FIFTH EDITION

COMPUTER NETWORKS



TANENBAUM | WETHERALL

Patent Owner's Ex. 2010
Samsung v. XiFi (PGR2025-00067)
Page 001

COMPUTER NETWORKS

FIFTH EDITION

ANDREW S. TANENBAUM

Vrije Universiteit
Amsterdam, The Netherlands

DAVID J. WETHERALL

University of Washington
Seattle, WA

PRENTICE HALL

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Paris Montreal Toronto
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Tapei Tokyo

Patent Owner's Ex. 2010
Page 002

Editorial Director: Marcia Horton
Editor-in-Chief: Michael Hirsch
Executive Editor: Tracy Dunkelberger
Assistant Editor: Melinda Haggerty
Editorial Assistant: Allison Michael
Vice President, Marketing: Patrice Jones
Marketing Manager: Yezan Alayan
Marketing Coordinator: Kathryn Ferranti
Vice President, Production: Vince O'Brien
Managing Editor: Jeff Holcomb
Senior Operations Supervisor: Alan Fischer
Manufacturing Buyer: Lisa McDowell
Cover Direction: Andrew S. Tanenbaum, David J. Wetherall, Tracy Dunkelberger
Art Director: Linda Knowles
Cover Designer: Susan Paradise
Cover Illustration: Jason Consalvo
Interior Design: Andrew S. Tanenbaum
AV Production Project Manager: Gregory L. Dulles
Interior Illustrations: Laserwords, Inc.
Media Editor: Daniel Sandin
Composition: Andrew S. Tanenbaum
Copyeditor: Rachel Head
Proofreader: Joe Ruddick
Printer/Binder: Courier/Westford
Cover Printer: Lehigh-Phoenix Color/ Hagerstown

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on appropriate page within text.

Many of the designations by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Copyright © 2011, 2003, 1996, 1989, 1981 Pearson Education, Inc., publishing as Prentice Hall. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, 501 Boylston Street, Suite 900, Boston, Massachusetts 02116.

Library of Congress Cataloging-in-Publication Data

Tanenbaum, Andrew S., 1944-
Computer networks / Andrew S. Tanenbaum, David J. Wetherall. -- 5th ed.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-13-212695-3 (alk. paper)
ISBN-10: 0-13-212695-8 (alk. paper)
1. Computer networks. I. Wetherall, D. (David) II. Title.
TK5105.5.T36 2011
004.6--dc22

2010034366

10 9 8 7 6 5 4 3 2 1—CRW—14 13 12 11 10



To Suzanne, Barbara, Daniel, Aron, Marvin, Matilde,
and the memory of Bram, and Sweetie π (AST)

To Katrin, Lucy, and Pepper (DJW)

This page intentionally left blank

CONTENTS

PREFACE xix

1 INTRODUCTION 1

1.1 USES OF COMPUTER NETWORKS, 3
- 1.1.1 Business Applications, 3
- 1.1.2 Home Applications, 6
- 1.1.3 Mobile Users, 10
- 1.1.4 Social Issues, 14

1.2 NETWORK HARDWARE, 17
- 1.2.1 Personal Area Networks, 18
- 1.2.2 Local Area Networks, 19
- 1.2.3 Metropolitan Area Networks, 23
- 1.2.4 Wide Area Networks, 23
- 1.2.5 Internetworks, 28

1.3 NETWORK SOFTWARE, 29
- 1.3.1 Protocol Hierarchies, 29
- 1.3.2 Design Issues for the Layers, 33
- 1.3.3 Connection-Oriented Versus Connectionless Service, 35
- 1.3.4 Service Primitives, 38
- 1.3.5 The Relationship of Services to Protocols, 40

1.4 REFERENCE MODELS, 41
- 1.4.1 The OSI Reference Model, 41
- 1.4.2 The TCP/IP Reference Model, 45
- 1.4.3 The Model Used in This Book, 48

Patent Owner's Ex. 2010
Page 006

1.4.4 A Comparison of the OSI and TCP/IP Reference Models*, 49
1.4.5 A Critique of the OSI Model and Protocols*, 51
1.4.6 A Critique of the TCP/IP Reference Model*, 53

1.5 EXAMPLE NETWORKS, 54
1.5.1 The Internet, 54
1.5.2 Third-Generation Mobile Phone Networks*, 65
1.5.3 Wireless LANs: 802.11*, 70
1.5.4 RFID and Sensor Networks*, 73

