g h i j k l m n o p q r s t u v w x y z |
| зашифрованный текст: | Q W E R T Y U I O P A S D F G H J K L Z X C V B N M |

Такая система называется **моноалфавитным подстановочным шифром (monoalphabetic substitution cipher)**, ключом к которому является 26-символьная строка, соответствующая полному алфавиту. В нашем случае открытый текст *attack* будет преобразован в зашифрованный текст *QZZQEA*.

На первый взгляд эта система может показаться надежной, ведь даже если криптоаналитику известен общий принцип шифрования (подстановка одного символа вместо другого), он все равно не знает, какой из $26! \approx 4 \times 10^{26}$ возможных вариантов ключа следует применить. В отличие от шифра Цезаря применение метода простого перебора в данном случае весьма сомнительно. Даже при затратах 1 нс на проверку одного варианта ключа миллиону параллельно работающих процессорных ядер понадобится около 10 000 лет, чтобы перепробовать их все.

Тем не менее подобный шифр легко взламывается даже при наличии довольно небольших фрагментов зашифрованного текста. Для атаки может быть использовано преимущество статистических характеристик естественных языков. Например, в английском языке буква *e* встречается в тексте чаще всего. Следом за ней по частоте использования идут буквы *t*, *o*, *a*, *n*, *i* и т. д. Самыми распространенными комбинациями из двух символов, **биграммами (digrams)**, являются *th*, *in*, *er*, *re* и *an*. Наиболее часто встречающимися комбинациями из трех символов, **триграммами (trigrams)**, являются *the*, *ing*, *and* и *ion*.

Криптоаналитик, пытающийся взломать моноалфавитный шифр, прежде всего сосчитает относительные частоты всех символов алфавита в зашифрованном тексте. Затем он попробует заменить наиболее часто встречающийся символ буквой *e*, а следующий по частоте — буквой *t*. Затем он найдет триграммы самой распространенной формы *tXe*, где, скорее всего, *X* — это *h*. Аналогичным образом, если последовательность *thYt* встречается достаточно часто, то, вероятно, *Y* обозначает символ *a*. Обладая этой информацией, криптоаналитик может поискать часто встречающуюся триграмму вида *aZW*, что, скорее всего, означает *and*. Продолжая угадывать буквы, биграммы и триграммы и зная, какие последовательности символов являются наиболее вероятными, криптоаналитик побуквенно восстанавливает исходный текст.

Другой метод заключается в угадывании слова или фразы целиком. Например, рассмотрим следующий зашифрованный текст, полученный от бухгалтерской фирмы (разбитый на блоки по пять символов):

```
CTBMN BYCTC BTJDS QXBNS GSTJC BTSWX CTQTZ CQVUJ
QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ
DSKSU JSNTK BGAQJ ZBGYQ TLCTZ BNYBN QJSW
```

В сообщении бухгалтерской фирмы, скорее всего, есть слово *financial*. Используя тот факт, что в этом слове буква *i* встречается дважды, разделенная четырьмя другими буквами, мы будем искать в зашифрованном тексте повторяющиеся символы, отстоящие друг от друга на это расстояние. В результате мы найдем 12 таких мест в тексте в позициях 6, 15, 27, 31, 42, 48, 56, 66, 70, 71, 76 и 82. Однако только в двух случаях, в позициях 31 и 42, следующий символ (соответствующий букве *n* в открытом тексте) повторяется в соответствующем месте. Из этих двух вариантов символ *a* имеет правильное расположение только для позиции 31. Теперь нам известно, что слово *financial* начинается в позиции 30. С этого момента выяснить ключ проще, применяя лингвистическую статистику английского языка и угадывая целые слова.

## 8.4.4. Перестановочные шифры

Подстановочные шифры «маскируют» символы открытого текста, но не меняют порядок их следования. **Перестановочные шифры (transposition ciphers)**, наоборот, меняют порядок следования символов, но не «маскируют» их. На илл. 8.10 показан простой перестановочный шифр с колоночной перестановкой. Ключом к такому шифру служит слово или фраза, в которых нет повторяющихся букв. В данном примере в качестве ключа используется слово MEGABUCK. Цель ключа — пронумеровать колонки. Первой колонкой становится колонка под буквой, расположенной ближе всего к началу алфавита, и т. д. Открытый текст записывается горизонтально в строках. Зашифрованный текст читается по колонкам, начиная с колонки с младшей ключевой буквой.

| M | E | G | A | B | U | C | K |
|---|---|---|---|---|---|---|---|
| 7 | 4 | 5 | 1 | 2 | 8 | 3 | 6 |
| p | l | e | a | s | e | t | r |
| a | n | s | f | e | r | o | n |
| e | m | i | l | l | i | o | n |
| d | o | l | l | a | r | s | t |
| o | m | y | s | w | i | s | s |
| b | a | n | k | a | c | c | o |
| u | n | t | s | i | x | t | w |
| o | t | w | o | a | b | c | d |

Открытый текст

pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Зашифрованный текст

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

**Илл. 8.10.** Перестановочный шифр

Чтобы взломать перестановочный шифр, прежде всего криптоаналитик должен понять, что он имеет дело именно с таким способом шифрования. Если посмотреть, насколько часто здесь употребляются символы *E*, *T*, *A*, *O*, *I*, *N* и т. д., можно легко заметить, что частота их употребления такая же, как в обычном открытом тексте. Становится ясно, что мы имеем дело с перестановочным шифром, поскольку при его использовании буквы представляют сами себя, и, соответственно, распределение частот остается неизменным.

Затем нужно угадать число колонок. Во многих случаях по контексту сообщения можно угадать слово или фразу. К примеру, криптоаналитик подозревает, что где-то в сообщении должно встретиться словосочетание *milliondollars*. Обратите внимание, что из-за присутствия этих слов в исходном тексте в зашифрованном тексте встречаются биграммы *MO*, *IL*, *LL*, *LA*, *IR* и *OS*. Символ *O* следует за символом *M* (то есть они стоят рядом по вертикали в колонке 4), поскольку они разделены в предполагаемой фразе дистанцией, равной длине ключа. Если бы использовался ключ длиной семь, тогда вместо перечисленных выше биграмм встречались бы следующие: *MD*, *IO*, *LL*, *LL*, *IA*, *OR* и *NS*. Таким образом, для каждой длины ключа в зашифрованном тексте образуется новый набор биграмм. Перебрав различные варианты, криптоаналитик зачастую довольно легко может определить длину ключа.

Остается узнать только порядок колонок. Если их число $k$ невелико, можно перебрать все $k(k-1)$ возможных комбинаций пар соседних колонок, сравнивая частоты образующихся биграмм со статистическими характеристиками английского языка. Пара с лучшим соответствием считается правильно позиционированной. Затем все оставшиеся колонки по очереди проверяются в сочетании с уже найденной парой. Предполагается, что колонка, в которой биграммы и триграммы дают максимальное совпадение со статистикой, является правильной. Весь процесс повторяется, пока не будет восстановлен порядок всех колонок. Есть шанс, что на данном этапе текст уже будет распознаваемым (например, если вместо слова *million* мы увидим *milloin*, то сразу станет ясно, где допущена ошибка).

Некоторые перестановочные шифры принимают блок фиксированной длины на входе и выдают блок фиксированной длины на выходе. Они полностью определяются списком, сообщающим порядок, в котором символы попадают в выходной блок. Например, шифр на илл. 8.10 можно рассматривать в виде шифра с 64-символьным блоком. Его выход описывается последовательностью чисел 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, ..., 62. То есть четвертая входная буква *a* первой появится на выходе, за ней последует двенадцатая *f* и т. д.

## 8.4.5. Одноразовые блокноты

Разработать шифр, который невозможно взломать, на самом деле довольно легко. Метод его создания известен на протяжении уже нескольких десятилетий. В качестве ключа выбирается произвольная битовая строка, длина которой совпадает с длиной исходного текста. Открытый текст также преобразуется в последовательность двоичных разрядов, например, с помощью стандартной

кодировки ASCII. Наконец, две эти строки поразрядно складываются по модулю 2 (операция XOR). Полученный в итоге зашифрованный текст невозможно взломать, поскольку в достаточно большом отрывке любая буква, биграмма или триграмма будет равновероятной. Этот метод называется **одноразовым блокнотом (one-time pad)** и теоретически является панацеей от любых атак (как существующих сегодня, так и будущих), независимо от вычислительных мощностей, которыми обладает взломщик. Объясняется это теорией информации: дело в том, что в зашифрованном сообщении не содержится никаких сведений для взломщика, поскольку любой открытый текст нужной длины является равновероятным кандидатом.

Пример практического использования одноразового блокнота показан на илл. 8.11. Для начала фраза «I love you» («Я люблю тебя») преобразуется в 7-битный ASCII-код. Затем составляется некий одноразовый блокнот 1, который складывается по модулю 2 с сообщением. В результате получается шифр. Чтобы его разгадать, криптоаналитику придется перебрать все возможные варианты одноразового блокнота, всякий раз проверяя, каким получается открытый текст. Например, если попробовать расшифровать послание с помощью блокнота 2 (см. илл. 8.11), получится текст «Elvis lives» («Элвис жив»). На самом деле для генерации любой последовательности из 11 символов в кодировке ASCII найдется одноразовый блокнот. Именно это мы имеем в виду, говоря, что в зашифрованном тексте не содержится никаких сведений: из него можно извлечь любое сообщение подходящей длины.

Одноразовые блокноты теоретически являются очень мощным инструментом, но на практике они имеют ряд недостатков. Во-первых, такой длинный ключ невозможно запомнить, поэтому и отправитель и получатель должны носить с собой письменную копию ключа. Если есть риск того, что одну из копий захватит взломщик, хранить письменные копии весьма нежелательно. Кроме того, полный объем данных, которые можно передать, ограничен размером доступного ключа. Может произойти ситуация, в которой шпион добудет много информации, но не сможет передать все эти сведения в центр, так как ему не хватит длины ключа. Еще одна проблема заключается в чувствительности метода к потерянным или вставленным символам. Если отправитель и получатель потеряют синхронизацию, все данные, начиная с этого места, будут искажены.

| | |
|---|---|
| Сообщение 1: | 1001001 0100000 1101100 1101111 1110110 1100101 0100000 1111001 1101111 1110101 0101110 |
| Блокнот 1: | 1010010 1001011 1110010 1010101 1010010 1100011 0001011 0101010 1010111 1100110 0101011 |
| Зашифрованный текст: | 0011011 1101011 0011110 0111010 0100100 0000110 0101011 1010011 0111000 0010011 0000101 |
| | |
| Блокнот 2: | 1011110 0000111 1101000 1010011 1010111 0100110 1000111 0111010 1001110 1110110 1110110 |
| Открытый текст 2: | 1000101 1101100 1110110 1101001 1110011 0100000 1101100 1101001 1110110 1100101 1110011 |

**Илл. 8.11.** Использование одноразового блокнота для шифрования сообщений и возможность получения произвольного открытого сообщения из зашифрованного текста с помощью другого блокнота

С появлением компьютеров метод одноразового блокнота мог бы получить практическое применение. Ключ можно хранить на специальном DVD-диске,

содержащем несколько гигабайтов информации. Это даже не вызвало бы особых подозрений, если записать перед ключом несколько минут фильма и положить диск в обычную коробку от DVD-диска. Конечно, в гигабитных сетях необходимость вставлять новый DVD-диск каждые 30 с быстро утомит. Получается, что диск с одноразовым ключом должен быть доставлен от отправителя к получателю еще до передачи сообщений, что делает этот подход весьма непрактичным. Также следует учесть, что DVD- и Blu-Ray-диски скоро выйдут из употребления, и на человека с таким диском в руках все будут смотреть с подозрением.

### *Квантовая криптография*

Интересно, что решение проблемы передачи по сети одноразового блокнота пришло из совершенно неожиданного источника — квантовой механики. Эта область все еще является экспериментальной, но при этом многообещающей. Если получится усовершенствовать данный метод, все задачи криптографии можно будет решать с помощью одноразовых блокнотов, ведь это самый надежный способ защиты информации. Ниже мы вкратце опишем суть технологии, называемой **квантовой криптографией**. В частности, мы рассмотрим протокол **BB84**, названный так в честь его создателей и года, в котором его описание было впервые опубликовано в работе Беннета и Брассара (Bennet and Brassard, 1984).

Допустим, пользователь Алиса хочет передать одноразовый блокнот другому пользователю, Бобу. Алиса и Боб называются **принципалами (principals)**, это главные герои в нашей истории. К примеру, Боб может быть банкиром, с которым хочет сотрудничать Алиса. Имена «Алиса» и «Боб» традиционно используются для обозначения принципалов практически во всех материалах, касающихся криптографии, с тех пор как Рон Ривест (Ron Rivest) использовал их впервые много лет назад (Ривест и др.; Rivest et al., 1978). Криптографы вообще обожают разного рода традиции. Если бы мы описали взаимоотношения Алекса и Барбары, нам бы никто не поверил (и стало бы понятно, что автор на самом деле далек от криптографии). А так, может быть, поверят. Поэтому пусть Алиса и Боб будут героями нашей книги.

Итак, если Алисе и Бобу удастся принять некоторый единый одноразовый блокнот, их переговоры будут полностью конфиденциальными. При этом возникает очевидный вопрос: как им обменяться секретным ключом, не используя физические носители (DVD-диск или USB-накопитель)? Мы можем предположить, что пользователи находятся на разных концах одного оптоволоконного кабеля, по которому они могут передавать и принимать световые импульсы. Бесстрашная шпионка Труди установила на пути этого кабеля активное подслушивающее устройство. Она может считывать сигналы, идущие в обоих направлениях. К тому же она может передавать в обе стороны фальшивые сообщения. Ситуация для Алисы и Боба, казалось бы, безнадежная, но на помощь приходит квантовая криптография.

Квантовая криптография основана на том, что световые лучи состоят из микроскопически малых порций, **фотонов**, обладающих рядом специфических свойств. Кроме того, пропуская свет через поляризационный фильтр, можно добиться его поляризации. Это знают те, кто носит солнцезащитные очки,

и фотографы. Световой луч (то есть поток фотонов), проходя через такой фильтр, поляризуется в направлении оси фильтра (например, вертикально). Если после этого пропустить луч через второй фильтр, интенсивность света на выходе будет пропорциональна квадрату косинуса угла между осями фильтров. Если оси расположить перпендикулярно, фотоны не смогут проникнуть через фильтры. Абсолютная ориентация осей в пространстве значения не имеет — важно только их взаимное расположение.

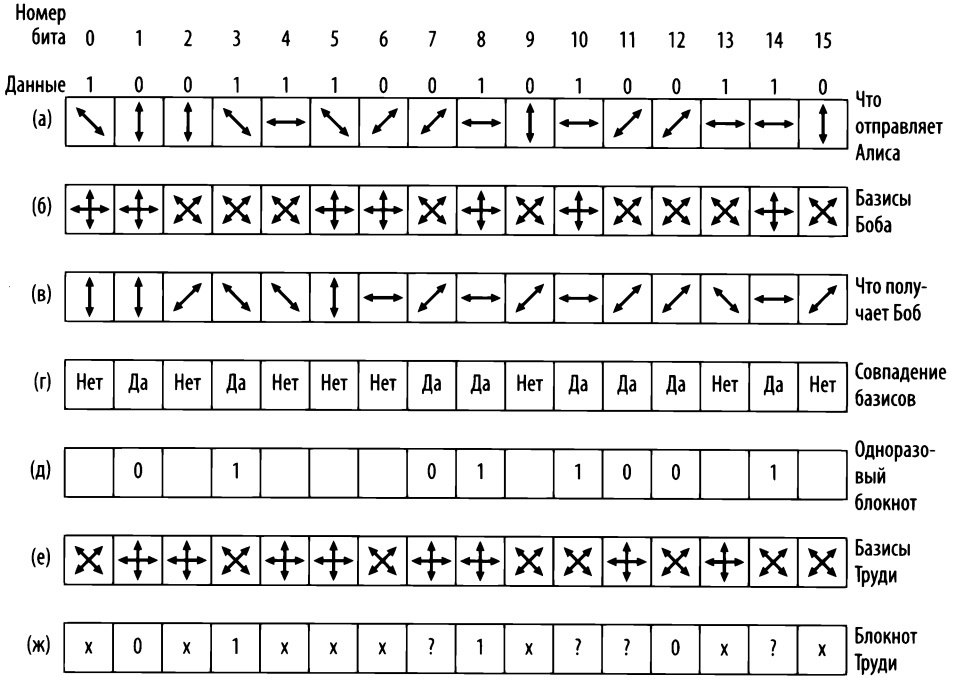
Чтобы сгенерировать одноразовый блокнот, Алисе понадобятся два набора поляризационных фильтров. Первый набор состоит из вертикального и горизонтального фильтров и называется **прямолинейным базисом (rectilinear basis)**. Базис — это просто система координат. Второй набор отличается от первого только тем, что он повернут на 45 градусов; то есть один фильтр можно представить в виде линии, идущей из нижнего левого угла в верхний правый, а другой — в виде линии из верхнего левого в нижний правый угол. Это **диагональный базис (diagonal basis)**. Итак, у Алисы есть два набора фильтров, и она может поставить любой из них на пути светового луча. На самом деле у нее не четыре отдельных фильтра, а кристалл, поляризация которого может на огромной скорости переключаться в одну из четырех позиций с помощью электричества. У Боба имеется такое же устройство. Тот факт, что у Алисы и Боба есть по два базиса, играет важную роль в квантовой криптографии.

В каждом базисе Алиса обозначает одно из направлений нулем, а другое единицей. В нашем примере вертикальному направлению она присвоила значение 0, а горизонтальному — 1. Затем, независимо от этого, для направления «нижний левый — верхний правый» Алиса выбрала значение 0, а для направления «верхний левый — нижний правый» — 1. Она отправляет эти варианты Бобу в виде открытого текста, прекрасно понимая, что ее сообщение может прочитать злоумышленник.

Теперь Алиса составляет одноразовый блокнот, допустим, с помощью генератора случайных чисел (это отдельная сложная тема), и передает его Бобу. Передача производится поразрядно, для каждого бита один из двух базисов выбирается случайным образом. Для передачи бита фотонная пушка испускает один фотон, поляризованный так, чтобы он мог пройти через базис, выбранный для этого бита. Например, базисы могут выбираться в такой последовательности: диагональный, прямолинейный, прямолинейный, диагональный, прямолинейный и т. д. Чтобы с помощью этих базисов передать одноразовый блокнот, состоящий из последовательности 1001110010100110, посылаются фотоны, показанные на илл. 8.12 (а). Для конкретного одноразового блокнота и последовательности базисов поляризация, которая используется для каждого бита, определяется однозначно. Биты, передаваемые одним фотоном за один раз, называются **кубитами (qubits)**.

Боб не знает, какой базис нужно использовать, поэтому он применяет их в случайном порядке для каждого прибывающего фотона; см. илл. 8.12 (б). Если базис для фотона выбран верно, Боб получает правильный бит. В противном случае значение бита будет случайным, так как фотон, проходя через поляризатор, повернутый на 45 градусов относительно его собственной поляризации, с одинаковой вероятностью попадет на направление, соответствующее единице

или нулю. Эта особенность фотонов является фундаментальным свойством в квантовой механике. Таким образом, некоторые биты будут получены правильно, некоторые — нет, но Боб не понимает, какие из них корректны. Полученный им результат показан на илл. 8.12 (в).




**Илл. 8.12.** Пример квантовой криптографии

Чтобы выяснить, какие базисы подставлены правильно, а какие — нет, Боб открытым текстом сообщает Алисе, что именно он использовал при приеме каждого бита. Затем она отвечает ему (также открытым текстом), какие базисы он подобрал верно (илл. 8.12 (г)). Владея этой информацией, Алиса и Боб могут составить битовую строку корректных предположений (илл. 8.12 (д)). В среднем длина этой строки равна половине полной длины исходной строки, но поскольку это знают обе стороны, они могут использовать строку корректных предположений в качестве одноразового блокнота. Все, что Алисе надо сделать, — это передать битовую строку, длина которой немного превышает удвоенную длину одноразового блокнота. Проблема решена.

Но погодите, мы же забыли про Труди! Предположим, что ей очень хочется узнать, о чем говорит Алиса, поэтому она внедряет в линию связи свой детектор и передатчик. К несчастью для Труди, она тоже не знает, через какой базис пропускать каждый фотон. Лучшее, что она может сделать, — это выбрать базисы случайным образом, как Боб (илл. 8.12 (е)). Когда Боб открытым текстом сообщает Алисе, какие базисы он использовал, а та отвечает ему, какие из них

верны, Труди, как и Боб, узнает, какие биты она угадала, а какие — нет. Как видно из рисунка, базисы Труди совпали с базисами Алисы в позициях 0, 1, 2, 3, 4, 6, 8, 12 и 13. Однако из ответа Алисы (илл. 8.12(г)) ей становится известно, что в одноразовый блокнот входят только биты 1, 3, 7, 8, 10, 11, 12 и 14. Четыре из них (1, 3, 8 и 12) были угаданы правильно, Труди их запоминает. Остальные биты (7, 10, 11 и 14) не были угаданы, и их значения остаются для Труди неизвестными. Таким образом, Бобу известен одноразовый блокнот, начинающийся с последовательности 01011001 (илл. 8.12 (д)), а все, что досталось Труди, — это обрывок 01?1??0? (илл. 8.12 (ж)).

Конечно, Алиса и Боб осознают, что Труди пытается захватить часть их одноразового блокнота, поэтому они стараются уменьшить количество информации, которая может ей достаться. Для этого они могут внести дополнительные изменения. Например, одноразовый блокнот можно поделить на блоки по 1024 бита и возвести каждый из блоков в квадрат, получая, таким образом, числа длиной 2048 бит. Затем можно использовать конкатенацию 2048-битных чисел в качестве одноразового блокнота. Имея лишь часть битовой строки, Труди никогда не сможет проделать эти преобразования. Действия над исходным одноразовым блокнотом, уменьшающие долю информации, получаемой Труди, называются **усилением секретности (privacy amplification)**. На практике вместо поблочного возведения в квадрат применяются сложные преобразования, в которых каждый выходной бит зависит от каждого входного.

Бедная Труди. Помимо того что она не представляет, как выглядит одноразовый блокнот, она понимает, что ее присутствие ни для кого не секрет. В конечном итоге ей приходится передавать каждый принятый бит Бобу, чтобы он думал, будто разговаривает с Алисой. Проблема в том, что лучшее, что может сделать Труди, — это передавать каждый принятый квантобит с использованием поляризации, с помощью которой он был принят ею. При этом примерно в половине случаев поляризация будет неправильной, что приведет к появлению множества ошибок в одноразовом блокноте Боба.

Когда, наконец, Алиса начинает отправлять данные, она шифрует их, используя сложный код с упреждающей коррекцией ошибок. С точки зрения Боба, ошибка в одном бите одноразовой последовательности — это то же самое, что ошибка передачи данных, произошедшая в одном бите. В любом случае результат состоит в получении неправильного значения бита. С помощью упреждающей коррекции ошибок, возможно, удастся восстановить потерянную информацию, но помимо этого можно посчитать ошибки. Если их количество намного превышает ожидания (связанные с вероятностью возникновения ошибок оборудования), становится очевидно, что линию прослушивает Труди, и Боб может принять меры (например, сказать Алисе, чтобы она переключилась на радиоканал, вызвать полицию и т. д.). Если бы Труди могла скопировать фотон и обработать свою копию, а исходный фотон переслать Бобу в неизменном виде, у нее был бы шанс остаться незамеченной, но на сегодняшний день методы клонирования фотонов отсутствуют. Но даже если Труди удастся получить копию фотона, это никак не уменьшит значение квантовой криптографии в создании одноразовых блокнотов.

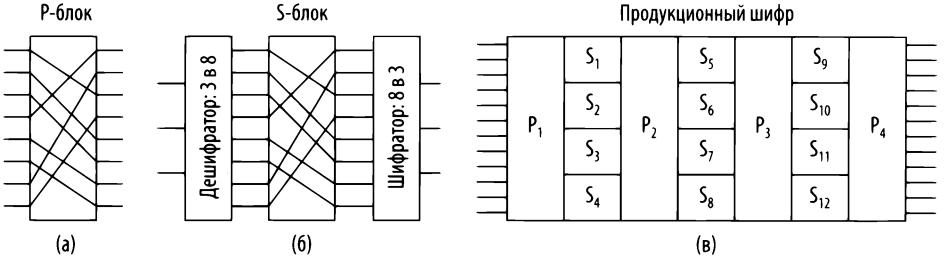
Хотя применение квантовой криптографии было продемонстрировано на примере 60-километрового оптоволокна, для этого требуется сложное и дорогое оборудование. В то же время сама идея выглядит многообещающе, особенно если решить проблемы с масштабированием и уровнем необходимых затрат. Чтобы узнать больше о квантовой криптографии, см. работу Клэнси и др. (Clancy et al., 2019).

## 8.5. АЛГОРИТМЫ С СИММЕТРИЧНЫМ КЛЮЧОМ

В современной криптографии применяются те же базовые принципы, что и в традиционной (перестановка и подстановка), но акценты расставлены иначе. Когда-то криптографы использовали простые алгоритмы шифрования. Сегодня, напротив, они стремятся создать настолько сложный и запутанный алгоритм, что даже если криптоаналитику попадут в руки целые горы зашифрованного текста, он не сможет извлечь из него никакой пользы без ключа.

Первый класс алгоритмов шифрования, который мы изучим, — **алгоритмы с симметричным ключом (symmetric-key algorithms)**. Их название говорит о том, что для шифрования и дешифрования сообщений применяется один и тот же ключ. На илл. 8.9 показан пример использования алгоритма с симметричным ключом. В частности, мы подробно рассмотрим **блочные шифры (block ciphers)**, которые принимают на входе $n$-битные блоки открытого текста и преобразуют их с использованием ключа в $n$-битный шифр.

Криптографические алгоритмы можно реализовать как аппаратно (что повышает скорость их работы), так и программно (для повышения гибкости). Несмотря на то что в основном мы рассматриваем алгоритмы и протоколы вне зависимости от конкретной реализации, в данном случае интересно обсудить шифровальное оборудование. Подстановки и перестановки могут осуществляться при помощи простых электрических цепей. На илл. 8.13 (а) показан **P-блок** (P означает «permutation» — «перестановка»). Это устройство используется для перестановки восьми входных разрядов. Если пронумеровать входные биты сверху вниз (01234567), выход данного P-блока будет выглядеть как 36071245. При определенной распайке проводов P-блок может выполнить любую операцию перестановки практически со скоростью света, так как никакие вычисления в нем не производятся, он лишь обеспечивает распространение сигнала.




**Илл. 8.13.** Базовые элементы продукционных шифров: (а) P-блок; (б) S-блок; (в) продукционный шифр

Это соответствует принципу Керкгоффса: взломщик знает, что используется метод перестановки битов, но он не знает, в каком порядке эти биты располагаются.

Подстановки выполняются **S-блоками** (S означает «substitution» — «подстановка, замена»), как показано на илл. 8.13 (б). В данном примере на вход подается 3-битный открытый текст, а на выходе появляется 3-битный зашифрованный текст. Для каждого входного сигнала выбирается одна из восьми выходных линий декодера путем установки ее в 1. Все остальные линии устанавливаются в 0. Затем эти восемь линий проходят через P-блок, представляющий собой вторую ступень S-блока. Третья ступень производит обратное кодирование одной из восьми линий в 3-битное двоичное число. При таком устройстве проводки восьмеричные числа 01234567 заменяются на 24506713 соответственно. То есть 0 заменяется числом 2, 1 — числом 4 и т. д. Опять же, при соответствующей распайке проводов P-блока внутри S-блока можно реализовать любой вариант подстановки. Помимо этого, P-блок может быть встроен в оборудование и работать на огромных скоростях, поскольку шифраторы и дешифраторы вносят лишь одну или две вентильные задержки (менее 1 нс), а время распространения сигнала внутри P-блока вполне может быть менее 1 пс.

Эффективность этих базовых элементов становится очевидной, если расположить каскадом целый ряд блоков (илл. 8.13 (в)), создав **продукционный шифр (product cipher)**. В нашем примере на первом этапе ($P_1$) 12 входных линий меняются местами. На второй ступени вход разбивается на четыре группы из 3 бит, с каждой из которых операция замены выполняется независимо ($S_1$–$S_4$). Такое расположение позволяет составить большой S-блок из множества мелких. Это хороший способ, так как небольшие S-блоки удобны при аппаратной реализации (к примеру, восьмиразрядный S-блок можно представить в виде 256-разрядной таблицы перекодировки), однако большие S-блоки довольно трудно построить (например, 12-разрядный S-блок потребует минимум $2^{12} = 4096$ перекрещенных проводов в средней стадии). Хотя этот метод является лишь частным случаем, он достаточно эффективный. Выход продукционного шифра можно сделать сложной функцией входа, используя достаточно большое количество дополнительных ступеней.

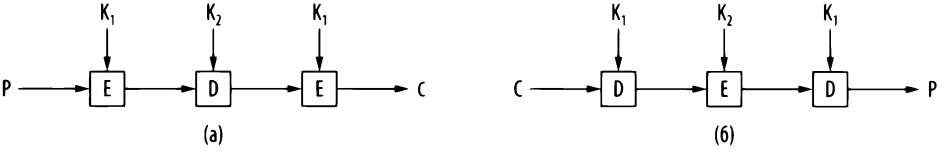
Продукционные шифры, работающие с $k$-битными входами и производящие $k$-битные последовательности, широко распространены. В частности, часто применяются шифры с $k$, равным 256. Аппаратная реализация обычно имеет не семь, как на илл. 8.13 (в), а по крайней мере 20 физических этапов. Программная реализация совершает минимум восемь циклических итераций, каждая из которых производит S-блочные подстановки в подблоках блока данных длиной от 64 до 256 бит, после чего производится перестановка, которая перемешивает результаты S-блоков. Часто также производятся две дополнительные перестановки в начале и в конце. В литературе эти итерации называются **раундами**.

## 8.5.1. Стандарт шифрования данных DES

В январе 1977 года правительство США приняло продукционный шифр, разработанный IBM, в качестве официального стандарта для незасекреченной информации. Этот шифр, получивший название **DES (Data Encryption**

**Standard — стандарт шифрования данных)**, широко применялся в промышленности для защиты информации. Хотя в своем исходном виде он уже небезопасен, его модифицированная версия по-прежнему изредка используется. Эффективность исходной версии сомнительна: изначально длина ключа равнялась 128 бит, но после консультаций с АНБ компания IBM «добровольно» уменьшила ее до 56 бит, что, по мнению криптографов, было слишком мало даже для того времени. Алгоритм DES действует практически так же, как показано на илл. 8.13 (в), но использует блоки большего размера. Открытый текст (в двоичном виде) разбивается на 64-битные блоки, каждый из которых шифруется по отдельности путем выполнения перестановок и подстановок, параметризованных 56-битным ключом в каждом из 16 последовательных раундов. По сути, это очень большой моноалфавитный подстановочный шифр на базе алфавита с 64-битными символами (о которых мы поговорим чуть позже).

Уже в 1979 году стало ясно, что 56-битный ключ слишком короткий, и IBM разработала обратно совместимую схему увеличения длины ключа за счет одновременного использования двух 56-битных ключей, что в совокупности давало 112-битный ключ (Такман; Tuchman, 1979). Эта новая схема, названная **тройным DES (Triple DES)**, используется до сих пор (илл. 8.14).




**Илл. 8.14.** (а) Тройное шифрование с помощью DES. (б) Дешифрование

Возникает два резонных вопроса. Первый: почему здесь используется два, а не три ключа? Второй: для чего выполняется шифрование, дешифрование и затем снова шифрование? Ответ следующий. В случае, когда компьютеру, использующему тройной DES, необходимо связаться с компьютером, на котором реализован одинарный DES, он может применить тройной DES, взяв в качестве обоих ключей одно и то же значение. Это даст тот же результат, что и применение одинарного DES. Такой подход упростил поэтапное внедрение тройного DES. Сегодня этот алгоритм можно считать устаревшим, но он по-прежнему используется в некоторых областях применения, которые трудно поддаются изменениям.

## 8.5.2. Улучшенный стандарт шифрования AES

Когда стало понятно, что DES (даже с тройным шифрованием) устаревает, **Национальный институт стандартов и технологий (National Institute of Standards and Technology, NIST)** — агентство Министерства торговли, занимающееся разработкой стандартов для Федерального правительства США, — решил создать новый криптографический стандарт для несекретных данных. Специалисты NIST ясно осознавали противоречия, связанные с DES. Они прекрасно понимали, что,

услышав о новом стандарте, все, кто хоть что-то смыслит в криптографии, решат, что и здесь есть лазейка, с помощью которой АНБ получит доступ к данным. Вряд ли в таких условиях люди согласятся применять новую технологию, и она, скорее всего, канет в Лету.

Исходя из этого, NIST выбрал неожиданное для правительственной бюрократии решение и организовал криптографический конкурс. В январе 1997 года ученые со всего мира были приглашены для представления своих разработок нового стандарта, который назвали **AES (Advanced Encryption Standard — продвинутый стандарт шифрования)**. Ниже перечислены требования к разработкам конкурсантов:

1. Алгоритм использует симметричный блочный шифр.
2. Все детали разработки общедоступны.
3. Поддерживаются длины ключей 128, 192 и 256 бит.
4. Возможна как программная, так и аппаратная реализация.
5. Алгоритм общедоступен или лицензирован на недискриминационных условиях.

Было рассмотрено 15 серьезных предложений. На открытых конференциях разработчики представляли свои проекты, а оппоненты должны были приложить максимум усилий, чтобы найти в них недостатки. В августе 1998 года NIST выбрал пять финалистов, руководствуясь критериями безопасности, эффективности, простоты, гибкости, а также требований к памяти (это важно для встроенных систем). Затем были проведены конференции, на которых высказывались дополнительные критические замечания.

В октябре 2000 года NIST объявил победителей конкурса — Йоана Дамена (Joan Daemen) и Винсента Рэймена (Vincent Rijmen), предложивших метод **Rijndael**. Название Rijndael (произносится примерно как «Райн-дол») представляет собой сокращение фамилий авторов разработки. В ноябре 2001 года Rijndael был объявлен стандартом правительства США и опубликован как FIPS 197[1]. Благодаря абсолютной открытости конкурса, техническим возможностям метода и тому факту, что победившая команда состояла из двух молодых бельгийских криптографов (которые вряд ли стали бы сотрудничать с NSA, предоставляя какие-то лазейки), Rijndael стал главным мировым криптографическим стандартом. AES в настоящее время является частью набора инструкций для некоторых процессоров.

Rhindael поддерживает длины ключей и размеры блоков от 128 до 256 бит с шагом в 32 бита. Длины ключей и блоков могут выбираться независимо друг от друга. При этом AES задает размер блока в 128 бит, а длину ключа — в 128, 192 или 256 бит. Вряд ли кто-то будет использовать 192-битные ключи, поэтому фактически есть два варианта AES: 128-битный блок с 128-битным ключом и 128-битный блок с 256-битным ключом.

---

[1] Federal Information Processing Standard — Федеральный стандарт обработки информации.

Ниже мы рассмотрим только один вариант алгоритма — 128/128, поскольку это стандарт для коммерческих приложений. При использовании 128-битного ключа размер ключевого пространства составляет $2^{128} \approx 3 \times 10^{38}$ ключей. Даже если АНБ удастся собрать компьютер с миллионом параллельных процессоров, каждый из которых способен вычислять один ключ в пикосекунду, на перебор всех значений потребуется около $10^{10}$ лет. К тому времени Солнце уже давно потухнет, и нашим потомкам придется читать распечатку ключа при свечах.

### *Rijndael*

С математической точки зрения Rijndael основан на теории полей Галуа, благодаря чему можно строго доказать некоторые его свойства, касающиеся секретности. Также можно рассмотреть его как код на языке C, не вдаваясь в математические подробности.

Как и в DES, в Rijndael применяются подстановки и перестановки. И там и там используется несколько итераций, число которых зависит от размера ключа и блока и равно 10 для 128-разрядного ключа и 128-битных блоков (при максимальном размере ключа и блоков число итераций равно 14). Однако, в отличие от DES, все операции выполняются над целым количеством байтов, что позволяет создавать эффективные реализации как в аппаратном, так и в программном исполнении. Из-за того что DES ориентирован на использование битов, его программные реализации работают медленно.

Алгоритм Rijndael обеспечивает не только высокую степень безопасности, но и отличную скорость. Хорошая программная реализация на компьютере с частотой 2 ГГц может шифровать данные со скоростью 700 Мбит/с. Этого достаточно для шифрования более десяти видеофайлов в формате 4K в режиме реального времени. Аппаратные реализации работают еще быстрее.

## 8.5.3. Режимы шифрования

Несмотря на всю эту сложность, по сути, AES (а также DES и любой другой блочный код) является моноалфавитным подстановочным шифром с большими длинами символов (128-битные символы в AES, 64-битные — в DES). Для любого отрывка открытого текста шифр при прогоне через один и тот же шифрующий блок будет всегда одинаковым. Скажем, если вы 100 раз зашифруете открытый текст *abcdefgh*, используя алгоритмы DES или AES с одним и тем же ключом, то вы 100 раз получите на выходе одинаковый зашифрованный текст. Взломщик может попытаться использовать это свойство при попытке расшифровки текста.

### *Режим электронной кодовой книги*

Чтобы понять, как использовать это свойство моноалфавитного подстановочного шифра для частичного взлома шифра, мы рассмотрим кодирование (тройное) по стандарту DES (поскольку изображать 64-разрядные блоки проще, чем 128-разрядные). Имейте в виду, что AES сталкивается с такими же проблемами. Самый очевидный способ кодирования длинного сообщения заключается в разбиении

его на отдельные блоки по 8 байт (64 бита) и кодировании этих блоков одним и тем же ключом по очереди. Последний блок при необходимости можно дополнить до 64 бит. Этот метод назван **режимом ECB (Electronic Code Book mode — режим электронной кодовой книги)** по аналогии с традиционными кодовыми книгами, в которых содержались слова и соответствующие им шифры (обычно пятизначные десятичные числа).

На илл. 8.15 показано начало компьютерного файла; это список годовых премий сотрудников компании. Файл состоит из последовательных 32-разрядных записей (по одной на сотрудника) следующего формата: 16 байт на имя, 8 байт на должность и 8 байт на премию. Каждый из шестнадцати 8-байтных блоков (пронумерованных от 0 до 15) кодируется шифром DES.

**Илл. 8.15.** Открытый текст файла, зашифрованного в виде 16 DES-блоков

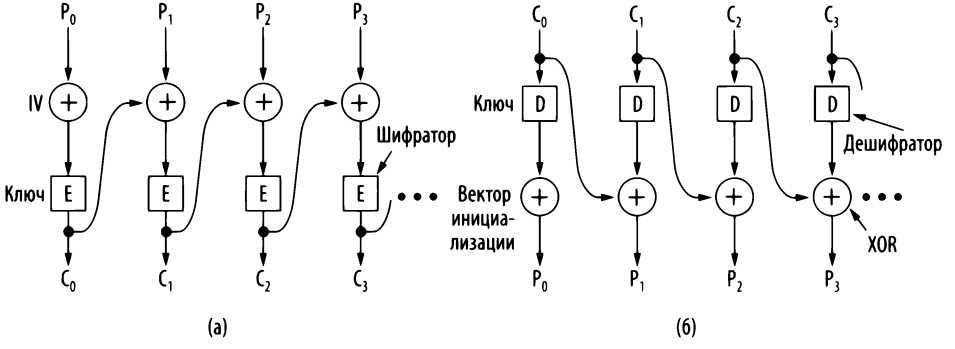
Лесли только что поругалась с начальством и не рассчитывает на большую премию. Ким, напротив, — любимица босса, и все это знают. Лесли может получить доступ к файлу после его зашифровки, но до его отправки в банк. Может ли Лесли исправить ситуацию, имея доступ только к зашифрованному файлу?

Это не проблема. Все, что нужно сделать, — это скопировать зашифрованный блок 12 (с премией Ким) и заменить им блок 4 (с премией Лесли). Даже не зная содержимого блока 12, Лесли может рассчитывать на гораздо более веселое Рождество. (Можно скопировать и зашифрованный блок 8, но скорее всего это обнаружат; да и Лесли, в общем-то, не жадная.)

### *Режим сцепления блоков шифра*

Чтобы противостоять подобным атакам, все блочные шифры можно модернизировать так, чтобы замена одного блока вызывала повреждение других блоков открытого текста после их расшифровки, превращая их (начиная с модифицированного места) в мусор. Один из таких методов — **сцепление блоков шифра (cipher block chaining)** (илл. 8.16). Перед зашифровкой каждый блок открытого текста складывается по модулю 2 с предыдущим уже зашифрованным блоком. Следовательно, блокам открытого текста не соответствуют одни и те же блоки зашифрованного текста, и этот метод не является большим моноалфавитным подстановочным шифром. Первый блок складывается по модулю 2 со случайно

выбранным **вектором инициализации (Initialization Vector, IV)**, который передается в виде открытого текста вместе с зашифрованными блоками.




**Илл. 8.16.** Сцепление блоков шифра. (а) Шифрование. (б) Дешифрование

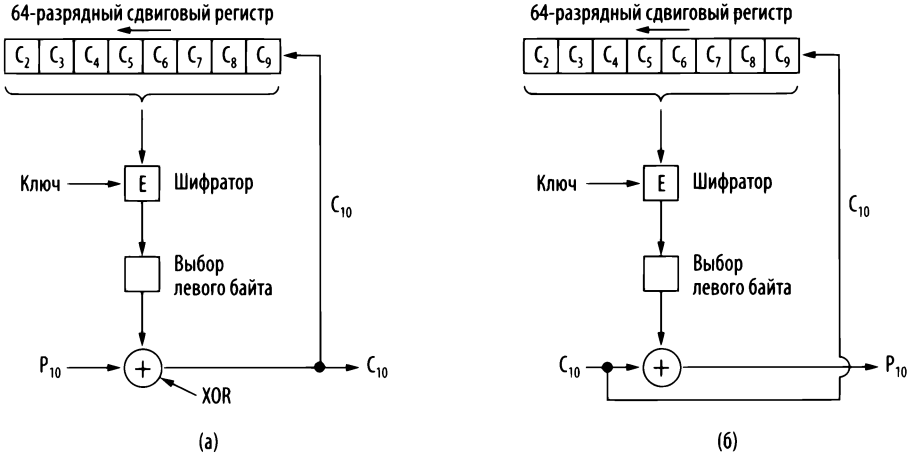
Рассмотрим сцепление блоков шифра на примере, представленном на илл. 8.16. Сначала мы вычисляем $C_0 = E(P_0 \text{ XOR } IV)$, затем $C_1 = E(P_1 \text{ XOR } C_0)$ и т. д. При дешифровании мы также используем операцию XOR, чтобы обратить этот процесс: $P_0 = IV \text{ XOR } D(C_0)$ и т. д. Следует заметить, что блок $i$ является функцией всех блоков открытого текста с 0 по $i - 1$, поэтому один и тот же исходный блок текста преобразуется в разные зашифрованные блоки в зависимости от их расположения. При таком способе шифрования преобразование, произведенное Лесли, приведет к появлению двух блоков бессмысленного содержания, начиная с поля премии Лесли. Для сообразительного сотрудника службы безопасности эта странность может послужить подсказкой в последующем расследовании.

Сцепление блоков шифра также обладает преимуществом, при котором одни и те же блоки открытого текста преобразуются в разные зашифрованные блоки, что усложняет криптоанализ. По правде говоря, именно поэтому данный метод и используется.

### *Режим шифрованной обратной связи*

Однако у метода сцепления блоков шифра есть один недостаток: прежде чем начнется дешифрование, должен появиться целый 64-битный блок данных. Для побайтового шифрования может применяться режим **шифрованной обратной связи (cipher feedback mode)** с использованием DES (тройного), как показано на илл. 8.17. Для AES принцип тот же, только применяется 128-разрядный сдвиговый регистр. На рисунке мы видим состояние шифровальной машины после того, как байты с 0-го по 9-й уже зашифрованы и отправлены. Когда приходит 10-й байт открытого текста (илл. 8.17 (а)), алгоритм DES обрабатывает 64-разрядный сдвиговый регистр, чтобы произвести 64-разрядный зашифрованный блок. Самый левый байт этого зашифрованного текста извлекается, складывается по модулю 2 с $P_{10}$ и передается по линии. Затем сдвиговый регистр смещается

влево на 8 разрядов. При этом байт $C_2$ извлекается с левого конца регистра, а байт $C_{10}$ вставляется на освободившееся место справа от $C_9$.




**Илл. 8.17.** Режим шифрованной обратной связи. (а) Шифрование. (б) Дешифрование

Обратите внимание, что содержимое сдвигового регистра зависит от всей предыстории открытого текста, так что повторяющиеся фрагменты исходного текста будут кодироваться каждый раз по-новому. Как и в случае сцепления блоков шифра, для запуска процесса требуется вектор инициализации.

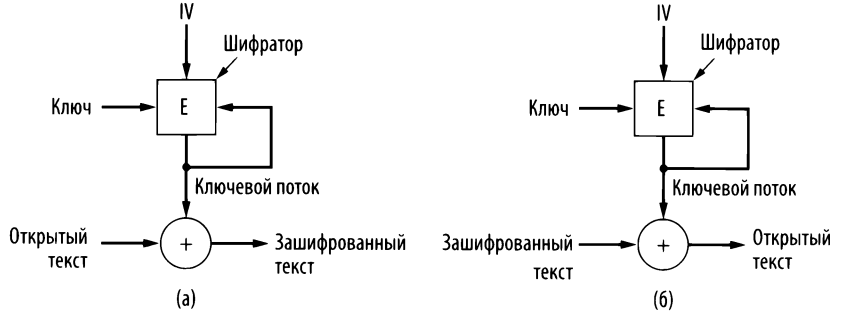
При использовании этого метода дешифрование аналогично шифрованию. В частности, содержимое сдвигового регистра *шифруется*, а не *дешифруется*, поэтому байт, который складывается по модулю 2 с $C_{10}$ для получения $P_{10}$, равен тому байту, который складывается по модулю 2 с $P_{10}$ для получения $C_{10}$. Пока содержимое двух сдвиговых регистров идентично, дешифрование выполняется корректно (илл. 8.17 (б)).

Проблемы с режимом шифрованной обратной связи возникают, когда при передаче зашифрованного текста один бит случайно инвертируется. При этом искажаются 8 байт, которые дешифруются во время нахождения поврежденного байта в сдвиговом регистре. Когда этот байт покидает сдвиговый регистр, открытый текст снова дешифруется корректно. Таким образом, последствия инверсии одного бита относительно локализованы: искажается столько битов, сколько помещается в сдвиговом регистре, остальная часть сообщения не повреждается.

### *Режим группового шифра*

Однако встречаются сценарии применения, в которых один испорченный при передаче бит приводит к порче 64 бит открытого текста, а это слишком много. Для таких случаев существует четвертый вариант, называемый **режимом группового (потокового) шифра (stream cipher mode)**. Его суть заключается в том, что выходной блок получают путем шифрования вектора инициализации

с использованием ключа. Затем этот выходной блок снова шифруется с помощью ключа, чтобы вычислить второй выходной блок, который, в свою очередь, шифруется для получения третьего, и т. д. Последовательность выходных блоков, называемая **ключевым потоком (keystream)**, может иметь произвольную длину. Она расценивается как одноразовый блокнот и складывается по модулю 2 с открытым текстом. В результате получается зашифрованный текст (илл. 8.18 (а)). Обратите внимание: вектор инициализации используется только на первом шаге. После этого шифруются выходные блоки. Кроме того, ключевой поток не зависит от данных, поэтому в случае необходимости он может быть вычислен заранее и совершенно нечувствителен к ошибкам передачи. Процесс дешифрования показан на илл. 8.18 (б).




**Илл. 8.18.** Групповой шифр. (а) Шифрование. (б) Дешифрование

Дешифрование происходит путем создания точно такого же ключевого потока на принимающей стороне. Поскольку он зависит только от вектора инициализации и ключа, ошибки передачи зашифрованного текста на него не влияют. Таким образом, ошибка в одном бите передаваемого шифра приводит к ошибке лишь в одном бите расшифрованного текста.

Важно никогда не использовать одну и ту же пару (ключ, вектор инициализации) в одном и том же групповом шифре, иначе каждый раз будет получаться одинаковый ключевой поток. Повторное использование ключевого потока может привести к взлому шифра при помощи **многократного использования ключевого потока (keystream reuse attack)**. Допустим, блок открытого текста $P_0$ шифруется с помощью ключевого потока, в результате вычисляется $P_0$ XOR $K_0$. Затем берется второй блок открытого текста $Q_0$ и шифруется тем же ключевым потоком (получаем $Q_0$ XOR $K_0$). Криптоаналитик, перехвативший оба блока зашифрованного текста, может просто сложить их вместе по модулю 2 и получить $P_0$ XOR $Q_0$, тем самым убирая ключ. Теперь у него есть XOR двух блоков открытого текста. Если один из них известен (или его можно угадать), найти второй не проблема. В любом случае взломать XOR двух блоков открытого текста можно, используя статистические свойства сообщения.

Скажем, при передаче английского текста чаще всего в потоке встречается XOR двух пробелов, за которой следует XOR одного пробела и буквы «е» и т. д.