1.6 NETWORK STANDARDIZATION*, 75
1.6.1 Who's Who in the Telecommunications World, 77
1.6.2 Who's Who in the International Standards World, 78
1.6.3 Who's Who in the Internet Standards World, 80

1.7 METRIC UNITS, 82

1.8 OUTLINE OF THE REST OF THE BOOK, 83

1.9 SUMMARY, 84

2 THE PHYSICAL LAYER 89

2.1 THE THEORETICAL BASIS FOR DATA COMMUNICATION, 90
2.1.1 Fourier Analysis, 90
2.1.2 Bandwidth-Limited Signals, 90
2.1.3 The Maximum Data Rate of a Channel, 94

2.2 GUIDED TRANSMISSION MEDIA, 95
2.2.1 Magnetic Media, 95
2.2.2 Twisted Pairs, 96
2.2.3 Coaxial Cable, 97
2.2.4 Power Lines, 98
2.2.5 Fiber Optics, 99

2.3 WIRELESS TRANSMISSION, 105
2.3.1 The Electromagnetic Spectrum, 105
2.3.2 Radio Transmission, 109
2.3.3 Microwave Transmission, 110
2.3.4 Infrared Transmission, 114
2.3.5 Light Transmission, 114

2.4 COMMUNICATION SATELLITES*, 116
2.4.1 Geostationary Satellites, 117
2.4.2 Medium-Earth Orbit Satellites, 121
2.4.3 Low-Earth Orbit Satellites, 121
2.4.4 Satellites Versus Fiber, 123

2.5 DIGITAL MODULATION AND MULTIPLEXING, 125
2.5.1 Baseband Transmission, 125
2.5.2 Passband Transmission, 130
2.5.3 Frequency Division Multiplexing, 132
2.5.4 Time Division Multiplexing, 135
2.5.5 Code Division Multiplexing, 135

2.6 THE PUBLIC SWITCHED TELEPHONE NETWORK, 138
2.6.1 Structure of the Telephone System, 139
2.6.2 The Politics of Telephones, 142
2.6.3 The Local Loop: Modems, ADSL, and Fiber, 144
2.6.4 Trunks and Multiplexing, 152
2.6.5 Switching, 161

2.7 THE MOBILE TELEPHONE SYSTEM*, 164
2.7.1 First-Generation (coco1G) Mobile Phones: Analog Voice, 166
2.7.2 Second-Generation (2G) Mobile Phones: Digital Voice, 170
2.7.3 Third-Generation (3G) Mobile Phones: Digital Voice and Data, 174

2.8 CABLE TELEVISION*, 179
2.8.1 Community Antenna Television, 179
2.8.2 Internet over Cable, 180
2.8.3 Spectrum Allocation, 182
2.8.4 Cable Modems, 183
2.8.5 ADSL Versus Cable, 185

2.9 SUMMARY, 186

3 THE DATA LINK LAYER 193

3.1 DATA LINK LAYER DESIGN ISSUES, 194
3.1.1 Services Provided to the Network Layer, 194
3.1.2 Framing, 197
3.1.3 Error Control, 200
3.1.4 Flow Control, 201

3.2 ERROR DETECTION AND CORRECTION, 202
3.2.1 Error-Correcting Codes, 204
3.2.2 Error-Detecting Codes, 209

3.3 ELEMENTARY DATA LINK PROTOCOLS, 215
3.3.1 A Utopian Simplex Protocol, 220
3.3.2 A Simplex Stop-and-Wait Protocol for an Error-Free Channel, 221
3.3.3 A Simplex Stop-and-Wait Protocol for a Noisy Channel, 222

3.4 SLIDING WINDOW PROTOCOLS, 226
3.4.1 A One-Bit Sliding Window Protocol, 229
3.4.2 A Protocol Using Go-Back-N, 232
3.4.3 A Protocol Using Selective Repeat, 239

3.5 EXAMPLE DATA LINK PROTOCOLS, 244
3.5.1 Packet over SONET, 245
3.5.2 ADSL (Asymmetric Digital Subscriber Loop), 248

3.6 SUMMARY, 251

4 THE MEDIUM ACCESS CONTROL SUBLAYER 257

4.1 THE CHANNEL ALLOCATION PROBLEM, 258
4.1.1 Static Channel Allocation, 258
4.1.2 Assumptions for Dynamic Channel Allocation, 260