Иными словами, имея XOR двух частей открытого текста, взломщик с высокой вероятностью вычислит обе части.

## 8.6. АЛГОРИТМЫ С ОТКРЫТЫМ КЛЮЧОМ

Исторически процесс передачи ключа всегда был слабым звеном почти во всех системах шифрования. Независимо от того, насколько прочна была сама криптосистема, если взломщик мог украсть ключ, система становилась бесполезной. До 1976 года все криптологи исходили из того, что ключ дешифрования должен быть идентичен ключу шифрования (или один можно легко получить из другого). При этом ключи должны быть у всех пользователей системы. Казалось, что проблема неустранима: необходимо защищать ключи от кражи и в то же время нужно распространять их среди пользователей, поэтому они не могут быть заперты в банковском сейфе.

В 1976 году два исследователя из Стэнфордского университета, Диффи (Diffie) и Хеллман (Hellman), предложили радикально новую криптосистему, в которой ключ дешифрования нельзя было получить из ключа шифрования — настолько они различались. Разработанные Диффи и Хеллманом алгоритмы шифрования $E$ и дешифрования $D$ (оба параметризованные ключом) должны были соответствовать следующим требованиям:

1. $D(E(P)) = P$.
2. Вывести $D$ из $E$ крайне сложно.
3. $E$ нельзя взломать при помощи произвольного открытого текста.

Первое требование состоит в том, что применив алгоритм дешифрования $D$ к зашифрованному сообщению $E(P)$, мы должны снова получить открытый текст $P$. Без этого авторизованный получатель просто не сможет расшифровать сообщение. Второй пункт говорит сам за себя. Третье требование необходимо, поскольку, как мы скоро увидим, взломщики могут экспериментировать с алгоритмом столько, сколько пожелают. В этих условиях нет никаких причин, по которым ключ шифрования нельзя было бы обнародовать.

Этот метод работает следующим образом. Допустим, Алиса, желая получать секретные сообщения, разрабатывает два алгоритма, удовлетворяющие перечисленным выше требованиям. Затем алгоритм шифрования и его ключ открыто публикуются; поэтому данный подход и называется **шифрованием с открытым ключом (public-key cryptography)**. Алиса может разместить открытый ключ, к примеру, на своей домашней страничке. Алгоритм шифрования, параметризованный этим ключом, мы обозначим как $E_A$. По аналогии алгоритм дешифрования (секретный), параметризованный персональным ключом Алисы, мы будем обозначать как $D_A$. Боб делает то же самое: обнародует $E_B$, но сохраняет в тайне $D_B$.

Теперь мы попробуем решить проблему установки надежного канала между Алисой и Бобом, которые ранее никогда не встречались. Оба ключа шифрования, $E_A$ и $E_B$, являются открытыми. Алиса берет свое первое сообщение $P$, вычисляет $E_B(P)$ и отправляет результат Бобу. Боб расшифровывает его с помощью своего

секретного ключа $D_B$, то есть вычисляет $D_B(E_B(P)) = P$. Никто другой не может прочитать зашифрованное сообщение $E_B(P)$, так как предполагается, что система шифрования надежна, а получить ключ $D_B$ на основании известного ключа $E_B$ очень трудно. Отсылая ответ $R$, Боб передает $E_A(R)$. Таким образом, Алиса и Боб получают надежный секретный канал связи.

Обратите внимание на используемую здесь терминологию. Шифрование с открытым ключом предполагает наличие у каждого пользователя двух ключей — открытого (все используют его для шифрования сообщений для этого пользователя) и закрытого (он нужен пользователю для дешифрования входящих сообщений). Мы будем и далее называть эти ключи *открытым (public)* и *закрытым (private)*, чтобы отличать их от *секретных (secret)* ключей, которые применяются для шифрования и дешифрования в обычной криптографии с симметричным ключом.

## 8.6.1. Алгоритм RSA

Единственная загвоздка состоит в том, чтобы найти алгоритмы, отвечающие всем трем требованиям. Преимущества шифрования с открытым ключом очевидны, поэтому многие исследователи неустанно работали над созданием таких алгоритмов (некоторые из них уже опубликованы). Группой исследователей Массачусетского технологического института был разработан один надежный метод (Ривест и др.; Rivest et al., 1978). Он получил название **RSA**, по начальным буквам фамилий трех авторов: Рона Ривеста, Ади Шамира и Леонарда Адлемана (Ron Rivest, Adi Shamir, Leonard Adleman). Вот уже четверть века этот метод выдерживает попытки взлома и считается очень надежным. На его основе построены многие практические системы безопасности. По этой причине Ривест, Шамир и Адлеман в 2002 году получили награду ACM — Премию Тьюринга (Turing Award). Главный недостаток RSA в том, что для обеспечения достаточного уровня защиты нужен ключ длиной по крайней мере 2048 бит (против 256 бит в алгоритмах с симметричными ключами). Из-за этого алгоритм работает довольно медленно.

В основе метода RSA лежат некоторые принципы теории чисел. Опишем в общих чертах, как пользоваться этим методом. Подробности можно найти в соответствующих источниках.

1. Выберем два больших простых числа $p$ и $q$ (скажем, длиной 1024 бита).
2. Вычислим $n = p \times q$ и $z = (p - 1) \times (q - 1)$.
3. Выберем число $d$, которое является взаимно простым с числом $z$.
4. Найдем такое число $e$, для которого справедливо равенство $e \times d = 1 \bmod z$.

Заранее вычислив эти параметры, можно начинать шифрование. Сначала разобьем весь открытый текст (рассматриваемый в качестве битовой строки) на блоки так, чтобы каждое сообщение $P$ попадало в интервал $0 \le P < n$. Это несложно сделать, если разбить открытый текст на блоки по $k$ бит, где $k$ — максимальное целое число, для которого справедливо неравенство $2^k < n$.

Чтобы зашифровать сообщение $P$, вычислим $C = P^e \pmod{n}$. Чтобы дешифровать $C$, вычислим $P = C^d \pmod{n}$. Можно доказать, что для всех значений $P$ в указанном диапазоне функции шифрования и дешифрования являются взаимно обратными. Чтобы выполнить шифрование, нужны $e$ и $n$. Для дешифрования требуются $d$ и $n$. Таким образом, открытый ключ состоит из пары $(e, n)$, а закрытый ключ — из пары $(d, n)$.

Надежность метода обеспечивается сложностью факторизации больших чисел. Если бы криптоаналитик мог разложить на множители публично известное число $n$, он бы нашел значения $p$ и $q$, а следовательно, и число $z$. После этого $e$ и $d$ можно найти с помощью алгоритма Евклида. К счастью, математики пытаются решить проблему факторизации больших чисел уже по меньшей мере 300 лет, и накопленный опыт говорит о том, что это очень трудная задача.

В свое время Ривест вместе с коллегами пришел к заключению, что разложение на множители числа из 500 цифр займет $10^{25}$ лет, если действовать методом простого перебора. Предполагалось использование наиболее известного алгоритма и компьютера, выполняющего одну инструкцию за 1 мкс. Если миллион чипов будет работать одновременно и выполнять одну инструкцию за 1 нс, все равно потребуется $10^{16}$ лет. Даже при экспоненциальном росте скоростей компьютеров потребуются многие годы, чтобы факторизовать числа из 500 цифр, а к тому времени наши потомки могут просто выбрать еще бо́льшие $p$ и $q$. При этом, вероятно, никого не удивит, что атаки тоже совершенствуются — на сегодняшний день их скорость значительно возросла.

На илл. 8.19 приведен тривиальный учебный пример работы алгоритма RSA. Здесь $p = 3$ и $q = 11$, что дает значения $n = 33$ и $z = 20$ (так как $(3 - 1) \times (11 - 1) = 20$). Число $d$ можно выбрать равным 7, так как числа 20 и 7 не имеют общих делителей. Значение $e$ можно найти, решив уравнение $7e = 1 \pmod{20}$, откуда следует, что $e = 3$. Зашифрованный текст $C$ получается из открытого сообщения $P$ по формуле $C = P^3 \pmod{33}$. Получатель расшифровывает сообщение по формуле $P = C^7 \pmod{33}$. В качестве примера на рисунке показано шифрование слова «SUZANNE».

| Открытый текст (P) | | Зашифрованный текст (C) | | | После дешифрования | |
|---|---|---|---|---|---|---|
| Символ | Число | $P^3$ | $P^3 \pmod{33}$ | $C^7$ | $C^7 \pmod{33}$ | Символ |
| S | 19 | 6859 | 28 | 13492928512 | 19 | S |
| U | 21 | 9261 | 21 | 1801088541 | 21 | U |
| Z | 26 | 17576 | 20 | 1280000000 | 26 | Z |
| A | 01 | 1 | 1 | 1 | 01 | A |
| N | 14 | 2744 | 5 | 78125 | 14 | N |
| N | 14 | 2744 | 5 | 78125 | 14 | N |
| E | 05 | 125 | 26 | 8031810176 | 05 | E |
| Вычисления отправителя | | | | Вычисления получателя | | |

**Илл. 8.19.** Пример работы алгоритма RSA

Поскольку выбранные для этого примера простые числа настолько малы, $P$ должно быть меньше 33, поэтому каждый блок открытого текста может содержать лишь одну букву. В результате получается моноалфавитный подстановочный шифр, что не слишком впечатляет. Если бы вместо этого мы выбрали числа $p$ и $q \approx 2^{512}$, мы бы получили $n \approx 2^{1024}$. Тогда каждый блок мог бы содержать до 1024 бит или 128 восьмиразрядных символов против 8 символов в случае метода DES или 16 символов для AES.

Отметим, что такое использование RSA аналогично применению симметричного алгоритма в режиме ECB, в котором одни и те же блоки на входе преобразуются в одинаковые блоки на выходе. Таким образом, для шифрования данных требуется сцепление блоков в том или ином виде. Но на практике RSA с открытым ключом используется только для передачи одноразовых 128- или 256-битных сеансовых ключей, после чего задействуется алгоритм с симметричным ключом наподобие AES. Система RSA слишком медленная, чтобы шифровать большие объемы данных, однако она широко применяется для распространения ключей.

## 8.6.2. Другие алгоритмы с открытым ключом

Хотя RSA по-прежнему популярен, это далеко не единственный известный алгоритм с открытым ключом. Первым таким алгоритмом стал **алгоритм на основе задачи о рюкзаке (knapsack algorithm)** Меркла и Хеллмана (Merkle and Hellman, 1978). Его принцип заключается в следующем. Допустим, имеется очень большое количество объектов разного веса. Их владелец кодирует сообщение, выбирая подмножество вещей и помещая их в рюкзак. Известен общий вес объектов в рюкзаке, а также список всех возможных объектов и их вес по отдельности. Список вещей в рюкзаке хранится в секрете. Считалось, что при некоторых дополнительных ограничениях определить возможный список объектов по известному общему весу невозможно путем вычислений. Поэтому эта задача легла в основу алгоритма с открытым ключом.

Разработчик алгоритма Ральф Меркл (Ralph Merkle) был настолько уверен в его надежности, что предложил $100 любому, кто сумеет его взломать. Ади Шамир («S» в группе RSA) мгновенно взломал его и получил награду. Это не смутило Меркла. Он усилил алгоритм и предложил за его взлом уже $1000. Рон Ривест («R» в группе RSA) тут же взломал улучшенную версию и получил награду. Леонард Адлеман («A» в группе RSA) остался без награды, поскольку Меркл уже не рискнул предложить $10 000 за взлом следующей версии. Несмотря на то что алгоритм рюкзака был в очередной раз исправлен, он не считается надежным и не используется на практике.

Остальные алгоритмы с открытым ключом основаны на сложности вычисления дискретных логарифмов или значений эллиптических кривых (Менезес и Ванстоун; Menezes and Vanstone, 1993). Первые были разработаны Эль Гамалем (El Gamal, 1985) и Шнорром (Schnorr, 1991). Вторые используют раздел математики, известный в очень узких кругах, — теорию эллиптических кривых.

Существует и ряд других схем, однако алгоритмы, использующие сложность факторизации больших чисел, вычисления дискретных логарифмов, разделенных по модулю на большое простое число, и значений эллиптических кривых,

безусловно, являются самыми важными. Эти задачи считаются по-настоящему сложными, так как математики пытаются их решить уже много лет без особого успеха. Эллиптические кривые здесь представляют особый интерес, поскольку задачи дискретного логарифмирования на такой кривой еще сложнее, чем задачи разложения на множители. Голландский математик Арьен Ленстра (Arjen Lenstra) предложил сравнивать криптографические алгоритмы по количеству энергии, которая тратится на их взлом. По его расчетам, чтобы взломать 228-битный RSA-ключ, требуется примерно столько же энергии, сколько нужно для того, чтобы вскипятить одну чайную ложку воды. Однако в случае ключа той же длины на базе эллиптической кривой на взлом уйдет столько же энергии, сколько нужно для того, чтобы вскипятить всю воду на планете. Перефразируя Ленстра, можно сказать, что даже испарив всю воду (включая воду в их телах), взломщики вряд ли добьются успеха.

## 8.7. ЦИФРОВЫЕ ПОДПИСИ

Подлинность документов (юридических, финансовых и др.) определяется наличием или отсутствием авторизованной собственноручной подписи (фотокопии не в счет). Чтобы компьютерные системы обмена сообщениями могли заменить пересылку бумажных документов, необходимо решить проблему подписи, исключив возможность подделки.

Задача разработки замены для подписи от руки довольно сложна. По сути, требуется система, с помощью которой одна сторона может отправить другой стороне подписанное сообщение, при этом:

1. Получатель может проверить заявленную личность отправителя.
2. Позднее отправитель не может отрицать содержание отправленного сообщения.
3. Получатель не имеет возможности изменить сообщение.

Первое требование имеет большое значение, к примеру, для финансовых систем. Когда компьютер клиента передает компьютеру банка заказ на покупку тонны золота, тот должен быть уверен, что устройство, с которого пришел запрос, действительно принадлежит данному клиенту, прежде чем снимать деньги с его счета. Другими словами, банк должен аутентифицировать клиента (и наоборот).

Соблюдение второго условия требуется для защиты банка от мошенничества. Предположим, что банк покупает заказанную тонну золота, после чего цена на этот металл резко падает. Недобросовестный клиент может подать на банк в суд, заявляя, что никогда не заказывал покупку. Банк может предъявить суду письмо с заказом, но клиент будет отрицать его подлинность. Свойство, при котором стороны не могут отрицать факт подписания контракта, называется **неотказуемостью (nonrepudiation)**. Схемы цифровых подписей, которые мы изучим далее, помогают ее обеспечить.

Третье требование необходимо для защиты клиента. Если цена на золото после его покупки банком резко взлетает, банк может создать подписанное сообщение,

в котором клиент заказывает не тонну золота, а один слиток. При таком сценарии банк, совершив мошенничество, забирает оставшуюся часть золота себе.

## 8.7.1. Подписи с симметричным ключом

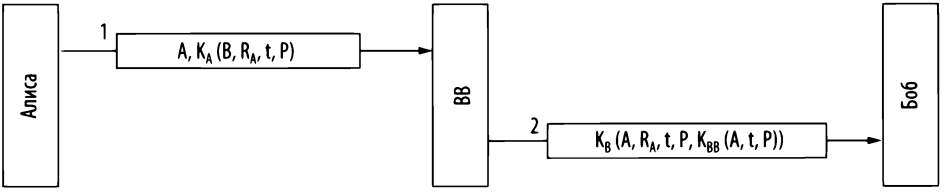
Один из методов реализации цифровых подписей состоит в создании центрального авторитетного органа, которому все доверяют, назовем его Большим Братом (Big Brother, BB). Каждый пользователь выбирает секретный ключ и лично относит его в его офис. Таким образом, к примеру, секретный ключ Алисы $K_A$ известен только ей самой и *BB*. Если вы забудете, что значит какой-либо символ или подстрочный индекс, вы всегда можете свериться с илл. 8.20. Здесь описаны самые важные обозначения, используемые в этом и последующих разделах.

| Обозначение | Описание |
|---|---|
| A | Алиса (отправитель) |
| B | Банковский менеджер Боб (получатель) |
| P | Открытый текст сообщения, которое хочет отправить Алиса |
| BB | Большой Брат (авторитетный центральный орган) |
| t | Временная метка (для подтверждения новизны) |
| $R_A$ | Случайное число, выбранное Алисой |
| **Симметричный ключ** | |
| $K_A$ | Секретный ключ Алисы (аналогично и в случае $K_B$, $K_{BB}$ и т. д.) |
| $K_A(M)$ | Сообщение M шифруется/дешифруется с помощью секретного ключа Алисы |
| **Асимметричные ключи** | |
| $D_A$ | Закрытый ключ Алисы (аналогично и в случае $D_B$ и т. д.) |
| $E_A$ | Открытый ключ Алисы (аналогично и в случае $E_B$ и т. д.) |
| $D_A(M)$ | Сообщение M шифруется/дешифруется с помощью закрытого ключа Алисы |
| $E_A(M)$ | Сообщение M шифруется/дешифруется с помощью открытого ключа Алисы |
| **Профиль сообщения** | |
| MD(P) | Профиль сообщения для открытого текста P |

**Илл. 8.20.** Алиса хочет отправить сообщение своему банковскому менеджеру: расшифровка ключей и символов

Чтобы отправить своему банковскому менеджеру Бобу подписанное незашифрованное сообщение *P*, Алиса формирует сообщение $K_A(B, R_A, t, P)$, зашифрованное ее ключом $K_A$ ($B$ — идентификатор Боба, $R_A$ — случайное число, выбранное Алисой, $t$ — временная метка для подтверждения новизны сообщения).

Затем она отсылает его *BB* (илл. 8.21). *BB* видит, что это сообщение от Алисы, расшифровывает его и передает Бобу. Сообщение для Боба содержит открытый текст сообщения Алисы $K_{BB}(A, t, P)$ и подпись *BB*. Получив его, Боб может выполнять заказ Алисы.




**Илл. 8.21.** Цифровая подпись BB

Что случится, если позже Алиса будет отрицать отправку этого сообщения? Прежде всего, все подают друг на друга в суд (по крайней мере, в США). В суде Алиса решительно утверждает, что она не отправляла Бобу спорное сообщение. Судья спрашивает Боба, почему он уверен, что данное сообщение пришло именно от Алисы, а не от Труди. Сначала Боб заявляет, что *BB* не принял бы сообщение от Алисы, если бы оно не было зашифровано ее ключом $K_A$. У Труди просто нет возможности отправить фальшивое сообщение от имени Алисы — Боб сразу бы это обнаружил.

Затем Боб театрально демонстрирует суду вещественное доказательство А: $K_{BB}(A, t, P)$. Боб говорит, что это сообщение подписано *BB*, а значит, Алиса является источником сообщения *P*. Судья просит *BB* (которому все доверяют) проверить подпись под этим сообщением. Когда *BB* подтверждает, что Боб говорит правду, судья решает дело в пользу Боба. Дело закрыто.

Теоретически проблема с протоколом цифровых подписей, показанным на илл. 8.21, может возникнуть, если Труди повторно воспроизведет любое из сообщений. Чтобы свести к минимуму такую вероятность, используются временные метки. Кроме того, Боб может просмотреть все недавние письма и проверить, не встречалось ли в них такое же $R_A$. При повторе $R_A$ сообщение считается дубликатом и отбрасывается. Очень старые (исходя из временной метки) письма также игнорируются. Для защиты от атаки мгновенного повторного воспроизведения Боб просто проверяет значение числа $R_A$ в каждом входящем сообщении, сверяя его со всеми $R_A$, полученными за последний час. Если в течение часа такое значение не приходило, Боб может быть уверен, что полученное сообщение является новым запросом.

## 8.7.2. Подписи с открытым ключом

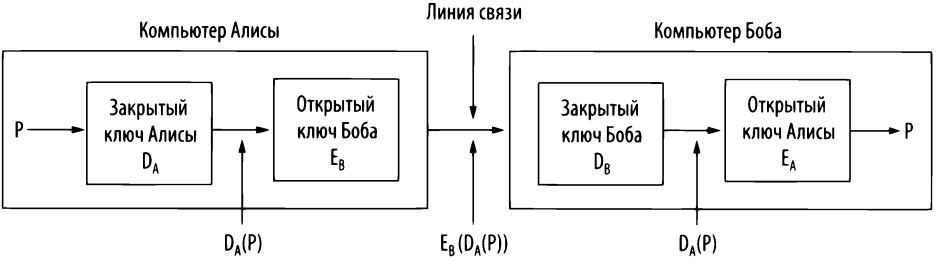
Главная проблема создания цифровых подписей с помощью шифрования с симметричным ключом — все должны согласиться доверять *BB*. Помимо этого, *BB* получает возможность читать подписываемые им сообщения. Самыми логичными кандидатами на управление сервером *BB* являются правительство, банки или нотариальные бюро. Однако им доверяют не все граждане. Было бы

здорово, если бы для получения подписи на электронном документе не требовался какой-либо доверенный полномочный орган.

К счастью, здесь нам поможет шифрование с открытым ключом. Предположим, такие алгоритмы шифрования и дешифрования, помимо обычного свойства $D(E(P)) = P$, обладают свойством $E(D(P)) = P$. Например, оно есть у алгоритма RSA, так что это предположение вполне обоснованно. В этом случае Алиса может отправить Бобу подписанное открытое сообщение $P$, переслав ему $E_B(D_A(P))$. Обратите внимание, что она знает свой собственный (закрытый) ключ дешифрования $D_A$, так же как и открытый ключ Боба $E_B$, поэтому сформировать такое сообщение ей по силам.

Приняв сообщение, Боб расшифровывает его как обычно, используя свой закрытый ключ $D_B$, что в результате дает $D_A(P)$, как показано на илл. 8.22. Он сохраняет зашифрованный текст в надежном месте, после чего декодирует его открытым ключом шифрования Алисы $E_A$, получая открытый текст.

Чтобы понять, как в данном случае работает цифровая подпись, предположим, что впоследствии Алиса отрицает отправку Бобу сообщения $P$. Когда дело доходит до суда, Боб предъявляет $P$ и $D_A(P)$. Судья легко может убедиться, что у Боба есть подлинное сообщение, зашифрованное ключом $D_A$, просто применив к нему ключ $E_A$. Боб не знает закрытого ключа Алисы, следовательно, получить зашифрованное этим ключом сообщение он мог только от нее. Сидя в тюрьме за лжесвидетельство и мошенничество, Алиса сможет заняться разработкой новых интересных алгоритмов с открытым ключом.




**Илл. 8.22.** Цифровая подпись, полученная при помощи шифрования с открытым ключом

Хотя схема шифрования с открытым ключом довольно элегантна, у нее есть серьезные недостатки. Правда, они скорее связаны не с самим алгоритмом, а со средой, в которой он работает. Прежде всего, Боб может доказать, что сообщение было отправлено Алисой, только пока ключ $D_A$ хранится в тайне. Если Алиса раскроет свой секретный ключ, этот аргумент потеряет актуальность, так как в этом случае нельзя определить источник сообщения (им может быть кто угодно, включая самого Боба).

Например, проблема может возникнуть, если Боб является биржевым брокером Алисы. Она заказывает ему покупку определенных акций. Сразу после этого их цена резко падает. Чтобы отречься от своего запроса, Алиса сообщает в полицию, что ее дом был обворован, и в том числе украден компьютер вместе

с секретным ключом. В зависимости от законов ее страны или штата Алиса может понести (или нет) юридическую ответственность, особенно если она заявляет, что обнаружила проникновение в квартиру, только вернувшись с работы, – через несколько часов после того, как это якобы произошло.

Другая проблема этой схемы цифровой подписи возникает, если Алиса поменяет свой ключ. Это абсолютно законно, более того, рекомендуется периодически менять ключ, чтобы гарантировать его высокую надежность. В этом случае, если дело дойдет до судебного разбирательства, судья попытается применить к подписи $D_A(P)$ *текущий ключ* $E_A$ и обнаружит, что в результате сообщение $P$ не получается. При этом Боб будет выглядеть довольно глупо.

В принципе, для цифровых подписей можно использовать любой алгоритм с открытым ключом. Стандартом фактически является RSA. Он применяется во многих программах, предназначенных для обеспечения безопасности. Однако в 1991 году NIST предложил использовать для нового **Стандарта цифровой подписи (Digital Signature Standard, DSS)** вариант алгоритма с открытым ключом Эль-Гамаля. Он основан не на трудности факторизации больших чисел, а на сложности вычисления дискретных логарифмов.

Как обычно, попытка правительства навязать новые криптографические стандарты вызвала много шума. Стандарт DSS критиковали за то, что он:

1) слишком засекречен (протокол, использующий алгоритм Эль-Гамаля, разрабатывался АНБ);
2) слишком медленный (при проверке подписей он работает в 10–40 раз медленнее алгоритма RSA);
3) слишком новый (алгоритм Эль-Гамаля еще не был тщательно проанализирован);
4) слишком ненадежен (фиксированная длина ключа – 512 бит).

При последующей переработке четвертый пункт утратил значение, так как было разрешено применять ключи длиной до 1024 бит. Однако первые два пункта актуальны и по сей день.

## 8.7.3. Профили сообщений

Многие методы цифровых подписей критикуют за то, что в них совмещаются две разные функции: аутентификация и секретность. Чаще всего требуется лишь первая функция. К тому же получить лицензию на экспорт обычно проще, если система обеспечивает только аутентификацию. Ниже описана схема аутентификации, не требующая шифрования всего сообщения.

Она основана на идее необратимой хеш-функции, которая принимает на входе участок открытого текста произвольной длины и по нему вычисляет строку битов фиксированной длины. Эта хеш-функция, которую часто называют **профилем сообщения (message digest, MD)**, обладает четырьмя важными свойствами:

1. На основе заданного сообщения $P$ можно легко вычислить $MD(P)$.
2. На основе $MD(P)$ практически невозможно определить $P$.

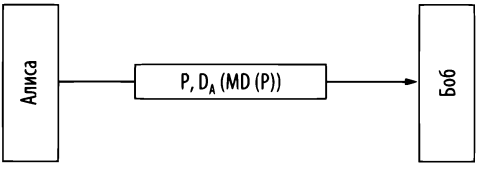
3. Для заданного сообщения $P$ невозможно подобрать сообщение $P'$, для которого будет выполняться равенство $MD(P') = MD(P)$.
4. Изменение даже одного бита входной последовательности приводит к совершенно другому результату.

Чтобы удовлетворять требованию 3, результат хеш-функции должен обладать длиной по крайней мере 128 бит (а желательно больше). Чтобы соответствовать требованию 4, хеш-функция должна очень сильно искажать входные значения, как и алгоритмы шифрования с симметричным ключом, рассмотренные выше.

Вычислить профиль сообщения по фрагменту открытого текста гораздо быстрее, чем зашифровать все сообщение с помощью алгоритма с открытым ключом. Поэтому профили сообщений могут использоваться для ускорения работы алгоритмов цифровых подписей. Чтобы понять, как это работает, мы вновь обратимся к протоколу цифровой подписи на илл. 8.21. Вместо передачи открытого текста $P$ вместе с $K_{BB}(A, t, P)$ теперь $BB$ вычисляет профиль сообщения $MD(P)$, применяя функцию хеширования $MD$ к открытому тексту $P$. Затем он помещает $K_{BB}(A, t, MD(P))$ как пятый элемент в список, зашифрованный ключом $K_B$, и отправляет его Бобу вместо $K_{BB}(A, t, P)$.

В случае возникновения спора Боб может предъявить на суде как открытый текст $P$, так и $K_{BB}(A, t, MD(P))$. По просьбе судьи $BB$ расшифровывает $K_{BB}(A, t, MD(P))$. В результате суду также предъявляется цифровая подпись $MD(P)$, подлинность которой гарантируется $BB$, и сам открытый текст $P$, подлинность которого суд должен выяснить. Поскольку создать другой открытый текст, соответствующий данной цифровой подписи, практически невозможно, суд убеждается в том, что Боб говорит правду. Использование профиля сообщения экономит время шифрования и затраты на транспортировку и хранение.

Профиль сообщения также применяется в системах шифрования с открытым ключом (илл. 8.23). Сначала Алиса вычисляет профиль сообщения для своего открытого текста. Затем она подписывает его и отправляет Бобу вместе с открытым текстом. Если во время передачи Труди подменит открытый текст $P$, Боб обнаружит это, вычислив $MD(P)$.

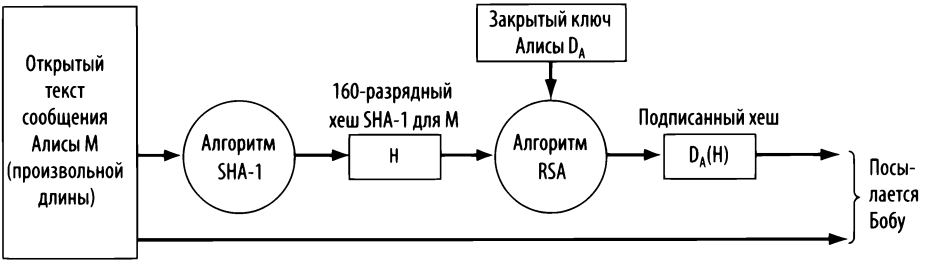



**Илл. 8.23.** Цифровая подпись с использованием профиля сообщения

## SHA-1, SHA-2 и SHA-3

Для вычисления профиля сообщения было предложено несколько вариантов функций. Долгое время в качестве такой функции широко использовался алгоритм **SHA-1 (Secure Hash Algorithm 1 — защищенный алгоритм хеширования 1)** (NIST, 1993). Прежде всего стоит отметить, что он был взломан в 2017 году и теперь постепенно выводится из эксплуатации в большинстве систем, о чем

будет подробно рассказано чуть позже. Как и все профили сообщения, этот алгоритм достаточно сложным образом перемешивает входные биты, и в результате каждый выходной бит зависит от каждого входного. Алгоритм SHA-1 был разработан АНБ и получил благословение NIST (в виде федерального стандарта FIPS 180-1). Он обрабатывает входные данные блоками по 512 бит и формирует профиль сообщения длиной 160 бит. Типичный случай отправки Алисой несекретного, но подписанного сообщения Бобу показан на илл. 8.24. Открытый текст обрабатывается алгоритмом SHA-1, на выходе получается 160-битный хеш SHA-1. Затем Алиса подписывает его закрытым ключом RSA и отправляет вместе с открытым текстом Бобу.




**Илл. 8.24.** Применение SHA-1 и RSA для создания подписей несекретных сообщений

При получении сообщения Боб сам вычисляет хеш-функцию с помощью SHA-1 и применяет открытый ключ Алисы к подписанному хешу, чтобы получить исходный хеш *H*. Если они совпадают, сообщение считается корректным. Поскольку Труди не может перехватить сообщение и изменить его так, чтобы значение *H* совпадало с контрольным, Боб легко узнает обо всех подменах, которые она совершила. Для сообщений, чья целостность важна, а секретность не имеет значения, часто применяется схема, показанная на илл. 8.24. При относительно небольших вычислительных затратах она гарантирует, что все изменения, внесенные на пути следования сообщения, будут с высокой степенью вероятности выявлены.

В настоящее время идет работа над новыми версиями SHA-1 с 224-, 256-, 384- и 512-разрядными значениями хеш-функций. Вместе они называются SHA-2. Помимо увеличения длины хешей, в этих версиях также была изменена функция для вычисления профиля, что позволило избавиться от потенциальных уязвимостей версии SHA-1 (а они были достаточно серьезными). В 2017 году SHA-1 был взломан специалистами Google и амстердамского исследовательского центра CWI. Точнее, им удалось сгенерировать **коллизии хеш-функций (hash collisions)**, что нивелирует защиту алгоритма SHA-1. Как и следовало ожидать, результатом этой атаки стало существенное возрастание интереса к SHA-2.

В 2006 году NIST организовал конкурс с целью создания нового стандарта хеш-функции, теперь известного как SHA-3. Это соревнование было завершено в 2012 году, а три года спустя был опубликован новый стандарт SHA-3 («Keccak»). Что интересно, NIST не предлагает сразу же отказаться от алгоритма

SHA-2 в пользу SHA-3, поскольку случаев успешного взлома SHA-2 еще не зафиксировано. Но даже несмотря на это, полезно на всякий случай иметь под рукой запасной вариант.

## 8.7.4. Атака «дней рождения»

В мире криптографии ничто не является тем, чем кажется. К примеру, можно предположить, что для взлома профиля сообщения, состоящего из $m$ разрядов, потребуется порядка $2^m$ операций. На самом деле часто достаточно $2^{m/2}$ операций, если использовать **атаку «дней рождения» (birthday attack)**. Впервые она была описана в уже ставшей классикой работе Юваля (Yuval, «How to Swindle Rabin», 1979).

Как уже упоминалось в разделе, посвященном службе DNS, атака «дней рождения» основана на том, что если имеется отображение $n$ входных значений (людей, сообщений и т. д.) на $k$ возможных выходных значений (дней рождения, профилей сообщения и т. д.), то имеется $n(n - 1)/2$ входных пар. Если $n(n - 1)/2 > k$, то довольно высока вероятность получения хотя бы одного совпадения. Таким образом, вероятность существования двух сообщений с одинаковыми профилями велика уже при $n > \sqrt{k}$. Это означает, что 64-разрядный профиль сообщения вполне возможно взломать, перебрав $2^{32}$ сообщений и отыскав два разных сообщения с одинаковым профилем.

Разберем практический пример. На кафедре вычислительной техники государственного университета появилась вакансия. На нее претендуют два кандидата, Том и Дик. Том работает в университете на два года дольше Дика, поэтому его кандидатура рассматривается первой. Если он получит эту должность, значит, Дику не повезло. Том знает, что заведующая кафедрой Мэрилин высоко ценит его работу, поэтому просит ее написать для него рекомендательное письмо декану факультета, который будет принимать решение. После отправки все письма становятся конфиденциальными.

Мэрилин просит своего секретаря Элен написать это письмо и дает ей примерное содержание. Когда письмо готово, Мэрилин просматривает его, вычисляет и подписывает его 64-разрядный профиль и отсылает этот подписанный профиль декану. Позже Элен должна отправить само письмо электронной почтой.

К несчастью для Тома, у Элен роман с Диком, и она хочет подставить Тома. Поэтому она пишет следующее письмо с 32 вариантами в квадратных скобках.

Уважаемый господин декан,

Это [*письмо* | *обращение*] отражает мое [*искреннее* | *откровенное*] мнение о проф. Томе Уилсоне, являющемся [*кандидатом* | *претендентом*] на профессорскую должность [*в настоящее время* | *в этом году*]. Я [*знакома* | *работала*] с проф. Уилсоном в течение [*почти* | *около*] шести лет. Он является [*выдающимся* | *блестящим*] исследователем, обладающим [*большим талантом* | *большими возможностями*] и известным [*во всем мире* | *не только в нашей стране*] своим [*серьезным* | *созидательным*] подходом к [*большому числу* | *широкому спектру*] [*сложных* | *перспективных*] вопросов.

Он также является [*высоко* | *весьма*] [*уважаемым* | *ценимым*] [*преподавателем* | *педагогом*]. Его студенты дают его [*занятиям* | *лекциям*] [*самые высокие* | *высочайшие*] оценки. Он самый [*популярный* | *любимый*] [*преподаватель* | *учитель*] [*нашей кафедры* | *нашего университета*].

[*Кроме* | *Помимо*] того, [*гранты* | *контракты*] проф. Уилсона [*существенно* | *значительно*] пополнили [*фонды* | *финансовые запасы*] нашей кафедры. Эти [*денежные* | *финансовые*] средства [*позволили нам* | *дали возможность*] [*выполнить* | *осуществить*] [*много* | *ряд*] [*важных* | *специальных*] программ, [*таких как* | *среди которых*] государственная программа 2025 года. Без этих средств было бы невозможным продолжение этой программы, такой [*важной* | *значительной*] для нас. Я настойчиво рекомендую вам предоставить ему эту должность.

К несчастью для Тома, закончив печатать это письмо, Элен тут же принимается за второе:

Уважаемый господин декан,

Это [*письмо* | *обращение*] отражает мое [*искреннее* | *откровенное*] [*мнение* | *суждение*] о проф. Томе Уилсоне, являющемся [*кандидатом* | *претендентом*] на профессорскую должность в [*настоящее время* | *этом году*]. Я [*знакома* | *работала*] с проф. Уилсоном в течение [*почти* | *около*] шести лет. Он является [*слабым* | *недостаточно талантливым*] [*исследователем* | *ученым*], почти неизвестным в той области науки, которой он занимается. В его работах практически не заметно понимания [*ключевых* | *главных*] [*проблем* | *вопросов*] современности.

[*Более* | *Кроме*] того, он также не является сколько-нибудь [*уважаемым* | *ценимым*] [*преподавателем* | *педагогом*]. Его студенты дают его [*занятиям* | *лекциям*] [*самые низкие* | *негативные*] оценки. Он самый непопулярный [*преподаватель* | *учитель*] нашей кафедры, [*славящийся* | *печально известный*] своей [*привычкой* | *склонностью*] [*высмеивать* | *ставить в неудобное положение*] студентов, осмелившихся задавать вопросы на его [*лекциях* | *занятиях*].

[*Кроме* | *Помимо*] того, [*гранты* | *контракты*] проф. Уилсона [*почти* | *практически*] не пополняют [*фондов* | *финансовых запасов*] нашей кафедры. Если не удастся быстро найти новый источник финансирования, [*мы будем вынуждены* | *нам придется*] [*закрыть* | *прекратить*] [*много* | *ряд*] [*важных* | *специальных*] программ, [*таких как* | *среди которых*] государственная программа 2025 года. К сожалению, при таких [*условиях* | *обстоятельствах*] я не могу [*предлагать* | *рекомендовать*] его вам на эту должность.

Затем Элен программирует свой компьютер на подсчет 232 профилей сообщения для каждого варианта обоих писем; вычисления занимают всю ночь. Есть шанс, что один профиль первого письма совпадет с одним из профилей второго письма. В противном случае Элен может добавить еще несколько вариантов слов и выражений в каждое письмо и попытаться еще раз вычислить профили

за выходные. Предположим, она нашла совпадение. Назовем положительный отзыв письмом *A*, а отрицательный — письмом *B*.

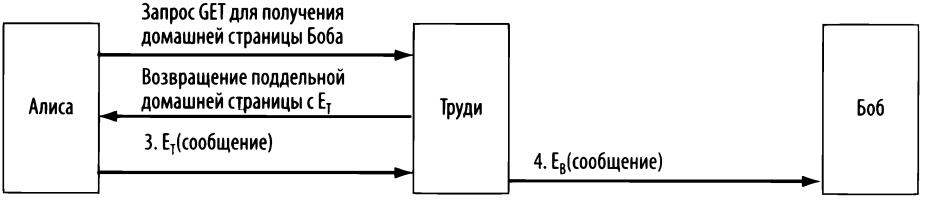
Элен отправляет письмо *A* по электронной почте Мэрилин на утверждение. Содержание письма *B* она сохраняет в тайне, никому его не показывая. Естественно, Мэрилин утверждает письмо, вычисляет 64-разрядный профиль сообщения, подписывает профиль и отсылает его декану. Независимо от нее, Элен отправляет декану письмо *B* (вместо письма *A*, которое нужно отправить на самом деле).

Получив письмо и подписанный профиль, декан запускает алгоритм вычисления профиля сообщений для письма *B*, видит, что его профиль совпадает с тем, что ему прислала Мэрилин, и увольняет Тома. Он не догадывается, что Элен удалось создать два письма с одинаковыми профилями сообщений и отправить ему совсем не тот вариант, который читала и подписала Мэрилин. (Вариант с хэппи-эндом: Элен сообщает Дику о своих проделках. Потрясенный, Дик порывает с ней. Элен в ярости бежит сознаваться во всем Мэрилин. Мэрилин звонит декану. Том получает должность.) При использовании алгоритма SHA-2 подобную атаку шифра достаточно сложно выполнить. Даже если компьютер сможет вычислять по 1 трлн профилей в секунду, потребуется более 32 000 лет, чтобы перебрать по 280 вариантов для обоих писем, и это все равно не даст стопроцентной гарантии совпадения. Конечно, если параллельно запустить 1 млн процессоров, то вместо 32 000 лет потребуется 2 недели.

## 8.8. УПРАВЛЕНИЕ ОТКРЫТЫМИ КЛЮЧАМИ

Криптография с использованием открытых ключей позволяет передавать секретные данные, не обладая открытым ключом заранее, а также создавать электронные подписи сообщений без необходимости в привлечении третьей, доверительной стороны. Наконец, подписанные профили сообщений позволяют получателю легко и надежно проверять целостность и аутентичность полученных данных.

Между тем мы обошли один важный вопрос: если Алиса и Боб друг друга не знают, как они обменяются открытыми ключами перед началом общения? Очевидное решение — выложить ключ на веб-сайте. Однако так делать нельзя, и вот почему. Представьте, что Алиса хочет найти открытый ключ Боба на его сайте. Как она это делает? Набирает URL сайта. Браузер ищет DNS-адрес домашней страницы Боба и передает запрос GET (илл. 8.25). К сожалению, Труди в этот




**Илл. 8.25.** Способ вторжения в систему с открытыми ключами

момент перехватывает запрос и отсылает Алисе фальшивую страницу, например копию страницы Боба, на которой вместо его ключа выложен открытый ключ Труди. После того как Алиса зашифрует свое первое сообщение с помощью $E_T$, Труди расшифрует его, прочтет, зашифрует с помощью открытого ключа Боба и перешлет ничего не подозревающему Бобу. Но гораздо хуже то, что Труди может менять сообщения перед повторным шифрованием и отправкой Бобу. Очевидно, необходим механизм секретного обмена открытыми ключами.

## 8.8.1. Сертификаты

Конечно, можно попытаться решить эту проблему с помощью **центра распространения ключей (Key Distribution Center, KDC)**, круглосуточно доступного в интернете и предоставляющего открытые ключи по требованию. Однако у этого решения есть множество недостатков; в частности, оно не поддается масштабированию — такой центр очень быстро станет узким местом. А если он не выдержит нагрузки, вся интернет-безопасность в один момент сойдет на нет.

По этой причине возникло новое решение, не требующее постоянного доступа к центру распространения ключей. На самом деле даже не обязательно, чтобы центр вообще работал онлайн. Вместо этого на него возлагается обязанность сертификации открытых ключей, принадлежащих как физическим, так и юридическим лицам. Организация, выполняющая эту задачу, в настоящее время называется **центром сертификации (Certification Authority, CA)**.

В качестве примера рассмотрим такую ситуацию: Боб хочет разрешить Алисе и другим лицам, с которыми он не знаком, устанавливать с ним защищенные соединения. Он приходит в CA со своим открытым ключом, а также паспортом или другим удостоверением личности и просит зарегистрировать ключ. CA выдает ему сертификат (похожий на тот, что показан на илл. 8.26) и подписывает хеш SHA-2 своим закрытым ключом. Затем Боб оплачивает услуги CA и получает документ, содержащий сертификат и его подписанный хеш (который в идеале должен передаваться по надежным каналам).

Настоящим удостоверяю, что открытый ключ
19836A8B03030CF83737E3837837FC3s87092827262643FFA82710382828282A
принадлежит
Роберту Джону Смиту
12345, Юниверсити-авеню
Беркли, Калифорния 94702
Дата рождения: 4 июля 1958 г.
Email: bob@superdupernet.com

Хеш SHA-2 данного сертификата подписан закрытым ключом CA

**Илл. 8.26.** Пример сертификата и подписанного хеша

Основная задача сертификата — связать открытый ключ с именем принципала (физического или юридического лица). Сами сертификаты никак не защищаются и не хранятся в тайне. Боб может выложить его на свой сайт и поставить

ссылку: «Здесь находится сертификат моего открытого ключа». Перейдя по ссылке, пользователь увидит и сертификат, и блок с подписью (подписанный хеш SHA-2 сертификата).

Теперь снова рассмотрим сценарий, показанный на илл. 8.25. Что может сделать Труди, перехватив запрос страницы Боба, отправленный Алисой? Она может выложить свой сертификат и блок с подписью на подложной странице, однако Алиса сразу догадается, что разговаривает не с Бобом: его имени в этом сертификате просто нет. Труди может изменить домашнюю страницу Боба на лету, заменив его открытый ключ своим собственным. Но Алиса может проверить сертификат алгоритмом SHA-2. Тогда она получит значение хеш-функции, которое не согласуется со значением, вычисленным по открытому ключу CA и блоку подписи. Труди не имеет доступа к закрытому ключу CA и никак не может сгенерировать блок подписи, содержащий хеш модифицированной страницы с открытым ключом Труди. Таким образом, Алиса может не беспокоиться о подлинности ключа Боба. Как мы и обещали, при такой схеме не требуется, чтобы CA постоянно работал онлайн. Тем самым ликвидируется потенциально узкое место системы.

Сертификат используется для связывания открытого ключа не только с принципалом, но и с **атрибутом**. Например, он может содержать такую информацию: «Данный открытый ключ принадлежит лицу старше 18 лет». Этим можно подтвердить статус принципала или убедиться в том, что ему разрешен доступ к некоторым специфическим данным, на которые накладываются возрастные ограничения. При этом имя владельца может и не раскрываться. Обычно владелец отсылает сертификат на веб-сайт, принципалу или процессу, запрашивающему информацию о возрасте клиента. В ответ генерируется случайное число, шифруемое с помощью открытого ключа (считываемого с сертификата). Если клиент (то есть владелец) сможет расшифровать его и отослать обратно, это будет служить подтверждением тому, что он действительно обладает указанными в сертификате характеристиками. Кроме того, это случайное число может быть использовано в качестве сеансового ключа для будущего соединения.

Сертификат может содержать атрибут еще в одном случае: если речь идет об объектно-ориентированной распределенной системе. Каждый объект обычно имеет несколько методов. Владелец объекта может предоставить каждому клиенту сертификат с битовой картой, где перечислены доступные ему методы. Эта битовая карта привязывается к открытому ключу с помощью подписанного сертификата. Если владелец сертификата докажет, что у него есть соответствующий закрытый ключ, он сможет воспользоваться методами, указанными в битовой карте. Особенность этого подхода состоит в том, что необязательно указывать имя владельца. Это полезно в ситуациях, в которых важна конфиденциальность.

## 8.8.2. X.509

Если бы все желающие подписать что-либо обращались в CA за сертификатами различных видов, обслуживание разнообразных форматов вскоре стало бы проблемой. Во избежание этого МСЭ разработал и утвердил специальный стандарт сертификатов. Он называется **X.509** и широко применяется в интернете.

Начиная с 1988 года вышло уже три версии X.509; мы рассмотрим новейшую из них — третью.

На стандарт X.509 сильное влияние оказала система OSI, из которой он позаимствовал худшие ее свойства, в частности принцип кодирования и именования. Удивительно, что IETF согласился с данной концепцией, учитывая, что практически во всех областях, начиная с машинных адресов и заканчивая транспортными протоколами и форматами электронных писем, IETF всегда игнорировал OSI и пытался сделать что-нибудь более толковое. IETF-версия X.509 описана в RFC 5280.

По сути, X.509 — это способ описания сертификатов. Основные поля сертификата перечислены на илл. 8.27. Из описаний, приведенных в правой колонке, должно быть понятно, для чего служит соответствующее поле. За дополнительной информацией обращайтесь к RFC 2459.

| Поле | Значение |
|---|---|
| Version | Версия X.509 |
| Serial number | Это число вместе с именем CA однозначно идентифицирует сертификат |
| Signature algorithm | Алгоритм генерации подписи сертификата |
| Issuer | X.500-имя CA |
| Validity period | Начало и конец периода годности |
| Subject name | Сущность, ключ которой сертифицируется |
| Public key | Открытый ключ сущности и идентификатор использующего его алгоритма |
| Issuer ID | Необязательный идентификатор, единственным образом определяющий эмитента (создателя) сертификата |
| Subject ID | Необязательный идентификатор, единственным образом определяющий владельца сертификата |
| Extensions | Возможны многочисленные расширения |
| Signature | Подпись сертификата (генерируется с помощью закрытого ключа CA) |

**Илл. 8.27.** Основные поля сертификата в стандарте X.509

Например, если Боб работает в отделе ссуд банка Money Bank, его X.500-адрес будет выглядеть так:

```
/C=US/O=MoneyBank/OU=Loan/CN=Bob/
```

где *C* — страна, *O* — организация, *OU* — отдел организации, *CN* — имя. CA и другие сущности именуются похожим образом. Серьезная проблема с системой именования X.500 заключается в том, что если Алиса пытается обратиться к Бобу по адресу bob@moneybank.com и имеет данные сертификата с именем в этом формате,

для нее вовсе не очевидно, что этот сертификат относится именно к тому Бобу, который ей нужен. К счастью, начиная с третьей версии, появилась возможность использовать имена DNS вместо имен X.500, что позволяет наконец забыть об этой проблеме.

Сертификаты кодируются с помощью языка **ASN.1 (Abstract Syntax Notation 1 — абстрактная синтаксическая нотация версии 1)**, используемого в модели OSI. Более подробную информацию о X.509 можно найти в книге Форда и Баума (Ford and Baum, 2000).