4.2 MULTIPLE ACCESS PROTOCOLS, 261
4.2.1 ALOHA, 262
4.2.2 Carrier Sense Multiple Access Protocols, 266
4.2.3 Collision-Free Protocols, 269
4.2.4 Limited-Contention Protocols, 274
4.2.5 Wireless LAN Protocols, 277

4.3 ETHERNET, 280
4.3.1 Classic Ethernet Physical Layer, 281
4.3.2 Classic Ethernet MAC Sublayer Protocol, 282
4.3.3 Ethernet Performance, 286
4.3.4 Switched Ethernet, 288

4.3.5 Fast Ethernet, 290
4.3.6 Gigabit Ethernet, 293
4.3.7 10-Gigabit Ethernet, 296
4.3.8 Retrospective on Ethernet, 298

4.4 WIRELESS LANS, 299
4.4.1 The 802.11 Architecture and Protocol Stack, 299
4.4.2 The 802.11 Physical Layer, 301
4.4.3 The 802.11 MAC Sublayer Protocol, 303
4.4.4 The 802.11 Frame Structure, 309
4.4.5 Services, 311

4.5 BROADBAND WIRELESS*, 312
4.5.1 Comparison of 802.16 with 802.11 and 3G, 313
4.5.2 The 802.16 Architecture and Protocol Stack, 314
4.5.3 The 802.16 Physical Layer, 316
4.5.4 The 802.16 MAC Sublayer Protocol, 317
4.5.5 The 802.16 Frame Structure, 319

4.6 BLUETOOTH*, 320
4.6.1 Bluetooth Architecture, 320
4.6.2 Bluetooth Applications, 321
4.6.3 The Bluetooth Protocol Stack, 322
4.6.4 The Bluetooth Radio Layer, 324
4.6.5 The Bluetooth Link Layers, 324
4.6.6 The Bluetooth Frame Structure, 325

4.7 RFID*, 327
4.7.1 EPC Gen 2 Architecture, 327
4.7.2 EPC Gen 2 Physical Layer, 328
4.7.3 EPC Gen 2 Tag Identification Layer, 329
4.7.4 Tag Identification Message Formats, 331

4.8 DATA LINK LAYER SWITCHING, 332
4.8.1 Uses of Bridges, 332
4.8.2 Learning Bridges, 334
4.8.3 Spanning Tree Bridges, 337
4.8.4 Repeaters, Hubs, Bridges, Switches, Routers, and Gateways, 340
4.8.5 Virtual LANs, 342

4.9 SUMMARY, 349

Patent Owner's Ex. 2010
Page 010

5 THE NETWORK LAYER 355

5.1 NETWORK LAYER DESIGN ISSUES, 355
- 5.1.1 Store-and-Forward Packet Switching, 356
- 5.1.2 Services Provided to the Transport Layer, 356
- 5.1.3 Implementation of Connectionless Service, 358
- 5.1.4 Implementation of Connection-Oriented Service, 359
- 5.1.5 Comparison of Virtual-Circuit and Datagram Networks, 361

5.2 ROUTING ALGORITHMS, 362
- 5.2.1 The Optimality Principle, 364
- 5.2.2 Shortest Path Algorithm, 366
- 5.2.3 Flooding, 368
- 5.2.4 Distance Vector Routing, 370
- 5.2.5 Link State Routing, 373
- 5.2.6 Hierarchical Routing, 378
- 5.2.7 Broadcast Routing, 380
- 5.2.8 Multicast Routing, 382
- 5.2.9 Anycast Routing, 385
- 5.2.10 Routing for Mobile Hosts, 386
- 5.2.11 Routing in Ad Hoc Networks, 389

5.3 CONGESTION CONTROL ALGORITHMS, 392
- 5.3.1 Approaches to Congestion Control, 394
- 5.3.2 Traffic-Aware Routing, 395
- 5.3.3 Admission Control, 397
- 5.3.4 Traffic Throttling, 398
- 5.3.5 Load Shedding, 401

5.4 QUALITY OF SERVICE, 404
- 5.4.1 Application Requirements, 405
- 5.4.2 Traffic Shaping, 407
- 5.4.3 Packet Scheduling, 411
- 5.4.4 Admission Control, 415
- 5.4.5 Integrated Services, 418
- 5.4.6 Differentiated Services, 421

5.5 INTERNETWORKING, 424
- 5.5.1 How Networks Differ, 425
- 5.5.2 How Networks Can Be Connected, 426
- 5.5.3 Tunneling, 429