## 8.8.3. Инфраструктуры систем с открытыми ключами

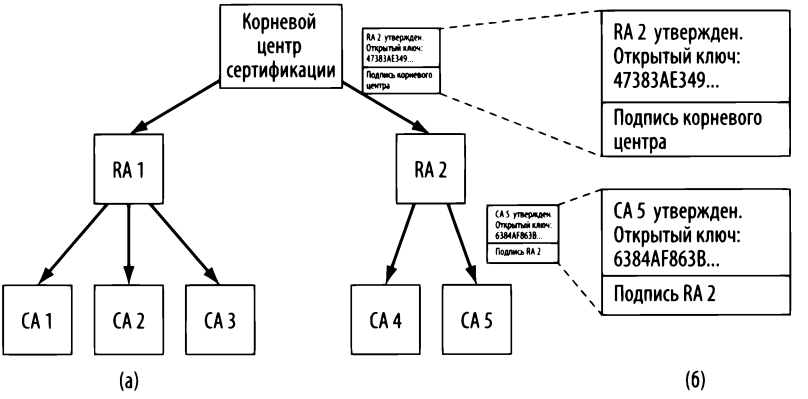
Понятно, что одного CA на весь мир недостаточно. Он бы быстро перестал функционировать из-за огромной нагрузки, да еще и стал бы эпицентром всех проблем, связанных с безопасностью сетей. Возможно, нужно создать целый набор таких CA под руководством одной организации, использующих один и тот же закрытый ключ для подписания сертификатов. Однако хотя это и решит проблему распределения нагрузки, возникнет вопрос утечки ключа. Если по всему миру будут разбросаны десятки серверов, хранящих закрытый ключ CA, велик шанс, что рано или поздно он будет украден или как-то иначе разглашен. Если ключ будет рассекречен, с мировой инфраструктурой электронной безопасности можно будет попрощаться. Вместе с тем наличие всего одного главного CA — это тоже риск.

Следующий вопрос — какая организация будет заведовать CA? Довольно трудно представить себе какую-то структуру, признанную законной и заслуживающей доверия в мировом масштабе. В некоторых странах предпочтительно, чтобы это было государственное учреждение, а где-то — наоборот, чтобы эта структура была какой угодно, но не правительственной.

По этим причинам был разработан альтернативный способ сертификации открытых ключей. Он известен под общим названием **инфраструктуры открытых ключей (Public Key Infrastructure, PKI)**. В этом разделе мы рассмотрим только общие принципы действия PKI, поскольку было внесено множество предложений по ее модификации и некоторые детали могут со временем измениться.

PKI состоит из множества компонентов — пользователей, CA, самих сертификатов, а также каталогов. Она предоставляет возможность структурировать эти компоненты и определяет стандарты, касающиеся различных документов и протоколов. Одним из простейших видов PKI является иерархия CA (илл. 8.28). На рисунке представлены три уровня, но их может быть как больше, так и меньше. CA верхнего уровня (корневой центр сертификации) сертифицирует CA второго уровня — назовем их **региональными (промежуточными) центрами сертификации (Regional Authorities, RA)**, так как они могут обслуживать некоторый географический регион (например, страну или континент). Этот термин не стандартизован. Названия для уровней иерархии вообще не оговариваются стандартом. RA занимаются легализацией реальных CA, эмитирующих сертификаты стандарта X.509 для физических и юридических лиц. Когда корневой центр

авторизует RA, он генерирует подтверждающий полномочия RA сертификат X.509, включает в него открытый ключ нового RA, подписывает его и передает RA. Аналогичным образом RA взаимодействуют с CA: выдают и подписывают сертификаты, содержащие открытые ключи CA и признающие легальность их деятельности.




**Илл. 8.28.** (а) Иерархия PKI. (б) Цепочка сертификатов

Итак, наша PKI работает следующим образом. Допустим, Алисе, чтобы пообщаться с Бобом, нужен его открытый ключ. Она находит сертификат, в котором содержится нужный ключ. Этот сертификат подписан CA 5. Однако Алиса никогда ничего не слышала о CA 5. Этим «CA» вполне может оказаться десятилетняя дочка Боба. Алиса обращается к CA 5 за подтверждением его легитимности. Он предъявляет сертификат, полученный от RA 2 и содержащий открытый ключ CA 5. Теперь, вооружившись открытым ключом CA 5, Алиса может удостовериться в том, что сертификат Боба действительно подписан CA 5, а значит, является легальным.

Если только RA 2 не является двенадцатилетним сыном Боба. Что ж, следующим шагом Алисы будет запрос подтверждения легитимности RA 2. Ответом будет служить сертификат с подписью корневого центра сертификации, содержащий открытый ключ RA 2. Вот теперь Алиса может не сомневаться, что она получила открытый ключ Боба, а не кого-то еще.

А если Алиса хочет узнать открытый ключ корневого CA? Как это сделать? Загадка. Предполагается, что его знают все. Например, он может быть «вшит» в ее браузер.

Но наш Боб — добряк, он не хочет доставлять Алисе лишних хлопот. Он понимает, что она захочет проверить легитимность CA 5 и RA 2, поэтому сам собирает нужные сертификаты и отправляет их вместе со своим. Теперь, зная открытый ключ корневой организации, Алиса может проверить по цепочке все интересующие ее компоненты. Ей не придется никого беспокоить для подтверждения. Поскольку все сертификаты подписаны, она может запросто уличить любые попытки подлога. Цепочка сертификатов, восходящая к корню, иногда

называется **доверительной цепочкой (chain of trust)** или **путем сертификации (certification path)**. Описанный метод широко применяется на практике.

Конечно, остается проблема определения владельца корневой организации. Лучше всего иметь несколько таких структур, причем связать с каждой из них свою иерархию RA и CA. На самом деле в современные браузеры действительно «зашиваются» открытые ключи более 100 корневых CA, иногда называемые **доверительными якорями (trust anchors)**. Как видите, можно избежать проблемы одного всемирного учреждения, занимающегося сертификацией.

Однако встает вопрос, какие доверительные якоря производители браузеров могут считать надежными, а какие — нет. На самом деле все сводится к тому, насколько конечный пользователь доверяет разработчику браузера и уверен в том, что решения генерируются грамотно, а доверительные якоря не принимаются от всех желающих за умеренную плату. Большинство браузеров позволяют пользователям проверять корневые ключи (обычно в виде сертификатов, подписанных корневым CA) и удалять те, которые кажутся сомнительными. Более подробную информацию о PKI можно найти в книге Стэплтона и Эпштейна (Stapleton and Epstein, 2016).

### Каталоги

PKI должна решать еще один вопрос. Он касается места хранения сертификатов (и цепочек, ведущих к какому-нибудь доверительному якорю). Можно заставить всех пользователей хранить свои сертификаты у себя. Это безопасно (так как невозможно подделать подписанные сертификаты незаметно), но не слишком удобно. В качестве каталога для сертификатов было предложено использовать DNS. Скорее всего, прежде чем соединиться с Бобом, Алисе все равно придется узнать его IP-адрес с помощью DNS. Так почему бы DNS не возвращать вместе с IP-адресом всю цепочку сертификатов?

Кому-то это кажется выходом из положения, но некоторые считают, что лучше иметь специализированные серверы с каталогами для хранения сертификатов X.509. Такие каталоги могли бы обеспечить возможность поиска с помощью имен X.500. Например, теоретически можно представить себе службу сервера каталогов, позволяющую получать ответы на запросы типа «Дайте мне полный список всех сотрудников с именем Алиса, работающих в отделах продаж по всей стране».

### Отзыв сертификата

Реальный мир полон разнообразных сертификатов — паспорта, водительские удостоверения и т. д. Иногда их необходимо аннулировать (например, водительское удостоверение объявляется недействительным за езду в нетрезвом виде). Та же проблема возникает и в мире цифровых технологий: лицо, предоставившее сертификат, может отозвать его за нарушение противоположной стороной каких-либо условий. Это необходимо делать и в том случае, если закрытый ключ сущности (а еще хуже, ключ CA) был рассекречен. Таким образом, PKI должна обеспечивать процедуру отзыва сертификата, хотя это все усложняет.

Прежде всего, CA должны периодически выпускать **список отозванных сертификатов (Certificate Revocation List, CRL)**. В нем перечисляются их порядковые номера. Поскольку в сертификатах содержится дата окончания срока годности, в CRL следует включать номера только тех из них, срок годности которых еще не истек. По истечении срока годности сертификаты перестают быть действительными автоматически, это не то же самое, что отозванные сертификаты. В обоих случаях использовать их больше нельзя.

К сожалению, наличие CRL означает, что перед использованием сертификата нужно убедиться в том, что его нет в списке. В противном случае от него следует отказаться. При этом сертификат может быть отозван сразу после выпуска самого свежего варианта списка. Получается, что единственный надежный способ узнать о состоянии сертификата — обратиться непосредственно к CA. Эти запросы придется отсылать при каждом использовании сертификата, так как нет никакой гарантии, что его не отозвали несколько секунд назад.

Еще больше усложняет ситуацию то, что иногда требуется восстановление отозванного сертификата. Например, если причиной отзыва была неуплата каких-либо взносов, после внесения необходимой суммы не остается никаких причин для отказа в восстановлении сертификата. Необходимость заниматься отзывом (и возможным восстановлением) сводит на нет такое ценное свойство сертификатов, как возможность их использования без обращения к CA.

Где же хранить списки недействительных сертификатов? Было бы удобно хранить их там же, где и сами сертификаты. Одна из стратегий подразумевает, что CA периодически выпускает CRL и каталоги обновляются (отозванные сертификаты просто удаляются из них). Если для хранения сертификатов каталоги не используются, можно кэшировать их в разных местах в сети. Поскольку CRL сам по себе является подписанным документом, любые попытки подлога тотчас будут замечены.

Если у сертификатов большие сроки годности, CRL также будут довольно длинными. Аналогично, число заблокированных кредитных карточек будет гораздо выше, если их срок годности равен 5 годам, а не 3 месяцам. Стандартный способ борьбы с длинными списками — редко выпускать сами CRL, но часто их обновлять. Помимо прочего, это снижает необходимую для распространения CRL пропускную способность.

## 8.9. ПРОТОКОЛЫ АУТЕНТИФИКАЦИИ

**Аутентификация** — это метод, с помощью которого процесс проверяет, является ли его собеседник тем, за кого он себя выдает. Проверка подлинности удаленного процесса при активных умышленных попытках проникновения представляет собой на удивление сложную задачу и требует применения сложных протоколов, основанных на криптографии. В данном разделе мы познакомимся с несколькими протоколами аутентификации, которые используются в незащищенных компьютерных сетях.

Следует отметить, что иногда аутентификацию путают с авторизацией. Аутентификация связана с вопросом подлинности процесса, с которым происходит

взаимодействие. Авторизация касается того, что этому процессу разрешено делать. Например, клиентский процесс обращается к файловому серверу и говорит: «Я процесс Скотта, и я хочу удалить файл cookbook.old». Файл-сервер должен ответить на два вопроса:

1. Действительно ли это процесс Скотта (аутентификация)?
2. Есть ли у Скотта право на удаление файла cookbook.old (авторизация)?

Только получив однозначный утвердительный ответ на оба вопроса, можно выполнить запрошенное действие. Ключевым является первый вопрос. После того как сервер узнает, с кем он разговаривает, для проверки прав доступа нужно лишь просмотреть содержимое локальных таблиц или баз данных. По этой причине в данном разделе мы сосредоточимся на аутентификации.

Общая схема, используемая практически всеми протоколами аутентификации, состоит в следующем. Алиса желает установить защищенное соединение с Бобом или считающимся надежным KDC. Затем в разных направлениях передается еще несколько сообщений. При этом Труди может их перехватить, изменить и повторно воспроизвести, чтобы обмануть Алису и Боба или просто сорвать сделку.

Тем не менее, когда протокол завершает свою работу, Алиса уверена, что разговаривает с Бобом, а он — что разговаривает с Алисой. Кроме того, в большинстве протоколов собеседники могут установить секретный **ключ сеанса (session key)**, чтобы использовать его в предстоящем обмене информацией. На практике, по соображениям производительности, поток данных кодируется с помощью алгоритма с симметричным ключом (обычно это AES), а шифрование с открытым ключом широко применяется для самих алгоритмов аутентификации и для установления ключа сеанса.

Существует несколько причин, по которым для каждого нового соединения используется новый, случайно выбранный ключ сеанса. Это снижение количества трафика, передаваемого с использованием закрытых и открытых ключей пользователя, уменьшение объема зашифрованного текста, который может получить злоумышленник, а также минимизация вреда, причиняемого в случае, если процесс даст сбой и дамп ядра (содержимое памяти) попадет не в те руки. Желательно, чтобы при этом в процессе хранился только ключ сеанса. Все постоянные ключи должны быть удалены после установления соединения.

## 8.9.1. Аутентификация на основе общего секретного ключа

Для нашего первого протокола аутентификации предположим, что у Алисы и Боба есть общий секретный ключ $K_{AB}$. О нем можно договориться лично или по телефону, но только не по незащищенной сети.

В основе данного протокола лежит принцип, актуальный для многих протоколов аутентификации: одна сторона отправляет другой случайное число, а та особым образом преобразует его и возвращает результат. Такие протоколы называют **запросно-ответными (challenge-response)**. При рассмотрении

протоколов аутентификации будут использоваться следующие условные обозначения:

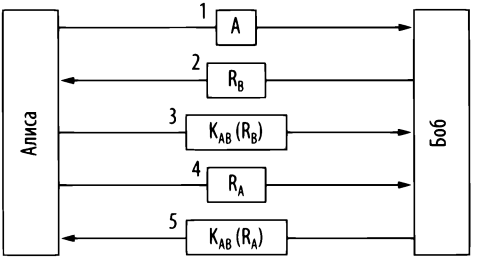
$A$ и $B$ — Алиса и Боб;

$R_i$ — запросы, где $i$ — индекс отправителя;

$K_i$ — ключи, где $i$ — индекс владельца ключа;

$K_S$ — ключ сеанса.

Последовательность сообщений данного протокола аутентификации с общим ключом показана на илл. 8.29. В первом сообщении Алиса отсылает свое удостоверение личности $A$ Бобу (тем способом, который ему понятен). Боб, конечно, не знает, от кого пришло это сообщение — от Алисы или от Труди, поэтому он выбирает большое случайное число $R_B$ и отправляет его в качестве запроса «Алисе» открытым текстом (сообщение 2). Затем Алиса шифрует это сообщение секретным ключом, общим для нее и Боба, и возвращает зашифрованный текст $K_{AB}(R_B)$ в сообщении 3. Когда Боб видит это сообщение, он сразу понимает, что оно пришло именно от Алисы, поскольку Труди не располагает ключом $K_{AB}$ и потому не может сформировать такое сообщение. Более того, поскольку запрос $R_B$ выбирался случайным образом из большого пространства чисел (скажем, 128-битных), вероятность того, что Труди уже видела этот запрос и соответствующий ответ в одном из предыдущих сеансов, крайне низка. И вряд ли ей удастся подобрать правильный ответ на запрос наугад.

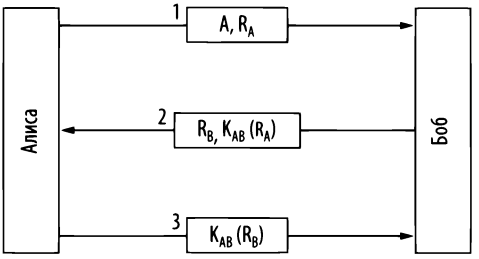



**Илл. 8.29.** Двусторонняя аутентификация с использованием запросно-ответного протокола

К этому моменту Боб уверен, что говорит с Алисой, однако она еще пока ни в чем не уверена. Алиса понимает, что Труди могла перехватить сообщение 1 и отправить обратно запрос $R_B$. Возможно, Боба уже нет в живых. Чтобы узнать, с кем она говорит, Алиса выбирает случайное число $R_A$ и отправляет его Бобу в виде открытого текста (сообщение 4). Когда Боб возвращает ответ $K_{AB}(R_A)$, это убеждает Алису в том, что она говорит именно с ним. После этого они могут установить временный ключ сеанса $K_S$, который можно переслать друг другу, закодировав его все тем же общим ключом $K_{AB}$.

Число сообщений в этом протоколе можно сократить, объединив в каждом из них ответ на предыдущее сообщение с новым запросом (илл. 8.30). Здесь Алиса сама в первом же сообщении отправляет запрос Бобу. Отвечая на него,

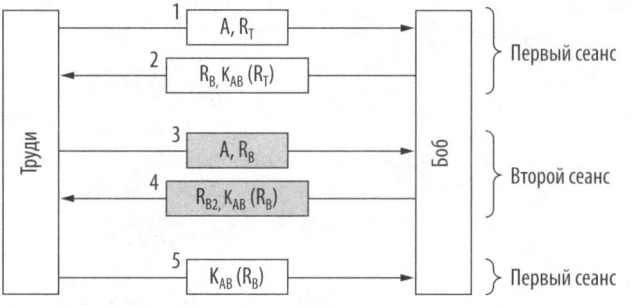
Боб помещает в это же сообщение свой запрос. Таким образом, вместо пяти сообщений понадобилось всего три.




**Илл. 8.30.** Укороченный двусторонний протокол аутентификации

Можно ли утверждать, что этот протокол лучше изначального? С одной стороны, да, ведь он короче. Но, к сожалению, применять его не рекомендуется. При некоторых обстоятельствах Труди может атаковать этот протокол, используя **зеркальную атаку (reflection attack)**. В частности, она может взломать его, если с Бобом можно открыть сразу несколько сеансов связи. Это вполне возможно, если Боб, скажем, является банком, который готов установить множество соединений с банкоматами одновременно.

Схема зеркальной атаки показана на илл. 8.31. Она начинается с того, что Труди, объявляя себя Алисой, отсылает запрос $R_T$. Боб, как обычно, отвечает своим собственным запросом $R_B$. Теперь, казалось бы, Труди в тупике. Что ей делать? Она ведь не знает $K_{AB}(R_B)$.




**Илл. 8.31.** Зеркальная атака

Однако Труди может открыть второй сеанс сообщением 3 и подать в качестве запроса Бобу его собственный запрос, взятый из сообщения 2. Боб спокойно шифрует его и отсылает обратно $K_{AB}(R_B)$ в сообщении 4. Сообщения второго сеанса выделены в нашей схеме серым цветом. Труди получила нужную информацию, поэтому она завершает первый сеанс и прерывает второй. Теперь Боб уверен, что Труди — это Алиса, поэтому предоставляет ей доступ к банковским счетам Алисы и позволяет перевести деньги с ее текущего счета на секретный счет в швейцарском банке без малейших колебаний.

Мораль истории такова:

*Разработать корректный протокол аутентификации гораздо сложнее, чем это может показаться.*

Следующие четыре общих правила помогают разработчикам избежать распространенных ошибок.

1. Инициатор сеанса должен подтверждать свою личность прежде, чем это сделает отвечающая сторона. Это помешает злоумышленнику получить ценную для него информацию, прежде чем он подтвердит свою личность.
2. Следует использовать два раздельных общих секретных ключа: один для инициатора сеанса, а другой для отвечающего: $K_{AB}$ и $K'_{AB}$.
3. Инициатор и отвечающий должны выбирать запросы из разных наборов. Например, инициатор может использовать четные числа, а отвечающий — нечетные.
4. Протокол должен уметь противостоять атакам, при которых запускается второй параллельный сеанс, информация для которого извлекается из первого сеанса (или наоборот).

Если нарушается хотя бы одно из этих правил, протокол становится уязвимым. В приведенном примере игнорировались все четыре правила, что привело к разрушительным последствиям.

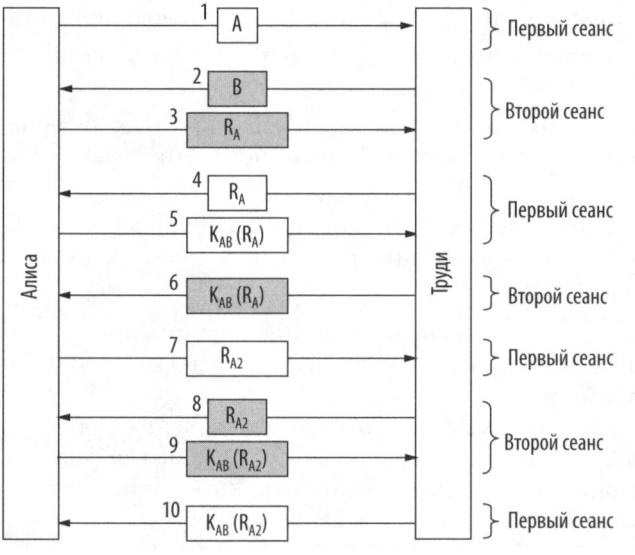
Вернемся к ситуации, показанной на илл. 8.29. Подвержен ли этот протокол зеркальным атакам? Точно сказать нельзя. Труди удалось справиться с нашим протоколом с помощью зеркальной атаки, поскольку он позволял запустить параллельный сеанс с Бобом и ввести его в заблуждение его собственным запросом. А что произойдет, если Алиса — обычный компьютер общего назначения, принимающий параллельные сеансы связи, а не человек? Посмотрим, что Труди сможет сделать.

Чтобы понять, каким образом Труди взламывает протокол, обратимся к илл. 8.32. Алиса объявляет свои идентификационные данные в сообщении 1. Труди это сообщение перехватывает и запускает собственный сеанс, отправляя сообщение 2 и прикидываясь Бобом. Здесь мы, как и раньше, изобразили серыми прямоугольниками сообщения второго сеанса. Алиса отвечает на сообщение 2: «Ты утверждаешь, что ты Боб? Докажи» (сообщение 3). Здесь Труди заходит в тупик: она не может подтвердить, будто она — это Боб.

Что же ей делать? Она возвращается к первому сеансу, где как раз наступает ее очередь отправки запроса. При этом отправляется запрос $R_A$, полученный в сообщении 3. Алиса любезно отвечает на это в сообщении 5, предоставляя тем самым Труди информацию, необходимую ей для создания сообщения 6 в сеансе 2. Теперь Труди может выбирать сеанс, так как она корректно ответила на запрос Алисы во втором сеансе. Сеанс 1 можно закрыть, отправить в сеансе 2 любое старое число и получить аутентифицированный сеанс связи с Алисой.

Но будучи перфекционисткой, Труди очень хочет показать, на что она способна. Вместо того чтобы отправить какое-нибудь старое число для завершения регистрации сеанса 2, она ждет, пока Алиса пришлет сообщение 7 (ее запрос для

сеанса 1). Конечно, Труди понятия не имеет, как на него ответить, поэтому она вновь проводит зеркальную атаку, отправляя запрос $R_{A2}$ в сообщении 8. Алиса очень кстати шифрует $R_{A2}$ в сообщении 9. Труди переключается на сеанс 1 и отправляет Алисе нужное число в сообщении 10. Откуда она берет это число? Очевидно, из сообщения 9, пришедшего от Алисы. С этого момента Труди может гордиться тем, что у нее есть два полностью аутентифицированных сеанса связи с Алисой.

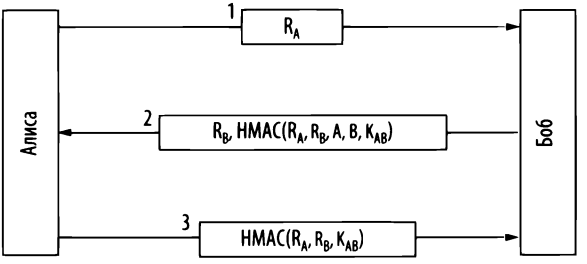



**Илл. 8.32.** Зеркальная атака протокола, показанного на илл. 8.29

Эта атака приводит к несколько иному результату, нежели протокол с тремя сообщениями, который мы видели на илл. 8.31. На этот раз Труди удается установить сразу два аутентифицированных соединения с Алисой. В предыдущем примере одно такое соединение было установлено с Бобом. Опять же, если бы протокол удовлетворял всем четырем перечисленным требованиям, эту атаку можно было бы остановить. Различные типы атак и методы борьбы с ними подробно обсуждаются в работе Берда и др. (Bird et al., 1993). В ней вы также найдете описание планомерного построения протоколов, корректность которых можно доказать. Однако даже самый простой из них довольно сложен, поэтому далее мы рассмотрим другой класс протоколов, работающих ничуть не хуже.

Итак, новый протокол аутентификации показан на илл. 8.33 (Берд и др.; Bird et al., 1993). В нем используется **код аутентификации сообщения на основе хеширования (Hashed Message Authentication Code, HMAC)**, гарантирующий целостность и подлинность сообщения. Простой и в то же время мощный код HMAC состоит из хеша на основе сообщения и общего ключа. Отправка HMAC вместе с сообщением не позволяет злоумышленнику изменить или подделать данные: смена любого бита приведет к неправильному хешу. Кроме того,

сгенерировать корректный хеш невозможно, не располагая ключом. Коды HMAC привлекательны тем, что их можно очень эффективно генерировать (быстрее по сравнению с выполнением алгоритма SHA-2 с последующим применением к результатам алгоритма RSA).




**Илл. 8.33.** Аутентификация с применением кодов HMAC

Для начала Алиса отправляет Бобу случайно выбранное число $R_A$ (сообщение 1). Такие случайно выбранные числа, которые применяются в протоколах безопасности только один раз, принято называть **нонсами (nonce)**; это сокращение от английского выражения «number used once» — «однократно используемое число». Боб при ответе выбирает собственный нонс $R_B$ и высылает его вместе с кодом HMAC. Этот код формируется путем построения структуры данных, состоящей из нонсов Алисы и Боба, их идентификаторов, а также общего секретного ключа $K_{AB}$. Затем вся эта структура с помощью хеш-функции (например, SHA-2) помещается в HMAC. После приема сообщения 2 у Алисы имеется значение $R_A$ (она сама его выбрала), значение $R_B$, полученное в виде открытого текста, два идентификатора и секретный ключ $K_{AB}$, который она и так знала. С помощью этих данных она может вычислить HMAC самостоятельно. Если он совпадает с кодом HMAC в сообщении, она убеждается, что говорит с Бобом. Ведь Труди не знает $K_{AB}$ и, следовательно, не может угадать HMAC, который следует отослать. Алиса отправляет Бобу HMAC, содержащий только два нонса.

Вопрос: может ли Труди взломать такой протокол? Нет, поскольку она не может заставить ни одну из сторон зашифровать или хешировать выбранное ею значение, как было показано на илл. 8.31 и 8.32. Оба кода HMAC содержат значения, выбранные отправителем, и Труди не способна их контролировать.

Коды HMAC — далеко не единственный вариант применения этой идеи. Довольно распространенная альтернативная схема заключается в шифровании элементов данных последовательно, с помощью сцепления блоков шифра.

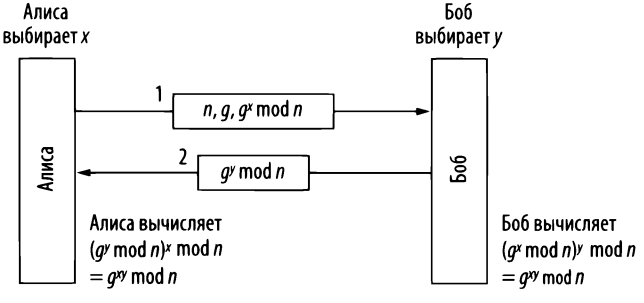
## 8.9.2. Установка общего ключа: протокол обмена ключами Диффи — Хеллмана

До сих пор мы подразумевали, что у Алисы и Боба есть общий секретный ключ. Теперь предположим, что такого ключа нет (поскольку до сих пор не разработана универсальная инфраструктура PKI для создания подписей и распространения

сертификатов). Как же его установить? Алиса может позвонить Бобу и передать ключ по телефону, но Боб, скорее всего, спросит: «Как вы докажете, что вы — Алиса, а не Труди?» Можно попытаться организовать встречу, на которую каждый придет с паспортом, водительскими правами и тремя кредитными картами. Однако, будучи занятыми людьми, Алиса и Боб могут месяцами искать удобную дату. К счастью, совершенно незнакомые люди могут установить общий секретный ключ среди бела дня, даже если злоумышленник старательно записывает каждое сообщение.

Протокол, позволяющий устанавливать общий секретный ключ людям, которые друг друга не знают, называется **протоколом обмена ключами Диффи — Хеллмана (Diffie — Hellman key exchange)** (Diffie and Hellman, 1976). Он работает следующим образом. Алиса и Боб договариваются о двух больших простых числах, $n$ и $g$, где $(n - 1)/2$ также является простым числом, кроме того, на число $g$ накладываются некоторые дополнительные ограничения. Эти числа могут быть публичными, поэтому каждый просто устанавливает значения $n$ и $g$ и открыто сообщает о них собеседнику. Затем Алиса выбирает большое (например, двоичное 1024-разрядное) число $x$ и держит его в тайне. Аналогично, Боб выбирает большое секретное число $y$.

Алиса запускает протокол обмена ключами, отправив Бобу сообщение (открытым текстом), содержащее $(n, g, g^x \bmod n)$, как показано на илл. 8.34. Боб отвечает ей сообщением, содержащим $g^y \bmod n$. Алиса возводит присланное Бобом число в степень $x$ и делит его по модулю на $n$, получая $(g^y \bmod n)^x \bmod n$. Боб выполняет аналогичные вычисления и получает $(g^x \bmod n)^y \bmod n$. Согласно законам модульной арифметики, оба вычисления должны быть равны $g^{xy} \bmod n$. Таким образом, как по мановению волшебной палочки, у Алисы и Боба появляется общий секретный ключ $g^{xy} \bmod n$.

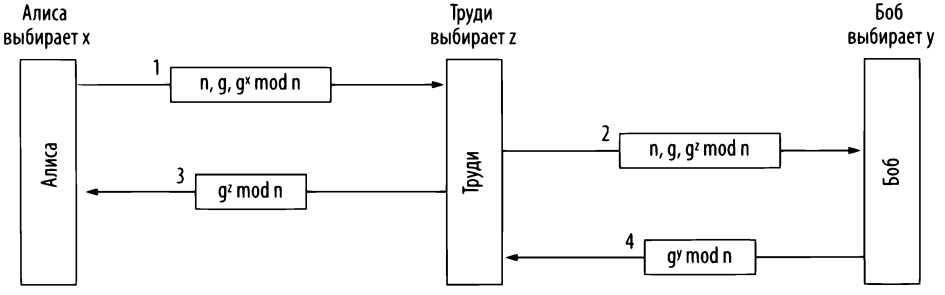



**Илл. 8.34.** Протокол обмена ключами Диффи — Хеллмана

Конечно, Труди видит оба сообщения. Ей известны значения $g$ и $n$ из первого сообщения. Если бы она могла вычислить значения $x$ и $y$, ей бы удалось получить секретный ключ. Беда в том, что, зная только $g^x \bmod n$, найти значение $x$ очень трудно. На сегодняшний день неизвестен алгоритм вычисления дискретного логарифма модуля очень большого простого числа.

Для примера возьмем значения $n = 47$ и $g = 3$ (абсолютно нереалистичные). Алиса выбирает значение $x = 8$, а Боб — $y = 10$. Эти числа хранятся в секрете. Сообщение Алисы Бобу содержит числа (47, 3, 28), так как $3^8 \bmod 47 = 28$. Боб отвечает Алисе числом 17. Алиса вычисляет $17^8 \bmod 47$ и получает 4. Боб считает $28^{10} \bmod 47$ и тоже получает 4. Таким образом, независимо друг от друга, Алиса и Боб определили, что значение секретного ключа равно 4. Чтобы найти ключ, Труди придется решить уравнение $3^v \bmod 47 = 28$. Это можно сделать путем полного перебора при использовании небольших чисел, но не в случае чисел длиной в несколько сотен битов. Сегодня все известные алгоритмы требуют для этого вычисления гигантских временных затрат, даже при использовании сверхбыстрых суперкомпьютеров с десятками миллионов ядер.

Несмотря на всю элегантность алгоритма Диффи — Хеллмана, есть одна проблема: когда Боб получит три числа (47, 3, 28), как удостовериться в том, что их отправила Алиса, а не Труди? Нет никакого способа узнать это. К сожалению, Труди может воспользоваться этим, чтобы обмануть Алису и Боба (илл. 8.35). Пока Алиса и Боб устанавливают значения $x$ и $y$, Труди выбирает свое случайное число $z$. Алиса отсылает Бобу сообщение 1. Труди перехватывает его и вместо него отправляет Бобу сообщение 2, используя правильные значения $g$ и $n$ (которые передавались открытым текстом), но со своим значением $z$ вместо $x$. Она также отсылает сообщение 3 обратно Алисе. Позднее Боб отправляет Алисе сообщение 4, которое Труди снова перехватывает и хранит у себя.




**Илл. 8.35.** Атака посредника

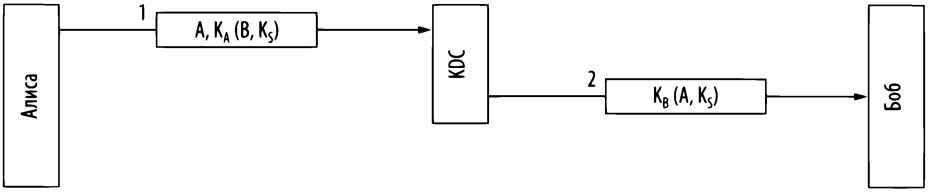
Теперь все занимаются вычислением остатков от деления. Алиса вычисляет значение секретного ключа $g^{xz} \bmod n$. Те же самые подсчеты производит Труди (для общения с Алисой). Боб вычисляет $g^{yz} \bmod n$, что также делает и Труди (для общения с Бобом). Думая, что она общается с Бобом, Алиса устанавливает ключ сеанса (с Труди), и точно так же поступает Боб. Каждое сообщение, отправляемое Алисой в шифрованном сеансе, перехватывается Труди, сохраняется, изменяется, если нужно, и отсылается (по желанию Труди) Бобу. То же самое происходит в обратном направлении. Труди видит все сообщения и может менять их по своему усмотрению, в то время как Алиса и Боб полагают, что у них имеется защищенный канал для связи друг с другом. Подобные действия

злоумышленника называются **«атакой посредника»** или **атакой типа «человек посередине» (man-in-the-middle attack)**[1].

## 8.9.3. Аутентификация с помощью центра распространения ключей

Итак, установление общего секретного ключа с незнакомцем почти удалась. Хотя, вполне возможно, что результат не стоит усилий. Чтобы общаться с $n$ людьми, нужно хранить $n$ ключей. Для людей, чей круг общения широк, хранение ключей может превратиться в серьезную проблему, особенно если все эти ключи придется хранить на физических токенах.

Другой подход состоит в том, чтобы ввести в систему доверенный **Центр распространения ключей (Key Distribution Center, KDC)**. Им может стать, к примеру, банк или какой-либо государственный орган. При такой схеме у каждого пользователя всего один ключ, общий с KDC. Через KDC проходят операции с сеансовыми ключами и ключами аутентификации. На илл. 8.36 представлена простейшая схема такого протокола аутентификации, где присутствуют две стороны и доверенный KDC.




**Илл. 8.36.** Первая попытка реализации протокола аутентификации на основе KDC

У этого протокола достаточно простой принцип действия: Алиса выбирает ключ сеанса $K_S$ и уведомляет KDC о том, что она хочет поговорить с Бобом, используя $K_S$. Это сообщение шифруется секретным ключом $K_A$, который известен только Алисе и KDC. KDC расшифровывает это сообщение и извлекает из него идентификатор личности Боба и ключ сеанса. Затем он формирует новое сообщение, которое содержит идентификатор личности Алисы и ключ сеанса, и отправляет его Бобу. Это сообщение зашифровывается секретным ключом $K_B$ (он известен только Бобу и KDC). Расшифровав сообщение, Боб понимает, что Алиса желает с ним поговорить, и узнаёт, какой ключ она хочет использовать.

Аутентификация в данном случае происходит сама собой. KDC знает, что сообщение 1 пришло от Алисы, так как больше никто не может зашифровать его секретным ключом Алисы. Аналогично, Боб знает, что сообщение 2 пришло от KDC, ведь кроме него их общий секретный ключ никому не известен.

---

[1] Также ее называют атакой по принципу пожарной цепочки (bucket brigade attack), поскольку она напоминает действия пожарной бригады прежних времен — передачу ведер с водой от пожарной машины к месту возгорания.

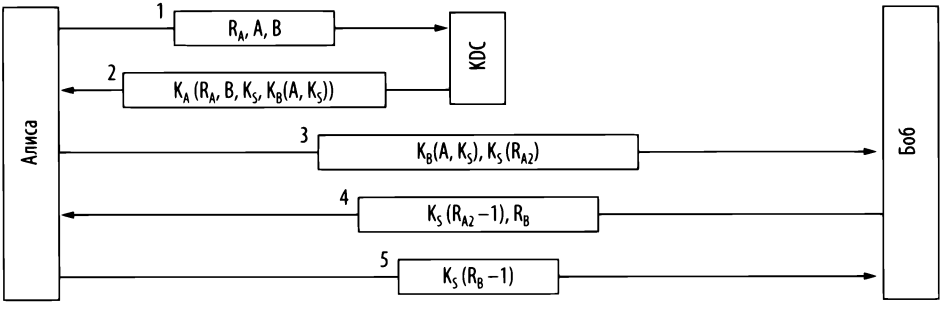
К сожалению, у этого протокола есть серьезный недостаток. Труди нужны деньги, и она предлагает Алисе некую легальную услугу на выгодных условиях. Алиса соглашается, и Труди получает работу. Выполнив ее, она вежливо просит Алису оплатить услугу банковским переводом. Алиса устанавливает ключ сеанса со своим менеджером Бобом и отправляет ему сообщение с просьбой перевести деньги на счет Труди.

Тем временем Труди возвращается к своим темным делам. Она копирует сообщение 2 (см. илл. 8.36) и запрос на перевод денег, следующий за ним. Затем она воспроизводит оба сообщения для Боба. Боб получает их и думает: «Должно быть, Алиса снова наняла Труди. Похоже, она хорошо справляется с работой». Боб перечисляет еще столько же денег со счета Алисы на счет Труди. Получив пятидесятый запрос на перевод, Боб выбегает из офиса, чтобы найти Труди и предложить ей кредит, чтобы она могла расширить свой чрезвычайно успешный бизнес. Атаки такого рода называют **атаками повторного воспроизведения (replay attack)**.

Существует несколько способов борьбы с ними. Первый заключается в том, чтобы поместить в каждое сообщение временную метку. Это позволяет отбрасывать все устаревшие сообщения. Однако системные часы в сети невозможно точно синхронизировать, поэтому нужен определенный срок годности временной метки. Труди может обмануть протокол, отправив повторное сообщение во время этого интервала.

Второе решение сводится к добавлению в сообщение однократно используемого числа — нонса. Каждая сторона должна запоминать все предыдущие нонсы и отвергать любое сообщение, содержащее использованный ранее нонс. При этом нонсы должны храниться вечно, иначе Труди попытается воспроизвести сообщение пятилетней давности. Кроме того, если компьютер потеряет список нонсов в результате сбоя, он снова станет уязвимым к атакам повторного воспроизведения. Можно комбинировать временные метки и нонсы, чтобы ограничить срок хранения последних, но это сильно усложняет протокол.

Более продвинутый метод взаимной аутентификации состоит в использовании многостороннего запросно-ответного протокола. Хорошо известный пример — **протокол аутентификации Нидхема — Шредера (Needham — Schroeder authentication protocol)** (Needham and Schroeder, 1978). Один из его вариантов показан на илл. 8.37.




**Илл. 8.37.** Протокол аутентификации Нидхема — Шредера

Работа протокола начинается с того, что Алиса заявляет KDC о своем желании поговорить с Бобом. Это сообщение содержит в качестве нонса большое случайно выбранное число $R_A$. KDC отсылает обратно сообщение 2 со случайным числом Алисы, ключом сеанса и **удостоверением (ticket)**[1], которое она может передать Бобу. Цель отправки случайного числа $R_A$ состоит в том, чтобы убедить Алису, что сообщение 2 свежее, а не повторно воспроизведенное. Идентификатор Боба также помещается в сообщение 2 на случай, если Труди вздумает заменить его идентификатор в сообщении 1 на свой, чтобы KDC зашифровал удостоверение в конце сообщения 2 ключом $K_T$ (ключ Труди) вместо $K_B$. Удостоверение, зашифрованное ключом $K_B$, помещается в зашифрованное сообщение, чтобы Труди не смогла заменить его чем-то другим, пока сообщение 2 добирается до Алисы.

Затем Алиса отсылает Бобу удостоверение вместе с новым случайным числом $R_{A2}$, зашифрованным ключом сеанса $K_S$. В сообщении 4 Боб отправляет обратно $K_S(R_{A2} - 1)$, чтобы доказать Алисе, что она разговаривает именно с ним. Передавать обратно $K_S(R_{A2})$ бессмысленно, так как Труди могла украсть это число из сообщения 3.

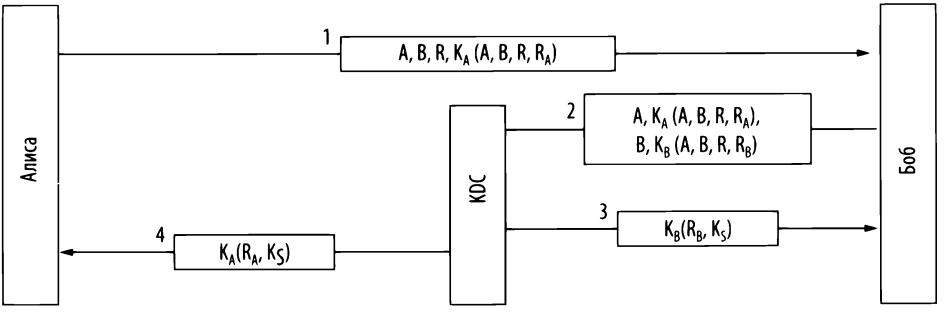
Получив сообщение 4, Алиса убеждается, что разговаривает с Бобом и что до сих пор повторные сообщения не использовались. Между отправкой случайного числа $R_{A2}$ и получением ответа на него в виде $K_S(R_{A2} - 1)$ проходит достаточно мало времени. Цель сообщения 5 — убедить Боба, что он действительно разговаривает с Алисой и что в этом сеансе связи также отсутствуют повторно воспроизведенные данные. Этот протокол исключает возможность атаки повторного воспроизведения ранее записанной информации, поскольку каждая сторона формирует запрос другой стороны и получает на него ответ.

Несмотря на кажущуюся солидность протокола, у него есть небольшой недостаток. Если Труди удастся как-то раздобыть старый ключ сеанса $K_S$, она сможет инициировать новый сеанс с Бобом, повторно воспроизведя сообщение 3 с использованием скомпрометированного ключа, и выдать себя за Алису (Деннинг и Сакко; Denning and Sacco, 1981). На этот раз Труди может украсть деньги со счета Алисы, даже не выполнив никаких услуг.

Позднее Нидхем и Шредер опубликовали протокол для исправления этой ситуации (Needham and Shroeder, 1987). В том же выпуске журнала Отуэй и Рис (Otway and Rees, 1987) представили свой протокол, решающий проблему более коротким путем. На илл. 8.38 показан слегка видоизмененный **протокол Отуэя — Риса (Otway — Rees protocol)**.

В протоколе Отуэя — Риса Алиса прежде всего формирует пару случайных чисел: $R$, необходимое в качестве общего идентификатора, и $R_A$, которое Алиса будет использовать в качестве запроса для Боба. Получив это сообщение, Боб формирует новое сообщение из зашифрованной части сообщения Алисы и аналогичной собственной части. Обе части сообщения, зашифрованные ключами $K_A$ и $K_B$, идентифицируют Алису и Боба: в них содержится общий идентификатор и запрос.

[1] В литературе их также называют билетами. — *Примеч. ред.*




**Илл. 8.38.** Протокол аутентификации Отуэя — Риса (в слегка упрощенном виде)

KDC сравнивает общие идентификаторы $R$ в обеих частях сообщения. Они могут не совпадать, если Труди подменила $R$ в сообщении 1 или заменила часть сообщения 2. Если оба числа $R$ совпадают, KDC считает сообщение от Боба достоверным. Затем он формирует ключ сеанса $K_S$ и шифрует его дважды: для Алисы и для Боба. Каждое сообщение содержит случайное число получателя как доказательство того, что эти сообщения сгенерировал KDC, а не Труди. К этому моменту Алиса и Боб обладают одним и тем же ключом сеанса и могут начать коммуникацию. После первого же обмена данными они увидят, что обладают одинаковыми копиями ключа сеанса $K_S$, и процесс аутентификации можно будет считать завершенным.

## 8.9.4. Аутентификация при помощи протокола Kerberos

Во многих реальных системах (включая Windows) применяется протокол аутентификации **Kerberos**, основанный на варианте протокола Нидхема — Шредера. Он назван по имени трехглавого пса из древнегреческой мифологии Кербера (или Цербера), охранявшего выход из царства Аида. Kerberos был разработан в Массачусетском технологическом институте, чтобы предоставить пользователям рабочих станций надежный доступ к сетевым ресурсам. Его основное отличие от протокола Нидхема — Шредера — предположение о довольно хорошей синхронизации всех часов в сети. Было разработано несколько версий протокола. Версия V5 наиболее широко применяется в промышленности и описана в RFC 4120. От более ранней версии V4 окончательно отказались, после того как в ней были найдены серьезные недостатки (Юй и др.; Yu et al., 2004). V5 усовершенствована по сравнению с V4 — внесено множество мелких изменений и улучшены некоторые характеристики. Например, протокол больше не опирается на устаревший DES. Более подробные сведения можно найти в книге Суда (Sood, 2012).

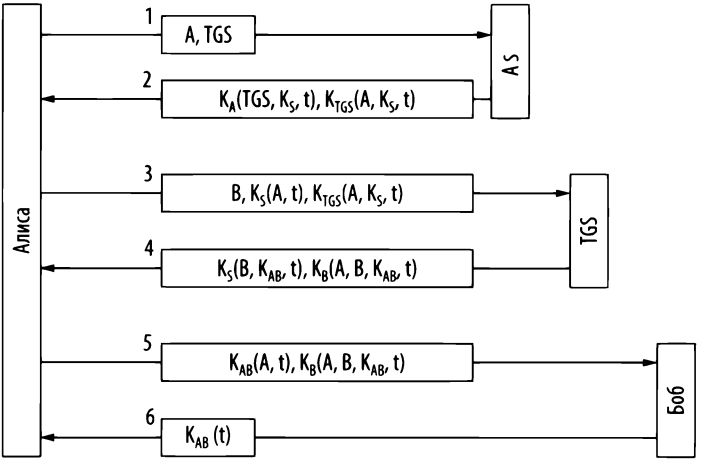
В работе Kerberos помимо Алисы (клиентской рабочей станции) принимают участие еще три сервера:

1. Сервер аутентификации (Authentication Server, AS): проверяет личность пользователей при входе в систему.

2. Сервер выдачи удостоверений (Ticket Granting Server, TGS): предоставляет удостоверения, подтверждающие полномочия объекта.
3. Сервер, предоставляющий услуги Алисе (Боб).

AS похож на KDC тем, что у него есть общий секретный пароль для каждого пользователя. Работа TGS состоит в выдаче удостоверений, убеждающих другие серверы в том, что обладатель удостоверения действительно является тем, за кого себя выдает.

Чтобы начать сеанс, Алиса усаживается за произвольную публичную рабочую станцию и вводит свое имя. Рабочая станция передает введенное имя и название TGS открытым текстом на AS (сообщение 1 на илл. 8.39). В ответ она получает ключ сеанса и удостоверение $K_{TGS}(A, K_S, t)$, предназначенное для TGS. Ключ сеанса зашифровывается секретным ключом Алисы, чтобы только она могла его расшифровать. Только после получения сообщения 2 рабочая станция запрашивает пароль Алисы, и никак не раньше. С помощью этого пароля формируется ключ $K_A$, чтобы расшифровывать сообщение 2 и извлечь из него ключ сеанса.




**Илл. 8.39.** Работа протокола Kerberos V5

После расшифровки рабочая станция сразу же уничтожает хранящийся в ее памяти пароль. Если вместо Алисы на рабочей станции попытается зарегистрироваться Труди, введенный ею пароль будет ошибочным. Рабочая станция это обнаружит, так как стандартная часть сообщения 2 окажется неверной.

После регистрации в сети Алиса может сообщить рабочей станции, что она хочет вступить в контакт с файловым сервером, то есть Бобом. Рабочая станция отсылает TGS сообщение 3 с просьбой выдать удостоверение для этого взаимодействия. Ключевым элементом запроса является удостоверение $K_{TGS}(A, K_S, t)$,

которое зашифровано секретным ключом TGS и используется для подтверждения личности отправителя. В ответном сообщении 4 TGS передает созданный им ключ сеанса $K_{AB}$, которым будут пользоваться Алиса и Боб. Он возвращает две версии этого ключа. Один ключ зашифрован ключом сеанса $K_S$, чтобы его могла прочитать Алиса. Второй ключ представляет собой еще одно удостоверение, зашифрованное ключом Боба $K_B$ (чтобы его мог прочитать Боб).

Труди может скопировать сообщение 3 и попытаться использовать его снова, но в этом ей помешает временная метка $t$, отправляемая вместе с этим сообщением. Труди не может заменить временную метку на более новую, так как не знает ключа сеанса $K_S$, которым пользуется Алиса для общения с TGS. Даже если Труди очень быстро воспроизведет сообщение 3, она все равно получит в ответ лишь сообщение 4, которое она не смогла расшифровать в первый раз и не сможет расшифровать во второй.

Теперь, с помощью нового удостоверения, Алиса может отправить Бобу ключ $K_{AB}$, чтобы установить с ним сеанс. Эти сообщения также содержат временные метки. Возможный ответ (сообщение 6) подтверждает, что Алиса говорит именно с Бобом, а не с Труди.

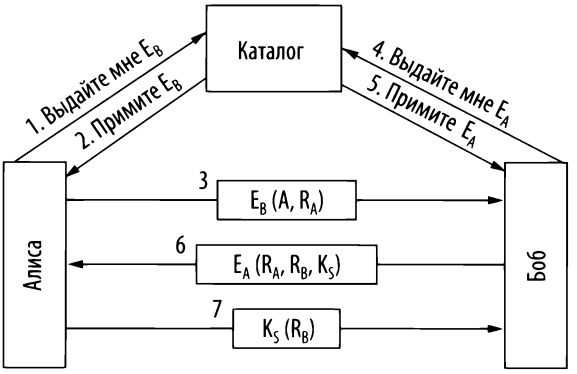
После этой серии обмена сообщениями Алиса сможет взаимодействовать с Бобом, используя ключ сеанса $K_{AB}$. Если после этого Алиса решит, что ей необходим другой сервер, например Кэрол ($C$, Carol), она может просто отправить TGS сообщение, аналогичное третьему, заменив в нем $B$ на $C$ (то есть идентификатор Боба на идентификатор Кэрол). В ответ TGS мгновенно пришлет ей удостоверение, зашифрованное ключом $K_C$. Алиса передаст его Кэрол, для которой оно будет служить гарантией подлинности Алисы.

Достоинство этого протокола в том, что теперь Алиса может получать защищенный доступ к любому серверу сети, и в то же время ее пароль никогда не передается. В действительности он лишь на несколько миллисекунд появляется на ее рабочей станции. Однако обратите внимание, что каждый сервер выполняет свою собственную процедуру авторизации. Когда Алиса предъявляет свое удостоверение Бобу, это всего лишь подтверждает, что она та, за кого себя выдает. При этом действия, которые Алиса может выполнять, определяет Боб.

Поскольку разработчики Kerberos не рассчитывали, что весь мир станет доверять единственному серверу аутентификации, они предусмотрели существование нескольких **областей (realms)**, каждая из которых имеет свой собственный AS и TGS. Чтобы получить удостоверение для сервера, расположенного в удаленной области, Алиса должна запросить его у своего TGS, чтобы оно было принято TGS нужной ей области. Если удаленный TGS зарегистрировался на локальном TGS (так же, как это делают локальные серверы), локальный TGS выдаст Алисе удостоверение, действительное на удаленном TGS. После этого она может получить у удаленного TGS удостоверения для серверов его области. Важно отметить, что для установления защищенного сеанса между двумя сторонами из разных областей необходимо, чтобы каждая из них доверяла TGS другой стороны. Иначе они просто не смогут работать вместе.

## 8.9.5. Аутентификация путем шифрования с открытым ключом

Взаимная аутентификация также может выполняться с помощью шифрования с открытым ключом. Сначала Алиса должна получить открытый ключ Боба. Если PKI реализована на основе сервера каталогов, выдающего сертификаты на открытые ключи, Алиса может запросить сертификат Боба (сообщение 1 на илл. 8.40). В ответе (сообщение 2) содержится сертификат X.509 с открытым ключом Боба. Проверив корректность подписи, Алиса может отправить Бобу сообщение со своим идентификатором и нонсом.




**Илл. 8.40.** Взаимная аутентификация с помощью открытого ключа

Когда Боб получает это сообщение, он не знает, от кого оно пришло — от Алисы или от Труди, — но делает вид, что все в порядке, и просит сервер каталогов выдать ему открытый ключ Алисы (сообщение 4). Вскоре он его получает (сообщение 5). Затем он отсылает Алисе сообщение 6, содержащее случайное число Алисы $R_A$, свой собственный нонс $R_B$ и предлагаемый ключ сеанса $K_S$.

Алиса расшифровывает полученное сообщение 6 своим закрытым ключом. Она видит в нем свое случайное число $R_A$ и очень этому рада: это подтверждает, что сообщение пришло от Боба, так как Труди не способна определить значение $R_A$. Кроме того, $R_A$ свидетельствует о свежести этого сообщения. Алиса соглашается на установление сеанса (сообщение 7). Когда Боб видит свое случайное число $R_B$, зашифрованное ключом сеанса (который он сформировал сам), он понимает, что Алиса получила сообщение 6 и проверила значение $R_A$. Боб счастлив.

Может ли Труди обойти этот протокол? Она может сфабриковать сообщение 3 и спровоцировать Боба на проверку Алисы, но Алиса увидит число $R_A$, которое она не отсылала, и не ответит. Труди не удастся убедительно подделать сообщение 7, так как она не знает содержимого запроса $R_B$ или ключа $K_S$, и не может определить их, поскольку не располагает закрытым ключом Алисы. Удача не на ее стороне.

## 8.10. ЗАЩИТА СОЕДИНЕНИЙ

Мы закончили изучение прикладных инструментов, рассмотрев большинство используемых методов и протоколов. В оставшейся части главы мы обсудим их практическое применение, а в завершение выскажем некоторые соображения по социальным вопросам безопасности.

Следующие разделы посвящены защите соединений. Мы узнаем, как передавать биты от отправителя получателю конфиденциально и без изменений, при этом не пропуская посторонние биты. Это далеко не все проблемы сетевой безопасности, но определенно одни из самых важных.

### 8.10.1. IPsec

IETF в течение многих лет мирился с отсутствием безопасности в интернете. Обеспечить ее было непросто из-за жарких споров о том, в какой части системы располагать средства защиты. Большинство экспертов по вопросам безопасности уверены в том, что по-настоящему надежная система должна выполнять сквозное шифрование и сквозное обеспечение целостности данных (то есть все это должно быть сделано на прикладном уровне). Другими словами, процесс-источник шифрует и/или ставит защиту целостности данных и отправляет их процессу-получателю, который, в свою очередь, дешифрует данные и/или проверяет их целостность. Это позволит заметить любые попытки взлома (даже на уровне операционной системы на любой из сторон). Проблема в том, что для обеспечения безопасности требуется вносить изменения во все приложения. Из этого следует, что лучше перенести шифрование на транспортный уровень или организовать новый специализированный подуровень между прикладным и транспортным. Он должен быть сквозным, но в то же время не требующим внесения изменений в приложения.

Противоположная точка зрения заключается в том, что пользователи не знают, как устроены средства безопасности, и просто не способны их корректно использовать. При этом никто не хочет менять существующие программы. Поэтому сетевой уровень должен выполнять проверку подлинности и/или шифровать сообщения незаметно для пользователя. Долгие годы дискуссий привели к победе такого подхода: был разработан стандарт безопасности, ориентированный на сетевой уровень. Одним из аргументов стало то, что шифрование на сетевом уровне не помешает тем, кто серьезно относится к безопасности, и при этом в какой-то мере убережет беспечных пользователей.

Результатом всех этих споров было создание стандарта **IPsec (IP security — IP-безопасность)**, описанного в ряде спецификаций RFC. Не всем пользователям требуется шифрование соединений (соответствующие процедуры могут занимать существенную долю вычислительных ресурсов). Однако вместо того чтобы делать шифрование необязательным, пользователю предлагается в случае необходимости выбирать нулевое шифрование. В RFC 2410 описаны такие достоинства нулевого шифрования, как простота, легкость реализации и высокая скорость.

Полноценный IPsec лежит в основе множества служб, алгоритмов и модулей. Существование большого количества отдельных служб объясняется тем, что люди не хотят постоянно платить сразу за все, поэтому нужные службы предоставляются на выбор. Например, пользователь, который просматривает в потоковом режиме фильм, размещенный на удаленном сервере, вероятно, может обойтись без шифрования (в отличие от владельца авторских прав на этот фильм). Самые главные службы обеспечивают конфиденциальность, целостность данных, защиту от взлома методом повторения сообщений (когда злоумышленник воспроизводит подслушанный разговор). Все они основаны на криптографии с симметричными ключами, поскольку здесь критична высокая производительность.

Для чего нужен целый набор алгоритмов? Дело в том, что алгоритм, который сегодня считается надежным, завтра может быть взломан. Если сделать IPsec независимым от алгоритмов, стандарт выживет даже в случае взлома одного из них. Ведь переключиться на использование другого алгоритма гораздо проще, чем целиком разрабатывать новую систему.

Разные модули нужны для того, чтобы иметь возможность защищать как одно TCP-соединение, так и весь трафик между парой хостов, а также весь трафик между парой защищенных маршрутизаторов и т. д.

Немного удивляет, что IPsec ориентирован на установление соединения, хотя расположен на уровне IP. На самом деле это не так странно, как кажется. Ведь безопасность можно обеспечить только созданием ключа и использованием его в течение какого-то времени. А это, по сути дела, разновидность соединения. К тому же все соединения нивелируют расходы на их установление за счет передачи большого количества пакетов. «Соединение» в контексте IPsec называется **сопоставлением безопасности (Security Association, SA)**. Это симплексное соединение между двумя конечными точками, с которым связан специальный идентификатор защиты. Если требуется передача защищенных данных в обоих направлениях, нужны две SA. Идентификаторы защиты содержатся в пакетах, следующих по этим надежным соединениям, и по прибытии используются для поиска ключей и другой важной информации.

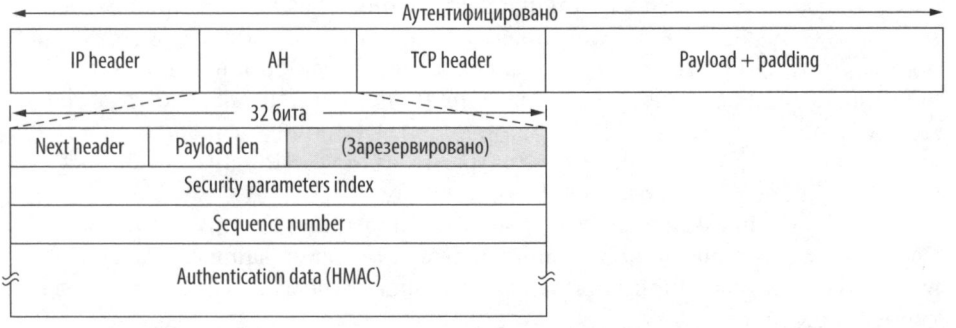
Технически IPsec состоит из двух основных частей. Первая часть описывает два новых заголовка, которые можно добавлять к пакету для передачи идентификатора защиты, данных контроля целостности и другой информации. Вторая, **ISAKMP (Internet Security and Key Management Protocol — интернет-безопасность и протокол управления ключами)**, предназначена для создания ключей и является средой. Основным протоколом для выполнения работы является **IKE (Internet Key Exchange — обмен ключами в интернете)**. Он прошел через множество модификаций, устранивших обнаруженные недостатки.

IPsec может работать в двух режимах. В **транспортном режиме (transport mode)** заголовок IPsec вставляется сразу после заголовка IP. Поле `Protocol` заголовка IP меняется так, чтобы было понятно, что далее следует заголовок IPsec (перед заголовком TCP). В заголовке IPsec содержится информация о безопасности, в частности идентификатор SA, новый порядковый номер и, возможно, проверка целостности пользовательских данных.

В **режиме туннелирования (tunnel mode)** весь IP-пакет вместе с заголовком вставляется внутрь нового IP-пакета с совершенно новым заголовком. Этот режим применяется, если туннель заканчивается не в пункте назначения. В некоторых случаях концом туннеля является шлюз безопасности, например корпоративный брандмауэр. Обычно это касается VPN. В этом режиме шлюз безопасности инкапсулирует и декапсулирует пакеты, проходящие через него. При такой организации компьютеры LAN компании не должны знать о стандарте IPsec. Об этом должен беспокоиться только шлюз безопасности.

Также режим туннелирования применяется, если несколько TCP-соединений объединяются вместе и обрабатываются в виде единого зашифрованного потока, поскольку в этом случае взломщик не может узнать, кто передает пакеты, в каком количестве и кому. А ведь иногда даже объем и назначение передаваемого трафика является ценной информацией. Например, если во время военного кризиса трафик между Пентагоном и Белым домом резко снижается, а трафик между Пентагоном и какой-нибудь военной базой в Колорадо так же резко возрастает, перехватчик может сделать из этого далеко идущие выводы. Изучение структуры потока пакетов (даже если они зашифрованы) называется **анализом трафика**. Режим туннелирования в некоторой степени усложняет такой анализ. Недостаток этого режима состоит в добавлении дополнительного IP-заголовка, что заметно увеличивает пакет. Транспортный режим, напротив, не так сильно влияет на размер пакетов.

Один из новых заголовков называется **заголовком аутентификации (Authentication Header, AH)**. С его помощью проверяется целостность данных и выполняется защита от взлома путем повторной передачи. Однако он не имеет никакого отношения к секретности (то есть к шифрованию данных). Применение AH в транспортном режиме показано на илл. 8.41. В стандарте IPv4 он располагается между заголовком IP (вместе со всеми необязательными полями) и заголовком TCP. В IPv6 он рассматривается просто как еще один заголовок расширения. Формат AH действительно очень близок к формату заголовка расширения стандарта IPv6. Иногда к полю `Payload` (Пользовательские данные) добавляют заполнение (padding), чтобы достичь определенной длины, необходимой алгоритму аутентификации (см. илл. 8.41).




**Илл. 8.41.** Заголовок аутентификации IPsec в транспортном режиме для IPv4

Рассмотрим заголовок AH. В поле `Next header` (Следующий заголовок) хранится значение, которое раньше находилось в поле `Protocol` (Протокол) заголовка IP (до того, как было заменено на 51, чтобы показать, что далее следует заголовок AH). Обычно здесь встречается код для TCP (6). Поле `Payload length` (Длина полезной нагрузки) хранит количество 32-разрядных слов заголовка AH минус 2.

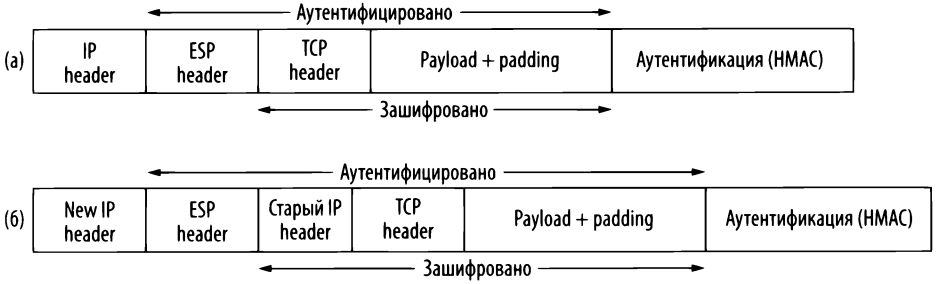
Поле `Security parameters index` (Указатель параметров безопасности) — это идентификатор соединения, который вставляется отправителем. Он ссылается на конкретную запись в базе данных получателя. В этой записи содержится общий ключ и другая информация о соединении. Если бы данный протокол был придуман МСЭ, а не IETF, это поле называлось бы `Virtual circuit number` (Номер виртуального канала).

Поле `Sequence number` (Порядковый номер) применяется для нумерации всех пакетов, отправляемых по SA. Все пакеты получают уникальные номера, даже если они отсылаются повторно. Другими словами, повторно передаваемый пакет имеет номер, отличный от номера оригинального пакета (даже если у них одинаковый порядковый номер TCP). Это поле служит для предотвращения взлома путем повторной передачи. Порядковые номера никогда не повторяются. Если же будут использованы все $2^{32}$ номера, для продолжения коммуникации устанавливается новая SA.

Наконец, поле переменной длины `Authentication data` (Данные аутентификации) содержит цифровую подпись пользовательских данных. При установлении SA стороны договариваются, какой алгоритм генерации подписей использовать. Обычно здесь не применяется шифрование с открытыми ключами, так как все известные алгоритмы этого типа слишком медленные, а пакеты нужно обрабатывать с очень большой скоростью. Протокол IPsec основан на шифровании с симметричными ключами, поэтому перед установлением SA отправитель и получатель должны договориться о значении общего ключа, применяемого при вычислении подписи. То есть IPsec использует код HMAC, который мало чем отличается от того кода, о котором мы говорили в разделе, посвященном аутентификации с использованием общего ключа. Вычисление этого кода выполняется гораздо быстрее, чем последовательный запуск SHA-2 и RSA.

Заголовок AH не позволяет шифровать данные, так что в основном он применяется в случаях, когда нужна проверка целостности, но не требуется секретность. Стоит отметить, что проверка целостности при помощи AH охватывает некоторые поля заголовка IP, которые не меняются при прохождении пакета между маршрутизаторами. К примеру, поле `TtL` (Время жизни) изменяется на каждом переходе, и проверка его не затрагивает. А вот IP-адрес источника проверяется, что делает невозможной его подмену.

Альтернативным заголовком IPsec является **ESP (Encapsulating Security Payload — безопасная инкапсуляция пользовательских данных)**. Как показано на илл. 8.42, он может применяться как в транспортном режиме, так и в режиме туннелирования.




**Илл. 8.42.** (а) ESP в транспортном режиме. (б) ESP в режиме туннелирования

Заголовок ESP состоит из двух 32-разрядных слов: `Security parameters index` (Указатель параметров безопасности) и `Sequence number` (Порядковый номер). Мы их уже встречали в заголовке AH. Третье слово, которое обычно следует за ними, при этом технически не являясь частью заголовка, это `Initialization vector` (Вектор инициализации); но если используется пустой алгоритм шифрования, это поле опускается.

ESP, как и AH, обеспечивает проверку целостности при помощи кода HMAC, однако вместо того чтобы включать хеш в заголовок, он вставляется после поля пользовательских данных (см. илл. 8.24). Такое расположение полей дает преимущество при аппаратной реализации метода: HMAC может подсчитываться во время передачи битов пользовательских данных по сети и добавляться в конце. Именно поэтому в Ethernet и других стандартах LAN CRC вставляется в трейлер, а не в заголовок. При использовании заголовка AH пакет нужно буферизовать и вычислять подпись, только после этого его можно отправлять. Это потенциально снижает число пакетов, которые можно передать за единицу времени.

Казалось бы, если ESP умеет делать все то же, что и AH (даже больше и при этом гораздо эффективнее), зачем вообще нужен AH? Причина скорее историческая. Изначально заголовок AH обеспечивал только проверку целостности, а ESP — только секретность. Позднее ESP стали использовать для проверки целостности, но разработчики AH не хотели, чтобы он канул в Лету после всей проделанной ими работы. Единственный (и довольно слабый) аргумент в пользу AH заключается в том, что с его помощью можно частично проверять заголовок IP, чего не умеет ESP. Еще один сомнительный довод состоит в том, что система, поддерживающая AH, но не поддерживающая ESP, может легче получить лицензию на экспорт, поскольку с помощью AH нельзя шифровать данные. Похоже, что этот заголовок все-таки исчезнет.

## 8.10.2. Виртуальные частные сети

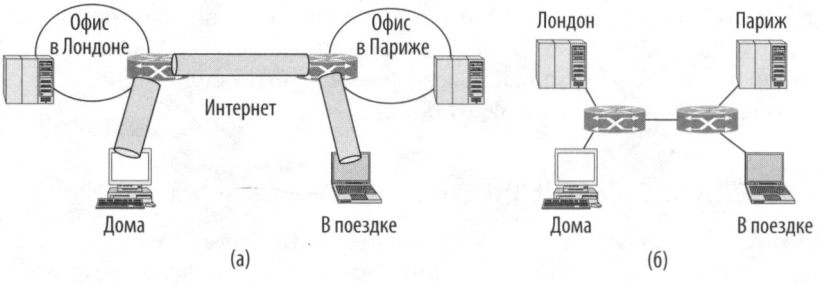
У многих компаний имеются филиалы, расположенные в разных городах или даже в разных странах. До появления сетей общего доступа обычным делом

было арендовать выделенную телефонную линию для организации связи между некоторыми (или всеми) парами подразделений. В некоторых компаниях такой подход применяется до сих пор. Сеть, состоящая из принадлежащих компании компьютеров и выделенных телефонных линий, называется **частной сетью (private network)**.

Частные сети работают хорошо и обладают высокой степенью защищенности. Если бы были доступны только выделенные линии, в подразделениях компании не возникали бы утечки трафика. Злоумышленникам пришлось бы физически подключаться к линиям, чтобы перехватить данные, а это не так просто. Проблема в том, что аренда выделенных каналов между двумя точками обходится очень дорого. Когда появились сети общего доступа, а позднее и интернет, у многих компаний возникло естественное желание использовать их для передачи данных (а может, и голоса). Правда, при этом не хотелось жертвовать безопасностью, свойственной частной сети.

В качестве ответа на этот запрос появились **виртуальные частные сети (Virtual Private Networks, VPN)**. Это оверлейные сети, которые работают поверх обычных общедоступных сетей, но обладают свойствами частных. Они называются виртуальными, поскольку это не более чем иллюзия, так же как виртуальные каналы — это не реальные каналы, а виртуальная память — не реальная память.

Часто VPN развертывают напрямую в интернете. Как правило, в каждом офисе устанавливается брандмауэр и создаются туннели через интернет между всеми парами офисов (илл. 8.43 (а)). Интернет удобен тем, что туннели можно устанавливать по требованию и, к примеру, подключать компьютер сотрудника, который находится дома или путешествует (при условии, что он имеет соединение с интернетом). Такая топология обеспечивает более высокую гибкость по сравнению с реальными выделенными линиями, но с точки зрения компьютеров внутри VPN она выглядит точно так же, как частная сеть (илл. 8.43 (б)). При запуске системы каждая пара брандмауэров должна договориться о параметрах SA, таких как набор услуг, режимов, алгоритмов и ключей. Если используется IPsec в режиме туннелирования, можно собрать весь трафик между любыми двумя парами офисов в один надежный поток и установить SA, обеспечив тем самым контроль целостности, секретности и даже определенную устойчивость к анализу трафика. Возможности VPN встроены во многие брандмауэры.




**Илл. 8.43.** (а) VPN. (б) Топология, видимая изнутри сети

Развернуть такую сеть можно и на некоторых обычных маршрутизаторах, но поскольку брандмауэры — это основа сетевой безопасности, вполне естественно начинать и заканчивать туннели именно на них, проводя четкую границу между компанией и интернетом.

Таким образом, естественная и наиболее распространенная комбинация — это брандмауэры, VPN и IPsec с ESP в режиме туннелирования.

После установления SA начинается передача данных. С точки зрения маршрутизатора, работающего в интернете, пакет, проходящий по туннелю VPN, — самый обычный пакет. Единственное, что его отличает от остальных, — это наличие заголовка IPsec после заголовка IP. Но поскольку дополнительные заголовки на процесс пересылки никак не влияют, заголовок IPsec маршрутизатору безразличен.

Другой подход, набирающий популярность, — реализация VPN с помощью интернет-провайдера. За счет использования MPLS (как обсуждалось в главе 5) пути для трафика VPN между офисами компании могут быть установлены через сеть интернет-провайдера. Эти пути отделяют трафик VPN от другого интернет-трафика и могут гарантировать определенную пропускную способность или другой уровень QoS.

Основное преимущество VPN состоит в том, что она совершенно прозрачна для любого пользовательского ПО. Установкой и управлением SA занимается брандмауэр. Единственный человек, знающий об устройстве сети, — системный администратор, который должен конфигурировать и поддерживать шлюзы безопасности (или администратор интернет-провайдера, настраивающий пути MPLS). Для всех остальных VPN мало чем отличается от частной сети на основе выделенной линии. Более подробную информацию об этих сетях вы можете найти в работе Ашрафа (Ashraf, 2018).

## 8.10.3. Безопасность в беспроводных сетях

Как ни парадоксально, с помощью VPN и брандмауэров очень просто создать систему, которая по логике абсолютно надежна, но на практике протекает, как решето. Такая ситуация может возникнуть, если в сети есть беспроводные устройства, передающие данные с помощью радиосигнала, который проходит прямо через брандмауэр в обоих направлениях. Радиус действия сетей 802.11 может составлять до 100 м, поэтому для перехвата информации шпион может просто приехать на автостоянку перед зданием фирмы, оставить в машине ноутбук с приемопередатчиком 802.11, записывающим все, что слышно в эфире, и пойти гулять по городу. Вернувшись под вечер, он обнаружит на диске ноутбука массу интересных сведений. Теоретически так быть не должно. Правда, теоретически ограбления банков тоже не должны происходить.

За многие проблемы безопасности стоит «поблагодарить» производителей беспроводных базовых станций (точек доступа), которые пытаются сделать свою продукцию удобной для пользователя. Обычно, как только пользователь вынимает устройство из коробки и подключает его к розетке, оно сразу начинает работать. И почти всегда без каких-либо мер безопасности — люди в зоне

действия радиопередатчика могут услышать секреты, о которых пользователь проболтается. Если же устройство подключить к Ethernet, весь трафик может оказаться на ноутбуке в припаркованной неподалеку машине. Беспроводная связь — это мечта шпиона, ставшая реальностью: информация сама идет в руки, только успевай ее ловить. Очевидно, что вопрос безопасности в таких сетях стоит куда острее, чем в проводных. В этом разделе мы рассмотрим некоторые методы, позволяющие обезопасить системы такого рода, уделив особое внимание Wi-Fi. Дополнительную информацию можно найти в книге Остерхаге (Osterhage, 2018).

Часть стандарта 802.11, изначально названная **802.11i**, описывает протокол безопасности канального уровня, не позволяющий беспроводному узлу читать или другим образом вмешиваться в сообщения, отправленные другой парой беспроводных узлов. Этот протокол также известен под коммерческим названием **WPA2 (Wi-Fi Protected Access 2 — защищенный доступ к Wi-Fi, версия 2)**. Простой WPA — это промежуточная схема, реализующая подмножество стандарта 802.11i. На смену ему пришел WPA2. В январе 2018 года была анонсирована следующая после WPA2 версия с оригинальным названием **WPA3**. Она использует 128-битное шифрование для личного пользования и 192-битное — для корпоративной версии. Протокол WPA3 отличается целым рядом улучшений по сравнению с WPA2. Вероятно, главным из них является переработанный механизм «рукопожатия» под названием «Dragonfly». Он предотвращает некоторые виды атак подбора пароля, которые создавали массу проблем в протоколе WPA2. На момент написания книги WPA3 применяется еще не так широко, как WPA2. Кроме того, в апреле 2019 года исследователи выявили вектор атаки, получивший название «DragonBlood», который сводит на нет многие из преимуществ WPA3 по обеспечению безопасности. В силу этого основное внимание в данном разделе будет уделено WPA2.

Мы кратко опишем протокол 802.11i, но сначала отметим, что он заменяет **WEP (Wired Equivalent Privacy — конфиденциальность на уровне проводных сетей)**, первое поколение протоколов безопасности 802.11. Протокол WEP был создан комитетом по сетевым стандартам. Его подход к разработке кардинально отличался от стратегии, например, института NIST, который выбрал дизайн алгоритма AES на открытом международном конкурсе. Это привело к удручающим результатам. Что же было не так? Как оказалось, с точки зрения безопасности практически все. Например, WEP зашифровывал конфиденциальные данные с помощью XOR с выходом поточного шифра. К сожалению, слабые механизмы ключей приводили к использованию данных несколько раз, поэтому их было просто расшифровать. В качестве еще одного примера можно привести проверку целостности, основанную на 32-битном CRC. Этот код эффективен для определения ошибок передачи, но он не является криптографически сильным механизмом борьбы со взломщиками.

Эти и другие недостатки сделали WEP легкой мишенью. Первая практическая демонстрация взлома WEP была проведена Адамом Стабблфилдом (Adam Stubblefield), когда он стажировался в компании AT&T (Stubblefield и др., 2002). Он смог реализовать и проверить атаку, описанную Флюрером и др. (Fluhrer et al., 2001), за одну неделю, потратив большую часть времени на

убеждение менеджеров предоставить ему Wi-Fi-карту для эксперимента. Программы, взламывающие пароли WEP за одну минуту, теперь находятся в свободном доступе, поэтому использовать WEP не рекомендуется. Он запрещает открытый доступ, но не обеспечивает никакой реальной защиты. Когда стало ясно, что WEP взломан, группа 802.11i поспешно собралась для решения этой проблемы. В результате в июне 2004 года был выпущен формальный стандарт.

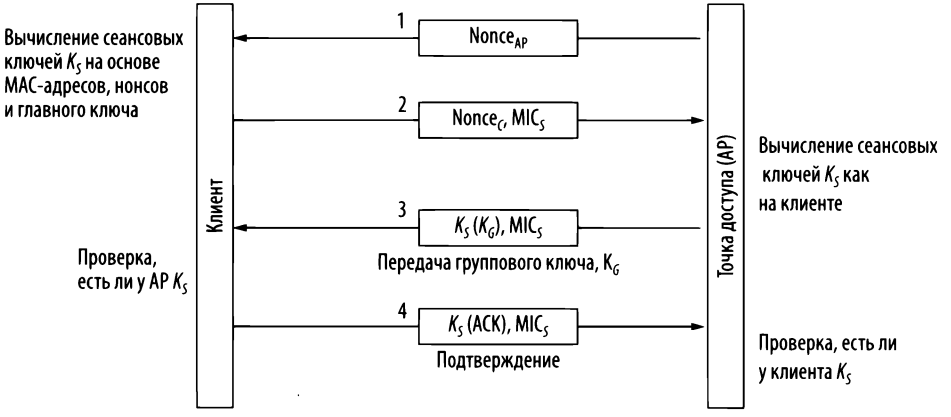
Теперь мы опишем 802.11i, который обеспечивает реальную безопасность, если он правильно настроен и применяется корректно. Существует два стандартных сценария, в которых задействован WPA2. Первый — это корпоративное использование, когда у компании есть отдельный сервер для аутентификации, хранящий имена пользователей и пароли. С помощью этих данных система определяет, может ли беспроводной клиент получить доступ к сети. В этом случае клиенты используют стандартные протоколы для того, чтобы аутентифицировать себя и войти в сеть. Основные стандарты — это **802.1X**, где точка доступа позволяет клиенту вести диалог с сервером аутентификации и наблюдать результат, а также **расширенный протокол аутентификации (Extendable Authentication Protocol, EAP)**. EAP (RFC 3748) описывает взаимодействие клиента и сервера аутентификации. По сути, он является средой, а другие стандарты определяют сообщения протокола. Мы не будем подробно изучать эти сообщения, для краткого обзора это не имеет значения.

Второй сценарий — это типичное домашнее использование без сервера аутентификации, но с единым общим паролем, с помощью которого клиенты получают доступ в беспроводную сеть. Это система не такая сложная, как в случае с сервером аутентификации, именно поэтому она используется в домашних условиях и в маленьких фирмах. Однако при этом она менее надежная. Основная разница в том, что при наличии сервера аутентификации каждый клиент получает ключ для шифрования трафика, неизвестный другим клиентам. При едином общем пароле для каждого клиента создается свой ключ, но у всех клиентов одинаковый пароль, и они могут узнать ключи друг друга, если захотят.

Ключи для шифрования трафика вычисляются как часть аутентификационного «рукопожатия». Эта процедура происходит сразу после того, как клиент связывается с беспроводной сетью и подтверждает подлинность на сервере аутентификации (если есть такой сервер). В начале «рукопожатия» у клиента есть либо общий пароль сети, либо пароль для сервера аутентификации. Пароль нужен для получения главного ключа. Однако этот ключ не используется напрямую для шифрования пакетов. Существует стандартная криптографическая практика: новый ключ сеанса создается для каждого периода использования и меняется для разных сеансов, а главный ключ держится в секрете. Во время «рукопожатия» вычисляется именно ключ сеанса.

Ключ сеанса рассчитывается четырехпакетным «рукопожатием» (илл. 8.44). Прежде всего AP (Access Point — точка доступа) отправляет случайное число для идентификации. Клиент также выбирает свой собственный нонс. Чтобы вычислить ключ сеанса $K_s$, клиент использует нонсы, свой MAC- и AP-адрес, а также главный ключ. Ключ сеанса разбивается на части, каждая из которых применяется для различных целей (мы опустим эти детали). Теперь у клиента есть

ключи сеанса, а у AP нет. Клиент отправляет свой нонс AP, и AP производит тот же самый расчет, чтобы получить те же ключи сеанса. Нонсы могут передаваться открыто, так как вычислить ключи на их основе невозможно без дополнительной, секретной информации. Сообщение клиента защищено **проверкой целостности сообщения (Message Integrity Check, MIC)**, которая проводится на основе ключа сеанса. После вычисления ключей сеанса AP может установить, что MIC верна и сообщение действительно пришло от клиента. MIC — это просто еще одно название кода аутентификации сообщения, подобного HMAC. Название «MIC» часто используется вместо «HMAC» в случае протоколов безопасности, чтобы не возникало путаницы с MAC-адресами.




**Илл. 8.44.** Генерация сеансового ключа «рукопожатием» в 802.11i

В последних двух сообщениях AP выдает клиенту общий ключ $K_G$, и клиент подтверждает его получение. Это позволяет AP удостовериться, что у клиента есть верные ключи сеанса, а клиенту — что они есть у AP. Общий ключ используется для передачи трафика по 802.11 LAN. Поскольку в результате «рукопожатия» оказывается, что у каждого клиента есть свои ключи шифрования, ни один из этих ключей не может быть использован AP для передачи пакетов всем клиентам беспроводной сети. Можно было бы отправить отдельную копию со своим ключом каждому клиенту. Вместо этого используется общий ключ, так что широковещательный трафик может быть передан один раз и получен всеми клиентами. Этот ключ должен обновляться по мере того, как клиенты уходят из сети и присоединяются к ней.

Наконец, мы подходим к той части, где ключи действительно применяются для обеспечения безопасности. Чтобы добиться конфиденциальности, целостности и аутентификации, в 802.11i могут использоваться два протокола. Один из них, **протокол целостности временного ключа (Temporary Key Integrity Protocol, TKIP)** был временным решением (как и WPA). Он был разработан для повышения безопасности старых и медленных карт 802.11, это была хоть какая-то защита (лучше, чем WEP), доступная после обновления прошивки.

Однако TKIP тоже был взломан, поэтому сегодня рекомендуется использовать протокол **CCMP**. Эта аббревиатура означает «Counter mode with Cipher block chaining message authentification code protocol» — «режим счетчика с протоколом аутентификации в режиме сцепления обратной связи». Мы будем использовать сокращение CCMP, а вы называйте его как угодно.

Принцип действия CCMP достаточно прост. Он использует шифрование AES с помощью ключа и блоков размером 128 бит. Ключ выводится из ключа сеанса. Чтобы обеспечить конфиденциальность, сообщения зашифровываются с помощью AES в режиме счетчика. Режимы шифров (см. раздел 8.2.3) предотвращают шифрование одних и тех же сообщений в одинаковые наборы битов. Режим счетчика внедряет счетчик в процесс шифрования сообщения. Чтобы обеспечить целостность, сообщение (включая поля заголовков) кодируется шифром в режиме обратной связи, и последний блок из 128 бит сохраняется как MIC. Затем и сообщение (закодированное в режиме счетчика), и MIC отсылаются. Как клиент, так и AP могут выполнять это шифрование или проверять его при получении беспроводного пакета. В случае многоадресных или широковещательных сообщений применяется групповой ключ.

## 8.11. БЕЗОПАСНОСТЬ ЭЛЕКТРОННОЙ ПОЧТЫ

Во время переписки двух удаленных пользователей сообщения обычно проходят на своем пути через десяток других компьютеров. Любой из компьютеров может читать и записывать проходящую через него почту. Вопреки распространенному мнению, конфиденциальности не существует. Тем не менее многие пользователи хотели бы отправлять электронные письма так, чтобы их мог прочитать только непосредственный адресат и никто другой: ни начальство, ни даже правительство. Это заставило некоторых разработчиков (как отдельных представителей сообщества, так и группы) применить к электронной почте криптографические принципы, изученные нами ранее. В следующих разделах мы обсудим популярную систему защиты электронной почты PGP, а также дадим общее представление о системе S/MIME.

### 8.11.1. PGP

Система **PGP (Pretty Good Privacy — «неплохая конфиденциальность»)** была создана всего одним человеком, Филом Циммерманом (Phil Zimmermann, 1995), активным сторонником конфиденциальности в интернете и автором фразы «Если конфиденциальность объявить вне закона, она будет доступна лишь преступникам». Выпущенная в 1991 году система PGP представляет собой полный пакет безопасности электронной почты, обеспечивающий конфиденциальность, аутентификацию, цифровые подписи и сжатие. Этот пакет удобен в использовании, содержит все исходные тексты программ и свободно распространяется в интернете. На сегодняшний день PGP широко используется благодаря своему качеству, стоимости (нулевой) и доступности на различных платформах (включая UNIX, Linux, Windows и Mac OS).

Изначально PGP кодировала данные с помощью блочного шифра **IDEA (International Data Encryption Algorithm — международный алгоритм шифрования данных)**, использующего ключи длиной 128 бит. Он был изобретен в Швейцарии в те времена, когда DES уже считался устаревшим, а AES еще не был придуман. Концептуально IDEA похож на DES и AES: в нем производится перемешивание битов в последовательности циклов, однако детали реализации функций отличаются от этих алгоритмов. Позже в качестве алгоритма шифрования был предложен AES, и теперь эта схема является общепринятой.

С первого же дня своего существования система PGP столкнулась с серьезными проблемами (Леви; Levy, 1993). Поскольку Циммерман никак не препятствовал распространению PGP в интернете, правительство США обвинило его в нарушении закона о запрете экспорта военного имущества. Следствие по этому делу длилось пять лет, но в один прекрасный день прекратилось — вероятно, по двум причинам. Во-первых, Циммерман не выкладывал PGP в сеть собственноручно. Его адвокат аргументировал позицию защиты тем, что обвиняемый никогда не занимался экспортом чего бы то ни было *сам* (а то, что создание сайта равносильно экспорту, еще нужно доказать). Во-вторых, со временем власти поняли следующее: для победы в суде пришлось бы убедить присяжных в том, что любой сайт, содержащий доступную для скачивания программу, связанную с конфиденциальностью, подпадает под действие закона о контрабанде оружия (то есть танков, подводных лодок, военных самолетов и ядерных боеголовок). Годы негативного освещения в СМИ также не пошли делу на пользу.

Вообще, существующие правила экспорта, мягко говоря, странные. Правительство решило, что размещение программы на веб-странице можно приравнять к нелегальному экспорту, и преследовало Циммермана по этому поводу целых пять лет. Однако если кто-то опубликует книгу с полным исходным кодом PGP на языке C (крупным шрифтом, да еще и с контрольной суммой на каждой странице для облегчения сканирования) и займется ее экспортом, государство и глазом не моргнет: книги по закону не являются военным имуществом. Вопреки известной пословице[1], в США порой сильнее меч, а не перо.

Еще одна проблема, с которой внезапно столкнулась система PGP, была связана с нарушением патентных прав. Корпорация RSA Security (владелец патента на RSA) заявила, что использование алгоритма RSA в PGP является посягательством на патент. Эта проблема разрешилась в последующих версиях, начиная с 2.6. Кроме того, в PGP использовался другой запатентованный алгоритм, IDEA, что поначалу тоже вызывало некоторые вопросы.

Так как PGP — это бесплатная система с открытым исходным кодом, различные группы и отдельные разработчики выпустили множество ее модификаций. Одни пытались обойти законы об экспорте оружия, другие — избежать применения запатентованных алгоритмов, а третьи работали над превращением PGP в коммерческий продукт с закрытым исходным кодом. Позже законы об экспорте

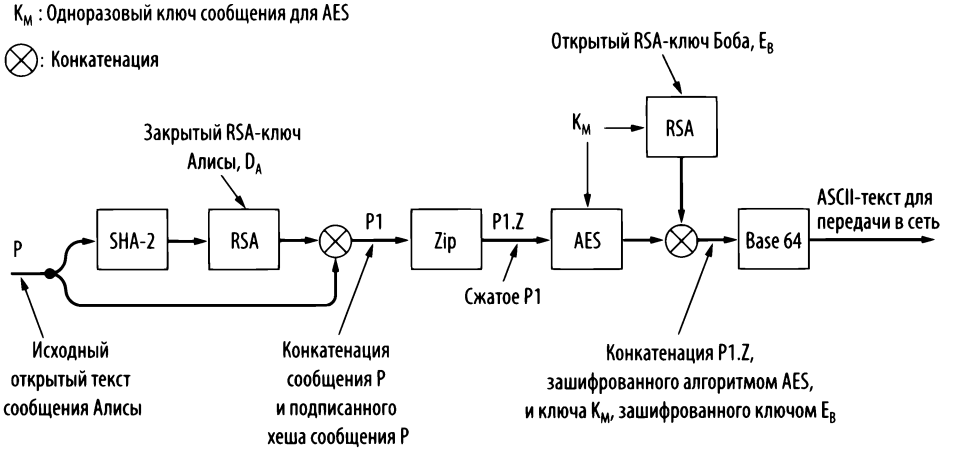
[1] Речь идет о фразе «The pen is mightier than the sword» («Перо сильнее меча») из пьесы английского писателя-романиста Эдуарда Бульвера-Литтона. — *Примеч. ред.*

оружия несколько смягчились (тем не менее продукцию, использующую AES, до сих пор нельзя поставлять за пределы США), а срок действия патента RSA закончился в сентябре 2000 года. Однако следствием всех этих проблем стало возникновение и распространение нескольких несовместимых версий PGP под разными названиями. Ниже мы рассмотрим классический вариант PGP (самый старый и самый простой), но используем в нашем объяснении AES и SHA-2 вместо IDEA и MD5. Еще одна популярная версия, Open PGP, описана в RFC 2440. Также можно отметить GNU Privacy Guard.

В системе PGP намеренно используются уже существующие криптографические алгоритмы, а не изобретаются новые. Все они прошли тщательную проверку ведущими криптоаналитиками мира и избежали влияния каких-либо государственных организаций, пытающихся снизить их эффективность. Последнее свойство имеет большое значение для всех, кто не склонен доверять правительству.

Система PGP обеспечивает сжатие текста, конфиденциальность и создание цифровых подписей, а также предоставляет широкие возможности управления ключами. При этом, как ни странно, в ней отсутствует функционал электронной почты. PGP напоминает препроцессор, который преобразует открытый текст в подписанный зашифрованный текст в формате base64. Разумеется, результат можно выслать по электронной почте. Некоторые реализации PGP на последнем шаге обращаются к пользовательскому агенту для отправки сообщения.

Чтобы понять, как работает система PGP, рассмотрим пример на илл. 8.45. Алиса хочет передать Бобу подписанное сообщение *P* открытым текстом в безопасном режиме. Система PGP поддерживает различные схемы шифрования (включая RSA и шифрование на основе эллиптических кривых), но в данном случае мы предположим, что у Алисы и Боба есть закрытый ($D_X$) и открытый ($E_X$) RSA-ключи. Также мы допустим, что им известны открытые ключи друг друга. Управление ключами в системе PGP мы рассмотрим чуть позже.




**Илл. 8.45.** Использование системы PGP для передачи сообщения

Прежде всего Алиса запускает на компьютере программу PGP. PGP хеширует сообщение Алисы $P$ с помощью алгоритма SHA-2, а затем шифрует полученный хеш ее закрытым RSA-ключом $D_A$. Получив это сообщение, Боб может расшифровать хеш открытым ключом Алисы и убедиться в его правильности. Даже если в этот момент кто-то другой (например, Труди) узнает хеш и расшифрует его общеизвестным открытым ключом Алисы, мощность алгоритма SHA-2 гарантирует невозможность создания другого сообщения с тем же хешем путем вычислений.

Зашифрованный хеш-код и оригинальное сообщение объединяются в единое сообщение *P1*, которое затем сжимается с помощью программы ZIP, использующей алгоритм Лемпеля — Зива (Ziv and Lempel, 1977). Назовем результат этого этапа *P1.Z*.

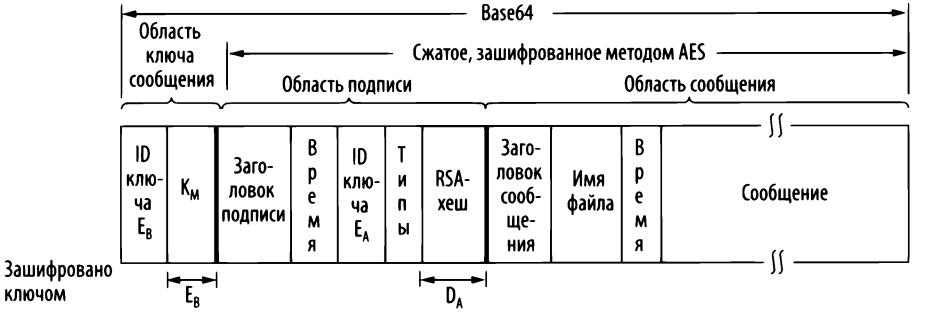
Далее PGP предлагает Алисе ввести случайную текстовую строку. При формировании 256-разрядного AES-ключа сообщения $K_M$ учитывается как содержание, так и скорость ввода. (В литературе по PGP этот ключ называют сеансовым, что является неправильным употреблением термина, так как никакого сеанса здесь нет.) Затем *P1.Z* шифруется алгоритмом AES с помощью ключа $K_M$, который, в свою очередь, шифруется открытым ключом Боба $E_B$. Два этих компонента объединяются и преобразуются в кодировку base64, о которой мы говорили в главе 7 при обсуждении стандартов MIME. Полученное в результате сообщение содержит только буквы, цифры и символы +, / и =. Таким образом, его можно поместить в тело письма стандарта RFC 822 с расчетом на то, что оно прибудет к адресату без изменений.

Приняв сообщение, Боб выполняет обратное преобразование base64 и расшифровывает AES-ключ своим закрытым RSA-ключом. Затем с помощью AES-ключа он декодирует сообщение и получает *P1.Z*. Распаковав zip-файл, Боб отделяет зашифрованный хеш-код от открытого текста и расшифровывает его открытым ключом Алисы. Если в результате обработки открытого текста алгоритмом SHA-2 получается тот же самый хеш-код, это означает, что сообщение $P$ действительно пришло от Алисы.

Надо отметить, что RSA используется здесь только в двух моментах: для зашифровки 256-разрядного SHA-2-хеша и 256-разрядного AES-ключа. Алгоритм RSA медленный, но в данном случае он должен зашифровать совсем немного данных. Более того, все эти 512 бит в высшей степени случайны, поэтому Труди придется очень сильно попотеть, чтобы угадать ключ. Основное шифрование выполняется алгоритмом AES — он на порядок быстрее RSA. Итак, PGP обеспечивает конфиденциальность, сжатие и цифровую подпись и делает это куда эффективнее, чем схема на илл. 8.22.

Система PGP поддерживает RSA-ключи разной длины. Подходящую длину можно выбрать самостоятельно. Скажем, если вы обычный пользователь, вам должно хватить ключа длиной 1024 бита. Чтобы защититься от современных спецслужб, лучше выбрать ключ минимум на 2048 бит. Ну а если вы боитесь, что ваши электронные письма прочитают инопланетяне, на 100 000 лет опережающие нас в технологиях, то вы всегда можете использовать ключи длиной 4096 бит. С другой стороны, учитывая, что RSA применяется только для шифрования небольшого количества битов, вероятно, стоит всегда выбирать «межпланетный» вариант.

Формат классического PGP-сообщения показан на илл. 8.46. Помимо него используются и многие другие форматы. Сообщение состоит из трех частей: области IDEA-ключа, области подписи и области сообщения. Первая часть, помимо самого ключа, содержит идентификатор (ID) открытого ключа, так как пользователям разрешено иметь несколько открытых ключей.




**Илл. 8.46.** PGP-сообщение

Область подписи содержит заголовок, который нас в данный момент не интересует. За ним следует временная метка и идентификатор открытого ключа отправителя, с помощью которого получатель сможет расшифровать хеш-код, используемый в качестве подписи. Следом идет идентификатор задействованных алгоритмов шифрования и хеширования (чтобы можно было пользоваться SHA-4 или RSA2, когда они появятся). Последним в области подписи располагается сам зашифрованный хеш-код.

Область сообщения также содержит заголовок, имя файла по умолчанию (на случай, если получатель сохранит файл на диске), время создания сообщения и, наконец, само сообщение.

Работе с ключами в системе PGP было уделено особое внимание, так как это ахиллесова пята всех систем защиты. Управление ключами осуществляется следующим образом. Локально у каждого пользователя есть две структуры данных: набор закрытых ключей и набор открытых ключей (иногда их называют «связками»). **Набор закрытых ключей (private key ring)** содержит одну или несколько индивидуальных пар ключей (закрытый/открытый). Несколько пар ключей нужны для того, чтобы пользователи могли их менять (периодически или при опасении, что тот или иной ключ скомпрометирован). Для этого не нужно аннулировать готовые к передаче или уже отправленные сообщения. У каждой пары ключей есть свой идентификатор, так что отправитель может сказать получателю, какой открытый ключ был использован для шифрования. Идентификатор сообщения состоит из 64 младших битов открытого ключа. За отсутствие конфликтов в идентификаторах открытых ключей отвечают сами пользователи. Закрытые ключи на диске зашифрованы специальным паролем произвольной длины, защищающим их от скрытых атак.