5.5.4 Internetwork Routing, 431
5.5.5 Packet Fragmentation, 432

5.6 THE NETWORK LAYER IN THE INTERNET, 436
5.6.1 The IP Version 4 Protocol, 439
5.6.2 IP Addresses, 442
5.6.3 IP Version 6, 455
5.6.4 Internet Control Protocols, 465
5.6.5 Label Switching and MPLS, 470
5.6.6 OSPF—An Interior Gateway Routing Protocol, 474
5.6.7 BGP—The Exterior Gateway Routing Protocol, 479
5.6.8 Internet Multicasting, 484
5.6.9 Mobile IP, 485

5.7 SUMMARY, 488

6 THE TRANSPORT LAYER 495

6.1 THE TRANSPORT SERVICE, 495
6.1.1 Services Provided to the Upper Layers, 496
6.1.2 Transport Service Primitives, 498
6.1.3 Berkeley Sockets, 500
6.1.4 An Example of Socket Programming: An Internet File Server, 503

6.2 ELEMENTS OF TRANSPORT PROTOCOLS, 507
6.2.1 Addressing, 509
6.2.2 Connection Establishment, 512
6.2.3 Connection Release, 517
6.2.4 Error Control and Flow Control, 522
6.2.5 Multiplexing, 527
6.2.6 Crash Recovery, 527

6.3 CONGESTION CONTROL, 530
6.3.1 Desirable Bandwidth Allocation, 531
6.3.2 Regulating the Sending Rate, 535
6.3.3 Wireless Issues, 539

6.4 THE INTERNET TRANSPORT PROTOCOLS: UDP, 541
6.4.1 Introduction to UDP, 541
6.4.2 Remote Procedure Call, 543
6.4.3 Real-Time Transport Protocols, 546

6.5 THE INTERNET TRANSPORT PROTOCOLS: TCP, 552
6.5.1 Introduction to TCP, 552
6.5.2 The TCP Service Model, 553
6.5.3 The TCP Protocol, 556
6.5.4 The TCP Segment Header, 557
6.5.5 TCP Connection Establishment, 560
6.5.6 TCP Connection Release, 562
6.5.7 TCP Connection Management Modeling, 562
6.5.8 TCP Sliding Window, 565
6.5.9 TCP Timer Management, 568
6.5.10 TCP Congestion Control, 571
6.5.11 The Future of TCP, 581

6.6 PERFORMANCE ISSUES*, 582
6.6.1 Performance Problems in Computer Networks, 583
6.6.2 Network Performance Measurement, 584
6.6.3 Host Design for Fast Networks, 586
6.6.4 Fast Segment Processing, 590
6.6.5 Header Compression, 593
6.6.6 Protocols for Long Fat Networks, 595

6.7 DELAY-TOLERANT NETWORKING*, 599
6.7.1 DTN Architecture, 600
6.7.2 The Bundle Protocol, 603

6.8 SUMMARY, 605

7 THE APPLICATION LAYER 611

7.1 DNS—THE DOMAIN NAME SYSTEM, 611
7.1.1 The DNS Name Space, 612
7.1.2 Domain Resource Records, 616
7.1.3 Name Servers, 619

7.2 ELECTRONIC MAIL*, 623
7.2.1 Architecture and Services, 624
7.2.2 The User Agent, 626
7.2.3 Message Formats, 630
7.2.4 Message Transfer, 637
7.2.5 Final Delivery, 643

7.3 THE WORLD WIDE WEB, 646
7.3.1 Architectural Overview, 647
7.3.2 Static Web Pages, 662
7.3.3 Dynamic Web Pages and Web Applications, 672
7.3.4 HTTP—The HyperText Transfer Protocol, 683
7.3.5 The Mobile Web, 693
7.3.6 Web Search, 695

7.4 STREAMING AUDIO AND VIDEO, 697
7.4.1 Digital Audio, 699
7.4.2 Digital Video, 704
7.4.3 Streaming Stored Media, 713
7.4.4 Streaming Live Media, 721
7.4.5 Real-Time Conferencing, 724

7.5 CONTENT DELIVERY, 734
7.5.1 Content and Internet Traffic, 736
7.5.2 Server Farms and Web Proxies, 738
7.5.3 Content Delivery Networks, 743
7.5.4 Peer-to-Peer Networks, 748