**Набор открытых ключей (public key ring)** содержит открытые ключи собеседников пользователя. Они нужны для шифрования ключей сообщений (последние привязаны к каждому сообщению). Все записи набора открытых ключей включают не только ключ, но и его 64-разрядный идентификатор, а также индикатор степени доверия пользователя этому ключу.

Степень доверия ключу зависит, например, от способа его получения. Предположим, открытые ключи хранятся на веб-сайте. Труди может атаковать этот сайт и подменить размещенный там открытый ключ Боба своим ключом. Если Алиса попытается воспользоваться фальшивым ключом, Труди организует атаку посредника на Боба.

Чтобы предотвратить такие атаки или хотя бы минимизировать их ущерб, Алисе нужно знать, насколько она может доверять «ключу Боба» в ее наборе открытых ключей. Если Боб лично передал ей ключ на компакт-диске (или более современном носителе), ее доверие к этому ключу максимально. В этом и состоит децентрализованный, контролируемый пользователем подход к управлению открытыми ключами, который отличает PGP от централизованной схемы PKI.

Впрочем, в некоторых случаях открытые ключи получают путем запроса к доверенному серверу ключей. Поэтому после стандартизации X.509 система PGP поддерживает эти сертификаты наряду с традиционным для PGP механизмом набора открытых ключей. Все современные версии PGP работают с X.509.

## 8.11.2. S/MIME

Смелым проектом IETF по обеспечению конфиденциальности электронной почты стала система под названием **S/MIME (Secure/MIME — защищенный MIME)**, описанная в спецификациях RFC 2632–2643. S/MIME обеспечивает аутентификацию, целостность данных, конфиденциальность и проверку подлинности информации. Это довольно гибкая система, поддерживающая множество криптографических алгоритмов. По названию можно догадаться, что S/MIME хорошо сочетается с MIME и позволяет защищать любые типы сообщений. В результате появился целый ряд новых заголовков MIME, например, для цифровых подписей.

В S/MIME нет жесткой иерархии сертификатов и отсутствует единый центр управления, что составляло проблему для более ранней системы **почты с повышенной секретностью (Privacy Enhanced Mail, PEM)**. Вместо этого пользователи могут работать с набором якорей доверия. До тех пор пока сертификат можно отследить до какого-нибудь якоря доверия, он считается корректным. Система S/MIME использует стандартные алгоритмы и протоколы, которые мы уже рассматривали, поэтому на этом мы закончим ее обсуждение. Более подробную информацию вы найдете в RFC.

# 8.12. ВЕБ-БЕЗОПАСНОСТЬ

Мы только что изучили две важные области, в которых требуется защита информации, — соединения и электронная почта. Можно сказать, что это были

аперитив и первое блюдо. Теперь самое время перейти к основному вопросу: безопасности данных во Всемирной паутине. Именно здесь большинство злоумышленников проворачивает свои темные дела. В следующих разделах мы рассмотрим некоторые проблемы, касающиеся веб-безопасности.

Эту тему можно условно разделить на три части. Первая связана с безопасным именованием объектов и ресурсов, вторая — с установлением аутентифицированных соединений, третья — с тем, что происходит, когда сайт отправляет клиенту исполняемый код. Мы обсудим все эти вопросы после ознакомления с рядом возможных угроз.

## 8.12.1. Угрозы

Практически каждую неделю газеты публикуют статьи о проблемах безопасности во Всемирной паутине. Ситуация действительно довольно неприятная. Давайте рассмотрим несколько реальных случаев взлома. Прежде всего, домашние страницы многочисленных организаций самых разных масштабов подвергались атакам взломщиков и заменялись подложными страницами. (В прессе людей, которые атакуют чужие компьютеры, называют хакерами, однако в профессиональном сообществе это слово скорее используется по отношению к великим программистам. Мы предпочитаем термин **«взломщик» (cracker)**.) К числу сайтов, которые однажды подверглись успешной атаке, относятся веб-страницы таких серьезных организаций, как Yahoo!, Вооруженные силы США, Американское бюро кредитной истории Equifax, ЦРУ, НАСА, а также газета New York Times. В большинстве случаев взломщики просто заменяли оригиналы на страницы с каким-нибудь забавным текстом, и уже через несколько часов сайты удавалось восстановить.

Конечно, происходили и гораздо более серьезные атаки. Многие сайты были выведены из строя за счет искусственно созданной чрезмерной нагрузки (DoS-атака), с которой сервер заведомо не может справиться. Зачастую такие нападения совершались сразу с большого количества компьютеров, которые злоумышленник уже взломал (DDoS-атака). Такие атаки настолько распространены, что уже перестали быть новостью. Тем не менее ущерб от них исчисляется миллионами долларов.

В 1999 году шведский взломщик проник на сайт Hotmail (корпорации Microsoft) и создал зеркало, на котором все желающие могли ввести имя любого пользователя этого сайта и прочесть всю его почту, включая архивы.

Русский 19-летний взломщик по имени Максим украл с сайта интернет-магазина номера 300 000 кредитных карт. Затем он обратился к их владельцам и заявил, что если они не заплатят ему $100 000, он опубликует номера кредиток в интернете. Они не поддались на шантаж, и тогда Максим исполнил свою угрозу, что нанесло серьезный ущерб многим невинным жертвам.

Двадцатитрехлетний студент из Калифорнии отправил по электронной почте в агентство новостей фальшивый пресс-релиз, в котором сообщалось об огромных убытках корпорации Emulex и об уходе в отставку ее генерального директора. Спустя несколько часов акции компании упали на 60 %,

в результате чего их держатели лишились более $2 млрд. Преступник заработал около четверти миллиона долларов, продав акции без покрытия[1] незадолго до своего ложного заявления. Хотя этот случай не является взломом веб-сайта, размещение такого объявления на сайте компании привело бы к аналогичному эффекту.

К сожалению, такие примеры можно перечислять долго. Теперь пора перейти к технической стороне дела. Более подробную информацию о проблемах веб-безопасности всех видов можно найти в работах Ду (Du, 2019), Шнайера (Schneier, 2004), Статтарда и Пинто (Stuttard and Pinto, 2007). Также множество описаний конкретных случаев вы найдете в интернете.

## 8.12.2. Безопасное именование ресурсов и DNSSEC

Давайте еще раз затронем проблему спуфинга DNS и начнем с самого простого случая. Допустим, Алиса хочет посетить веб-сайт Боба. Она набирает в браузере нужный URL и через несколько секунд видит страницу. Но настоящая ли она? Может, да, а может, нет. Не исключено, что Труди снова взялась за старое. Предположим, она перехватила исходящие пакеты Алисы и изучила их. Отыскав HTTP-запрос `GET` к сайту Боба, взломщица сама зашла на эту страницу, изменила ее и отправила ни о чем не подозревающей Алисе. Хуже того, Труди может сильно снизить цены в интернет-магазине Боба, тем самым сделав его товары очень привлекательными. Это резко повышает вероятность того, что Алиса вышлет номер своей кредитной карты «Бобу» (с целью приобрести что-нибудь по выгодной цене).

Одним из недостатков традиционной атаки посредника является то, что Труди должна иметь возможность перехватывать исходящий трафик Алисы и подделывать входящий. На практике она должна прослушивать телефонную линию Алисы или Боба (поскольку прослушивать оптоволоконный магистральный кабель — задача непростая). Конечно, перехват телефонных разговоров возможен, но это требует больших усилий, а Труди не только умна, но и ленива.

Кроме того, Алису можно обмануть проще: например, спуфинг DNS, о котором мы говорили в разделе 8.2.3. В двух словах это выглядит так. Злоумышленник сохраняет некорректные данные сопоставления сервиса на промежуточном сервере имен, чтобы сервер направлял пользователей на поддельный IP-адрес. Если пользователь захочет подключиться к сервису, он извлечет этот адрес и в итоге свяжется не с реальным сервером, а со злоумышленником.

Настоящая проблема в том, что служба DNS разрабатывалась в те времена, когда интернет был чисто исследовательской сетью, работавшей в нескольких сотнях университетов, и ни Алиса, ни Боб, ни Труди не были приглашены на этот праздник жизни. Вопрос защиты информации тогда еще не стоял; задача была

[1] Вид спекуляции на бирже, когда торговец берет акции в кредит и продает их по текущей цене. После падения акций он выкупает их по новой, низкой стоимости и возвращает их брокеру. — *Примеч. ред.*

в том, чтобы интернет вообще функционировал. Но с годами все кардинально изменилось, поэтому в 1994 году IETF создал рабочую группу, которая должна была обеспечить безопасность DNS. Этот проект под названием **DNSSEC (DNS Security – защита DNS)** действует до сих пор; первая версия представлена в спецификации RFC 2535, а обновление – в RFC 4033–4035. К сожалению, DNSSEC не развернут в полном масштабе, поэтому многие DNS-серверы все еще уязвимы для атак подмены.

Концептуально система DNSSEC очень проста. Она основана на шифровании с открытыми ключами. В каждой зоне DNS (см. главу 7) имеется два ключа – открытый и закрытый. Вся информация, отправляемая DNS-сервером, подписывается с помощью закрытого ключа зоны источника, и принимающая сторона может легко проверить подлинность данных.

DNSSEC предоставляет три основные службы:

1. Подтверждение источника данных.
2. Распространение открытых ключей.
3. Аутентификацию транзакций и запросов.

Первая служба является основной: она проверяет, что пришедшие данные были одобрены владельцем зоны. Вторая служба используется для безопасного хранения и извлечения открытых ключей. Третья позволяет защититься от атак повторного воспроизведения и подмены сервера. Заметим, что конфиденциальность здесь не обеспечивается: такая задача не стоит, поскольку вся информация DNS считается открытой. Изначально предполагалось, что процесс внедрения системы займет несколько лет. Поэтому нужно было организовать взаимодействие между серверами с поддержкой DNSSEC и остальными серверами. Это подразумевает невозможность внесения в протокол каких-либо изменений. Теперь рассмотрим эту систему более детально.

Записи DNS группируются в **наборы записей ресурсов (Resource Record Set, RRSET)**. В таком наборе объединены все записи с одинаковым именем, классом и типом. Например, RRSET может состоять из нескольких записей *A*, если DNS-имя преобразуется в первичный и вторичный IP-адрес. Наборы расширяются за счет некоторых новых типов записей (они рассмотрены ниже). Каждый RRSET хешируется (например, с помощью SHA-2). Хеш подписывается закрытым ключом зоны (например, с применением RSA). Единицей передаваемой клиентам информации является подписанный RRSET. Получив его, клиент может проверить, действительно ли набор был подписан закрытым ключом зоны отправителя. Если подпись корректна, данные принимаются. Поскольку каждый RRSET содержит собственную подпись, эти наборы можно кэшировать где угодно, даже на не слишком надежных серверах, не опасаясь за их безопасность.

Система DNSSEC вводит несколько новых типов записей. Первая из них – *DNSKEY*. В ней указывается открытый ключ зоны, пользователя, хоста или другого принципала, криптографический алгоритм генерации подписи, наименование протокола передачи и некоторые другие данные. Открытый ключ хранится в незащищенном виде. Сертификаты X.509 не используются из-за их

громоздкости. В поле алгоритма содержится значение 1 для MD5/RSA и другие значения для остальных комбинаций. Поле протокола может указывать на применение IPsec или другого протокола защиты соединений (если он вообще используется).

Второй новый тип записи — *RRSIG*. В ней содержится подписанный хеш, сформированный согласно алгоритму, который указан в *DNSKEY*. Подпись охватывает все записи RRSET (в том числе все *DNSKEY*), за исключением ее самой. Здесь также указано время начала и конца действия подписи, имя ее владельца и некоторая дополнительная информация.

Система DNSSEC устроена так, что для большей надежности закрытый ключ зоны можно хранить вне сети. Один или два раза в день содержимое базы данных зоны вручную переносится (например, на таком безопасном носителе, как старый добрый компакт-диск) на автономный компьютер, где расположен закрытый ключ. Здесь генерируются подписи для всех RRSET, и полученные таким образом записи *RRSIG* вновь записываются на защищенное устройство и возвращаются на главный сервер зоны. Таким образом, закрытый ключ хранится на безопасном носителе в сейфе и извлекается лишь для того, чтобы подписать ежедневное обновление RRSET на автономном компьютере. По окончании генерации подписей все копии ключа удаляются из памяти, а носитель отправляется в сейф. Эта процедура сводит электронную защиту информации к физической, что гораздо понятнее для пользователей.

Метод предварительного подписания RRSET существенно ускоряет процесс обработки запросов, ведь благодаря ему отпадает необходимость в шифровании на лету. Правда, для хранения всех ключей и подписей в базах данных DNS требуется большой объем дискового пространства. Из-за подписи некоторые записи увеличиваются в размере десятикратно.

Получив подписанный RRSET, клиент применяет открытый ключ зоны отправителя для расшифровки хеша, затем вычисляет хеш самостоятельно и сравнивает два значения. Если они совпадают, данные считаются корректными. Но эта процедура не решает вопрос получения клиентом открытого ключа зоны. Один из возможных подходов сводится к тому, что надежный сервер передает клиенту ключ по защищенному соединению (например, при помощи IPsec).

Однако на практике предполагается, что у клиентов уже есть открытые ключи всех доменов верхнего уровня. Если Алиса пожелает посетить сайт Боба, она запросит у службы DNS набор RRSET для сайта bob.com. Этот набор будет включать IP-адрес и запись *DNSKEY* с открытым ключом Боба и будет подписан доменом верхнего уровня (com), поэтому Алиса сможет легко проверить его подлинность. На илл. 8.47 показан пример содержимого RRSET.

Теперь, вооружившись заверенной копией открытого ключа Боба, Алиса запрашивает у DNS-сервера Боба IP-адрес сайта www.bob.com. Этот RRSET подписан закрытым ключом Боба, поэтому Алиса может проверить подлинность подписи возвращенного Бобом набора. Если Труди каким-то образом внедрит фальшивый RRSET в один из кэшей, Алиса заметит это, так как запись *RRSIG* будет неправильной.

| Имя домена | Время жизни | Класс | Тип | Значение |
|---|---|---|---|---|
| bob.com. | 86400 | IN | A | 36.1.2.3 |
| bob.com. | 86400 | IN | DNSKEY | 3682793A7B73F731029CE2737D... |
| bob.com. | 86400 | IN | RRSIG | 86947503A8B848F5272E53930C... |

**Илл. 8.47.** Пример RRSET для bob.com. Запись DNSKEY содержит открытый ключ Боба. Запись RRSIG содержит хеш записей A и DNSKEY, подписанный сервером домена верхнего уровня com для подтверждения их подлинности

Впрочем, система DNSSEC также предоставляет криптографический механизм для связывания ответов с соответствующими запросами, что исключает возможность проведения атаки подмены данных, о которой мы говорили в начале главы. Эта мера (необязательная) заключается в том, что к ответу добавляется хеш запроса, подписанный закрытым ключом респондента. Поскольку у Труди нет закрытого ключа сервера домена верхнего уровня (com), она не сможет подделать ответ этого сервера на запрос интернет-провайдера Алисы. Конечно, Труди может опередить настоящий ответ, но фальшивка будет отслежена по неправильной подписи хешированного запроса.

DNSSEC поддерживает и некоторые другие типы записей. Так, для хранения сертификатов (например, стандарта X.509) можно использовать запись *CERT*. Зачем она нужна? Дело в том, что некоторые хотят превратить DNS в PKI. Случится это на самом деле или нет, пока неизвестно. На этом мы заканчиваем обсуждение DNSSEC. Более подробную информацию вы можете найти в спецификациях RFC.

## 8.12.3. Протокол TLS

Защита имен ресурсов — неплохое начало, но веб-безопасность этим не исчерпывается. Теперь мы поговорим об установлении защищенных соединений. Это довольно сложная тема (как и все связанное с безопасностью).

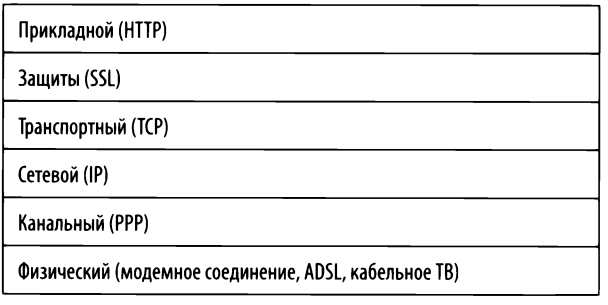
Когда веб-технологии стали достоянием общественности, их применяли для распространения статических страниц. Но вскоре некоторые компании задумались об использовании Всемирной паутины для выполнения финансовых транзакций, таких как покупка товаров по кредитным картам, онлайновые банковские операции, электронная торговля ценными бумагами. Для этого требовались защищенные соединения, поэтому в 1995 году компания Netscape Communications, на тот момент доминирующий на рынке разработчик браузера, представила пакет безопасности **SSL (Secure Sockets Layer — уровень защищенных сокетов)**. Теперь он называется **TLS (Transport Layer Security — защита транспортного уровня)**. Сегодня это программное обеспечение вместе с соответствующим протоколом получило широкое распространение (в частности, оно используется в браузерах Firefox, Brave, Safari и Chrome), и потому его стоит рассмотреть более подробно.

Итак, SSL создает защищенное соединение между двумя сокетами, при этом обеспечивается:

1. Согласование параметров между клиентом и сервером.
2. Аутентификация сервера клиентом.
3. Конфиденциальная передача данных.
4. Защита целостности данных.

Поскольку все перечисленные пункты нам уже знакомы, мы не будем на них подробно останавливаться.

Расположение SSL в структуре типичного стека протоколов показано на илл. 8.48. Фактически между прикладным и транспортным уровнями появляется новый уровень, который принимает запросы от браузера и отсылает их TCP для передачи серверу. После установления защищенного соединения основная задача SSL заключается в обеспечении сжатия и шифрования. При использовании протокола HTTP поверх SSL он называется **HTTPS (Secure HTTP – защищенный HTTP)**, хотя по сути это тот же HTTP. Правда, часто доступ осуществляется через новый порт (443) вместо стандартного (80). К слову, SSL используется не только в веб-браузерах, хотя это самое распространенное применение. Также он обеспечивает взаимную аутентификацию.

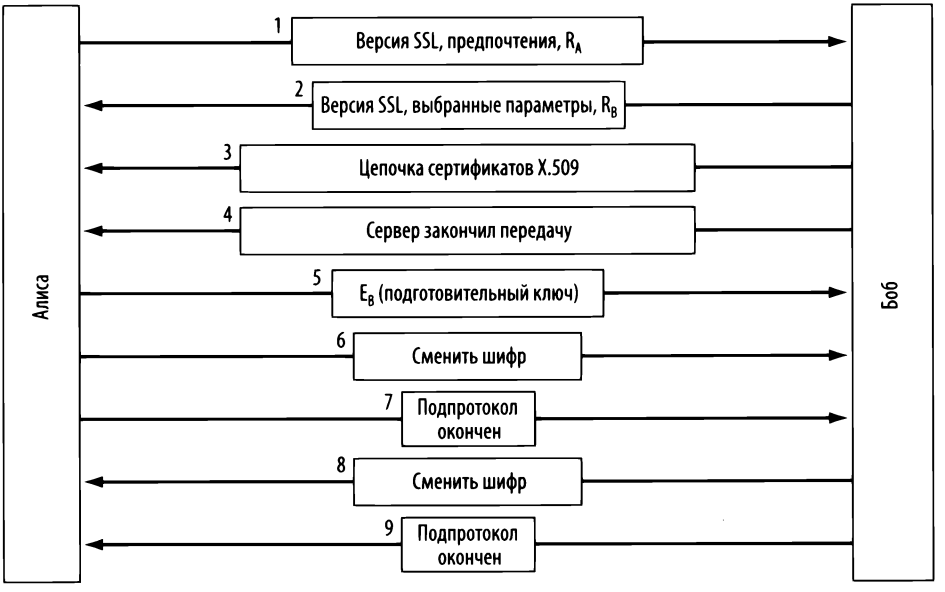



**Илл. 8.48.** Уровни (и протоколы), используемые обычным домашним браузером с SSL

Существует несколько версий протокола SSL. Ниже мы обсудим только версию 3, так как она является самой популярной. SSL предусматривает множество различных вариантов: с наличием или отсутствием сжатия, тем или иным алгоритмом шифрования, а также инструментами ограничения экспорта в криптографии. Последнее в основном предназначено для того, чтобы убедиться в корректном применении серьезного шифрования. Его можно использовать, только если данные передаются в пределах США. В иных случаях длину ключа ограничивают 40 битами, что криптографы воспринимают как насмешку. Но Netscape была вынуждена ввести это ограничение, чтобы получить лицензию на экспорт от правительства США.

SSL состоит из двух подпротоколов, один из которых предназначен для установления защищенного соединения, а второй – для его использования. Рассмотрим первый подпротокол (илл. 8.49). Все начинается с сообщения 1,

в котором Алиса отправляет Бобу запрос на установление соединения. В запросе указывается версия SSL, а также предпочтения Алисы относительно сжатия и алгоритмов шифрования. Также в нем содержится нонс $R_A$, который будет применен впоследствии.




**Илл. 8.49.** Упрощенный вариант подпротокола SSL установления соединения

Теперь очередь Боба. Он выбирает один из алгоритмов, поддерживаемых Алисой, и отсылает собственный нонс $R_B$ (сообщение 2). Затем он отправляет сертификат со своим открытым ключом (сообщение 3). Если сертификат не подписан какой-нибудь уважаемой организацией, Боб отсылает цепочку сертификатов, по которым Алиса может убедиться, что его сертификату действительно можно доверять. Все браузеры, включая тот, что установлен у Алисы, изначально снабжаются примерно сотней открытых ключей. Если среди присланных Бобом сертификатов встретится один из этих ключей, Алиса сможет восстановить по нему ключ Боба и проверить его. В этот момент Боб может прислать и другие сообщения (например, запрос на получение сертификата Алисы с ее открытым ключом). После выполнения своей части протокола Боб отправляет сообщение 4, в котором говорит, что настала очередь Алисы.

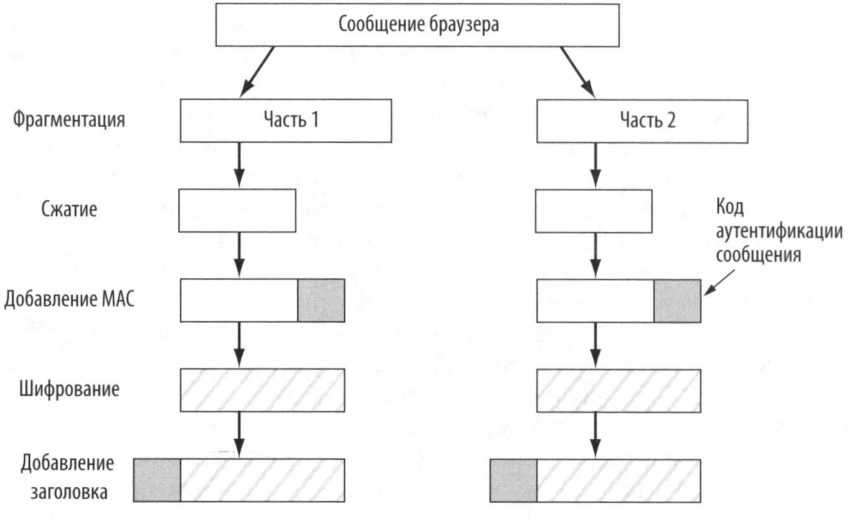
Алиса отвечает Бобу случайно выбранным 384-разрядным **подготовительным ключом (premaster key)**, который она зашифровала открытым ключом Боба (сообщение 5). Действующий ключ сеанса вычисляется при помощи подготовительного ключа и нонсов обеих сторон. Это достаточно сложная процедура. После получения сообщения 5 оба собеседника могут вычислить ключ сеанса. Для этого Алиса просит Боба переключиться на новый шифр (сообщение 6), а также сообщает, что она считает подпротокол

установления соединения оконченным (сообщение 7). Боб соглашается с ней (сообщения 8 и 9).

Несмотря на то что Алиса знает, кто такой Боб, сам Боб Алису не знает (если только у нее нет открытого ключа и сертификата к нему, что довольно необычно для физического лица). Поэтому первым сообщением Боба вполне может быть просьба войти в систему, используя полученные ранее имя пользователя и пароль. Впрочем, протокол входа находится за рамками SSL. Так или иначе, по окончании этой серии запросов-подтверждений можно переходить к передаче данных.

Как уже говорилось, SSL поддерживает многочисленные криптографические алгоритмы. Один из них использует для шифрования тройной DES с тремя отдельными ключами и SHA для обеспечения целостности сообщений. Эта комбинация алгоритмов работает довольно медленно, поэтому ее использовали в основном при выполнении банковских операций и в других случаях, требующих высокого уровня защиты. Для шифрования данных в обычных приложениях электронной коммерции применялся алгоритм RC4 с 128-разрядным ключом, а для аутентификации сообщений — алгоритм MD5. В качестве исходных данных RC4 использует 128-разрядный ключ, который значительно увеличивается в процессе работы алгоритма. С помощью этого внутреннего числа генерируется ключевой поток, который затем суммируется по модулю 2 с открытым текстом. В результате получается обычный потоковый шифр (см. илл. 8.18). В экспортных версиях также используется алгоритм RC4 со 128-разрядными ключами, но при этом 88 разрядов являются открытыми, что позволяет достаточно легко взломать шифр.

Для передачи данных используется второй подпротокол (илл. 8.50). Поступающие от браузера сообщения разбиваются на единицы данных размером




**Илл. 8.50.** Передача данных с использованием SSL

до 16 Кбайт. Если включено сжатие, то каждая единица сжимается отдельно. Далее по двум нонсам и подготовительному ключу вычисляется закрытый ключ, который объединяется со сжатым текстом. Результат хешируется алгоритмом, о котором договорились стороны (чаще всего это MD5). Полученный хеш добавляется к каждому фрагменту в виде **кода аутентификации сообщения (Message Authetication Code, MAC)**. Сжатый фрагмент вместе с MAC кодируется согласованным алгоритмом с симметричным ключом (обычно это суммирование по модулю 2 с ключевым потоком RC4). Наконец, присоединяется заголовок фрагмента, и тот передается по TCP-соединению как обычно.

Здесь стоит отметить следующее. Как уже упоминалось, в RC4 используется ряд слабых ключей, которые легко взламываются, поэтому сочетание SSL и RC4 уже давно считается не слишком надежным (Флюрер и др.; Fluhrer et al., 2001). Браузеры, позволяющие пользователю выбирать набор шифров, нужно настраивать на постоянное использование тройного DES со 168-разрядными ключами и SHA-2, невзирая на то что такая комбинация работает медленнее, чем RC4 с MD5. Еще лучше — перейти на браузеры, поддерживающие TLS (преемник SSL, описанный ниже).

С SSL связана еще одна проблема: у Алисы и Боба может не быть сертификатов. К тому же они не всегда проверяют ключи на соответствие сертификатам, даже располагая последними.

В 1996 году корпорация Netscape Communications направила SSL на стандартизацию в IETF. Результатом стал стандарт TLS. Он описан в RFC 5246.

TLS основан на 3-й версии протокола SSL. Хотя разница между ними сравнительно небольшая, все же отличий достаточно для того, чтобы SSL версии 3 и TLS были несовместимы. Например, в целях усиления ключа сеанса (усложнения его дешифровки) был изменен способ его вычисления по подготовительному ключу и нонсам. Из-за этой несовместимости большинство браузеров применяют оба протокола, и TLS превращается обратно в SSL, если нужно. Этот подход называется SSL/TLS. Первая реализация протокола TLS увидела свет в 1999 году, после чего в августе 2008 года появилась версия 1.2, а в марте 2018 года — версия 1.3. TLS поддерживает более сильные наборы шифров (в частности, AES), а также шифрование полей **указания имени сервера (Server Name Indication, SNI)**, которые можно использовать для идентификации веб-сайта, посещаемого пользователем (в случае их передачи в виде открытого текста).

## 8.12.4. Выполнение недоверенного кода

С точки зрения веб-безопасности именование ресурсов и установление соединений требуют повышенного внимания. Но это далеко не все вопросы, касающиеся этой сферы. Одна из наиболее сложных проблем связана с тем, что нам все чаще приходится запускать на локальных компьютерах сторонний недоверенный код. Ниже мы кратко рассмотрим некоторые возникающие в связи с этим проблемы и методы их решения.

### *Использование скриптов в браузере*

Поначалу веб-страницы представляли собой статичные HTML-файлы без исполняемого кода. Теперь же они, как правило, содержат небольшие программы, обычно написанные на языке **JavaScript** (и иногда скомпилированные в более эффективный формат **Web Assembly**). Поскольку скачивание и выполнение такого **переносимого кода (mobile code)** очевидным образом угрожает безопасности, были разработаны различные методы минимизации рисков.

У JavaScript нет формальной модели политики безопасности, зато есть длинная история реализаций с проблемами конфиденциальности. При этом каждая компания-разработчик пытается решить этот вопрос по-своему. Основная мера защиты нацелена на то, чтобы при отсутствии программных ошибок нельзя было нанести ущерб путем чтения и записи произвольных файлов, получения доступа к конфиденциальным данным других веб-страниц и т. д. В таких случаях говорят, что код выполняется в **песочнице**, то есть в **изолированной среде (sandboxed environment)**. Однако ошибки все же случаются.

Корнем всех проблем является уже сам факт запуска стороннего кода на вашем компьютере — вы сами напрашиваетесь на неприятности. С точки зрения безопасности это то же самое, что пригласить в гости вора и пытаться внимательно следить за тем, чтобы он не проник из кухни в гостиную. Если вы на секунду отвлечетесь, может произойти все что угодно. Загвоздка в том, что переносимый код позволяет использовать эффектную графику и обеспечивает быстрое взаимодействие с пользователем. Многие веб-дизайнеры считают, что это гораздо важнее безопасности, тем более что риску подвергается чей-то чужой компьютер, а не их собственный.

Предположим, что веб-сайт, содержащий ваши личные данные, предлагает вам дать обратную связь в виде произвольного текста, который видят все остальные пользователи. Идея в том, чтобы клиенты могли сообщить компании, понравились ли им предоставленные услуги. Но если сайт недостаточно тщательно проводит очистку данных в форме обратной связи, то злоумышленник может, помимо текста, разместить в поле небольшой фрагмент JavaScript-кода. Теперь представьте, что вы зашли на этот сайт и решили прочитать отзывы других пользователей. При этом JavaScript-код будет отправлен в ваш браузер. Однако браузер не знает, что это часть текста обратной связи. Он воспринимает его как обычный JavaScript-код, который можно встретить на любой другой веб-странице, и начинает его выполнять. Это позволяет вредоносному коду выкрасть все конфиденциальные данные, которые ваш браузер использует для этого сайта (например, файлы cookie), и отправить их злоумышленнику. Такие атаки называют **межсайтовым скриптингом (Cross-Site Scripting, CSS)**. Родственная разновидность атак, **подделка межсайтовых запросов (Cross-Site Request Forgery, CSRF)**, позволяет злоумышленнику выдавать себя за пользователя.

Еще одной потенциальной проблемой является недостаточная безопасность самого движка JavaScript. Например, используя уязвимость браузера, вредоносный JavaScript-код может взять под контроль браузер или даже операционную систему. Эта атака называется **теневой загрузкой (drive-by download)**: пользователь заходит на веб-сайт и, не зная об этом, подвергается заражению. Причем

сам сайт может и не быть вредоносным — JavaScript-код может находиться в рекламе или, как мы видели ранее, в поле обратной связи. Особую известность получила атака на Google и ряд других IT-компаний, так называемая **операция «Аврора»**. В ходе этой атаки злоумышленники применили теневую загрузку, чтобы охватить как можно больше компьютеров компании и получить доступ к репозиториям исходного кода.

### *Расширения браузера*

Наряду с увеличением возможностей веб-страниц с помощью кода сегодня также активно развивается рынок **браузерных расширений (browser extensions)**, **аддонов**, или **дополнений (add-ons)**, а также **плагинов**, или **подключаемых модулей (plug-in)**. Это компьютерные программы, расширяющие функциональность браузеров. Плагины позволяют интерпретировать или отображать определенный тип контента (PDF-документы или флеш-анимацию). Расширения и аддоны предоставляют новые функции (например, улучшенное управление паролями) или способы взаимодействия со страницами (маркировка страниц, просмотр сопутствующих товаров и т. д.).

Установить аддон, расширение или плагин очень просто: достаточно найти нужную программу в Сети и пройти по ссылке. Таким образом код загружается и встраивается в браузер. Все эти программы написаны в разных средах, в зависимости от того, в какой браузер они встраиваются. В любом случае в первом приближении они становятся частью доверенной вычислительной базы браузера. То есть если установленный код содержит ошибки, это может нарушить работу всего браузера.

Существуют еще две очевидные проблемы. Первая: программа может осуществлять вредоносные действия, например собирать личную информацию и отсылать ее на удаленный сервер. При этом браузер будет считать, что пользователь установил расширение именно с этой целью. Вторая проблема в том, что плагины дают браузеру возможность считывать новые типы контента. Часто это сам по себе полноценный язык программирования; хорошие примеры — PDF и Flash. Когда пользователи просматривают страницы с PDF и Flash, плагины выполняют соответствующий код. Он должен быть безопасным, ведь в браузере могут быть уязвимости, которыми код может воспользоваться. В силу этих причин аддоны и плагины лучше ставить по мере необходимости и скачивать только из надежных источников.

### *Трояны и другое вредоносное ПО*

Трояны и прочие вредоносные программы — еще одна форма недоверенного кода. Обычно пользователи устанавливают их, даже не подозревая об этом. Либо они думают, что программа безобидна, либо запускают ее в скрытом режиме, открыв вложение, прикрепленное к письму, и в результате на компьютер устанавливается вредоносное ПО. Обычно оно первым делом заражает другие программы (на диске или в оперативной памяти), после чего запуск любой из них будет вести к выполнению вредоносного кода. Он может распространиться

на другие компьютеры, зашифровать все документы на диске (с целью получения выкупа), проследить за вашими действиями и сделать множество других неприятных вещей. Некоторые вредоносные программы заражают загрузочный сектор жесткого диска, чтобы запускаться на этапе начальной загрузки компьютера. Подобное ПО превратилось в огромную проблему интернета и принесло многомиллиардные убытки. Какого-либо простого решения проблемы не существует. Возможно, положение спасет создание нового поколения операционных систем на основе защищенных микроядер, а также жесткое разделение пользователей, процессов и ресурсов.

## 8.13. СОЦИАЛЬНЫЕ ВОПРОСЫ

Интернет и технологии сетевой безопасности — это сферы, в которых сталкиваются социальные вопросы, государственная политика и технологии. Зачастую это приводит к серьезным последствиям. Ниже мы кратко рассмотрим три проблемы: приватность, свободу слова и авторские права. Стоит ли говорить, что это лишь верхушка айсберга. Более подробную информацию вы найдете в работах Андерсона (Anderson, 2008a), Баазе и Генри (Baase and Henry, 2017), Бернала (Bernal, 2018) и Шнайера (Schneier, 2004). Множество материалов по этой теме можно отыскать в интернете. Достаточно набрать в любой поисковой системе слова «приватность» (privacy), «цензура» (censorship) и «авторское право» (copyright).

### 8.13.1. Приватная и анонимная коммуникация

Имеют ли люди право на неприкосновенность частной жизни? Хороший вопрос. Четвертая поправка к Конституции США запрещает государству обыскивать жилье граждан, их имущество и личные бумаги без достаточных на то оснований. Перечень обстоятельств, в которых выдается ордер на обыск, ограничен. Таким образом, вопрос приватности стоит на повестке дня уже более 200 лет (по крайней мере, в США).

Что изменилось за последнее десятилетие? Теперь правительство может легко шпионить за гражданами, а те, в свою очередь, — без особых усилий избегать слежки. В XVIII веке для получения доступа к личным документам подданного требовалось по нескольку дней добираться до его фермы верхом, что было весьма непросто. В наши дни телефонные компании и интернет-провайдеры при виде ордера с готовностью предоставляют властям возможность прослушки. Это сильно упрощает работу полицейских, к тому же они больше не рискуют упасть с лошади.

Повсеместное распространение смартфонов выводит слежку государства за гражданами на новый уровень. Многие хранят на смартфоне всю информацию о своей жизни, а для его разблокировки часто используют технологию распознавания лиц. Это означает, что если, к примеру, сотрудник полиции хочет получить доступ к телефону подозреваемого, а тот отказывается снять блокировку, достаточно всего лишь подержать смартфон перед лицом задержанного. Как ни

странно, мало кто предполагает такое развитие событий, когда активирует распознавание лиц (или более раннюю функцию распознавания отпечатков пальцев).

Однако использование криптографии сильно меняет дело. Тот, кто потрудился скачать и установить PGP и сгенерировать хорошо защищенный, надежный ключ, может быть уверен, что никто в мире не сможет прочесть его электронную почту (даже при наличии ордера). Правительства прекрасно это понимают, и, разумеется, им это не нравится. В реальности неприкосновенность частной жизни означает, что уполномоченным органам очень трудно следить не только за преступниками всех мастей, но и за журналистами, а также за политическими оппонентами. Неудивительно, что многие правительства запрещают использование и экспорт криптографии. К примеру, до 1999 года во Франции любая негосударственная криптография была вне закона (если только государству не предоставлялись все ключи).

В этом Франция не была одинока. В апреле 1993 года правительство США объявило о своем желании создать **аппаратный криптопроцессор (clipper chip)** и сделать его стандартом для любых сетевых коммуникаций. При этом, как было заявлено, гражданам гарантировалось право на приватность. Вместе с тем упоминалось, что государство сможет расшифровывать весь трафик криптопроцессоров за счет использования **системы депонирования ключей (key escrow)**; эта система предоставила бы властям доступ ко всем ключам. И хотя представители правительства обещали использовать эту возможность только при наличии распоряжения суда, понятно, что такое заявление вызвало большой скандал. Сторонники приватности осуждали весь план от начала до конца, а сотрудники правоохранительных органов его восхваляли. В итоге правительство сдало позиции и отказалось от своей идеи.

Огромное количество материалов, посвященных конфиденциальности цифровой информации, доступно на сайте Фонда электронных рубежей (Electronic Frontier Foundation, EFF) по адресу www.eff.org.

### Анонимные ремейлеры

С помощью PGP, SSL и других технологий можно установить между двумя сторонами безопасное, аутентифицированное соединение, защищенное от слежки со стороны третьих лиц. Но иногда приватность лучше всего обеспечивается как раз *отсутствием* аутентификации, точнее — с помощью установления анонимных соединений. Анонимность востребована как при передаче сообщений между двумя пользователями, так и в новостных группах.

Приведем несколько примеров. Во-первых, политические диссиденты, живущие при авторитарном режиме, во избежание репрессий должны сохранять анонимность. Во-вторых, различные нарушения во многих коммерческих, образовательных, правительственных и других организациях зачастую выявляются благодаря осведомителям, желающим оставаться неизвестными. В-третьих, приверженцы непопулярных социальных, политических или религиозных убеждений могут общаться на тематических форумах или по электронной почте под вымышленными именами. В-четвертых, многие предпочитают обсуждать проблемы алкоголизма, психических расстройств, сексуальных домогательств,

жестокого обращения с детьми или принадлежность к преследуемым меньшинствам в конференциях, где люди могут оставаться анонимными. Конечно, помимо этого существует масса других примеров.

Рассмотрим конкретный пример. В 1990-х годах критики одной нетрадиционной религиозной группы опубликовали свои взгляды в сети USENET через **анонимный ремейлер (anonymous remailer)**. Это сервер, позволявший создавать псевдонимы и отправлять ему письма, которые затем публиковались под выбранными псевдонимами. При этом выявить настоящий источник было невозможно. Некоторые из этих публикаций раскрывали данные, которые, по мнению представителей секты, являлись коммерческой тайной либо были защищены авторским правом. В ответ на эти разоблачения религиозная группа подала в суд, поскольку в округе, где находился сервер, раскрытие коммерческой тайны и нарушение закона об авторском праве считались преступлением. Суд заставил владельцев ремейлера предоставить истинные имена тех, кто скрывался под псевдонимами и писал разоблачения (кстати, это не первый раз, когда религиозная группа недовольна раскрытием ее секретов: в 1536 году Уильям Тиндейл (William Tyndale) был сожжен за перевод Библии на английский).

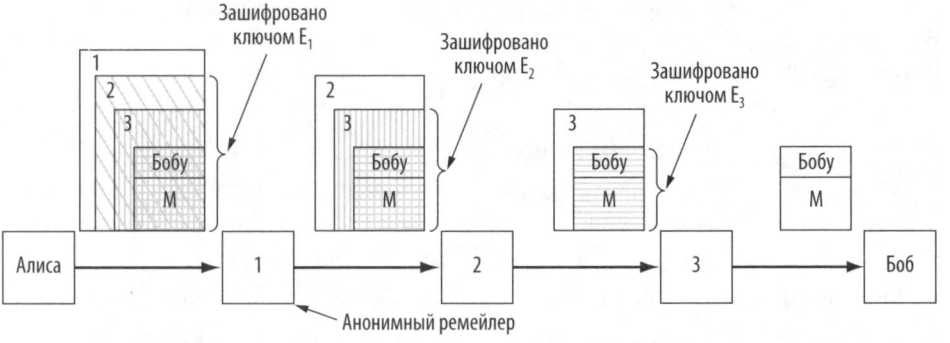
Значительная часть интернет-сообщества была сильно возмущена таким грубым нарушением принципов приватности. Все пришли к выводу, что сервер, хранящий таблицу соответствия реальных электронных адресов и псевдонимов (ремейлер первого типа), оказался бесполезным. Этот случай побудил многих разработчиков к созданию анонимных ремейлеров, способных противостоять атакам со стороны судебной системы.

Ремейлеры нового типа, так называемые **ремейлеры шифропанков (cypherpunk remailer)**, работают следующим образом. Пользователь создает электронное письмо с обычными заголовками RFC 822 (естественно, за исключением поля `From:`), шифрует его открытым ключом ремейлера и отправляет на сервер. Затем сервер отбрасывает заголовки RFC 822, расшифровывает содержимое и переадресовывает сообщение. В ремейлере нет учетных записей, и он не ведет журнал, поэтому даже в случае конфискации сервера никаких следов писем, прошедших через него, обнаружено не будет.

Многие пользователи, которым нужна анонимность, передают свои сообщения через несколько анонимных ремейлеров, как показано на илл. 8.51. В данном примере Алиса хочет отправить Бобу поздравление с Днем святого Валентина. Для нее очень важна анонимность, поэтому она использует три ремейлера. Она сочиняет письмо $M$ и вставляет заголовок, содержащий адрес электронной почты Боба. Затем сообщение шифруется открытым ключом ремейлера 3 $E_3$ (показано горизонтальной штриховкой). К нему прибавляется заголовок с электронным адресом ремейлера 3 (он передается открытым текстом). В результате получается сообщение, показанное на илл. 8.51 между ремейлерами 2 и 3.

После этого Алиса шифрует сообщение открытым ключом ремейлера 2 $E_2$ (это показано вертикальной штриховкой) и предваряет его открытым заголовком,

содержащим электронный адрес ремейлера 2. Получившееся сообщение показано на рисунке между ремейлерами 1 и 2. Затем Алиса шифрует его открытым ключом ремейлера 1 $E_1$, добавляет адрес этого ремейлера и, наконец, отправляет письмо. Итоговое сообщение показано справа от Алисы.




**Илл. 8.51.** Использование трех анонимных ремейлеров для передачи письма Алисы Бобу

Когда письмо Алисы достигает ремейлера 1, от него отсекается внешний заголовок. Тело сообщения расшифровывается и пересылается на ремейлер 2. Аналогичные шаги производятся и на двух других серверах.

Хотя проследить путь финального сообщения до Алисы и без того невероятно трудно, многие ремейлеры принимают дополнительные меры предосторожности. Например, они могут задерживать письма на какой-то случайный промежуток времени, переставлять их местами, добавлять или удалять всякий мусор в конце сообщения — в общем, делать все возможное, чтобы запутать тех, кто отслеживает трафик и пытается выявить автора того или иного письма, прошедшего через анонимный ремейлер. Описание такого ремейлера можно найти в классической работе Мазьера и Каашука (Mazières and Kaashoek, 1998).

Анонимной может быть не только электронная почта. К примеру, существуют сервисы анонимного просмотра веб-страниц. Они работают по такому же принципу многослойного пути, где каждый узел знает только следующий узел в цепочке. Этот метод называется **«луковой» маршрутизацией (onion routing)**: каждый узел как будто снимает новый слой, чтобы определить, куда передавать пакет. Пользователь может настроить браузер на использование такого сервиса в качестве прокси-сервера; широко известный пример такого подхода — браузер Tor (Берначи и др.; Bernaschi et al., 2019). После этого все HTTP-запросы направляются в сеть анонимайзера, которая запрашивает нужную страницу и возвращает ее пользователю. При этом в качестве источника запроса для веб-сайта выступает не сам пользователь, а выходной узел сети анонимайзера. Если эта сеть не хранит журналы активности, то определить, кто на самом деле запрашивал страницу, невозможно (даже в случае судебного разбирательства, поскольку эта информация просто отсутствует).

## 8.13.2. Свобода слова

Анонимность затрудняет для третьей стороны получение сведений о частных беседах. Другим ключевым социальным вопросом, несомненно, является свобода слова и ее противоположность — цензура, то есть попытка правительства ограничить спектр информации, которую граждане могут читать и публиковать. Всемирная паутина с ее миллионами страниц — настоящий рай для цензуры. В зависимости от типа и идеологии режима под запрет могут попадать сайты, содержащие следующее:

1. Материалы, не предназначенные для детей и подростков.
2. Материалы, выражающие ненависть к группам лиц по причине их этнического происхождения, религиозных убеждений, сексуальной ориентации и т. д.
3. Информация о демократии и демократических ценностях.
4. Описания исторических событий, не совпадающие с официальной версией.
5. Руководства по взлому различных замков, созданию ядерного оружия, шифрованию сообщений и т. д.

Как правило, неугодные сайты просто блокируются.

Иногда такая политика приводит к неожиданным результатам. Например, некоторые публичные библиотеки установили на своих компьютерах веб-фильтры, блокирующие доступ к порносайтам и таким образом делающие Паутину безопасной для детей. Фильтры сверяют адреса сайтов с «черными списками» и проверяют содержимое страниц на наличие бранных слов. Однажды в округе Лаудон, штат Вирджиния, фильтр заблокировал поиск информации о раке молочной железы — запрос содержал слово «breast» (женская грудь, молочная железа). Посетитель подал в суд на правительство округа. А в Ливерморе, штат Калифорния, возмущенный родитель засудил библиотеку за то, что там *не был* установлен фильтр (он застал своего 12-летнего сына за просмотром порнографии). Так что же делать библиотеке?

Многие люди никак не могут понять, что Всемирная паутина — действительно *всемирная*. Она охватывает весь земной шар. Государства по-разному смотрят на то, что можно размещать в Сети. Например, в ноябре 2000 года французский суд постановил, что Yahoo! (калифорнийская корпорация) должна закрыть французским пользователям доступ к аукциону нацистских памятных вещей, так как обладание такими предметами идет вразрез с законодательством Франции. Yahoo! обжаловала это решение в американском суде, который встал на ее сторону. Однако вопрос о том, законы какой страны должны применяться в подобных ситуациях, остается открытым.

Надо сказать, что Yahoo! долгие годы была одной из самых успешных интернет-компаний, но ничто не длится вечно. В 2017 году она была куплена компанией Verizon почти за $5 млрд. Цена этой сделки была снижена на $350 млн из-за утечки данных, затронувшей миллиард пользователей сервисов Yahoo!, что еще раз подчеркивает важность обеспечения безопасности.

Представьте, что случится, если какой-нибудь суд штата Юта вынесет решение о том, что Франция должна заблокировать сайты, посвященные винам, ведь распространение такой информации нарушает строгие законы Юты об алкоголе? Точно так же Китай может запретить все сайты, где рассказывается про демократию, так как это не в интересах Поднебесной. Должны ли иранские законы о религии применяться в либеральной Швеции? Может ли Саудовская Аравия заблокировать сайты о правах женщин? Эта проблема — настоящий ящик Пандоры.

Джон Гилмор (John Gilmore) однажды сказал: «Сеть воспринимает цензуру как разрушенный участок дороги и идет в обход». Конкретной реализацией этой мысли стал **сервис Eternity (Eternity service)** (Андерсон; Anderson, 1996). Его цель — гарантировать, что опубликованные материалы не исчезнут и не будут переписаны заново. Пользователь должен лишь указать, как долго нужно хранить информацию, оплатить услугу (стоимость зависит от сроков и объема данных) и загрузить материалы на сервер. После этого никто, включая самого пользователя, не сможет их удалить или отредактировать.

Как реализовать такой сервис на практике? Проще всего организовать пиринговую систему, в которой документы размещаются на десятках серверов участников проекта, каждый из которых получает свою долю вознаграждения (это служит стимулом для вступления в проект). Серверы должны располагаться в самых разных юрисдикциях для максимальной устойчивости системы. Списки из десяти случайно выбранных серверов следует надежно хранить в разных местах, чтобы в случае взлома одного из серверов остальные продолжали работу. Власти, желающие уничтожить неугодную информацию, никогда не будут до конца уверены в том, что найдены все копии. Кроме того, систему можно сделать самовосстанавливающейся: если станет известно об уничтожении каких-то экземпляров документов, держатели остальных копий попытаются найти новые места хранения на замену выбывшим из строя.

Сервис Eternity стал первой попыткой противостояния цензуре в Сети. С тех пор были предложены и другие системы, и некоторые из них даже нашли свое воплощение. Были добавлены новые возможности, такие как шифрование, анонимность, отказоустойчивость. Зачастую документы разбивают на фрагменты и хранят их на нескольких серверах. Среди таких систем можно упомянуть Freenet (Кларк и др.; Clarke et al., 2002), PASIS (Уайли и др.; Wylie et al., 2000) и Publius (Уолдман и др.; Waldman et al., 2000).

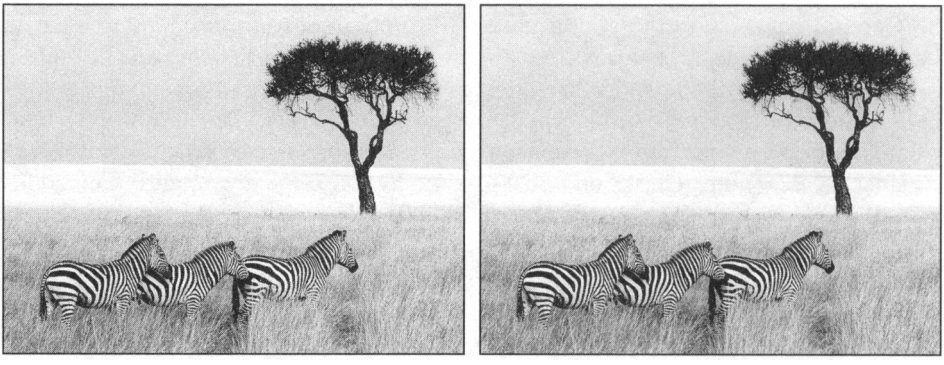
Наряду с фильтрацией и цензурой все большей проблемой становится распространение заведомо ложных данных, то есть **дезинформация**. Сегодня она используется для влияния на политические, социальные и финансовые процессы. Так, в 2016 году широкий резонанс получило создание злоумышленниками ряда сайтов с фальшивыми данными о кандидатах в президенты США и распространение ссылок на эти сайты в социальных сетях. В других случаях ложные сведения использовались с целью снижения для инвесторов цен на недвижимость. К сожалению, выявить дезинформацию, а тем более сделать это до того, как она станет достоянием общественности, весьма непросто.

### *Стеганография*

В странах, где цензура применяется особенно широко, всегда существуют диссиденты, которые пытаются ее обойти. Криптография, конечно, позволяет (не всегда законным путем) отправлять секретные сообщения так, чтобы никто не смог узнать их смысл. Однако если государство считает Алису своим врагом, то сам факт ее общения с Бобом может поставить его в такое же положение (даже при нехватке математиков репрессивные правительства понимают концепцию транзитивного замыкания). Здесь могли бы помочь анонимные ремейлеры, но если они запрещены внутри страны, а для отправки сообщения за границу нужна лицензия на экспорт, они бесполезны. Но Всемирная паутина всегда найдет решение.

Люди, которым требуется приватность, зачастую пытаются сохранить в тайне саму коммуникацию. Наука сокрытия сообщений называется **стеганографией (steganography)**; ее название происходит от греческого слова, которое можно перевести как «скрытое письмо». Собственно, древние греки сами использовали стеганографию. Геродот описывал метод секретной переписки военачальников: посыльному брили голову, набивали татуировку с сообщением и ждали, пока волосы снова отрастут, после чего отправляли его в путь. Современные технологии базируются на той же концепции, разве что пропускная способность стала выше, а задержки — ниже (и при этом не требуются услуги парикмахера).

В качестве примера рассмотрим илл. 8.52 (а). На этой фотографии, сделанной одним из авторов данной книги в Кении, изображены три зебры, созерцающие акацию. Фотография на илл. 8.52 (б), выглядит точно так же, но обладает одной интересной особенностью. Дело в том, что в нее внедрен полный текст пяти самых известных пьес Шекспира: *Гамлет*, *Король Лир*, *Макбет*, *Венецианский купец* и *Юлий Цезарь*; суммарный объем текста превышает 700 Кбайт.



**Илл. 8.52.** (а) Три зебры и дерево. (б) Три зебры, дерево и полный текст пяти пьес Уильяма Шекспира

Как же это было сделано? Стеганографический канал работает следующим образом. Размер исходного изображения составляет 1024 × 768 пикселей. Каждый пиксель представлен тремя 8-битными числами, по одному для красного (R),

зеленого (G) и синего (B) каналов интенсивности цвета. Итоговый цвет пикселя формируется посредством линейной суперпозиции трех этих цветов. При стеганографическом шифровании младший бит каждого из трех цветовых значений RGB используется в качестве скрытого канала. Таким образом, в пикселе помещаются три бита секретных данных (по одному в каналах красного, зеленого и синего цветов). С учетом размера нашей фотографии в ней поместится 1024 × 768 × 3 бит, или 294 912 байт, скрытой информации.

Полный текст пяти пьес Шекспира с небольшим предисловием занимает 734 891 байт. Сначала он был сжат до 274 Кбайт стандартным алгоритмом. После этого данные были зашифрованы с использованием IDEA и размещены в младших битах значений интенсивностей цветов. Как можно увидеть (хотя это как раз и невозможно), присутствие текстовой информации на фото совершенно не заметно, даже на ее большой полноцветной версии. Глаз человека с трудом отличает 21-разрядные цвета от 24-разрядных.

Приведенные на илл. 8.52 черно-белые фотографии с низким разрешением не отражают в полной мере мощные возможности стеганографии. Чтобы дать более глубокое представление о том, как она работает, мы подготовили демонстрационный пример, представленный на веб-сайте данной книги. Этот пример включает полноцветную версию нашей фотографии в высоком разрешении со встроенными в нее пятью пьесами и позволяет опробовать в деле инструменты для вставки и извлечения текста из изображений.

Для скрытого обмена информацией диссиденты могут создать веб-сайт, заполненный политически «правильными» или нейтральными фотографиями (например, Великого Вождя, местных спортивных команд, теле- и кинозвезд и т. п.). Разумеется, они будут содержать стеганографические сообщения. Учитывая то, что сообщения перед внедрением в графический файл сжимаются и шифруются, даже если кто-то и заподозрит их присутствие «внутри» изображений, отличить их от белого шума, а тем более расшифровать их смысл будет очень трудно. Конечно, фотографии для стеганографической обработки должны быть отсканированы или сняты самостоятельно: если взять картинку из интернета и записать в нее секретное сообщение, можно выявить «второе дно» простым побитовым сравнением. Пример того, как можно встроить звукозапись в неподвижное изображение, можно найти в работе Чаудхари и Шаубе (Chaudhary and Chaube, 2018).

Стеганографические сообщения записываются не только в графические файлы, но и, к примеру, в звуковые. Скрытую информацию можно поместить в VoIP-звонок путем манипуляций с задержками пакетов и искажением звука. Данные внедряются даже в поля заголовка пакета (Любач и др.; Lubacz et al., 2010). Форматирование текста и расположение тегов в HTML-файле также может нести скрытую информацию.

Мы рассмотрели стеганографию в контексте проблемы свободы слова, однако у этой технологии есть масса иных применений. Очень часто авторы электронных изображений вшивают в файлы секретные сообщения, которые в случае необходимости смогут подтвердить их авторские права. Если кто-нибудь украдет удачную фотографию и разместит ее на своем сайте без указания авторства,

законный владелец сможет доказать в суде свои права на изображение, предъявив стеганографическую подпись. Этот метод называется **технологией цифровых водяных знаков (watermarking)**. Подробнее см. работу Муйко и Эрнандеса (Muyco and Hernandez, 2019).

Стеганография является областью активных исследований, о чем говорит проведение конференций, целиком посвященных данной технологии. Среди работ по этой теме можно отметить следующие: (Хегарти и Кин; Hegarty and Keane, 2018), (Кумар; Kumar, 2018), (Патил и др.; Patil et al., 2019).

## 8.13.3. Авторское право

Приватность и цензура — это те сферы, в которых сталкиваются технологические аспекты и общественные интересы. Третьей такой областью является защита авторских прав. **Авторское право (copyright)** предоставляет создателям **интеллектуальной собственности (Intellectual Property)** — писателям, поэтам, художникам, композиторам, музыкантам, фотографам, кинорежиссерам, хореографам и т. д. — исключительное право на извлечение выгоды из результатов своей работы. Авторское право выдается на определенный срок, который обычно равен сроку жизни автора, плюс 50 лет (или 75 лет в случае корпоративного авторского права). По окончании этого срока интеллектуальная собственность становится достоянием общественности, и каждый волен распоряжаться ею как хочет. Так, например, проект Gutenberg (www.gutenberg.org) разместил в Сети более 50 тысяч общедоступных произведений (в том числе работы Шекспира, Диккенса и Твена). В 1998 году Конгресс США разрешил продлить срок действия авторских прав еще на 20 лет по запросу Голливуда. Представители киноиндустрии утверждали, что без принятия этой меры больше никто ничего не создаст. Таким образом, защита авторских прав на оригинал анимационного фильма о Микки Маусе (1928) была продлена до 2024 года, после чего кто угодно сможет арендовать кинотеатр и вполне законно показывать этот фильм, не получая разрешение от Walt Disney Company. В то же время патенты действуют всего лишь 20 лет, а люди продолжают совершать открытия и создавать изобретения.

Вопрос о защите авторских прав вышел на первый план, когда число пользователей Napster, сервиса для обмена музыкой, достигло 50 млн. Хотя на самом деле Napster не копировал музыку, проблемой стала его центральная база данных с информацией о записях, имеющихся у пользователей. Суд постановил, что это косвенно нарушает закон об авторских правах: система помогала участникам преступать закон. В общем-то, никто не жалуется на то, что идея авторских прав плоха (хотя многим не нравится то, что компании имеют куда большие привилегии в этом плане, чем простые граждане), однако следующее поколение технологий распространения музыкальных записей уже поднимает важные этические вопросы.

В качестве примера рассмотрим одноранговую сеть, участники которой вполне законно обмениваются файлами: музыкой, ставшей достоянием общественности, домашними видео, религиозной литературой (не составляющей коммерческой тайны) и т. д. Возможно, какая-то небольшая часть этих файлов

защищена законом об авторских правах. Допустим, все пользователи проекта постоянно находятся в сети (по ADSL или кабелю). На каждом компьютере хранится список того, что есть на его жестком диске, а также список всех компьютеров сети. В поисках конкретного файла можно выбрать случайного пользователя и узнать, есть ли у него нужный материал. Если нет, проверяются остальные компьютеры из имеющегося списка, затем — из списков, хранящихся у других участников, и т. д. Компьютеры отлично справляются с такой работой. Найдя нужный файл, можно просто скопировать его.

Если найденная работа защищена авторским правом, вероятнее всего, пользователь его нарушает (хотя при международной передаче важно знать, чей закон применяется: в некоторых странах загружать файлы в систему нельзя, а скачивать можно). А виноват ли поставщик информации? Является ли преступлением хранение на своем жестком диске легально скачанной музыки, за которую были заплачены деньги, при условии, что к диску имеют доступ посторонние лица? Если в ваш незапертый загородный домик проникает вор с ноутбуком и сканером, снимает копию с книги, защищенной авторским правом, и уходит восвояси, разве *вы* виноваты в том, что не защитили чужие авторские права?

Существует еще одна проблема, связанная с авторскими правами. Между Голливудом и компьютерной индустрией ведется ожесточенная борьба. Киностудии требуют усилить защиту интеллектуальной собственности, а компьютерщики говорят, что они не обязаны охранять голливудские ценности. В октябре 1998 года Конгресс принял **Закон об авторском праве в цифровую эпоху (Digital Millenium Copyright Act, DMCA)**, в котором говорится, что взлом механизма защиты объекта авторского права, а также распространение сведений о методе взлома является преступлением. Аналогичный акт был принят и в Евросоюзе. Разумеется, мало кто думает, что пираты с Дальнего Востока должны иметь возможность нелегально копировать чужие работы, однако многие считают, что новый закон нарушил баланс между интересами владельцев авторских прав и интересами общества.

Взять хотя бы такой пример. В сентябре 2000 года некий музыкальный консорциум в попытках построить непробиваемую систему онлайн-продаж аудиозаписей организовал конкурс и пригласил всех желающих попробовать ее взломать (это действительно важный этап при создании новой системы защиты). Группа экспертов по безопасности данных из разных университетов под руководством профессора Эдварда Фельтена (Edward Felten) из Принстона приняла вызов и взломала систему. Выводы, сделанные в ходе исследования, они описали в статье и направили ее на конференцию USENIX, посвященную защите информации. Доклад был рассмотрен коллегией и принят. Однако незадолго до конференции Фельтен получил письмо от Американской ассоциации звукозаписывающих компаний (Recording Industry Association of America, RIAA) с угрозами судебного разбирательства за нарушение DMCA в случае публикации статьи.

В ответ на это ученые подали иск в федеральный суд по поводу легальности публикации научных статей по защите информации. Опасаясь, что окончательное решение суда будет вынесено не в пользу консорциума, его представители

были вынуждены снять свои претензии к Фельтену, и на этом инцидент был исчерпан. Несомненно, отзыв иска был продиктован тем, что дело изначально было проигрышное для музыкального консорциума: он сам устроил конкурс на взлом системы, а после угрожал судом тем, кому это удалось. После того как конфликт был улажен, статья все-таки была напечатана (Крейвер и др.; Craver et al., 2001). Очевидно, впереди нас ждет еще много подобных споров.

Тем временем люди обменивались контентом, защищенным авторским правом, через одноранговые сети. Со своей стороны правообладатели стали рассылать пользователям и интернет-провайдерам автоматические **уведомления о нарушении закона DMCA**. Сначала владельцы авторских прав пытались предупреждать пользователей (и привлекать их к суду) индивидуально, однако это оказалось неудобно и неэффективно. Теперь они судятся с интернет-провайдерами за отсутствие реакции на нарушение DMCA клиентами. Это ненадежные основания для обвинений, поскольку узлы одноранговых сетей часто выдают ложные сведения о том, что они предлагают для скачивания (Куэвас и др.; Cuevas et al., 2014; Сантос и др.; Santos et al., 2011), и в результате злоумышленником могут посчитать даже ваш принтер (Пиатек и др.; Piatek et al., 2008). Однако этот подход позволил правообладателям добиться некоторого успеха: в декабре 2019 года федеральный суд США обязал компанию Cox Communications выплатить владельцам авторских прав $1 млрд за ненадлежащее реагирование на уведомления о нарушении прав.

В связи с этим также встает вопрос о **доктрине добросовестного использования (fair use doctrine)**, ставшей результатом судебных решений во многих странах. Она состоит в том, что покупатели защищенной продукции имеют некие ограниченные права на копирование этих материалов, включая частичное использование в научных целях, применение в качестве обучающего материала в школах и колледжах, а также создание архивных резервных копий на тот случай, если исходный носитель выйдет из строя. Чтобы определить, является ли использование добровольным, нужно ответить на следующие три вопроса: 1) является ли оно коммерческим, 2) каков процент скопированных данных, 3) влияет ли копирование на объем продаж. Так как DMCA и аналогичные законы Евросоюза не позволяют обходить системы защиты от копирования, они заодно запрещают и легальное добросовестное использование. По сути, DMCA отбирает у потребителей историческое право активно поддерживать продавцов контента. Налицо столкновение интересов.

Еще одно явление, которое затмевает собой по уровню смещения баланса между правообладателями и потребителями даже DMCA, — это **доверенные вычисления (trusted computing)**. Этот проект продвигается такими отраслевыми организациями, как **Группа доверенных вычислений (Trusted Computing Group, TCG)**, и разрабатывается совместными усилиями Intel и Microsoft. Идея состоит в том, чтобы обеспечить тщательное наблюдение за действиями пользователя (например, за нелегальным воспроизведением музыки) на уровне ниже операционной системы с целью не допустить нежелательного поведения. Это возможно благодаря небольшому чипу **TPM (Trusted Platform Module — модуль доверенной платформы)**, в работу которого сложно вмешаться. Сегодня многие

ПК поставляются со встроенным TPM. Он позволяет программам правообладателей манипулировать компьютером таким образом, что пользователь не может на это повлиять. Это ставит вопрос о том, кто является «доверенным» в этой системе. Очевидно, не пользователь. Стоит ли говорить, что эта схема вызвала огромный общественный резонанс. Конечно, здорово, что индустрия наконец обратила внимание на проблемы безопасности, однако то, что большинство усилий направлено на ужесточение законов об авторских правах, а не на борьбу с вирусами, взломщиками, мошенниками и другими проблемами, волнующими большинство пользователей, — весьма прискорбно.

Коротко говоря, авторам законов и юристам теперь предстоит в течение долгих лет пытаться урегулировать взаимоотношения владельцев авторских прав и потребителей. Виртуальный социум ничем не отличается от реального: здесь постоянно сталкиваются интересы различных групп, что приводит к борьбе за власть и судебным разбирательствам. Обычно рано или поздно люди находят некий компромисс — до появления следующей инновационной технологии.

## 8.14. РЕЗЮМЕ

Безопасность представляет собой соединение таких важных свойств, как приватность, целостность и доступность. К сожалению, зачастую трудно сказать, насколько безопасной является система. Однако мы можем неукоснительно соблюдать принципы безопасности, сформулированные Зальцером и Шредером и другими исследователями.

Разумеется, злоумышленники будут пытаться взломать систему, комбинируя базовые составляющие атаки — разведку (попытки выяснить, что, где и при каких условиях происходит), прослушивание (изучение содержимого трафика), подмену данных (выдача себя за другого) и нарушение работы (DoS-атаки). Все эти элементы могут быть чрезвычайно сложными. Для защиты от атак сетевые администраторы устанавливают брандмауэры, системы обнаружения и предотвращения вторжений. Они могут быть развернуты как во всей сети, так и на отдельном хосте, и работают на основе сигнатур или аномалий. В любом случае важным показателем полезности таких решений является число ложноположительных результатов (ложных тревог) и ложноотрицательных результатов (незамеченных атак). В силу так называемой ошибки базовой оценки высокая доля ложноположительных результатов сильно снижает эффективность системы обнаружения вторжений, особенно если атаки происходят редко при большом количестве событий.

Криптография — это инструмент обеспечения конфиденциальности информации, а также проверки ее целостности и аутентичности. Все современные криптографические системы основаны на принципе Керкгоффса, гласящем, что алгоритм должен быть общеизвестным, а ключ — секретным. Чтобы зашифровать открытый текст, многие алгоритмы выполняют сложные преобразования с заменами и перестановками. Между тем если на практике удастся реализовать принципы квантовой криптографии, то с помощью одноразовых блокнотов можно будет создать действительно непробиваемые криптосистемы.

Криптографические алгоритмы делятся на два типа: с симметричным и с открытым ключом. Алгоритмы с симметричным ключом при шифровании искажают значения битов в последовательности итераций, параметризованных ключом. К числу наиболее популярных алгоритмов этого типа относятся тройной DES и AES (Rijndael). Такие алгоритмы могут работать в режиме электронного шифроблокнота, сцепления блоков шифра, потокового шифра и др.

В алгоритмах с открытым ключом для шифрования и дешифрования используются разные ключи, причем ключ дешифрования невозможно вычислить по ключу шифрования. Это позволяет публиковать открытый ключ. Один из самых распространенных алгоритмов такого типа — RSA. Его надежность обусловлена сложностью факторизации больших чисел. Также существуют алгоритмы на основе эллиптических кривых.

Юридические, коммерческие и другие документы необходимо подписывать. Существуют различные методы генерации цифровых подписей, основанные на алгоритмах как с симметричным ключом, так и с открытым. Обычно подписываемые сообщения хешируются с помощью SHA-2 или SHA-3, после чего подписываются полученные хеши, а не исходные сообщения.

Управление открытыми ключами реализуется при помощи сертификатов. Это документы, связывающие между собой открытые ключи и обладающих ими принципалов. Сертификаты подписываются доверенным органом или тем, кто способен предъявить действительную цепочку промежуточных сертификатов, ведущих к нему. Начальное звено этой цепочки (доверенный корневой центр сертификации) должно быть известно заранее, но обычно сертификаты многих из них встроены в браузер.

Эти криптографические методы позволяют обезопасить сетевой трафик. На сетевом уровне работает система IPsec: она шифрует потоки пакетов между хостами. Брандмауэры фильтруют как входящий, так и исходящий трафик, анализируя используемые протоколы и порты. VPN имитируют старые сети на основе выделенных линий, чтобы обеспечить необходимые параметры безопасности. Наконец, в беспроводных сетях требуется серьезная защита, чтобы сообщения не читали все подряд, и такие протоколы, как 802.11i, ее обеспечивают. Рекомендуется выстраивать эшелонированную защиту из нескольких механизмов.

При установлении соединения стороны должны идентифицировать себя и при необходимости определить общий ключ сеанса. Для этого применяются различные протоколы аутентификации, в том числе подразумевающие наличие третьей, доверенной стороны, а также протоколы Диффи — Хеллмана, Kerberos и шифрование с открытым ключом.

Безопасность электронной почты достигается с помощью сочетания методов, представленных в этой главе. К примеру, PGP сжимает сообщение, а потом шифрует его с помощью секретного ключа. В свою очередь, секретный ключ шифруется открытым ключом получателя. Кроме того, вычисляется хеш-функция сообщения. Проверка целостности выполняется за счет того, что хеш перед отправкой подписывается.

Еще одной важной темой является веб-безопасность, и начинается она с защиты имен ресурсов. DNSSEC предотвращает спуфинг DNS. Большинство сайтов

электронной коммерции использует протокол TLS для установления надежных, аутентифицированных сеансов между клиентом и сервером. Для борьбы с проблемами, связанными с переносимым кодом, разработаны разные методы, среди которых выполнение в изолированной среде (песочнице) и подписание кода.

Наконец, интернет поднимает много проблем на стыке технологий и государственной политики. К их числу относятся вопросы обеспечения приватности, свободы слова и авторских прав. Решение этих проблем требует участия специалистов из разных сфер. Учитывая, что технологии сегодня развиваются быстро, а сфера законодательства и госуправления — медленно, рискнем предположить, что к моменту публикации следующего издания данной книги эти вопросы все еще будут актуальны. Если прогноз не сбудется, мы подарим каждому читателю по головке сыра.

## ВОПРОСЫ И ЗАДАЧИ

1. Рассмотрите принцип полной опосредованности. На какое нефункциональное требование к системе повлияет его строгое соблюдение?
2. В главе 3 описывается применение кодов CRC для выявления ошибочных сообщений. Объясните, почему CRC нельзя использовать для обеспечения целостности сообщений.
3. Какой тип сканирования отражает сетевой журнал, представленный ниже? Дайте максимально полный ответ, указав, какие хосты являются активными и какие порты при этом открыты или закрыты.

```
Time            From            To                    Flags    Other info
21:03:59.711106 brutus.net.53 > host201.caesar.org.21: F 0:0(0) win 2048 (ttl 48, id 55097)
21:04:05.738307 brutus.net.53 > host201.caesar.org.21: F 0:0(0) win 2048 (ttl 48, id 50715)
21:05:10.399065 brutus.net.53 > host202.caesar.org.21: F 0:0(0) win 3072 (ttl 49, id 32642)
21:05:16.429001 brutus.net.53 > host202.caesar.org.21: F 0:0(0) win 3072 (ttl 49, id 31501)
21:09:12.202997 brutus.net.53 > host024.caesar.org.21: F 0:0(0) win 2048 (ttl 52, id 47689)
21:09:18.215642 brutus.net.53 > host024.caesar.org.21: F 0:0(0) win 2048 (ttl 52, id 26723)
21:10:22.664153 brutus.net.53 > host003.caesar.org.21: F 0:0(0) win 3072 (ttl 53, id 24838)
21:10:28.691982 brutus.net.53 > host003.caesar.org.21: F 0:0(0) win 3072 (ttl 53, id 25257)
21:11:10.213615 brutus.net.53 > host102.caesar.org.21: F 0:0(0) win 4096 (ttl 58, id 61907)
21:11:10.227485 host102.caesar.org.21 > brutus.net.53: R 0:0(0) ack 4294947297 win 0 (ttl 25, id 38400)
```

4. Какой тип сканирования отражает сетевой журнал, представленный ниже? Дайте максимально полный ответ, указав, какие хосты являются активными и какие порты при этом открыты или закрыты.

```
Time               From              To              Flags        Other info
20:31:49.635055 IP 127.0.0.1.56331 > 127.0.0.1.22:    Flags [FPU], seq 149982695, win 4096, urg 0, length 0
20:31:49.635123 IP 127.0.0.1.56331 > 127.0.0.1.80:    Flags [FPU], seq 149982695, win 3072, urg 0, length 0
20:31:49.635162 IP 127.0.0.1.56331 > 127.0.0.1.25:    Flags [FPU], seq 149982695, win 4096, urg 0, length 0
20:31:49.635200 IP 127.0.0.1.25    > 127.0.0.1.56331: Flags [R.],  seq 0, ack 149982696, win 0, length 0
20:31:49.635241 IP 127.0.0.1.56331 > 127.0.0.1.10000: Flags [FPU], seq 149982695, win 3072, urg 0, length 0
20:31:49.635265 IP 127.0.0.1.10000 > 127.0.0.1.56331: Flags [R.],  seq 0, ack 149982696, win 0, length 0
20:31:50.736353 IP 127.0.0.1.56332 > 127.0.0.1.80:    Flags [FPU], seq 150048230, win 1024, urg 0, length 0
20:31:50.736403 IP 127.0.0.1.56332 > 127.0.0.1.22:    Flags [FPU], seq 150048230, win 3072, urg 0, length 0
```

5. Алиса хочет связаться с сайтом www.vu.nl, но запись для этого домена на ее сервере имен была отравлена, поэтому пакеты направляются на компьютер, контролируемый злоумышленником. В какой мере взломщик способен нарушить приватность, целостность и аутентичность информации в следующих случаях: (а) незашифрованная коммуникация (http) между Алисой и сайтом www.vu.nl, (б) зашифрованная коммуникация (https) между Алисой и www.vu.nl, при этом сайт использует самоподписанный сертификат, (в) зашифрованная коммуникация (https) между Алисой и www.vu.nl, при которой сайт использует сертификат, подписанный законным центром сертификации?
6. Система обнаружения вторжений проверила за день 1 000 000 событий. 50 раз она подняла тревогу, и в 10 случаях тревога оказалась ложной. В действительности за этот день было всего проведено 70 атак. Рассчитайте показатели точности, полноты, F-меры и доли верных результатов для этой системы обнаружения вторжений.
7. Объясните, в чем состоит ошибка базовой оценки при использовании системы обнаружения вторжений с показателями из предыдущей задачи.
8. Вы проводите атаку внемаршрутного захвата TCP-соединения против компьютера Герберта. Сначала вам нужно узнать, авторизовался ли он со своего компьютера на FTP-сервере в домене vusec.net (напомним, что FTP использует для команд порт получателя 21). Оба компьютера работают на ОС Linux и реализуют исходную спецификацию RFC 5961 (эта комбинация описана в разделе 8.2). Как выяснить, что Герберт авторизован на FTP-сервере, применив метод внемаршрутного взлома TCP?
9. Расшифруйте следующее сообщение, составленное с помощью моноалфавитного подстановочного шифра. Открытый текст, состоящий только из букв, представляет собой хорошо известный отрывок из поэмы Льюиса Кэрролла.

   kfd ktbd fzm eubd kfd pzyiom mztx ku kzyg ur bzha kfthcm
   ur mftnm zhx mfudm zhx mdzythc pzq ur ezsszcdm zhx gthcm
   zhx pfa kfd mdz tm sutythc fuk zhx pfdkfdi ntcm fzld pthcm
   sok pztk z stk kfd uamkdim eitdx sdruid pd fzld uoi efzk
   rui mubd ur om zid uok ur sidzkf zhx zyy ur om zid rzk
   hu foiia mztx kfd ezindhkdi kfda kfzhgdx ftb boef rui kfzk

10. Взломайте следующий колоночный перестановочный шифр. Открытый текст взят из популярной книги о компьютерных сетях, поэтому в нем с большой вероятностью может встретиться слово «information». Открытый текст полностью состоит из букв (без пробелов). Зашифрованный текст разбит на блоки по пять символов для удобства.

prort elhfo osdte taxit matec hbcni wtseo datnr tuebc eyeao ncrin nfeee aoeai nirog m

11. Алиса закодировала свои сообщения для Боба перестановочным шифром. Для дополнительной защиты она зашифровала ключ перестановочного шифра методом подстановки и сохранила шифр на своем компьютере.

Труди заполучила закодированный перестановочный ключ. Может ли она расшифровать сообщения Алисы? Ответ поясните.

12. Боб зашифровал сообщение для Алисы подстановочным шифром. Для надежности он зашифровал сообщение еще раз, используя перестановочный шифр. Будет ли итоговый шифротекст другим, если Боб применит сначала перестановочный, а затем — подстановочный шифр? Обоснуйте свой ответ.
13. Подберите 77-битный одноразовый блокнот, с помощью которого из шифра на илл. 8.11 получается текст «Hello World».
14. Вы шпион, и у вас очень кстати есть библиотека с бесконечным количеством книг (как и у вашего оператора). Изначально вы договорились использовать в качестве одноразового блокнота роман «Властелин колец». Объясните, как можно было бы использовать все имеющиеся книги для создания бесконечно длинного одноразового блокнота.
15. Квантовая криптография требует наличия фотонной пушки, способной при необходимости испускать одиночный фотон, соответствующий 1 биту. Подсчитайте, сколько фотонов переносит 1 бит по оптоволоконной линии с пропускной способностью 250 Гбит/с. Допустим, длина фотона равна длине волны, то есть 1 мкм (применительно к данной задаче). Также предполагается, что скорость света в оптоволокне равна 20 см/нс.
16. Если при использовании квантовой криптографии злоумышленник перехватит и заново сгенерирует фотоны, некоторые значения будут приняты неверно, а значит, ошибки появятся и в одноразовом блокноте Боба. Какая доля его блокнота будет содержать ошибки (в среднем)?
17. Один из базовых принципов криптографии гласит, что все сообщения должны быть избыточными. Но мы знаем, что это свойство позволяет злоумышленнику понять, правильно ли он угадал секретный ключ. Рассмотрите два типа избыточности. При первом типе начальные $n$ бит открытого текста содержат известную последовательность, при втором — конечные $n$ бит сообщения включают хеш. Эквивалентны ли эти типы избыточности с точки зрения безопасности? Поясните свой ответ.
18. Банковская система использует следующий формат сообщений о транзакциях: 2 байта для идентификатора отправителя, 2 байта для идентификатора получателя и 4 байта для переводимой суммы. Эти сообщения шифруются перед отправкой. Какие элементы нужно добавить в эти сообщения, чтобы они соответствовали двум криптографическим принципам, о которых шла речь в этой главе?
19. Группа преступников, ведущих незаконный бизнес, опасается прослушки их цифровых коммуникаций со стороны полиции. Чтобы не допустить этого, они используют систему сквозного шифрования сообщений с шифром, не поддающимся криптоанализу. С помощью каких двух методов полиция все же может перехватить их разговоры?
20. Было подсчитано, что компьютеру для взлома шифров, снабженному миллионом процессоров и способному обрабатывать один ключ за 1 нс, понадобится

$10^{16}$ лет для взлома 128-битной версии AES. Давайте сосчитаем, какое время уйдет на сокращение этого срока до одного года. Для достижения этой цели компьютеры должны работать в $10^{16}$ раз быстрее. Если закон Мура (он гласит, что вычислительные мощности компьютеров удваиваются через каждые 18 месяцев) будет действовать и в дальнейшем, сколько лет пройдет до того момента, когда многопроцессорный компьютер сможет сократить время взлома шифра до одного года?

21. Ключи AES могут иметь длину 256 бит. Сколько вариантов ключей доступно в таком режиме? Сможете ли вы найти в какой-нибудь науке, например физике, химии или астрономии, понятия, оперирующие подобными величинами? Найдите в интернете материалы, посвященные большим числам. Сделайте выводы из этого исследования.
22. Рассмотрите следующий пример шифрования со сцеплением блоков. Допустим, вместо преобразования одного бита 0 в бит 1 в поток зашифрованного текста после блока $C_i$ был вставлен дополнительный бит 0. Какая часть открытого текста будет при этом повреждена?
23. Сравните режим сцепления блоков шифра с режимом шифрованной обратной связи с точки зрения количества операций шифрования, необходимых для передачи большого файла. Какой режим эффективнее и насколько?
24. Алиса и Боб общаются друг с другом, используя шифрование с открытым ключом. Кто может извлечь открытый текст $P$ из сообщения $E_B(D_A(P))$ и какие шаги для этого нужно предпринять?
25. Представьте, что прошло уже несколько лет, вы стали преподавателем и проводите лекцию по компьютерным сетям. Вы объясняете студентам, что в алгоритме шифрования RSA открытые и закрытые ключи состоят из пар значений ($e$, $n$) и ($d$, $n$) соответственно. Возможные значения $e$ и $d$ зависят от $z$. В свою очередь, $z$ зависит от $n$. Один из студентов говорит вам, что эта схема слишком сложная, и предлагает ее упростить. Вместо того чтобы выбирать значение $d$ как взаимно простое число по отношению к $z$, предлагается выбирать значение $e$ как взаимно простое число по отношению к $n$. Тогда можно найти такое значение $d$, при котором $e \times d$ равно единице, разделенной по модулю на $n$. В этом случае $z$ уже можно не использовать. Как такая модификация повлияет на количество усилий, необходимых для взлома шифра?
26. Допустим, пользователь обнаружил, что его закрытый RSA-ключ ($d1$, $n1$) совпадает с открытым RSA-ключом ($e2$, $n2$) другого пользователя. То есть $d1 = e2$ и $n1 = n2$. Стоит ли этому пользователю подумать об изменении своего открытого и закрытого ключа? Обоснуйте ответ.
27. Один из недостатков протокола передачи цифровой подписи на илл. 8.21 состоит в том, что в случае сбоя Боб может потерять содержимое своей оперативной памяти. Какую проблему это порождает и как ее можно решить?
28. На илл. 8.23 мы видим, как Алиса передает Бобу подписанное сообщение. Если Труди заменит $P$, Боб заметит это. А что будет, если Труди заменит и $P$, и подпись?

29. У цифровых подписей есть потенциальная уязвимость, и виной этому ленивые пользователи. После составления договора в электронной коммерции обычно требуется, чтобы клиент подписал его своим хешем SHA. Если пользователь не озаботится проверкой соответствия хеша и контракта, он имеет шанс случайно подписать другой договор. Допустим, вездесущая и бессмертная мафия решила сделать на этом деньги. Бандиты создают платный веб-сайт (с порнографией, азартными играми и т. п.) и запрашивают у новых клиентов номера кредитных карт. Затем пользователю отсылается договор, в котором говорится, что он желает воспользоваться услугами и оплатить их кредиткой. Договор нужно подписать, однако владельцы сайта знают, что большинство пользователей не будет сверять хеш с контрактом. Покажите, каким образом злоумышленники смогут купить бриллианты в легальном ювелирном интернет-магазине за счет ничего не подозревающих клиентов.
30. В математическом классе 25 учащихся. Предположим, все они родились в первой половине года — между 1 января и 30 июня. Какова вероятность того, что по крайней мере у двух из них совпадают дни рождения? Предполагается, что никто из учеников не родился 29 февраля.
31. После того как Элен созналась Мэрилин в своем обмане в деле с предоставлением должности Тому, Мэрилин решила устранить проблему, записывая свои сообщения на диктофон, с тем чтобы ее новый секретарь просто их перепечатывала. Мэрилин планирует проверять на своем терминале набранный текст, чтобы быть уверенной в его правильности. Может ли новый секретарь по-прежнему применить атаку «дней рождения», чтобы фальсифицировать сообщение, и если да, то как? *Подсказка:* может.
32. Рассмотрите пример неудачной попытки получения Алисой открытого ключа Боба (см. илл. 8.25). Допустим, они уже установили общий закрытый ключ, однако Алиса все же хочет узнать открытый ключ Боба. Существует ли в данной ситуации безопасный способ его получения? Если да, то как это сделать?
33. Алиса хочет пообщаться с Бобом, используя шифрование с открытым ключом. Она устанавливает соединение с кем-то, кто, по ее предположению, является Бобом. Она спрашивает у него открытый ключ, и тот отсылает его вместе с сертификатом X.509, подписанным корневым CA (центром сертификации). У Алисы уже есть открытый ключ корневого CA. Что теперь должна сделать Алиса, чтобы удостовериться, что она действительно общается с Бобом? Будем считать, что Бобу безразлично, с кем он общается (то есть под именем Боба мы здесь понимаем, например, какую-нибудь государственную организацию).
34. Допустим, система использует PKI, базирующийся на древовидной иерархии CA. Алиса желает пообщаться с Бобом и после установления соединения получает от него сертификат, подписанный CA *X*. Допустим, Алиса никогда не слышала про *X*. Что ей нужно сделать, чтобы удостовериться, что на той стороне канала связи находится именно Боб?

35. Может ли IPsec с заголовком аутентификации (AH) использоваться в транспортном режиме, если один из компьютеров находится за NAT-блоком? Обоснуйте свой ответ.
36. Алиса хочет отправить Бобу сообщение, используя хеши SHA-2. Она консультируется с вами по поводу подходящего алгоритма. Что вы посоветуете?
37. Назовите одно из преимуществ кодов HMAC по сравнению с хешами SHA-2, подписанными с помощью RSA.
38. Назовите одну причину, по которой брандмауэры следует настраивать на анализ входящего трафика. Теперь назовите одну причину, по которой их нужно настраивать на анализ исходящего трафика. Как вы думаете, насколько успешен такой анализ?
39. Допустим, организация установила защищенную VPN для обеспечения безопасной связи между своими сайтами в интернете. Джим, сотрудник этой организации, использует VPN для общения со своим руководителем, Мэри. Опишите тип коммуникации между Джимом и Мэри, который бы не требовал шифрования или других механизмов защиты, а также другой тип коммуникации, для которого эти средства, наоборот, нужны. Обоснуйте свой ответ.
40. Измените одно сообщение в протоколе, изображенном на илл. 8.31, таким образом, чтобы он стал устойчивым к зеркальной атаке. Объясните, почему это сработало.
41. Для установления секретного ключа между Алисой и Бобом используется алгоритм Диффи — Хеллмана. Алиса отправляет Бобу (227, 5, 82). Боб отвечает (125). Секретное число Алисы $x = 12$, Боба — $y = 3$. Покажите, как Алиса и Боб могут вычислить секретный ключ.
42. Даже если два пользователя никогда не встречались, не обменивались секретами и не имеют сертификатов, они все равно могут установить общий закрытый ключ с помощью алгоритма Диффи — Хеллмана.
    а) Объясните, почему этот алгоритм уязвим для атаки посредника.
    б) Будет ли он уязвим для этой атаки, если значения $n$ или $g$ будут секретными?
43. Почему в протоколе на илл. 8.36 $A$ отправляется открытым текстом вместе с зашифрованным ключом сеанса?
44. Для чего используются временные метки и нонсы — с целью обеспечить приватность, целостность, доступность, аутентификацию или неотказуемость? Обоснуйте свой ответ.
45. При обсуждении протокола, показанного на илл. 8.36, было упомянуто, что размещение 32 нулевых битов в начале каждого незашифрованного сообщения представляет угрозу безопасности. Допустим, все сообщения начинаются со случайного для каждого пользователя числа; по сути, оно представляет собой второй секретный ключ, известный только пользователю и KDC-центру. Исключает ли это возможность применения атаки на основе открытого текста? Почему?

46. Основными параметрами безопасности являются приватность, целостность, доступность, аутентификация и неотказуемость. Можно ли обеспечить каждое из этих свойств путем шифрования с открытым ключом? Если да, объясните, каким образом.
47. Рассмотрите основные аспекты безопасности, перечисленные в предыдущей задаче. Можно ли обеспечить каждое из этих свойств за счет использования профилей сообщения? Если да, объясните как.
48. В протоколе Нидхема — Шредера Алиса формирует два запроса, $R_A$ и $R_{A2}$. Не является ли это излишеством? Можно ли здесь обойтись одним запросом?
49. Предположим, что организация применяет для аутентификации Kerberos. Как повлияет на систему выход из строя сервера аутентификации или выдачи тикетов с точки зрения безопасности и доступности сервиса?
50. Алиса использует протокол аутентификации с открытым ключом (илл. 8.40), чтобы аутентифицировать коммуникацию с Бобом. Однако, отправляя сообщение 7, Алиса забыла зашифровать $R_B$. Таким образом, Труди знает значение $R_B$. Нужно ли Алисе и Бобу повторять процедуру аутентификации с новыми параметрами, чтобы удостовериться, что коммуникация безопасна? Обоснуйте свой ответ.
51. В протоколе аутентификации с открытым ключом, показанном на илл. 8.40, в сообщении 7 случайное число $R_B$ зашифровано ключом $K_S$. Так ли необходимо это шифрование или $R_B$ можно было отправить обратно открытым текстом? Обоснуйте свой ответ.
52. У кассовых аппаратов, использующих магнитные карты и PIN-коды, есть существенный недостаток: кассир-злоумышленник может модифицировать считывающее устройство, чтобы сохранять всю информацию с карт, включая PIN-код, а затем проводить дополнительные поддельные транзакции. В следующем поколении кассовых терминалов будут использоваться карты с полноценным центральным процессором, клавиатурой и небольшим дисплеем. Разработайте для этой системы протокол, который не сможет взломать кассир.
53. Вы получили от своего банка электронное письмо с сообщением о подозрительной активности в отношении вашего счета. Однако, перейдя по встроенной ссылке и авторизовавшись на сайте банка, вы не видите никаких новых транзакций. Решив, что письмо было направлено вам по ошибке, вы выходите из системы. Но когда вы снова авторизуетесь на сайте на следующий день, вы обнаруживаете, что все ваши деньги были переведены на неизвестный счет. Что же произошло?
54. В силу каких *двух* причин система PGP сжимает сообщения?
55. Можно ли передать сообщение PGP с помощью многоадресной рассылки? Какие ограничения будут применяться?
56. Предположим, что все в интернете используют PGP. Можно ли в этом случае отправить PGP-сообщение на произвольный интернет-адрес и быть

полностью уверенным в том, что на стороне получателя оно будет корректно расшифровано? Поясните свой ответ.

57. Протокол передачи данных SSL подразумевает наличие двух нонсов и подготовительного ключа. Какую роль играют нонсы (если она вообще у них есть)?
58. Представьте себе изображение размером 2048 на 1536 пикселей. Вы хотите скрыть в нем файл размером 2,5 Мбайт. Какую часть файла вы сможете внедрить в это изображение посредством стеганографии? Какую часть файла получится внедрить, если сжать его в четыре раза? Продемонстрируйте ваши расчеты.
59. Изображение на илл. 8.52 (б) содержит ASCII-текст пяти пьес Шекспира. Можно ли спрятать между зебрами не текст, а музыку? Если да, то как и в каком объеме? Если нет, то почему?
60. Имеется текстовый файл размером 60 Мбайт, который нужно спрятать с помощью стеганографии в изображении — в младших битах каждого цвета. Каким должен быть размер картинки, чтобы можно было скрыть в ней весь текстовый файл? Какой размер потребуется, если текстовый файл будет сжат в три раза? Выдайте ответ в пикселях и продемонстрируйте расчеты. Предполагается, что используется изображение с соотношением сторон 3:2 (например, 3000 × 2000 пикселей).
61. Алиса была постоянным пользователем анонимного ремейлера первого типа. Она могла публиковать сколько угодно сообщений в своей любимой конференции alt.fanclub.alice, и все знали, что их автор — Алиса, так как все письма подписывались одним псевдонимом. При правильной работе системы у Труди не было возможности писать от имени Алисы. После того как все анонимные ремейлеры первого типа были отключены, Алиса перешла на шифропанковские ремейлеры и создала новую тему в конференции. Предложите Алисе актуальный способ защиты от Труди, которая пытается писать от ее имени.
62. В 2018 году исследователи выявили у современных процессоров две уязвимости, Spectre и Meltdown. Узнайте, как работает атака Meltdown, и объясните, какие принципы безопасности при разработке процессоров соблюдались недостаточно (что и привело к появлению этой уязвимости). Обоснуйте свой ответ. Что могло побудить разработчиков отказаться от строгого соблюдения этих принципов?
63. Во время туристических поездок вы подключаетесь к Wi-Fi отеля, используя уникальный пароль. Объясните, как злоумышленники могут прослушивать ваш канал связи.
64. Найдите в интернете описание какого-нибудь интересного случая, касающегося приватности, и напишите о нем небольшой доклад на одну страницу.
65. Напишите программу, шифрующую входные данные путем суммирования по модулю 2 с ключевым потоком. Найдите или напишите сами хороший генератор случайных чисел для создания ключевого потока. Программа должна действовать как фильтр, принимая открытый текст со стандартного

устройства ввода и выдавая шифр на стандартном устройстве вывода (и наоборот). У программы должен быть один параметр: ключ, запускающий генератор случайных чисел.

66. Напишите процедуру для вычисления хеша SHA-2 блока данных. У процедуры должно быть два параметра: указатель на входной буфер и указатель на 20-байтный выходной буфер. Чтобы найти спецификацию SHA-2, поищите в интернете FIPS 180-1, в этом документе содержится полное описание.

67. Напишите функцию, которая принимает поток ASCII-символов и шифрует его с помощью режима сцепления блоков. Размер блока должен быть 8 байт. Программа должна преобразовывать открытый текст со стандартного устройства ввода в зашифрованный текст на стандартном устройстве вывода. Для решения этой задачи вы можете воспользоваться любой подходящей системой, чтобы определить конец входного потока и/или понять, когда добавить заполнение, чтобы завершить блок. Вы можете выбрать формат вывода данных, главное, чтобы он был однозначным. Программа должна получать два параметра:

    1. Указатель на вектор инициализации.
    2. Номер $k$, представляющий смещение шифра подстановки, так чтобы значение ASCII было зашифровано $k$-м значением перед ним в алфавите.

    Например, если $x = 3$, тогда «A» шифруется как «D», «B» — как «E» и т. д. Сделайте разумные допущения с учетом последнего значения в наборе ASCII. Четко зафиксируйте в своем коде все сделанные вами допущения, касающиеся входного потока и алгоритма шифрования.

68. Цель этой задачи — дать лучшее представление о механизме работы RSA. Напишите функцию, которая получает в качестве параметров простые числа $p$ и $q$, рассчитывает открытый и секретный RSA-ключи с помощью этих параметров и выводит $n$, $z$, $d$ и $e$ в качестве выходных данных. Также функция должна принимать поток значений ASCII и шифровать его с помощью вычисленных ключей RSA. Программа должна получать открытый текст со стандартного устройства ввода и выдавать зашифрованный текст на стандартном устройстве вывода. Шифрование должно производиться посимвольно: каждый символ из входных данных шифруется независимо от остальных. Для решения этой задачи можно воспользоваться любой подходящей системой, чтобы определить конец входного потока. Вы можете выбрать формат вывода данных, главное, чтобы он был однозначным. Четко зафиксируйте в своем коде все допущения, касающиеся входного потока и алгоритма шифрования.

# ГЛАВА 9

# Рекомендации для чтения и библиография

Мы закончили изучение компьютерных сетей, но это только начало. Многие интересные темы не были рассмотрены во всей полноте, а некоторые — вообще пропущены за неимением места. Данная глава содержит список дополнительной литературы для читателей, желающих продолжить ознакомление с компьютерными сетями.

## 9.1. ЛИТЕРАТУРА ДЛЯ ДАЛЬНЕЙШЕГО ЧТЕНИЯ

Существует большое количество книг, касающихся всех аспектов компьютерных сетей и распределенных систем. Среди журналов, часто публикующих статьи по этой теме, стоит выделить следующие два: *IEEE/ACM Transactions on Networking* и *IEEE Journal on Selected Areas in Communications*.

В периодических изданиях *ACM Special Interest Groups on Data Communications* (SIGCOMM) и *Mobility of Systems, Users, Data, and Computing* (SIGMOBILE) публикуется множество интересных статей, особенно на тему новых открытий и разработок. Это издания *Computer Communication Review* и *Mobile Computing and Communications Review*.

Кроме того, Институт инженеров электротехники и электроники (IEEE) выпускает еще три журнала: *IEEE Internet Computing*, *IEEE Network Magazine* и *IEEE Communications Magazine* — в них содержатся обзоры, учебные статьи и информация об исследованиях по компьютерным сетям. Первые два в основном посвящены архитектуре, стандартам и программному обеспечению, а третий — технологиям коммуникаций (оптоволоконной, спутниковой связи и т. д.).

Ежегодно или раз в два года проводятся несколько конференций, о которых рассказывается в многочисленных статьях, посвященных сетям. В частности, обратите внимание на конференции *SIGCOMM*, *NSDI* (Symposium on Networked Systems Design and Implementation), *MobiSys* (Conference on Mobile Systems, Applications, and Services), *SOSP* (Symposium on Operating Systems Principles) и *OSDI* (Symposium on Operating Systems Design and Implementation).