7.6 SUMMARY, 757

8 NETWORK SECURITY 763

8.1 CRYPTOGRAPHY, 766
8.1.1 Introduction to Cryptography, 767
8.1.2 Substitution Ciphers, 769
8.1.3 Transposition Ciphers, 771
8.1.4 One-Time Pads, 772
8.1.5 Two Fundamental Cryptographic Principles, 776

8.2 SYMMETRIC-KEY ALGORITHMS, 778
8.2.1 DES—The Data Encryption Standard, 780
8.2.2 AES—The Advanced Encryption Standard, 783
8.2.3 Cipher Modes, 787
8.2.4 Other Ciphers, 792
8.2.5 Cryptanalysis, 792

8.3 PUBLIC-KEY ALGORITHMS, 793
8.3.1 RSA, 794
8.3.2 Other Public-Key Algorithms, 796

8.4 DIGITAL SIGNATURES, 797
8.4.1 Symmetric-Key Signatures, 798
8.4.2 Public-Key Signatures, 799
8.4.3 Message Digests, 800
8.4.4 The Birthday Attack, 804

8.5 MANAGEMENT OF PUBLIC KEYS, 806
8.5.1 Certificates, 807
8.5.2 X.509, 809
8.5.3 Public Key Infrastructures, 810

8.6 COMMUNICATION SECURITY, 813
8.6.1 IPsec, 814
8.6.2 Firewalls, 818
8.6.3 Virtual Private Networks, 821
8.6.4 Wireless Security, 822

8.7 AUTHENTICATION PROTOCOLS, 827
8.7.1 Authentication Based on a Shared Secret Key, 828
8.7.2 Establishing a Shared Key: The Diffie-Hellman Key Exchange, 833
8.7.3 Authentication Using a Key Distribution Center, 835
8.7.4 Authentication Using Kerberos, 838
8.7.5 Authentication Using Public-Key Cryptography, 840

8.8 EMAIL SECURITY*, 841
8.8.1 PGP—Pretty Good Privacy, 842
8.8.2 S/MIME, 846

8.9 WEB SECURITY, 846
8.9.1 Threats, 847
8.9.2 Secure Naming, 848
8.9.3 SSL—The Secure Sockets Layer, 853
8.9.4 Mobile Code Security, 857

8.10 SOCIAL ISSUES, 860
8.10.1 Privacy, 860
8.10.2 Freedom of Speech, 863
8.10.3 Copyright, 867

8.11 SUMMARY, 869

9 READING LIST AND BIBLIOGRAPHY 877

9.1 SUGGESTIONS FOR FURTHER READING*, 877
- 9.1.1 Introduction and General Works, 878
- 9.1.2 The Physical Layer, 879
- 9.1.3 The Data Link Layer, 880
- 9.1.4 The Medium Access Control Sublayer, 880
- 9.1.5 The Network Layer, 881
- 9.1.6 The Transport Layer, 882
- 9.1.7 The Application Layer, 882
- 9.1.8 Network Security, 883

9.2 ALPHABETICAL BIBLIOGRAPHY*, 884

INDEX 905

1

INTRODUCTION

Each of the past three centuries was dominated by a single new technology. The 18th century was the era of the great mechanical systems accompanying the Industrial Revolution. The 19th century was the age of the steam engine. During the 20th century, the key technology was information gathering, processing, and distribution. Among other developments, we saw the installation of worldwide telephone networks, the invention of radio and television, the birth and unprecedented growth of the computer industry, the launching of communication satellites, and, of course, the Internet.

As a result of rapid technological progress, these areas are rapidly converging in the 21st century and the differences between collecting, transporting, storing, and processing information are quickly disappearing. Organizations with hundreds of offices spread over a wide geographical area routinely expect to be able to examine the current status of even their most remote outpost at the push of a button. As our ability to gather, process, and distribute information grows, the demand for ever more sophisticated information processing grows even faster.

Although the computer industry is still young compared to other industries (e.g., automobiles and air transportation), computers have made spectacular progress in a short time. During the first two decades of their existence, computer systems were highly centralized, usually within a single large room. Not infrequently, this room had glass walls, through which visitors could gawk at the great electronic wonder inside. A medium-sized company or university might have had

Patent Owner's Ex. 2010
Page 017

one or two computers, while very large institutions had at most a few dozen. The idea that within forty years vastly more powerful computers smaller than postage stamps would be mass produced by the billions was pure science fiction.