Ниже мы перечислим дополнительную литературу, сгруппированную по главам этой книги. Большая часть приведенных рекомендаций представляет собой ссылки на издания (или отдельные главы) с учебной или обзорной информацией. Полные ссылки приводятся в разделе 9.2.

## 9.1.1. Введение

Comer (Комер), «The Internet Book», 4-е издание

В эту книгу стоит заглянуть всем, кто ищет простое и понятное описание интернета. В ней доступным для новичков языком говорится об истории, развитии, технологиях, протоколах и службах интернета. Книга заинтересует и более подготовленных читателей благодаря большому количеству затронутых тем.

*Computer Communication Review*, 50th Anniversary Issue, Oct. 2019

В 2019 году состоялась 50-я конференция ACM SIGCOMM, и в специальном выпуске журнала рассказывается о том, как выглядели компьютерные сети и SIGCOMM[1] на заре своего становления и как они изменились со временем. Несколько бывших руководителей SIGCOMM опубликовали статьи о том, как обстояли дела в прошлом и в каком направлении следует развиваться в будущем. Также обсуждается взаимосвязь между научными исследованиями в области компьютерных сетей и промышленным применением этих технологий. Говорится и об эволюции самого журнала.

Crocker, S.D. (Крокер), *The Arpanet and Its Impact on the State of Networking*

В честь 50-летнего юбилея сети ARPANET, предшественницы интернета, журнал *IEEE Computer* собрал за виртуальным круглым столом шестерых разработчиков этой сети, чтобы поговорить о ней и о том, какое огромное влияние она оказала на весь мир. Участниками круглого стола стали Бен Баркер (Ben Barker), Винт Серф (Vint Cerf), Стив Крокер (Steve Crocker), Боб Кан (Bob Kahn), Лен Клейнрок (Len Kleinrock) и Джефф Рулифсон (Jeff Rulifson). В ходе обсуждения выяснилось много интересных фактов, в частности то, что хотя проект сети ARPANET был изначально рассчитан на ряд лучших исследовательских университетов США, их представители поначалу не видели в нем никакой ценности и не хотели к нему присоединяться.

Crovella and Krishnamurthy (Кровелла и Кришнамурти), «Internet Measurement»

Как узнать, хорошо ли работает интернет? Вопрос не тривиальный, ведь за интернет никто не отвечает. Эта книга описывает методы, разработанные для оценки работы интернета, от сетевой инфраструктуры до приложений.

*IEEE Internet Computing*, Jan.-Feb. 2000

Первый выпуск журнала *IEEE Internet Computing* в новом тысячелетии вполне ожидаемо содержит размышления людей, приложивших руку к созданию интернета в предыдущую эпоху, о том, каким он может стать в будущем. В обсуждении приняли участие такие эксперты, как Пол Бэран (Paul Baran), Лоуренс Робертс (Lawrence Roberts), Леонард Клейнрок (Leonard Kleinrock), Стивен Крокер (Stephen Crocker), Дэнни Коэн (Danny Cohen), Боб Меткалф

[1] Специальная группа Ассоциации вычислительной техники по передаче данных. — *Примеч. ред.*

(Bob Metcalfe), Билл Гейтс (Bill Gates), Билли Джой (Billy Joy) и др. Оцените, насколько верными оказались их прогнозы спустя два десятилетия.

Kurose and Ross (Куроуз и Росс), «Computer Networking: A Top-Down Approach»[1]

Эта книга очень схожа по содержанию с той, которую вы держите в руках. Но есть отличие — после вступительной части описание стека протоколов начинается с верхнего (прикладного) уровня и постепенно опускается до канального уровня. Здесь нет главы о физическом уровне, зато есть отдельные главы, посвященные безопасности и мультимедийным данным.

McCullough (Маккалоу), «How the Internet Happened: From Netscape to the iPhone»

Если вы искали легкое чтиво об истории интернета с начала 1990-х годов до наших дней, эта книга — то, что вам нужно! Она рассказывает о множестве компаний, продуктов и устройств, сыгравших важную роль в развитии и росте интернета, включая Netscape, Internet Explorer, AOL, Yahoo, Amazon, Google, Napster, Netflix, PayPal, Facebook и iPhone.

Naughton (Нотон), «A Brief History of the Future»

Кто же все-таки изобрел интернет? На это претендует множество людей. И некоторые из них по праву этого заслуживают. Пол Бэран (Paul Baran), написавший отчет о коммутации пакетов, специалисты различных университетов, разработавшие архитектуру ARPANET, сотрудники BBN, которые запрограммировали первые IMP, Боб Кан (Bob Kahn) и Винт Серф (Vint Cerf), создавшие TCP/IP и т. д. Эта книга рассказывает историю интернета, по крайней мере до 2000 года, и содержит множество анекдотов.

Severance (Северанс), «Introduction to Networking: How the Internet Works»

Если вы хотите изучить тему компьютерных сетей, прочитав только сто, а не тысячу страниц, вероятно, вам стоит обратиться к этой книге. Это пособие с кратким и легким стилем изложения освещает большинство ключевых моментов, включая сетевые архитектуры, канальный уровень, протокол IP, службу DNS, транспортный и прикладной уровни, протокол SSL и модель OSI. Все это — с забавными рисованными иллюстрациями.

## 9.1.2. Физический уровень

Boccardi et al. (Боккарди и др.), «Five Disruptive Technology Directions for 5G»

Апологеты сотовых сетей 5G утверждают, что внедрение этой технологии изменит мир. Но как именно? В статье описаны пять революционных новшеств сетей 5G: устройство-ориентированные архитектуры, волны миллиметрового диапазона, технология MIMO, повышение интеллектуальности устройств и нативная поддержка межкомпьютерной коммуникации.

[1] Куроуз Дж., Росс К. «Компьютерные сети. 2-е изд.». СПб., издательство «Питер».

Hu and Li (Ху и Ли), «Satellite-Based Internet: A Tutorial»

Спутниковый доступ в интернет отличается от использования наземных линий связи. Здесь должны учитываться не только задержки, но и маршрутизация и коммутация. Авторы рассматривают проблемы применения спутниковых систем для интернет-доступа.

Hui (Хуэй), «Introduction to Fiber-Optic Communications»

Название этой книги — «Введение в оптоволоконные коммуникации» — хорошо отражает ее содержание. Здесь есть главы, посвященные оптическим волокнам, источникам света, фотоприемникам, оптическим усилителям, оптическим системам передачи данных и многим другим темам. Книга написана с углублением в технические детали, и для ее полного понимания требуется соответствующий опыт.

Lamparter et al. (Лампартер и др.), «Multi-Gigabit over Copper Access Networks»

Сегодня никто не будет спорить с тем, что лучшим способом высокоскоростной доставки данных в дома пользователей является технология FTTH («оптоволокно в дом»). Однако замена оптоволокном всех имеющихся проводов — весьма дорогостоящее решение. В этой статье авторы рассматривают комбинированные варианты прокладки кабелей, более целесообразные в краткосрочной и среднесрочной перспективе. В частности, предлагается проводить оптоволокно до крупных офисных зданий или многоквартирных жилых домов, а внутри использовать уже существующую проводку и инфраструктуру.

Pearson (Пирсон), «Fiber Optic Communication for Beginners: The Basics»

Если вы хотите быстро узнать, что представляют собой оптоволоконные технологии, то эта маленькая 42-страничная книжка — для вас. Вы узнаете о преимуществах оптоволокна и получите представление о таких вещах, как оптическая электроника, пассивные элементы, моды волокна, кабели, разъемы, сращивание волокон и тестирование.

Stockman and Coomans (Стокман и Куманс), «Fiber to the Tap: Pushing Coaxial Cable Networks to Their Limits»

Авторы этой статьи считают, что сети кабельного телевидения еще не дошли до предела своих возможностей и могут обеспечить скорость передачи данных на уровне нескольких гигабитов в секунду. Они рассматривают различные части кабельной системы и предлагают возможные способы обеспечения таких скоростей. Для полного понимания содержания статьи потребуется некоторое знакомство с технологиями.

### 9.1.3. Канальный уровень

Lin and Costello (Линь и Костелло), «Error Control Coding», 2-е издание

Коды для обнаружения и коррекции ошибок — основа надежных компьютерных сетей. Этот популярный учебник объясняет некоторые из самых важных

кодов, от простых линейных кодов Хэмминга до более сложных, имеющих малую плотность кодов проверки на четность. Автор постарался использовать минимум необходимой алгебры, но ее все равно довольно много.

Kurose and Ross (Куросе и Росс), «Computer Networking»

Глава 6 этой книги посвящена канальному уровню. В ней также есть раздел, посвященный подключению к центрам обработки данных.

Stallings (Сталлингс), «Data and Computer Communications», 10-е издание

Во второй части данной книги рассказывается о цифровой передаче данных и различных каналах, в том числе об обнаружении ошибок, их контроле с повторной передачей и управлении потоком.

## 9.1.4. Подуровень управления доступом к среде

Alloulah and Huang (Аллула и Хуан), «Future Millimeter-Wave Indoor Systems»

В силу все более плотного использования радиочастот в диапазоне 5 ГГц и ниже разработчики средств связи присматриваются к более высоким частотам, чтобы обеспечить менее загруженную полосу пропускания. Теоретически можно использовать диапазон 30–300 ГГц, но на этих частотах радиоволны поглощаются водой (например, дождем), поэтому они больше подходят для использования в помещении. В данной статье рассматриваются некоторые проблемы и области применения стандарта 802.11ad и других систем, использующих волны миллиметрового диапазона.

Bing (Бинг), «Wi-Fi Technologies and Applications»

Стандарт IEEE 802.11 сегодня является общепризнанным стандартом беспроводной связи, и эта книга станет хорошим справочным пособием для тех, кто хочет узнать о нем больше. В ней обсуждаются полосы частот, многоантенные системы и различия между версиями стандарта 802.11. Также рассматриваются такие альтернативные решения, как LTE-U и LAA. Завершает книгу глава о методах модуляции.

Colbach (Кольбах), «Bluetooth Tutorial: Design, Protocol and Specifications for BLE»

Стандарт Bluetooth широко используется для подключения мобильных устройств с использованием радиоволн ближнего радиуса действия. Данная книга предлагает довольно подробное рассмотрение этой технологии, включая ее архитектуру, протоколы и области применения. Рассматриваются все версии стандарта, от Bluetooth 1.0 до Bluetooth 5.

Kasim (Касим), «Delivering Carrier Ethernet»

В настоящее время Ethernet является не только технологией местной связи. Теперь Ethernet используют и в качестве надежного канала передачи на большие расстояния. В книге собраны подробные эссе на эту тему.

Perlman (Перлман), «Interconnections», 2-е издание

Это заслуживающая доверия и в то же время увлекательно написанная книга о мостах, маршрутизаторах и маршрутизации в целом. Автор книги участвовала в разработке алгоритмов, применяемых в мосте связующего дерева в сетях стандарта IEEE 802, и она, несомненно, является одним из ведущих мировых экспертов в различных вопросах сетевых технологий.

Spurgeon and Zimmerman (Сперджен и Циммерман), «Ethernet: The Definitive Guide», 2-е издание

После некоторой вводной информации о кабелях, фреймах, согласовании параметров, технологии PoE и системах сигнализации здесь идут главы, посвященные сетям Ethernet со скоростями 10, 100 и 1000 Мбит/с, а также 10, 40 и 100 Гбит/с. Следующие главы посвящены кабельным соединениям, коммутации, производительности и устранению неполадок. Книга больше ориентирована на практику, чем на теорию.

### 9.1.5. Сетевой уровень

Comer (Комер), «Internetworking with TCP/IP», том 1, 5-е издание

Пятое издание наиболее полного труда о стеке протоколов TCP/IP. Первая половина книги посвящена преимущественно IP и связанным с ним протоколам сетевого уровня. В остальных главах, также заслуживающих внимания, в основном описываются более высокие уровни.

Hallberg (Халльберг), «Quality of Service in Modern Packet Networks»

Подавляющую часть современного веб-трафика составляют мультимедийные данные, что делает крайне актуальным вопрос, касающийся QoS. Эта книга освещает множество тем, включая комплексное и дифференцированное обслуживание, очереди и диспетчеризацию пакетов, предотвращение перегрузок, оценку QoS и многое другое.

Grayson et al. (Грейсон и др.), «IP Design for Mobile Networks»

Телефонные сети и интернет нашли точку соприкосновения: сети мобильной связи реализуются на базе IP. Эта книга посвящена разработке сетей с использованием IP, поддерживающего сервис мобильной телефонной связи.

Nucci and Papagiannaki (Нуччи и Пападжаннаки), «Design, Measurement and Management of Large-Scale IP Networks»

Мы много говорили о том, как сети работают, но не о том, как интернет-провайдеры их проектируют, развертывают и управляют ими. Данная книга восполняет этот пробел: в ней описываются современные методы организации трафика и то, как провайдеры оказывают услуги, используя сети.

Perlman (Перлман), «Interconnections», 2-е издание

В главах с 12-й по 15-ю автор рассматривает многочисленные вопросы разработки алгоритмов одноадресной и групповой рассылки как для WAN, так и для LAN и их реализацию в различных протоколах. Но интереснее всего, безусловно, глава 18 — в ней автор делится своими личными впечатлениями о многолетней работе с сетевыми протоколами. Эта глава информативна и полна юмора; ее следует прочесть всем разработчикам протоколов.

Stevens (Стивенс), «TCP/IP Illustrated», том 1

Главы с 3-й по 10-ю содержат доступное и дополненное примерами описание протокола IP и взаимосвязанных с ним протоколов (ARP, RARP и ICMP).

Feamster et al. (Фимстер и др.), «The Road to SDN»

В этой обзорной статье рассказывается о том, какой ход мыслей в свое время привел к появлению программно-конфигурируемых сетей (SDN), берущих начало в централизованных системах управления телефонными сетями. Упоминается и о том, какие обстоятельства (технические и политические) привели к активному внедрению SDN в конце 2000-х годов.

Swami et al. (Свами и др.), «Software-defined Networking-based DDoS Defense Mechanisms»

SDN взаимодействуют с проблемами безопасности (а точнее, DDoS-атаками) в двух случаях. Во-первых, они сами подвергаются таким атакам. Во-вторых, при DDoS-атаке SDN-системы могут помочь в деле защиты сети. В этой обзорной статье рассматриваются многие работы, посвященные двум вариантам развития событий.

Varghese (Варгезе), «Network Algorithmics»

Мы много говорили о том, как маршрутизаторы и прочие сетевые элементы взаимодействуют между собой. Эта книга другая: в ней рассказывается о реальной разработке маршрутизаторов, способных передавать пакеты на потрясающих скоростях. Ее необходимо прочитать, чтобы разобраться в этом и многих других смежных вопросах. Автор — признанный авторитет в хорошо продуманных алгоритмах, которые применяются на практике для создания высокоскоростных сетевых элементов в программном и аппаратном обеспечении.

## 9.1.6. Транспортный уровень

Comer (Комер), «Internetworking with TCP/IP», том 1, 5-е издание

Как уже говорилось выше, автор написал наиболее полный труд о наборе протоколов TCP/IP. Вторая половина книги посвящена протоколам UDP и TCP.

Pyles et al. (Пайлс и др.), «Guide to TCP/IP: IPv6 and IPv4»

Еще одна книга по TCP, IP и родственным протоколам. В отличие от других работ, здесь довольно много внимания уделяется IPv6; в частности, есть главы, посвященные переходу на использование этой версии протокола.

Stevens (Стивенс), «TCP/IP Illustrated», том 1

Главы с 17-й по 24-ю содержат доступное описание протокола TCP, снабженное примерами.

Feamster and Livingood (Фимстер и Ливингуд), «Internet Speed Measurement: Current Challenges and Future Recommendations»

Авторы обсуждают проблемы измерения скорости интернета по мере того, как сети доступа становятся все более быстрыми. В статье описываются принципы проектирования средств измерения скорости интернета и трудности, с которыми приходится сталкиваться по мере ускорения сетей.

## 9.1.7. Прикладной уровень

Ahsan et al. (Ахсан и др.), «DASHing Towards Hollywood»

Хотя протоколы DASH и HLS используют HTTP для совместимости с интернетом, обе эти технологии основаны на TCP. При этом протокол TCP на первое место ставит надежность доставки данных и соблюдение исходного порядка пакетов, а не скорость передачи. В этой статье показано, как, используя определенный вариант TCP, можно устранить паузы в воспроизведении потокового видео за счет недопущения блокировки очереди.

Berners-Lee et al. (Бернерс-Ли и др.), «The World Wide Web»

Взгляд на историю Всемирной паутины и на ее дальнейшее развитие со стороны человека, который ее придумал, и его коллег по CERN. Статья посвящена архитектуре Всемирной паутины, URL, протоколу HTTP, языку HTML, а также перспективам на будущее. Приводится сравнение с другими распределенными информационными системами.

Chakraborty et al. (Чакраборти и др.), «VoIP Technology: Applications and Challenges»

Старые аналоговые телефонные системы постепенно отмирают и во многих странах уже практически исчезли. На смену им приходит IP-телефония (VoIP). Если вы хотите подробно с ней ознакомиться, обратитесь к этой книге. Помимо прочего, в работе рассматривается принцип действия IP-телефонии, используемые протоколы, проблемы QoS, применение IP-телефонии в беспроводных сетях, проблемы производительности и методы оптимизации, устранение перегрузок и многое другое.

Dizdarevic et al. (Диздаревич и др.), «A Survey of Communication Protocols for Internet of Things...»

Сегодня интернет вещей (IoT) все больше набирает обороты, однако протоколы, посредством которых устройства взаимодействуют с серверами и облаками, носят крайне разрозненный характер. В большинстве случаев эти протоколы работают на прикладном уровне поверх TCP, но их слишком много — к ним относятся REST HTTP, MQTT, CoAP, AMQP, DDS, XMPP и даже HTTP/2.0. Они описаны в данной статье, при этом затрагиваются вопросы производительности, величины задержки, потребления энергии,

безопасности и т. д. В документе также приводится более 130 ссылок на источники.

Goralski (Горальски), «The Illustrated Network: How TCP/IP Works in a Modern Network»

Название этой книги — «Сетевые технологии наглядно: как в современной сети работает TCP/IP» — несколько сбивает с толку. В ней действительно подробно рассматривается протокол TCP, но наряду с ним обсуждается и множество других сетевых протоколов и технологий. В частности, поднимаются следующие темы: протоколы и уровни, TCP/IP, технологии канального уровня, оптические сети, IPv4 и IPv6, ARP, маршрутизация, мультиплексирование, пиринг, BGP, многоадресная рассылка, MPLS, DHCP, DNS, FTP, SMTP, HTTP, SSL и многое другое.

Held (Хелд), «A Practical Guide to Content Delivery Networks», 2-е издание

Книга дает утилитарное представление о работе CDN, уделяя особое внимание практическим соображениям относительно разработки и функционирования эффективной CDN.

Li et al. (Ли и др.), «Two Decades of Internet Video Streaming: A Retrospective View»

Сегодня интернет заполнен потоковыми видео. Большую часть его пропускной способности теперь потребляют Netflix, YouTube и другие стриминговые сервисы. В данной статье рассматривается история и технологические составляющие потоковой передачи видео.

Simpson (Симпсон), «Video Over IP», 2-е издание

Автор рассказывает о том, как может использоваться IP-технология, чтобы передавать видео по сети (как в интернете, так и в разработанных для этого частных сетях). Примечательно, что эта книга ориентирована на профессионалов в сфере видео, интересующихся сетями, а не наоборот.

Wittenberg (Виттенберг), «Understanding Voice Over IP Technology»

В этой книге рассматривается работа IP-телефонии, от передачи аудиоданных с применением IP-протоколов и проблем QoS до SIP и стека протоколов H.323. Материал достаточно подробный и при этом доступный: он разбит на блоки для более легкого восприятия информации.

## 9.1.8. Сетевая безопасность

Anderson (Андерсон), «Making Security Sustainable»

IoT заставляет нас совершенно иначе взглянуть на безопасность. Раньше производитель автомобилей отправлял прототипы новой модели в государственные органы для проверки. В случае одобрения модели он запускал массовое производство и выпускал миллионы ее идентичных копий. Сегодня автомобили подключены к интернету и практически еженедельно получают обновления программного обеспечения, поэтому старый подход не работает.

В статье Андерсон рассматривает эту проблему и множество связанных с ней вопросов безопасности, с которыми нам придется иметь дело в ближайшем будущем.

Anderson (Андерсон), «Security Engineering», 2-е издание

Здесь вы найдете описание причудливых сочетаний методов безопасности, их использования и злоупотребления ими. В техническом плане она сложнее, чем *Secrets and Lies*, но проще, чем *Network Security* (см. ниже). После вводной части даются основы методов защиты информации, затем следуют целые главы, посвященные разным практическим сферам применения, включая банковские системы, системы управления атомной энергетикой, безопасную печать, биометрию, физическую защиту, средства радиоэлектронной войны, защиту в телекоммуникациях, электронную коммерцию и защиту авторских прав.

Fawaz and Shin (Фаваз и Шин), «Security and Privacy in the Internet of Things»

IoT сегодня переживает бурный рост. Скоро десятки миллиардов устройств будут подключены к интернету, включая автомобили, кардиостимуляторы, дверные замки и многое другое. Несмотря на крайнюю важность обеспечения безопасности и конфиденциальности во многих сферах применения IoT, этот вопрос часто упускается из виду. Авторы предлагают решение этой проблемы.

Ferguson et al. (Фергюсон и др.), «Cryptography Engineering»

Много книг посвящено тому, как работают популярные шифровальные алгоритмы. В данной книге рассказывается, как использовать криптографию: почему шифровальные протоколы разработаны именно таким образом и как соединить их в систему, которая удовлетворит вашим целям безопасности. Это довольно компактная книга, и прочесть ее необходимо каждому, кто разрабатывает системы, зависящие от криптографии.

Fridrich (Фридрих), «Steganography in Digital Media»

Стеганография применялась еще в Древней Греции, где с пустых дощечек удаляли воск, записывали секретное сообщение и снова покрывали табличку воском. В настоящее время видео, аудио и другой контент в интернете предоставляют каналы для секретных сообщений. В книге обсуждаются современные методы для сокрытия и обнаружения информации в изображениях.

Kaufman et al. (Кауфман и др.), «Network Security», 2-е издание

Эту заслуживающую доверия и остроумную книгу в первую очередь следует читать, если вас интересует дополнительная техническая информация об алгоритмах и протоколах безопасности сетей. В ней подробно освещаются алгоритмы и протоколы с закрытым и открытым ключом, хеширование сообщений, аутентификация, протокол Kerberos, PKI, IPsec, SSL/TLS и обеспечение безопасности электронной почты. Все темы снабжены примерами. Глава 26, посвященная фольклору на тему защиты информации, — это

настоящий шедевр. В деле обеспечения безопасности важны все детали. Если вы собираетесь разработать действительно полезную систему защиты, эта глава даст вам много важной информации в виде примеров из реальной жизни.

Schneier (Шнайер), «Secrets and Lies»[1]

Прочитав книгу *Cryptography Engineering* от корки до корки, вы станете знатоком криптографических алгоритмов. Если же вы после этого так же внимательно прочтете книгу *Secrets and Lies* (что займет уже гораздо меньше времени), вы поймете, что одними алгоритмами дело не ограничивается. Слабость большинства систем защиты связана не с плохими алгоритмами или слишком короткими ключами, а с пороками в среде, окружающей эти системы. Эта книга является прекрасным нетехническим описанием систем безопасности и рассматривает проблему в самом широком смысле.

Skoudis and Liston (Скудис и Листон), «Counter Hack Reloaded», 2-е издание

Как остановить взломщика? Надо думать так же, как он. В этой книге показан взгляд на сеть с позиции злоумышленника. Автор утверждает, что защита информации должна быть одной из функций всей сетевой системы в целом, она не должна додумываться и прикручиваться в виде специальной технологии к уже существующим сетям. Рассматриваются почти все типы самых распространенных атак, включая социальный инжиниринг, рассчитанный на незнание пользователем систем электронной безопасности.

Ye et al. (Е и др.), «A Survey on Malware Detection Using Data Mining Techniques»

Сегодня вредоносное ПО существует повсеместно, в силу чего на большинстве компьютеров используются программы для защиты от него. Но как разработчики выявляют и классифицируют вредоносные программы? В этой обзорной статье описано состояние дел в сфере разработки вредоносного ПО и систем защиты, а также методы выявления вирусов путем интеллектуального анализа данных.

## 9.2. АЛФАВИТНЫЙ СПИСОК ЛИТЕРАТУРЫ[2]

**ABRAMSON, N. (Абрамсон):** «Internet Access Using VSATs'», *IEEE Commun. Magazine*, vol. 38, pp. 60–68, July 2000.

**ADAR, E., and HUBERMAN, B. A. (Адар и Губерман):** «Free Riding on Gnutella», *First Monday*, Oct. 2000.

**AHMED, A., SHAFIQ, Z., HARKEERAT, B., and KHAKPOUR, A. (Ахмед и др.):** «Suffering from Buffering? Detecting QoE Impairments in Live Video Streams», *Int'l Conf. on Network Protocols*, IEEE, 2017.

[1] Шнайдер Б. «Секреты и ложь. Безопасность данных в цифровом мире». СПб., издательство «Питер».

[2] Согласно английскому алфавиту. — *Примеч. ред.*

**AHSAN, A., MCQUISTIN, S.M., PERKINS, C., and OTT, J. (Ахсан и др.):** «DASHing Towards Hollywood», *Proc. Ninth ACM Multimedia Systems Conf.*, ACM, pp. 1–12, 2018.

**ALLMAN, M., and PAXSON, V. (Оллман и Паксон):** «On Estimating End-to-End Network Path Properties», *Proc. SIGCOMM '99 Conf.*, ACM, pp. 263–274, 1999.

**ALLOULAH, M., and HUANG, H. (Аллула и Хуан):** «Future Millimeter-Wave Indoor Systems: A Blueprint for Joint Communication and Sensing», *IEEE Computer*, vol. 52, pp. 16–24, July 2019.

**ALTAMINI, S., and SHIRMOHAMMADI, S. (Альтамини и Ширмохамади):** «Client-server Cooperative and Fair DASH Video Streaming», *Proc. 29th Workshop on Network and Operating System Support for Digital Audio and Video*, ACM, pp. 1–6, June 2019.

**ANDERSON, C. (Андерсон):** «The Long Tail: Why the Future of Business is Selling Less of More», revised updated ed., New York: Hyperion, 2008a.

**ANDERSON, R. J. (Андерсон):** «Making Security Sustainable», *Commun. of the ACM*, vol. 61, pp. 24–25, March 2018.

**ANDERSON, R. J.:** «Security Engineering: A Guide to Building Dependable Distributed Systems», 2nd ed., New York: John Wiley & Sons, 2008b.

**ANDERSON, R. J.:** «Free Speech Online and Offline», *IEEE Computer*, vol. 25, pp. 28–30, June 2002.

**ANDERSON, R. J.:** «The Eternity Service», *Proc. Pragocrypt Conf.*, CTU Publishing House, pp. 242–252, 1996.

**ANDREWS, J. G., BUZZO, S., CHOI, W., HANLY, S. V., LOZANO, A., SOONG, A. C. K., and ZHANG, J.C. (Эндрюс и др.):** «What Will 5G Be?»,' *IEEE J. on Selected Areas in Commun.*, vol. 32, pp. 1065–1082, June 2014.

**ANTONAKAKIS, M., PERDISCI, R., DAGON, D., LEE, W., and FEAMSTER, N. (Антонакакис и др.):** «Building a Dynamic Reputation System for DNS», *USENIX Security Symposium*, pp. 273–290, 2010.

**APTHORPE, N., HUANG, D., REISMAN D., NARAYANAN, A., and FEAMSTER, N. (Апторп и др.):** «Keeping the Smart Home Private with Smart(er) Traffic Shaping», *Proceedings on Privacy Enhancing Technologies*, pp. 128–48, 2019.

**ASHRAF, Z. (Ашраф):** «Virtual Private Networks in Theory and Practice», Munich: Grin Verlag, 2018.

**ATENCIO, L. (Атенсио):** «The Joy of JavaScript», Shelter Island, NY: Manning Publications, 2020.

**AXELSSON, S. (Аксельсон):** «The Base-rate Fallacy and It's Implications of the Difficulty of Intrusion Detection», *Proc. Conf. on Computer and Commun. Security*, ACM, pp. 1–7, 1999.

**BAASE, S., and HENRY, T. (Баазе и Генри):** «A Gift of Fire: Social, Legal, and Ethical Issues for Computing Technology», 5th ed., Upper Saddle River, NJ: Pearson Education, 2017.

**BALLARDIE, T., FRANCIS, P., and CROWCROFT, J. (Балларди и др.):** «Core Based Trees (CBT)», *Proc. SIGCOMM '93 Conf.*, ACM, pp. 85–95, 1993.

**BARAN, P. (Бэран):** «On Distributed Communications: I. Introduction to Distributed Communication Networks», *Memorandum RM-420-PR*, Rand Corporation, Aug. 1964.

**BASU, S., SUNDARRAJAN, A., GHADERTI, J., SHAKKOTTAI, S., and SITARAMAN, R. (Басу и др.):** «Adaptive TTL-Based Caching for Content Delivery»,' *IEEE/ACM Trans. on Networking*, vol. 26, pp. 1063–1077, June 2018.

**BELLMAN, R. E. (Беллман):** «Dynamic Programming», Princeton, NJ: Princeton University Press, 1957.

**BELLOVIN, S. (Белловин):** «The Security Flag in the IPv4 Header», RFC 3514, Apr. 2003.

**BELSNES, D. (Белснес):** «Flow Control in the Packet Switching Networks», *Commun.* Networks, Uxbridge, England: Online, pp. 349–361, 1975.

**BENNET, C.H., and BRASSARD, G. (Беннет и Брассар):** «Quantum Cryptography: Public Key Distribution and Coin Tossing», *Proc. Int'l Conf. on Computer Systems and Signal Processing*, pp. 175–179, 1984.

**BERESFORD, A., and STAJANO, F. (Бересфорд и Стаяно):** «Location Privacy in Pervasive Computing», *IEEE Pervasive Computing*, vol. 2, pp. 46–55, Jan. 2003.

**BERNAL, P. (Бернал):** «The Internet, Warts and All», Cambridge, U.K.: Cambridge University Press, 2018.

**BERNASCHI, M., CELESTINI, A., GUARINO, S., LOMBARDI, F., and MASTROSTEFANO, E. (Берначи и др.):** «Spiders Like Onions: on the Network of Tor Hidden Services», *Proc. World Wide Web Conf.*, ACM, pp. 105–115, May 2019.

**BERNERS-LEE, T., CAILLIAU, A., LOUTONEN, A., NIELSEN, H. F., and SECRET, A. (Бернерс-Ли и др.):** «The World Wide Web», *Commun. of the ACM*, vol. 37, pp. 76–82, Aug. 1994.

**BERTSEKAS, D., and GALLAGER, R. (Берцекас и Галлагер):** «Data Networks», 2nd ed., Upper Saddle River, NJ: Prentice Hall, 1992.

**BHATTI, S. N., and CROWCROFT, J. (Бхатти и Кроукрофт):** «QoS Sensitive Flows: Issues in IP Packet Handling», *IEEE Internet Computing*, vol. 4, pp. 48–57, July–Aug. 2000.

**BIHAM, E., and SHAMIR, A. (Бихам и Шамир):** «Differential Fault Analysis of Secret Key Cryptosystems», *Proc. 17th Ann. Int'l Cryptology Conf.*, Springer-Verlag LNCS 1294, pp. 513–525, 1997.

**BING, B. (Бинг):** «Wi-Fi Technologies and Applications», Seattle: Amazon, 2017.

**BIRD, R., GOPAL, I., HERZBERG, A., JANSON, P.A., KUTTEN, S., MOLVA, R., and YUNG, M. (Берд и др.):** «Systematic Design of a Family of Attack-Resistant Authentication Protocols», *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 679–693, June 1993.

**BIRRELL, A. D., and NELSON, B. J. (Биррелл и Нельсон):** «Implementing Remote Procedure Calls», *ACM Trans. on Computer Systems*, vol. 2, pp. 39–59, Feb. 1984.

**BIRYUKOV, A., SHAMIR, A., and WAGNER, D. (Бирюков и др.):** «Real Time Cryptanalysis of A5/1 on a PC», *Proc. Seventh Int'l Workshop on Fast Software Encryption*, Springer-Verlag LNCS 1978, pp. 1–8, 2000.

**BISCHOF, Z., BUSTAMANTE, F., and FEAMSTER, N. (Бишоф и др.):** «Characterizing and Improving the Reliability of Broadband Internet Access» (CQ *The 46th Research Conf. on Commun., Information, and Internet Policy (TPRC)*, SSRN, 2018.

**BOCCARDI, F., HEATH, R. W., LOZANO, A., MARZETTA, T. L., and POPOVSKI, P. (Боккарди и др.):** «Five Disruptive Technology Directions for 5G», *IEEE Commun. Magazine*, vol. 52, pp. 74–80, Feb. 2014.

**BOGGS, D., MOGUL, J., and KENT, C. (Боггс и др.):** «Measured Capacity of an Ethernet: Myths and Reality», *Proc. SIGCOMM '88 Conf.*, ACM, pp. 222–234, 1988.

**BORISOV, N., GOLDBERG, I., and WAGNER, D. (Борисов и др.):** «Intercepting Mobile Communications: The Insecurity of 802.11», *Seventh Int'l Conf. on Mobile Computing and Networking*, ACM, pp. 180–188, 2001.

**BOSSHART, P., DALY, D., GIBB, G., IZZARD, M., MCKEOWN, N., REXFORD, J., and WALKER, D. (Босхарт и др.):** «P4: Programming Protocol-Independent Packet Processors», *Computer Commun. Review*, vol. 44, pp. 87–95, Apr., 2014.

**BOSSHART, P., GIBB, G., KIM, H.-S., VARGHESE, G., MCKEOWN, N., IZZARD, M., MUJICA, F., and HOROWITZ, M. (Босхарт и др.):** «Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN», *Computer Commun. Review*, vol. 43, pp. 99–110, Apr., 2013.

**BRADEN, R. (Брейден):** «Requirements for Internet Hosts—Communication Layers», RFC 1122, Oct. 1989.

**BRADEN, R., BORMAN, D., and PARTRIDGE, C. (Брейден и др.):** «Computing the Internet Checksum», RFC 1071, Sept. 1988.

**BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., and SHENKER, S. (Бреслау и др.):** «Web Caching and Zipf-like Distributions: Evidence and Implications», *Proc. INFOCOM Conf.*, IEEE, pp. 126–134, 1999.

**BRONZINO, F., SCHMITT, P., AYOUBI, S., MARTINS, G., TEIXEIRA, R., and FEAMSTER, N. (Бронзино и др.):** «Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience», *ACM SIGMETRICS*, 2020.

**BUSH, V. (Буш):** «As We May Think», *Atlantic Monthly*, vol. 176, pp. 101–108, July 1945.

**CALDER, M., FAN, X., HU, Z., KATZ-BASSETT, E., HEIDEMANN, J. and GOVINDAN, R. (Колдер и др.):** «Mapping the Expansion of Google's Serving Infrastructure», *ACM SIGCOMM Internet Measurement Conf.*, ACM, pp. 313–326, 2013.

**CAPETANAKIS, J.I. (Капетанакис):** «Tree Algorithms for Packet Broadcast Channels», *IEEE Trans. on Information Theory*, vol. IT-5, pp. 505–515, Sept. 1979.

**CASADO, M., FREEDMAN, M.J., PETIT, J., LUO, J., MCKEOWN, N., and SCHENKER, S. (Касадо и др.):** «Ethane: Taking Control of the Enterprise», *Proc. SIGCOMM 2007 Conf.*, ACM, pp. 1–12, 2007.

**CASTAGNOLI, G., BRAUER, S., and HERRMANN, M. (Кастаньоли и др.):** «Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits», *IEEE Trans. on Commun.*, vol. 41, pp. 883–892, June 1993.

**CERF, V., and KAHN, R. (Серф и Кан):** «A Protocol for Packet Network Interconnection», *IEEE Trans. on Commun.*, vol. COM-2, pp. 637–648, May 1974.

**CHAKRABORTY, T., MISRA, S., and PRASAD, R. (Чакраборти и др.):** «VoIP Technology: Applications and Challenges», Berlin: Springer, 2019.

**CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W., WALLACH, D., BURROWS, M., CHANDRA, T., FIKES, A., and GRUBER, R. (Чан и др.):** «Bigtable: A Distributed Storage System for Structured Data», *Proc. OSDI 2006 Symp.*, USENIX, pp. 15–29, 2006.

**CHASE, J.S., GALLATIN, A.J., and YOCUM, K.G. (Чейз и др.):** «End System Optimizations for High-Speed TCP», IEEE *Commun. Magazine*, vol. 39, pp. 68–75, Apr. 2001.

**CHAUDHARY, A, and CHAUBE, M.K. (Чаудхари и Чаубе):** «Hiding MP3 in Colour Image Using Whale Optimization», *Proc. Second Int'l Conf. on Vision, Image, and Signal Processing*, ACM, Art. 54, 2018.

**CHEN, S., and NAHRSTEDT, K. (Чэнь и Нарштедт):** «An Overview of QoS Routing for Next-Generation Networks», *IEEE Network Magazine*, vol. 12, pp. 64–69, Nov./Dec. 1998.

**CHEN, X., FEIBISH, S., KORAL, Y., REXFORD, J., ROTTENSTREICH, O., MONETTI, S., WANG, T. (Чэнь и др.):** «Fine-Grained Queue Measurement in the Data Plane», CoNext, ACM, Dec. 2019.

**CHIU, D., and JAIN, R. (Чиу и Джейн):** «Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks», *Comput. Netw. ISDN Syst.*, vol. 17, pp. 1–4, June 1989.

**CLANCY, T. C., MCGWIER, R. W., and CHEN, L. (Клэнси и др.):** «Post-Quantum Cryptography and 5G Security: A Tutorial», *Proc. WiSec*, ACM, pp. 285–287, 2019.

**CLARK, D. D. (Кларк):** «The Design Philosophy of the DARPA Internet Protocols», *Proc. SIG-COMM '88 Conf.*, ACM, pp. 106–114, 1988.

**CLARK, D. D.:** «Window and Acknowledgement Strategy in TCP», RFC 813, July 1982.

**CLARK, D. D., JACOBSON, V., ROMKEY, J., and SALWEN, H. (Кларк и др.):** «An Analysis of TCP Processing Overhead», *IEEE Commun. Magazine*, vol. 27, pp. 23–29, June 1989.

**CLARK, D. D., SHENKER, S., and ZHANG, L. (Кларк и др.):** «Supporting Real-Time Applications in an Integrated Services Packet Network», *Proc. SIGCOMM '92 Conf.*, ACM, pp. 14–26, 1992.

**CLARKE, A. C. (Кларк):** «Extra-Terrestrial Relays», *Wireless World*, 1945.

**CLARKE, I., MILLER, S. G., HONG, T. W., SANDBERG, O., and WILEY, B. (Кларк и др.):** «Protecting Free Expression Online with Freenet», *IEEE Internet Computing*, vol. 6, pp. 40–49, Jan.–Feb. 2002.

**CODING, M. (Кодинг):** «JavaScript for Beginners», Seattle: Amazon, 2019.

**COHEN, B. (Коэн):** «Incentives Build Robustness in BitTorrent», *Proc. First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

**COLBACH, B. (Кольбах):** «Bluetooth Tutorial: Design, Protocol and Specifications for BLE – Bluetooth Low Energy 4.0 and Bluetooth 5», Seattle: Amazon Kindle, 2019.

**COMER, D. E. (Комер):** The Internet Book, 4th ed., Upper Saddle River, NJ: Prentice Hall, 2007.

**COMER, D. E.:** «Internetworking with TCP/IP», vol. 1, 6th ed., Upper Saddle River, NJ: Pren- tice Hall, 2013.

**CRAVER, S. A., WU, M., LIU, B., STUBBLEFIELD, A., SWARTZLANDER, B., WALLACH, D. W., DEAN, D., and FELTEN, E. W. (Крейвер и др.):** «Reading Between the Lines: Lessons from the SDMI Challenge», *Proc. 10th USENIX Security Symp.*, USENIX, 2001.

**CROCKER, S. D. (Крокер):** «The Arpanet and Its Impact on the State of Networking», *IEEE Computer*, vol. 52, pp-14–23, Oct. 2019.

**CROVELLA, M., and KRISHNAMURTHY, B. (Кровелла и Кришнамурти):** «Internet Measurement», New York: John Wiley & Sons, 2006.

**CUEVAS, R., KRYCZKA, M., GINZALEZ, R., CUEVAS, A., and AZCORRZ, A. (Куэвас):** «Torrent-Guard: Stopping Scam and Malware Distribution in the BitTorrent Ecosystem», *Com puter Networks*, vol. 59, pp. 77–90, 2014.

**DAEMEN, J., and RIJMEN, V. (Дамен и Рэймен):** «The Design of Rijndael», Berlin: Springer-Verlag, 2002.

**DAGON, D., ANTONAKAKIS, M., VIXIE, P., JINMEI, T., and LEE, W. (Дейгон и др.):** «Increased DNS Forgery Resistance Through 0x20-bit Encoding», *Proceedings of the 15th ACM Conf. on Computer and Commun. Security*, ACM, pp. 211–222, 2008.

**DALAL, Y., and METCLFE, R. (Далал и Меткалф):** «Reverse Path Forwarding of Broadcast Packets», *Commun. of the ACM*, vol. 21, pp. 1040–1048, Dec. 1978.

**DAN, K., KITAGAWA, N., SAKURABA, S., and YAMAI, N. (Дэн и др.):** «Spam Domain Detection Method Using Active DNS Data and E-Mail Reception Log», *Proc. 43rd Computer Softw. and Appl. Conf.*, IEEE, pp. 896–899, 2019.

**DAVIE, B., and FARREL, A. (Дейви и Фаррел):** «MPLS: Next Steps», San Francisco: Morgan Kaufmann, 2008.

**DAVIE, B., and REKHTER, Y. (Дейви и Рехтер):** «MPLS Technology and Applications», San Francisco: Morgan Kaufmann, 2000.

**DAVIES, J. (Дэвис):** «Understanding IPv6», 2nd ed., Redmond, WA: Microsoft Press, 2008.

**DAVIS, J. (Дэвис):** «Wifi Technology: Advances and Applications», New York: NY Research Press, 2018.

**DAY, J. D. (Дэй):** «The (Un)Revised OSI Reference Model», *Computer Commun.* Rev., vol. 25, pp. 39–55, Oct. 1995.

**DAY, J. D., and ZIMMERMANN, H. (Дэй и Циммерман):** «The OSI Reference Model», *Proc. of the IEEE*, vol. 71, pp. 1334–1340, Dec. 1983.

**DE ANDRADE, M., MAIER, M., MCGARRY, M., REISSLEIN, M. (Де Андраде и др.):** «Passive optical network (PON) supported networking», *Optical Switching and Networking*, 2014.

**DE MARCO, G., and KOWALSKI, D. (Де Марко и Ковальски):** «Contention Resolution in a Nonsynchronized Multiple Access Channel», *J. of Theoretical Computer Science*, vol. 689, pp. 1–13, Aug. 2017.

**DEAN, J., and GHEMAWAT, S. (Дин и Гемават):** «MapReduce: a Flexible Data Processing Tool», *Commun. of the ACM*, vol. 53, pp. 72–77, Jan. 2008.

**DEERING, S. E. (Диринг):** «SIP: Simple Internet Protocol», *IEEE Network Magazine*, vol. 7, pp. 16–28, May/June 1993.

**DEERING, S. E., and CHERITON, D. (Диринг и Черитон):** «Multicast Routing in Datagram Networks and Extended LANs», *ACM Trans. on Computer Systems*, vol. 8, pp. 85–110, May 1990.

**DEMERS, A., KESHAV, S., and SHENKER, S. (Демерс и др.):** «Analysis and Simulation of a Fair Queueing Algorithm», *Internetwork: Research and Experience*, vol. 1, pp. 3–26, Sept. 1990.

**DENNING, D. E., and SACCO, G. M. (Деннинг и Сакко):** «Timestamps in Key Distribution Protocols», *Commun. of the ACM*, vol. 24, pp. 533–536, Aug. 1981.

**DIFFIE, W., and HELLMAN, M. E. (Диффи и Хеллман):** «Exhaustive Cryptanalysis of the NBS Data Encryption Standard», *IEEE Computer*, vol. 10, pp. 74–84, June 1977.

**DIFFIE, W., and HELLMAN, M. E. (Диффи и Хеллман):** «New Directions in Cryptography», *IEEE Trans. on Information Theory*, vol. IT-2, pp. 644–654, Nov. 1976.

**DIJKSTRA, E. W. (Дейкстра):** «A Note on Two Problems in Connexion with Graphs», *Numer. Math.*, vol. 1, pp. 269–271, Oct. 1959.

**DIZDAREVIC, J., CARPIO, D., JUKAN, A., and MASIP-BRUIN, X. (Диздаревич и др.):** «A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration», *ACM Computing Surveys*, vol. 51, Art. 116, Jan. 2019.

**DONAHOO, M., and CALVERT, K. (Донаху и Калверт):** «TCP/IP Sockets in C», 2nd ed., San Francisco: Morgan Kaufmann, 2009.

**DONAHOO, M., and CALVERT, K. (Донаху и Калверт):** «TCP/IP Sockets in Java», 2nd ed., San Francisco: Morgan Kaufmann, 2008.

**DORFMAN, R. (Дорфман):** «Detection of Defective Members of a Large Population», *Annals Math. Statistics*, vol. 14, pp. 436–440, 1943.

**DU, W. (Ду):** «Computer & Internet Security: A Hands-on Approach», 2nd ed., Seattle: Amazon, 2019.

**DUTCHER, B. (Датчер):** «The NAT Handbook», New York: John Wiley & Sons, 2001.

**EL GAMAL, T. (Эль Гамаль):** «A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms», *IEEE Trans. on Information Theory*, vol. IT-1, pp. 469–472, July 1985.

**ESPOSITO, V. (Эспозито):** «Cryptography for Beginners: a Useful Support for Understanding», Seattle: Amazon Digital Services, 2018.

**FALL, K. (Фолл):** «A Delay-Tolerant Network Architecture for Challenged Internets», *Proc. SIG-COMM 2003 Conf.*, ACM, pp. 27–34, Aug. 2003.

**FAWAZ, K., and SHIN, K. G. (Фаваз и Шин):** «Security and Privacy in the Internet of Things», *IEEE Computer*, vol. 52, pp. 40–49, Apr. 2019.

**FEAMSTER, N., BALAKRISHNAN, H., REXFORD, J., SHAIKH, A., and VAN DER MERWE, J. (Фимстер и др.):** «The Case for Separating Routing from Routers», *Proc. SIGCOMM Workshop on Future Directions in Network Architecture*, ACM, pp. 5–12, 2004.

**FEAMSTER, N., and LIVINGOOD, J. (Фимстер и Ливингуд):** «Internet Speed Measurement: Current Challenges and Future Recommendations», *Commun. of the ACM*, ACM, 2020.

**FEAMSTER, N., REXFORD, J., and ZEGURA, E. (Фимстер и др.):** «The Road to SDN», *ACM Queue*, vol. 11, p. 20, Dec. 2013.

**FENNER, B., HANDLEY, M., HOLBROOK, H., and KOUVELAS, I. (Феннер и др.):** «Protocol Independent Multicast-Sparse Mode (PIM-SM)», RFC 4601, Aug. 2006.

**FERGUSON, N., SCHNEIER, B., and KOHNO, T. (Фергюсон и др.):** «Cryptography Engineering: Design Prin- ciples and Practical Applications», New York: John Wiley & Sons, 2010.

**FLETCHER, J. (Флетчер):** «An Arithmetic Checksum for Serial Transmissions», *IEEE Trans. on Commun.*, vol. COM-0, pp. 247–252, Jan. 1982.

**FLOYD, S., HANDLEY, M., PADHYE, J., and WIDMER, J. (Флойд и др.):** «Equation-Based Congestion Control for Unicast Applications», *Proc. SIGCOMM 2000 Conf.*, ACM, pp. 43–56, Aug. 2000.

**FLOYD, S., and JACOBSON, V. (Флойд и Джейкобсон):** «Random Early Detection for Congestion Avoidance», *IEEE/ACM Trans. on Networking*, vol. 1, pp. 397–413, Aug. 1993.

**FLUHRER, S., MANTIN, I., and SHAMIR, A. (Флюрер и др.):** «Weakness in the Key Scheduling Algorithm of RC4», *Proc. Eighth Ann. Workshop on Selected Areas in Cryptography*, Springer-Verlag LNCS 2259, pp. 1–24, 2001.

**FONTUGNE, R., ABRY, P., FUKUDA, K., VEITCH, D., BORGNAT, P., and WENDT, H. (Фонтюнь и др.):** «Scaling in Internet Traffic: A 14 Year and 3 Day Longitudinal Study, With Multiscale Analyses and Random Projections», *IEEE/ACM Trans. on Networking*, vol. 25, pp. 2152–2165, Aug. 2017.

**FORD, B. (Форд):** «Structured Streams: A New Transport Abstraction», *Proc. SIGCOMM 2007 Conf.*, ACM, pp. 361–372, 2007.

**FORD, L. R., Jr., and FULKERSON, D. R. (Форд и Фалкерсон):** «Flows in Networks», Princeton, NJ: Princeton University Press, 1962.

**FORD, W., and BAUM, M. S. (Форд и Баум):** «Secure Electronic Commerce», Upper Saddle River, NJ: Pren tice Hall, 2000.

**FORNEY, G.D. (Форни):** «The Viterbi Algorithm», *Proc. of the IEEE*, vol. 61, pp. 268–278, Mar. 1973.

**FOSTER, N., HARRISON, R., FREEDMAN, M., MONSANTO, C., REXFORD, J., STORY, A., and WALKER, D. (Фостер и др.):** «Frenetic: A Network Programming Language», *ACM Sigplan Notices*, vol. 46, pp. 279–291, Sep. 2011.

**FRANCIS, P. (Фрэнсис):** «A Near-Term Architecture for Deploying Pip», *IEEE Network Magazine*, vol. 7, pp. 30–37, May/June 1993.

**FRASER, A. G. (Фрейзер):** «Towards a Universal Data Transport System», *IEEE J. on Selected Areas in Commun.*, vol. 5, pp. 803–816, Nov. 1983.

**FRIDRICH, J. (Фридрих):** «Steganography in Digital Media: Principles, Algorithms, and Applications», Cambridge: Cambridge University Press, 2009.

**FULLER, V., and LI, T. (Фуллер и Ли):** «Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan», RFC 4632, Aug. 2006.

**GALLAGER, R.G. (Галлагер):** «A Minimum Delay Routing Algorithm Using Distributed Computation», *IEEE Trans. on Commun.*, vol. COM-5, pp. 73–85, Jan. 1977.

**GALLAGER, R.G.:** «Low-Density Parity Check Codes», *IRE Trans. on Information Theory*, vol. 8, pp. 21–28, Jan. 1962.

**GARCIA-LUNA-ACEVES, J. (Гарсия-Луна-Асевес):** «Carrier-Sense Multiple Access with Collision Avoidance and Detection», *Proc. 20th Int'l Conf. on Modelling, Analysis, and Simulation of Wireless and Mobile Systems*, ACM, pp. 53–61, Nov. 2017.

**GETTYS, J. (Геттис):** «Bufferbloat: Dark Buffers in the Internet», *IEEE Internet Computing*, IEEE, p. 96, 2011.

**GILDER, G. (Гилдер):** «Metcalfe's Law and Legacy», *Forbes ASAP*, Sept. 13, 1993.

**GORALSKI, W. (Горальски):** «The Illustrated Network: How TCP/IP Works in a Modern Network», 2nd ed., San Francisco: Morgan Kaufmann, 2017.

**GRAYSON, M., SHATZKAMER, K., and WAINNER, S. (Грейсон и др.):** «IP Design for Mobile Networks», Indianapolis, IN: Cisco Press, 2009.

**GROBE, K., and EISELT, M. (Гроуб и Эйзелт):** «Wavelength Division Multiplexing: A Practical Engineering Guide», New York: John Wiley & Sons, 2013.

**GROBE, K., and ELBERS, J. (Гроуб и Элберс):** «PON in Adolescence: From TDMA to WDM-PON», *IEEE Commun. Magazine*, vol. 46, pp. 26–34, Jan. 2008.

**GROSS, G., KAYCEE, M., LIN, A., MALIS, A., and STEPHENS, J. (Гросс и др.):** «The PPP Over AAL5», RFC 2364, July 1998.

**GUPTA, A., HARRISON, R., CANINI, M., FEAMSTER, N., REXFORD, J., and WILLINGER, W. (Гупта и др.):** «Sonata: Query-driven Streaming Network Telemetry», *Proc. SIGCOMM 2018 Conf.*, ACM, pp. 357–371, 2018.

**HA, S., RHEE, I., and LISONG, X. (Ха и др.):** «CUBIC: A New TCP-Friendly High-Speed TCP Variant», *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, June 2008.

**HALLBERG, G. (Халльберг):** «Quality of Service in Modern Packet Networks», Seattle: Amazon, 2019.

**HALPERIN, D., HEYDT-BENJAMIN, T., RANSFORD, B., CLARK, S., DEFEND, B., MORGAN, W., FU, K., KOHNO, T., and MAISEL, W. (Халперин и др.):** «Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses», *IEEE Symp. on Security and Privacy*, pp. 129–142, May 2008.

**HALPERIN, D., HU, W., SHETH, A., and WETHERALL, D. (Халперин и др.):** «802.11 with Multiple Antennas for Dummies», *Computer Commun. Rev.*, vol. 40, pp. 19–25, Jan. 2010.

**HAMMING, R.W. (Хэмминг):** «Error Detecting and Error Correcting Codes», *Bell System Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

**HARTE, L. (Харте):** «Introduction to Cable TV (Catv): Systems, Services, Operation, and Technology», Morrisville, NC: DiscoverNet Publishing, 2017.

**HARTE, L, BROMLEY, B, and DAVIS, M. (Харте и др.):** «Introduction to CDMA», Fayetteville, NC: Phoenix Global Support, 2012.

**HARTE, L., KELLOGG, S., DREHER, R., and SCHAFFNIT, T. (Харте и др.):** «The Comprehensive Guide to Wireless Technology», Fuquay-Varina, NC: APDG Publishing, 2000.

**HAWKINS, J. (Хокинс):** Carrier Ethernet, Hanover, MD: Ciena, 2016.

**HAWLEY, G. T. (Хоули):** «Historical Perspectives on the U.S. Telephone Loop», *IEEE Commun. Magazine*, vol. 29, pp. 24–28, Mar. 1991.

**HEGARTY, M. T., and KEANE, A. J. (Хегарти и Кин):** «Steganography, The World of Secret Communications», Amazon CreateSpace, 2018.

**HELD, G. (Хелд):** «A Practical Guide to Content Delivery Networks», 2nd ed., Boca Raton, FL: CRC Press, 2010.

**HEUSSE, M., ROUSSEAU, F., BERGER-SABBATEL, G., DUDA, A. (Хойс и др.):** «Performance Anomaly of 802.11b», *Proc. INFOCOM Conf.*, IEEE, pp. 836–843, 2003.

**HIERTZ, G., DENTENEER, D., STIBOR, L., ZANG, Y., COSTA, X., and WALKE, B. (Херц и др.):** «The IEEE 802.11 Universe», *IEEE Commun. Magazine*, vol. 48, pp. 62–70, Jan. 2010.

**HOE, J. (Хо):** «Improving the Start-up Behavior of a Congestion Control Scheme for TCP», *Proc. SIGCOMM '96 Conf.*, ACM, pp. 270–280, 1996.

**HU, Y., and LI, V.O.K. (**Ху и Ли**):** «Satellite-Based Internet: A Tutorial», *IEEE Commun. Magazine*, vol. 30, pp. 154–162, Mar. 2001.

**HUANG, T. Y., JOHARI, R., MCKEOWN, N., TRUNNELL, M. and WATSON, M. (Хуан и др.):** «A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Ser vice», *Proc. SIGCOMM 2014 Conf.*, ACM, pp. 187–198, 2014.

**HUI, R. (Хуэй):** «Introduction to Fiber-Optic Communications», London: Academic Press, 2020.

**HUITEMA, C. (Уитема):** «Routing in the Internet», 2nd ed., Upper Saddle River, NJ: Prentice Hall, 1999.

**HULL, B., BYCHKOVSKY, V., CHEN, K., GORACZKO, M., MIU, A., SHIH, E., ZHANG, Y., BALAKRISHNAN, H., and MADDEN, S. (Халл и др.):** «CarTel: A Distributed Mobile Sensor Com puting System», *Proc. Sensys 2006 Conf.*, ACM, pp. 125–138, Nov. 2006.

**HUSTON, G. (Хьюстон):** «The Death of Transit and Beyond», 2018.

**IRMER, T. (Ирмер):** «Shaping Future Telecommunications: The Challenge of Global Standardiza tion», *IEEE Commun. Magazine*, vol. 32, pp. 20–28, Jan. 1994.

**JACOBSON, V. (Джейкобсон):** «Compressing TCP/IP Headers for Low-Speed Serial Links», RFC 1144, Feb. 1990.

**JACOBSON, V. (Джейкобсон):** «Congestion Avoidance and Control», *Proc. SIGCOMM '88 Conf.*, ACM, pp. 314–329, 1988.

**JUANG, P., OKI, H., WANG, Y., MARTONOSI, M., PEH, L., and RUBENSTEIN, D. (Цзюань и др.):** «Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet», *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 96–107, Oct. 2002.

**KAMOUN, F., and KLEINROCK, L. (Камоун и Клейнрок):** «Stochastic Performance Evaluation of Hierarchical Routing for Large Networks», *Computer Networks*, vol. 3, pp. 337–353, Nov. 1979.

**KARAGIANNIS, V., VENITO, A., COELHO, R., BORKOWSKI, M, and FOHLER, G. (Карагианнис и др.):** «Edge Computing with Peer to Peer Interactions: Use Cases and Impact», *Proc. Workshop on Fog Computing and the IoT*, ACM, pp. 46–50, Apr. 2019.

**KARN, P. (Карн):** «MACA—A New Channel Access Protocol for Packet Radio», *ARRL/CRRL Amateur Radio Ninth Computer Networking Conf.*, pp. 134–140, 1990.

**KARN, P. and PARTRIDGE, C. (Карн и Партридж):** «Improving Round-Trip Time Estimates in Reliable Transport Protocols», *ACM SIGCOMM Computer Commun. Review*, ACM, pp. 2–7, 1987.

**KASIM, A. (Касим):** «Delivering Carrier Ethernet: Extending Ethernet Beyond the LAN», New York: McGraw-Hill, 2008.

**KATABI, D., HANDLEY, M., and ROHRS, C. (Катаби и др.):** «Congestion Control for High Bandwidth- Delay Product Networks», *Proc. SIGCOMM 2002 Conf.*, ACM, pp. 89–102, 2002.

**KATZ, D., and FORD, P.S. (Кац и Форд):** «TUBA: Replacing IP with CLNP», *IEEE Network Magazine*, vol. 7, pp. 38–47, May/June 1993.

**KAUFMAN, C., PERLMAN, R., and SPECINER, M. (Кауфман и др.):** «Network Security», Upper Saddle River, NJ: Prentice Hall, 2002.

**KENT, C., and MOGUL, J. (Кент и Могул):** «Fragmentation Considered Harmful», *Proc. SIGCOMM '87 Conf.*, ACM, pp. 390–401, 1987.

**KHANNA, A. and ZINKY, J. (Кханна и Зинки):** «The Revised ARPANET Routing Metric», *Proc. SIGCOMM '89 Conf.*, ACM, pp. 45–56, 1989.

**KIM, H., REICH, J., GUPTA, A., SHAHBAZ, M., FEAMSTER, N. and CLARK, R. (Ким и др.):** «Kinetic: Verifiable Dynamic Network Control», *12th USENIX Sym. on Networked Systems Design and Implementation*, ACM, pp. 59–72, 2015.

**KINNEAR, E., MCMANUS, P., and WOOD, C. (Киннир и др.):** «Oblivious DNS over HTTPS», IETF Network Working Group Internet Draft, 2019.

**KLEINROCK, L. (Клейнрок):** «Power and Other Deterministic Rules of Thumb for Probabalistic Problems in Computer Communications», *Proc. Int'l Conf. on Commun.*, pp. 43.1.1– 43.1.10, 1979.

**KLEINROCK, L., and TOBAGI, F. (Клейнрок и Тобаги):** «Random Access Techniques for Data Transmission over Packet-Switched Radio Channels», *Proc. Nat. Computer Conf.*, pp. 187–201, 1975.

**KOHLER, E., HANDLEY, H., and FLOYD, S. (Колер и др.):** «Designing DCCP: Congestion Control without Reliability», *Proc. SIGCOMM 2006 Conf.*, ACM, pp. 27–38, 2006.

**KOOPMAN, P. (Купман):** «32-Bit Cyclic Redundancy Codes for Internet Applications», *Proc. Intl. Conf. on Dependable Systems and Networks.*, IEEE, pp. 459–472, 2002.

**KRAFT, J, and WASHINGTON, L. (Крафт и Вашингтон):** «An Introduction to Number Theory with Cryptography», 2nd ed., London: Chapman and Hall, 2018.

**KUMAR, R. (Кумар):** «All about Steganography and Detection of Stegano Images», Riga, Latvia: Lap Lambert Academic Publishing, 2018.

**KUROSE, J., and ROSS, K. (Курose и Росс):** «Computer Networking: A Top-Down Approach», 7th ed. Upper Saddle River, NJ: Pearson, 2016.

**KUSZYK, A., and HAMMOUDEH, M. (Кушик и Хаммудех):** «Contemporary Alternatives to Traditional Processor Design in the Post Moore's Law Era», *Proc. Second Int'l Conf. on Future Networks and Distributed Systems*, ACM, Art. 46, 2018.

**LABOVITZ, C., AHUJA, A., BOSE, A., and JAHANIAN, F. (Лейбовиц и др.):** «Delayed Internet Routing Convergence», *IEEE/ACM Trans. on Networking*, vol. 9, pp. 293–306, June 2001.

**LAINO, J. (Лайно):** «The Telecom Handbook», New York: CMP Books, 2017.

**LAM, C. K. M., and TAN, B. C. Y. (Лам и Тан):** «The Internet Is Changing the Music Industry», *Commun. of the ACM*, vol. 44, pp. 62–66, Aug. 2001.

**LAMPARTER, O., FANG, L., BISCHOFF, J.-C., REITMANN, M., SCHWENDENER, R., ZASOWSKI, T. (Лампартер и др.):** «Multi-Gigabit over Copper Access Networks: Architectural Evolution and Techno-Economic Analysis», *IEEE Commun. Magazine*, vol. 57, pp. 22–27, Aug. 2019.

**LE FEUVRE, J., CONCOLATO, C., BOUZAKARIA, N., and NGUYEN, V. (Ле Февр и др.):** «MPEG-DASH for Low Latency and Hybrid Streaming Services», *Proc. 23rd Int'l conf. on Multimedia*, ACM, pp. 751–752, June 2015.

**LEMON, J. (Лемон):** «Resisting SYN Flood DOS Attacks with a SYN Cache», *Proc. BSDCon Conf.*, USENIX, pp. 88–98, 2002.

**LEVY, S. (Леви):** «Crypto Rebels», *Wired*, pp. 54–61, May/June 1993.

**LI, B., WANG, Z., LIU, J., and ZHU, W. (Ли и др.):** «Two Decades of Internet Video Streaming: A Retrospective View», *ACM Trans. on Multimedia Computing*, vol. 9, Art. 33, Oct. 2013.

**LI, M., AGRAWAL, D., GANESAN, D., and VENKATARAMANI, A. (Ли и др.):** «Block-Switched Networks: A New Paradigm for Wireless Transport», *Proc. NSDI 2009 Conf.*, USENIX, pp. 423–436, 2009.

**LI, Z., LEVIN, D., SPRING, N., and BHATTACHARJEE, B. (Ли и др.):** «Internet Anycast: Performance, Problems, and Potential», *Proc. SIGCOMM 2018 Conf.*, pp. 59–73, Aug. 2018.

**LIN, S., and COSTELLO, D. (Линь и Костелло):** «Error Control Coding», 2nd ed., Upper Saddle River, NJ: Pearson Education, 2004.

**LIOGKAS, N., NELSON, R., KOHIER, E., ZHANG, L.** Exploiting BitTorrent for fun (but not profit), 2006.

**LUBACZ, J., MAZURCZYK, W., and SZCZYPIORSKI, K. (Любач и др.):** «Vice over IP», *IEEE Spectrum*, pp. 42–47, Feb. 2010.

**MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S. and TURNER, J. (Маккиоун и др.):** «OpenFlow: Enabling Innovation in Campus Networks», *Computer Commun. Review*, vol. 38, pp. 69–74, Apr. 2008.

**MACEDONIA, M. R. (Македония):** «Distributed File Sharing», *IEEE Computer*, vol. 33, pp. 99–101, 2000.

**MALIS, A., and SIMPSON, W. (Малис и Симпсон):** «PPP over SONET/SDH», RFC 2615, June 1999.

**MANGLA, T., HALEPOVIC, E., AMMAR, M. and ZEGURA, E. (Мангла и др.):** «eMIMIC: Estimating HTTP-Based Video QoE Metrics from Encrypted Network Traffic», *Network Traffic Measurement and Analysis Conf.*, IEEE, pp. 1–8, 2018.

**MASSEY, J. L. (Мэсси):** «Shift-Register Synthesis and BCH Decoding», *IEEE Trans. on Information Theory*, vol. IT-5, pp. 122–127, Jan. 1969.

**MATSUI, M. (Мацуи):** «Linear Cryptanalysis Method for DES Cipher», *Advances in Cryptology – Eurocrypt 1993 Proceedings*, Springer-Verlag LNCS 765, pp. 386–397, 1994.

**MAZIERES, D., and KAASHOEK, M. F. (Мазьер и Каашук):** «The Design, Implementation, and Operation of an Email Pseudonym Server», *Proc. Fifth Conf. on Computer and Commun. Security*, ACM, pp. 27–36, 1998.

**MCCULLOUGH, B. (Маккалоу):** «How the Internet Happened: From Netscape to the iPhone», New York: Liveright, 2018.

**MENASCHE, D. S., ROCHA, D. A., ANTONIO, A., LI, B., TOWSLEY, D. and VENKATARAMANI, A. (Менаше и др.):** «Content Availability and Bundling in Swarming Systems», *IEEE/ACM Trans. on Networking*, IEEE, pp. 580–593, 2013.

**MENEZES, A. J., and VANSTONE, S. A. (Менезес и Ванстоун):** «Elliptic Curve Cryptosystems and Their Implementation», *Journal of Cryptology*, vol. 6, pp. 209–224, 1993.

**MERKLE, R. C., and HELLMAN, M. (Меркл и Хеллман):** «Hiding and Signatures in Trapdoor Knapsacks», *IEEE Trans. on Information Theory*, vol. IT-4, pp. 525–530, Sept. 1978.

**METCALFE, R.M. (Меткалф):** «Metcalfe's Law after 40 Years of Ethernet», *IEEE Computer*, vol. 46, pp. 26–31, 2013.

**METCALFE, R. M. (Меткалф):** «Computer/Network Interface Design: Lessons from Arpanet and Ethernet», *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 173–179, Feb. 1993.

**METCALFE, R. M., and BOGGS, D. R. (Меткалф и Богтс):** «Ethernet: Distributed Packet Switching for Local Computer Networks», *Commun. of the ACM*, vol. 19, pp. 395–404, July 1976.

**METZ, C. (Мец):** «Interconnecting ISP Networks», *IEEE Internet Computing*, vol. 5, pp. 74–80, Mar.–Apr. 2001.

**MISHRA, P.P., KANAKIA, H., and TRIPATHI, S. (Мишра и др.):** «On Hop by Hop Rate-Based Congestion Control», *IEEE/ACM Trans. on Networking*, vol. 4, pp. 224–239, Apr. 1996.

**MITRA, J., and NAYAK, T. (Митра и Найек):** «Reconfigurable Very High Throughput Low Latency VLSI (FPGA Design Architecture of CRC 32», *Integration*, vol. 56, pp. 1–14, Jan. 2017.

**MOGUL, J. (Могул):** «IP Network Performance», in *Internet System Handbook*, D.C. Lynch and M.Y. Rose (eds.), Boston: Addison-Wesley, pp. 575–575, 1993.

**MOGUL, J., and DEERING, S. (Могул и Диринг):** «Path MTU Discovery», RFC 1191, Nov. 1990.

**MOGUL, J., and MINSHALL, G. (Могул и Миншалл):** «Rethinking the Nagle Algorithm», *Comput. Commun. Rev.*, vol. 31, pp. 6–20, Jan. 2001.

**MOY, J. (Мой):** «Multicast Routing Extensions for OSPF», *Commun. of the ACM*, vol. 37, pp. 61–66, Aug. 1994.

**MUYCO, S. D., and HERNANDEZ, A. A. (Муйко и Эрнандес):** «Least Significant Bit Hash Algorithm for Digital Image Watermarking Authentication», *Proc. Fifth Int'l Conf. on Computing and Art. Intell*, ACM, pp. 150–154, 2019.

**NAGLE, J. (Нейгл):** «On Packet Switches with Infinite Storage», *IEEE Trans. on Commun.*, vol. COM-5, pp. 435–438, Apr. 1987.

**NAGLE, J.:** «Congestion Control in TCP/IP Internetworks», *Computer Commun.* Rev., vol. 14, pp. 11–17, Oct. 1984.

**NAUGHTON, J. (Нотон):** «A Brief History of the Future», Woodstock, NY: Overlook Press, 2000.

**NEEDHAM, R. M., and SCHROEDER, M. D. (Нидхем и Шредер):** «Authentication Revisited», *Operating Systems Rev.*, vol. 21, p. 7, Jan. 1987.

**NEEDHAM, R. M., and SCHROEDER, M. D. (Нидхем и Шредер):** «Using Encryption for Authentication in Large Networks of Computers», *Commun. of the ACM*, vol. 21, pp. 993–999, Dec. 1978.

**NELAKUDITI, S., and ZHANG, Z.-L. (Нелакудити и Чжан):** «A Localized Adaptive Proportioning Approach to QoS Routing», *IEEE Commun. Magazine*, vol. 40, pp. 66–71, June 2002.

**NIST:** «Secure Hash Algorithm», U.S. Government Federal Information Processing Standard 180, 1993.

**NORTON, W. B. (Нортон):** «The Internet Peering Playbook: Connecting to the Core of the Internet», DrPeering Press, 2011.

**NUCCI, A., and PAPAGIANNAKI, D. (Нуччи и Пападжаннаки):** «Design, Measurement and Management of Large-Scale IP Networks», Cambridge: Cambridge University Press, 2008.

**NUGENT, R., MUNAKANA, R., CHIN, A., COELHO, R., and PUIG-SUARI, J. (Ньюджент и др.):** «The Cube-Sat: The PicoSatellite Standard for Research and Education», *Proc. SPACE 2008 Conf.*, AIAA, 2008.

**OLEJNIK, L., CASTELLUCIA, C., and DIAZ, C. (Олейник и др.):** «The Leaking Battery», *Data Privacy Management and Security Assurance* Springer, pp. 254–263.

**ORAN, D. (Оран):** «OSI IS-IS Intra-domain Routing Protocol», RFC 1142, Feb. 1990.

**OSTERHAGE, W. (Остерхаге):** «Wireless Network Security», 2nd ed., Boca Raton, FL: CRC Press, 2018.

**OTWAY, D., and REES, O. (Отуэй и Рис):** «Efficient and Timely Mutual Authentication», *Operating Systems Rev.*, pp. 8–10, Jan. 1987.

**PADHYE, J., FIROIU, V., TOWSLEY, D., and KUROSE, J. (Падхай и др.):** «Modeling TCP Throughput: A Simple Model and Its Empirical Validation», *Proc. SIGCOMM '98 Conf.*, ACM, pp. 303–314, 1998.

**PALMER, M., KRUGER, T., CHANDRASEKARAN, N., and FELDMANN, A. (Палмер и др.):** «The QUIC Fix for Optimal Video Streaming», *Proc. Workshop on Evolution, Performance, and Interoperability of QUIC*, ACM, pp. 43–49, Dec. 2018.

**PARAMESWARAN, M., SUSARLA, A., and WHINSTON, A. B. (Парамешваран и др.):** «P2P Networking: An Information-Sharing Alternative», *IEEE Computer*, vol. 34, pp. 31–38, July 2001.

**PAREKH, A., and GALLAGER, R. (Парех и Галлагер):** «A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple-Node Case», *IEEE/ACM Trans. on Networking*, vol. 2, pp. 137–150, Apr. 1994.

**PAREKH, A., and GALLAGER, R.:** «A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case», *IEEE/ACM Trans. on Networking*, vol. 1, pp. 344–357, June 1993.

**PARTRIDGE, C., HUGHES, J., and STONE, J. (Партридж и др.):** «Performance of Checksums and CRCs over Real Data», *Proc. SIGCOMM '95 Conf.*, ACM, pp. 68–76, 1995.

**PARTRIDGE, C., MENDEZ, T., and MILLIKEN, W. (Партридж и др.):** «Host Anycasting Service», RFC 1546, Nov. 1993.

**PATIL, P., BUBANE, V., and PANDHARE, N. (Патил и др.):** «Audio Steganography», Riga, Latvia: Lap Lambert Academic Publishing, 2019.

**PAXSON, V., and FLOYD, S. (Паксон и Флойд):** «Wide-Area Traffic: The Failure of Poisson Modeling», *IEEE/ACM Trans. on Networking*, vol. 3, pp. 226–244, June 1995.

**PEARSON, E. (Пирсон):** «Fiber Optic Communications For Beginners: The Basics», Fiber Optic Assoc., 2015.

**PERKINS, C. E. (Перкинс):** «RTP: Audio and Video for the Internet», Boston: Addison-Wesley, 2003.

**PERKINS, C. E.:** «IP Mobility Support for IPv4», RFC 3344, Aug. 2002.

**PERKINS, C. E. (ed.):** «Ad Hoc Networking», Boston: Addison-Wesley, 2001.

**PERKINS, C. E.:** «Mobile IP Design Principles and Practices», Upper Saddle River, NJ: Prentice Hall, 1998.

**PERKINS, C. E., and ROYER, E. (Перкинс и Ройер):** «The Ad Hoc On-Demand Distance-Vector Protocol», in *Ad Hoc Networking*, edited by C. Perkins, Boston: Addison-Wesley, 2001.

**PERLMAN, R. (Перлман):** «Interconnections», 2nd ed., Boston: Addison-Wesley, 2000.

**PERLMAN, R.:** «Network Layer Protocols with Byzantine Robustness», Ph.D. thesis, M.I.T., 1988.

**PERLMAN, R.:** «An Algorithm for the Distributed Computation of a Spanning Tree in an Extended LAN», *Proc. SIGCOMM '85 Conf.*, ACM, pp. 44–53, 1985.

**PERLMAN, R., and KAUFMAN, C. (Перлман и Кауфман):** «Key Exchange in IPsec», *IEEE Internet Computing*, vol. 4, pp. 50–56, Nov.-Dec. 2000.

**PERROS, H. G. (Перрос):** «Connection-Oriented Networks: SONET/SDH, ATM, MPLS and Optical Networks», New York: John Wiley & Sons, 2005.

**PETERSON, L., ANDERSON, T., KATTI, S., MCKEOWN, N. PARULKAR, G., REXFORD, J., SATYANARAYANAN, M., SUNAY, O. and VAHDAT, A. (Петерсон и др.):** «Democratizing the Network Edge», *Computer Commun. Review*, vol. 49, pp. 31–36, Apr. 2019.

**PETERSON, W. W., and BROWN, D. T. (Петерсон и Браун):** «Cyclic Codes for Error Detection», *Proc. IRE*, vol. 49, pp. 228–235, Jan. 1961.

**PIATEK, M., ISDAL, T., ANDERSON, T., KRISHNAMURTHY, A., and VENKATARAMANI, V. (Пиатек и др.):** «Do Incentives Build Robustness in BitTorrent?», *Proc. NSDI 2007 Conf.*, USENIX, pp. 1–14, 2007.

**PIATEK, M., KOHNO, T., and KRISHNAMURTHY, A. (Пиатек и др.):** «Challenges and Directions for Monitoring P2P File Sharing Networks — or Why My Printer Received a DMCA Takedown Notice», *Third Workshop on Hot Topics in Security*, USENIX, July 2008.

**POSTEL, J. (Постел):** «Internet Control Message Protocols», RFC 792, Sept. 1981.

**PYLES, J., CARRELL, J. L., and TITTEL, E. (Пайлс и др.):** «Guide to TCP/IP: IPv6 and IPv4», 5th ed., Boston: Cengage Learning, 2017.

**QUINLAN, J., and SREENAN, C. (Куинлан и Сринан):** «Multi-profile Ultra High Definition (UHD) AVC and HEVC 4K DASH Datasets», *Proc. Ninth Multimedia Systems Conf.*, ACM, pp. 375–380, June 2018.

**RABIN, J., and MCCATHIENEVILE, C. (Рабин и МакКэтиНевил):** «Mobile Web Best Practices 1.0», W3C Recommendation, July 2008.

**RAMACHANDRAN, A., DAS SARMA, A., FEAMSTER, N. (Рамачандран и др.):** «Bit Store: An Incentive-Compatabile Solution for Blocked Downloads in BitTorrent», *Proc. Joint Workshop on Econ. Networked Syst. and Incentive-Based Computing*, 2007.

**RAMACHANDRAN, S., GRYYA, T., DAPENA, K., and THOMAS, P. (Рамачандран и др.):** «The Truth about Faster Internet: It's Not Worth It», *The Wall Street Journal*, p. A1, 2019.

**RAMAKRISHNAN, K. K., FLOYD, S., and BLACK, D. (Рамакришнан и др.):** «The Addition of Explicit Congestion Notification (ECN) to IP», RFC 3168, Sept. 2001.

**RAMAKRISHNAN, K. K., and JAIN, R. (Рамакришнан и Джейн):** «A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer», *Proc. SIG-COMM '88 Conf.*, ACM, pp. 303–313, 1988.

**RIBEZZO, G., SAMELA, G., PALMISANO, V., DE CICCO, L., and MASCOLO, S. (Рибеццо и др.):** «A DASH Video Streaming for Immersive Contents», *Proc. Ninth Multimedia Systems Conf.*, ACM, pp. 525–528, June 2018.

**RIVEST, R. L. (Ривест):** «The MD5 Message-Digest Algorithm», RFC 1320, Apr. 1992.

**RIVEST, R. L., SHAMIR, A., and ADLEMAN, L. (Ривест и др.):** «On a Method for Obtaining Digital Signatures and Public Key Cryptosystems», *Commun. of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

**ROBERTS, L. G. (Робертс):** «Extensions of Packet Communication Technology to a Hand Held Personal Terminal», *Proc. Spring Joint Computer Conf.*, AFIPS, pp. 295–298, 1972.

**ROBERTS, L. G.:** «Multiple Computer Networks and Intercomputer Communication», *Proc. First Symp. on Operating Systems Prin.*, ACM, pp. 3.1–3.6, 1967.

**ROSE, M. T. (Роуз):** «The Simple Book», Upper Saddle River, NJ: Prentice Hall, 1994.

**ROSE, M. T.:** «The Internet Message», Upper Saddle River, NJ: Prentice Hall, 1993.

**RUIZ-SANCHEZ, M. A., BIERSACK, E. W., and DABBOUS, W. (Руис-Санчес и др.):** «Survey and Taxonomy of IP Address Lookup Algorithms», *IEEE Network Magazine*, vol. 15, pp. 8–23, Mar.-Apr. 2001.

**SALTZER, J. H., REED, D. P., and CLARK, D. D. (Зальцер и др.):** «End-to-End Arguments in System Design», *ACM Trans. on Computer Systems*, vol. 2, pp. 277–288, Nov. 1984.

**SANTOS, F. R., DA COSTA CORDEIRO, W. L., GASPARY, L. P., and BARCELLOS, M. P. (Сантос и др.):** «Funnel: Choking Polluters in BitTorrent File Sharing Communities», *IEEE Trans. on Network and Service Management*, vol. 8, pp. 310–321, April 2011.

**SAROIU, S., GUMMADI, K., and GRIBBLE, S. (Сарою и др.):** «Measuring and Analyzing the Characteristics of Napster & Gnutella Hosts», *Multim. Syst.*, vol. 9, pp. 170–184, Aug. 2003.

**SCHMITT, P., EDMUNDSON, A., MANKIN, A. and FEAMSTER, N. (Шмитт и др.):** «Oblivious DNS: Practical Privacy for DNS Queries», Proc. *on Privacy Enhancing Technologies*, pp. 228–244, 2019.

**SCHNEIER, B. (Шнайер):** «Secrets and Lies», New York: John Wiley & Sons, 2004.

**SCHNORR, C. P. (Шнорр):** «Efficient Signature Generation for Smart Cards», *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.

**SCHWARTZ, M., and ABRAMSON, N. (Шварц и Абрамсон):** «The AlohaNet: Surfing for Wireless Data», *IEEE Commun. Magazine*, vol. 47, pp. 21–25, Dec. 2009.

**SENN, J. A. (Сенн):** «The Emergence of M-Commerce», *IEEE Computer*, vol. 33, pp. 148–150, Dec. 2000.

**SEVERANCE, C. R. (Северанс):** «Introduction to Networking: How the Internet Works», Amazon Cre ateSpace, 2015.

**SHAIKH, A., REXFORD, J., and SHIN, K. (Шейх и др.):** «Load-Sensitive Routing of Long-Lived IP Flows», *Proc. SIGCOMM '99 Conf.*, ACM, pp. 215–226, Sept. 1999.

**SHALUNOV, S., and CARLSON, R. (Шалунов и Карлсон):** «Detecting Duplex Mismatch on Ethernet», *Passive and Active Network Measurement*, Springer-Verlag LNCS 3431, pp. 3135–3148, 2005.

**SHANNON, C. (Шеннон):** «A Mathematical Theory of Communication», *Bell System Tech. J.*, vol. 27, pp. 379–423, July 1948; and pp. 623–656, Oct. 1948.

**SHREEDHAR, M., and VARGHESE, G. (Шридхар и Варгезе):** «Efficient Fair Queueing Using Deficit Round Robin», *Proc. SIGCOMM '95 Conf.*, ACM, pp. 231–243, 1995.

**SIGANOS, G., FALOUTSOS, M., FALOUTSOS, P., and FALOUTSOS, C. (Сиганос и др.):** «Power Laws and the AS-level Internet Topology», *IEEE/ACM Trans. on Networking*, vol. 11, pp. 514–524, Aug. 2003.

**SIMPSON, W. (Симпсон):** «Video Over IP», 2nd ed., Burlington, MA: Focal Press, 2008.

**SIMPSON, W.:** «The Point-to-Point Protocol (PPP)», RFC 1661, July 1994a.

**SIMPSON, W.:** «PPP in HDLC-like Framing», RFC 1662, July 1994b.

**SIU, K., and JAIN, R. (Сиу и Джейн):** «A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic», *Computer Commun. Review*, vol. 25, pp. 6–20, Apr. 1995.

**SKOUDIS, E., and LISTON, T. (Скудис и Листон):** «Counter Hack Reloaded», 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2006.

**SMITH, D. K., and ALEXANDER, R. C. (Смит и Александер):** «Fumbling the Future», New York: William Morrow, 1988.

**SOOD, K. (Суд):** «Kerberos Authentication Protocol: Cryptography and Network Security», Riga, Latvia: Lap Lambert Academic Publishing, 2012.

**SOTIROV, A., STEVENS, M., APPELBAUM, J., LENSTRA, A., MOLNAR, D., OSVIK, D., and DE WEGER, B. (Сотиров и др.):** «MD5 Considered Harmful Today», *Proc. 25th Chaos Commun. Congress*, Verlag Art d'Ameublement, 2008.

**SOUTHEY, R. (Саути):** «The Doctors», London: Longman, Brown, Green and Longmans, 1848.

**SPURGEON, C., and ZIMMERMAN, A. (Сперджен и Циммерман):** «Ethernet: The Definitive Guide», 2nd ed., Sebastapol, CA: O'Reilly, 2014.

**STALLINGS, W. (Сталлингс):** «Data and Computer Commun», 10th ed., Upper Saddle River, NJ: Pearson Education, 2013.

**STAPLETON, J., and EPSTEIN, W. C. (Стэплтон и Эпштейн):** «Security without Obscurity: A Guide to PKI Operations», Boca Raton, FL: CRC Press, 2016.

**STEVENS, W. R. (Стивенс):** «TCP/IP Illustrated: The Protocols», Boston: Addison Wesley, 1994.

**STEVENS, W. R., FENNER, B., and RUDOFF, A. M. (Стивенс и др.):** «UNIX Network Programming: The Sockets Network API», Boston: Addison-Wesley, 2004.

**STOCKMAN, G.-J., and COOMANS, W. (Стокман и Куманс):** «Fiber to the Tap: Pushing Coaxial Cable Networks to Their Limits», *IEEE Commun. Magazine*, vol. 57, pp. 34–39, Aug. 2019.

**STUBBLEFIELD, A., IOANNIDIS, J., and RUBIN, A. D. (Стабблфилд и др.):** «Using the Fluhrer, Mantin, and Shamir Attack to Break WEP», *Proc. Network and Distributed Systems Security Symp.*, ISOC, pp. 1–11, 2002.

**STUTTARD, D., and PINTO, M. (Статтард и Пинто):** «The Web Application Hacker's Handbook», New York: John Wiley & Sons, 2007.

**SU, S. (Су):** «The UMTS Air Interface in RF Engineering», New York: McGraw-Hill, 2007.

**SUN, L., MKWAWA, I. H., JAMMEH, E., and IFEACHOR, E. (Сунь и др.):** «Guide to Voice and Video over IP: For Fixed and Mobile Networks», Berlin: Springer, 2015.

**SUNDARESAN, S., De DONATO, W., FEAMSTER, N., TEIXEIRA, R., CRAWFORD, S. and PESCAPE, A. (Сундаресан и др.):** «Broadband Internet Performance: A View from the Gateway», *Proc. SIGCOMM 2011 Conf.*, ACM, pp. 134–145, 2011.

**SUNSHINE, C. A., and DALAL, Y. K. (Саншайн и Далал):** «Connection Management in Transport Protocols», *Computer Networks*, vol. 2, pp. 454–473, 1978.

**SWAMI, R., DAVE, M., and RANGA, V. (Свами и др.):** «Software-defined Networking-based DDoS Defense Mechanisms», *ACM Computing Surveys*, vol. 52, Art. 28, April 2019.

**TAN, K., SONG, J., ZHANG, Q., and SRIDHARN, M. (Тань и др.):** «A Compound TCP Approach for High-Speed and Long Distance Networks», *Proc. INFOCOM Conf.*, IEEE, pp. 1–12, 2006.

**TANENBAUM, A. S., and BOS, H. (Таненбаум и Бос):** «Modern Operating Systems», 4th ed., Upper Saddle River, NJ: Prentice Hall, 2015.

**TOMLINSON, R. S. (Томлинсон):** «Selecting Sequence Numbers», Proc. *SIGCOMM/SIGOPS Interprocess Commun. Workshop*, ACM, pp. 11–23, 1975.

**TUCHMAN, W. (Такман):** «Hellman Presents No Shortcut Solutions to DES», *IEEE Spectrum*, vol. 16, pp. 40–41, July 1979.

**TURNER, J. S. (Тернер):** «New Directions in Communications (or Which Way to the Information Age)», *IEEE Commun. Magazine*, vol. 24, pp. 8–15, Oct. 1986.

**VANHOEF, M., and PIESSENS, F. (Ванхоф и Писсенс):** «Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2», *Proc. 2017 SIGSAC Conf. on Computer and Commun. Security*, ACM, pp. 1313–1328, 2017.

**VARGHESE, G. (Варгезе):** «Network Algorithmics», San Francisco: Morgan Kaufmann, 2004.

**VARGHESE, G., and LAUCK, T. (Варгезе и Лаук):** «Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility», *Proc. 11th Symp. on Operating Systems Prin.*, ACM, pp. 25–38, 1987.

**VERIZON BUSINESS:** «2009 Data Breach Investigations Report», Verizon, 2009.

**VITERBI, A. (Витерби):** «CDMA: Principles of Spread Spectrum Communication», Upper Saddle River, NJ: Prentice Hall, 1995.

**WAITZMAN, D., PARTRIDGE, C., and DEERING, S. (Вайцман и др.):** «Distance Vector Multicast Routing Protocol», RFC 1075, Nov. 1988.

**WALDMAN, M., RUBIN, A. D., and CRANOR, L. F. (Уолдман и др.):** «Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System», *Proc. Ninth USENIX Security Symp.*, USENIX, pp. 59–72, 2000.

**WALTERS, R. (Уолтерс):** «Spread Spectrum: Hedy Lamarr and the Mobile Phone», Kindle, 2013.

**WANG, B., and REN, F. (Ван и Рен):** «Improving Robustness of DASH Against Network Uncertainty», *2019 Int'l Conf. on Multimedia and Expo*, IEEE, pp. 448–753, July 2019.

**WANG, Z., and CROWCROFT, J. (Ван и Кроукрофт):** «SEAL Detects Cell Misordering», *IEEE Network Magazine*, vol. 6, pp. 8–9, July 1992.

**WARE, R., MUKERJEE, M. K., SESHAN, S. and SHERRY J. (Уэр и др.):** «Modeling BBR's Interactions with Loss-Based Congestion Control», *Internet Measurement Conference*, 2019.

**WARNEKE, B., LAST, M., LIEBOWITZ, B., and PISTER, K. S. J. (Варнеке и др.):** «Smart Dust: Communicating with a Cubic Millimeter Computer», *IEEE Computer*, vol. 34, pp. 44–51, Jan. 2001.

**WEI, D., CHENG, J., LOW, S., and HEGDE, S. (Вэй и др.):** «FAST TCP: Motivation, Architecture, Algorithms, Performance», *IEEE/ACM Trans. on Networking*, vol. 14, pp. 1246–1259, Dec. 2006.

**WEISER, M. (Вайзер):** «The Computer for the Twenty-First Century», *Scientific American*, vol. 265, pp. 94–104, Sept. 1991.

**WHITE, G. (Уайт):** «Low Latency DOCSIS Overview And Performance Characteristics», *A Technical Paper prepared for SCTE-ISBE*, 2019.

**WITTENBERG, N. (Виттенберг):** «Understanding Voice Over IP Technology», Clifton Park, NY: Delmar Cengage Learning, 2009.

**WOOD, L., IVANCIC, W., EDDY, W., STEWART, D., NORTHAM, J., JACKSON, C., and DA SILVA CURIEL, A. (Вуд и др.):** «Use of the Delay-Tolerant Networking Bundle Protocol from Space», *Proc. 59th Int'l Astronautical Congress*, Int'l Astronautical Federation, pp. 3123–3133, 2008.

**WU, T. (У):** «Network Neutrality, Broadband Discrimination», *Journal on Telecom. and High-Tech. Law*, vol. 2, pp. 141–179, 2003.

**WYLIE, J., BIGRIGG, M. W., STRUNK, J. D., GANGER, G. R., KILICCOTE, H., and KHOSLA, P. K. (Уайли и др.):** «Survivable Information Storage Systems», *IEEE Computer*, vol. 33, pp. 61–68, Aug. 2000.

**YE, Y., LI, T., ADJEROH, D., and ITENGAR, S. S. (Е и др.):** «A Survey on Malware Detection Using Data Mining Techniques», *ACM Computing Surveys*, vol. 50, Art. 41, June 2017.

**YU, T., HARTMAN, S., and RAEBURN, K. (Юй и др.):** «The Perils of Unauthenticated Encryption: Kerberos Version 4», *Proc. NDSS Symposium*, Internet Society, Feb. 2004.

**YUVAL, G. (Юваль):** «How to Swindle Rabin», *Cryptologia*, vol. 3, pp. 187–190, July 1979.

**ZHANG, Y., BRESLAU, L., PAXSON, V., and SHENKER, S. (Чжан и др.):** «On the Characteristics and Origins of Internet Flow Rates», *Proc. SIGCOMM 2002 Conf.*, ACM, pp. 309–322, 2002.

**ZHANG, Y., YUAN, X., and TZENG, N.-F. (Чжан и др.):** «Pseudo-Honeypot: Toward Efficient and Scalable Spam Sniffer», *Proc. 49th Int'l Conf. on Dependable Systems and Networks*, IEEE, pp. 435–446, 2019.

**ZIMMERMANN, P. R. (Циммерман):** «The Official PGP User's Guide», Cambridge, MA: M.I.T. Press, 1995a.

**ZIPF, G. K. (Ципф):** «Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology», Boston: Addison-Wesley, 1949.

**ZIV, J., and LEMPEL, Z. (Зив и Лемпель):** «A Universal Algorithm for Sequential Data Compression», *IEEE Trans. on Information Theory*, vol. IT-3, pp. 337–343, May 1977.

*Эндрю Таненбаум, Ник Фимстер, Дэвид Уэзеролл*

# Компьютерные сети

## 6-е издание

*Перевели с английского С. Черников*

*Научный редактор М. Капранов*


| | |
|---|---|
| Руководитель дивизиона | *Ю. Сергиенко* |
| Ведущий редактор | *Е. Строганова* |
| Литературный редактор | *Ю. Лесняк* |
| Художественный редактор | *В. Мостипан* |
| Корректоры | *С. Беляева, Н. Викторова* |
| Верстка | *Л. Егорова* |

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 03.2023. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 — Книги печатные профессиональные, технические и научные.

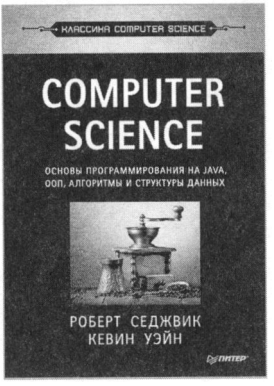
Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 26.01.23. Формат 70×100/16. Бумага офсетная. Усл. п. л. 79,980. Тираж 1700. Заказ 752.

Отпечатано с готовых файлов заказчика в АО «Первая Образцовая типография»,
филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ». 432980, Россия, г. Ульяновск, ул. Гончарова, 14

*Роберт Седжвик, Кевин Уэйн*

# COMPUTER SCIENCE: ОСНОВЫ ПРОГРАММИРОВАНИЯ НА JAVA, ООП, АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ



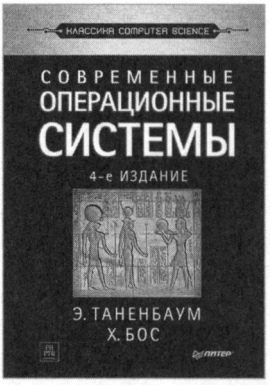

Преподаватели Принстонского университета Роберт Седжвик и Кевин Уэйн создали универсальное введение в Computer Science на языке Java, которое идеально подходит как студентам, так и профессионалам. Вы начнете с основ, освоите современный курс объектно-ориентированного программирования и перейдете к концепциям более высокого уровня: алгоритмам и структурам данных, теории вычислений и архитектуре компьютеров.

И главное — вся теория рассматривается на практических и ярких примерах: прикладная математика, физика и биология, числовые методы, визуализация данных, синтез звука, обработка графики, финансовое моделирование и многое другое.

*Э. Таненбаум, Х. Бос*

# СОВРЕМЕННЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

## 4-е издание




Эндрю Таненбаум представляет новое издание своего всемирного бестселлера, необходимое для понимания функционирования современных операционных систем. Оно существенно отличается от предыдущего и включает в себя сведения о последних достижениях в области информационных технологий.

Например, глава о Windows Vista теперь заменена подробным рассмотрением Windows 8.1 как самой актуальной версии на момент написания книги. Появился объемный раздел, посвященный операционной системе Android. Был обновлен материал, касающийся Unix и Linux, а также RAID-систем. Гораздо больше внимания уделено мультиядерным и многоядерным системам, важность которых в последние несколько лет постоянно возрастает. Появилась совершенно новая глава о виртуализации и облачных вычислениях. Добавился большой объем нового материала об использовании ошибок кода, о вредоносных программах и соответствующих мерах защиты.

В книге в ясной и увлекательной форме приводится множество важных подробностей, которых нет ни в одном другом издании.

Э. ТАНЕНБАУМ
Н. ФИМСТЕР, Д. УЭЗЕРОЛЛ

# КОМПЬЮТЕРНЫЕ СЕТИ

## ШЕСТОЕ ИЗДАНИЕ

КЛАССИКА COMPUTER SCIENCE

Перед вами шестое издание самой авторитетной книги по современным сетевым технологиям, написанное признанным экспертом Эндрю Таненбаумом в соавторстве со специалистом компании Google Дэвидом Уэзероллом и профессором Чикагского университета Ником Фимстером. Первая версия этого классического труда появилась на свет в далеком 1980 году, и с тех пор каждое издание книги неизменно становилось бестселлером. В книге последовательно изложены основные концепции, определяющие современное состояние компьютерных сетей и тенденции их развития. Авторы подробно объясняют устройство и принципы работы аппаратного и программного обеспечения, рассматривают все аспекты и уровни организации сетей — от физического до прикладного. Изложение теоретических принципов дополняется яркими, показательными примерами функционирования интернета и компьютерных сетей различного типа. Большое внимание уделяется сетевой безопасности.

Шестое издание полностью переработано с учетом изменений, произошедших в сфере сетевых технологий за последние годы, и, в частности, освещает такие технологии, как DOCSIS, 4G и 5G, беспроводные сети стандарта 802.11ax, 100-гигабитные сети Ethernet, интернет вещей, современные транспортные протоколы CUBIC TCP, QUIC и BBR, программно-конфигурируемые сети и многое другое.

**Тема:** Компьютерные сети / **Уровень пользователя:** опытный

ISBN:978-5-4461-1766-6

ПИТЕР®
WWW.PITER.COM
интернет-магазин
Заказ книг:
(812) 703-73-74
books@piter.com
PiterBooks
PiterForPeople
ThePiterBooks
Company/piter
PEARSON
Prentice Hall