The merging of computers and communications has had a profound influence on the way computer systems are organized. The once-dominant concept of the ''computer center'' as a room with a large computer to which users bring their work for processing is now totally obsolete (although data centers holding thousands of Internet servers are becoming common). The old model of a single computer serving all of the organization's computational needs has been replaced by one in which a large number of separate but interconnected computers do the job. These systems are called **computer networks**. The design and organization of these networks are the subjects of this book.

Throughout the book we will use the term ''computer network'' to mean a collection of autonomous computers interconnected by a single technology. Two computers are said to be interconnected if they are able to exchange information. The connection need not be via a copper wire; fiber optics, microwaves, infrared, and communication satellites can also be used. Networks come in many sizes, shapes and forms, as we will see later. They are usually connected together to make larger networks, with the **Internet** being the most well-known example of a network of networks.

There is considerable confusion in the literature between a computer network and a **distributed system**. The key distinction is that in a distributed system, a collection of independent computers appears to its users as a single coherent system. Usually, it has a single model or paradigm that it presents to the users. Often a layer of software on top of the operating system, called **middleware**, is responsible for implementing this model. A well-known example of a distributed system is the **World Wide Web**. It runs on top of the Internet and presents a model in which everything looks like a document (Web page).

In a computer network, this coherence, model, and software are absent. Users are exposed to the actual machines, without any attempt by the system to make the machines look and act in a coherent way. If the machines have different hardware and different operating systems, that is fully visible to the users. If a user wants to run a program on a remote machine, he† has to log onto that machine and run it there.

In effect, a distributed system is a software system built on top of a network. The software gives it a high degree of cohesiveness and transparency. Thus, the distinction between a network and a distributed system lies with the software (especially the operating system), rather than with the hardware.

Nevertheless, there is considerable overlap between the two subjects. For example, both distributed systems and computer networks need to move files around. The difference lies in who invokes the movement, the system or the user.

† ''He'' should be read as ''he or she'' throughout this book.

Patent Owner's Ex. 2010
Page 018

Although this book primarily focuses on networks, many of the topics are also important in distributed systems. For more information about distributed systems, see Tanenbaum and Van Steen (2007).

1.1 USES OF COMPUTER NETWORKS

Before we start to examine the technical issues in detail, it is worth devoting some time to pointing out why people are interested in computer networks and what they can be used for. After all, if nobody were interested in computer networks, few of them would be built. We will start with traditional uses at companies, then move on to home networking and recent developments regarding mobile users, and finish with social issues.

1.1.1 Business Applications

Most companies have a substantial number of computers. For example, a company may have a computer for each worker and use them to design products, write brochures, and do the payroll. Initially, some of these computers may have worked in isolation from the others, but at some point, management may have decided to connect them to be able to distribute information throughout the company.

Put in slightly more general form, the issue here is **resource sharing**. The goal is to make all programs, equipment, and especially data available to anyone on the network without regard to the physical location of the resource or the user. An obvious and widespread example is having a group of office workers share a common printer. None of the individuals really needs a private printer, and a high-volume networked printer is often cheaper, faster, and easier to maintain than a large collection of individual printers.

However, probably even more important than sharing physical resources such as printers, and tape backup systems, is sharing information. Companies small and large are vitally dependent on computerized information. Most companies have customer records, product information, inventories, financial statements, tax information, and much more online. If all of its computers suddenly went down, a bank could not last more than five minutes. A modern manufacturing plant, with a computer-controlled assembly line, would not last even 5 seconds. Even a small travel agency or three-person law firm is now highly dependent on computer networks for allowing employees to access relevant information and documents instantly.

For smaller companies, all the computers are likely to be in a single office or perhaps a single building, but for larger ones, the computers and employees may be scattered over dozens of offices and plants in many countries. Nevertheless, a sales person in New York might sometimes need access to a product inventory

database in Singapore. Networks called **VPNs** (**Virtual Private Networks**) may be used to join the individual networks at different sites into one extended network. In other words, the mere fact that a user happens to be 15,000 km away from his data should not prevent him from using the data as though they were local. This goal may be summarized by saying that it is an attempt to end the "tyranny of geography."

In the simplest of terms, one can imagine a company's information system as consisting of one or more databases with company information and some number of employees who need to access them remotely. In this model, the data are stored on powerful computers called **servers**. Often these are centrally housed and maintained by a system administrator. In contrast, the employees have simpler machines, called **clients**, on their desks, with which they access remote data, for example, to include in spreadsheets they are constructing. (Sometimes we will refer to the human user of the client machine as the "client," but it should be clear from the context whether we mean the computer or its user.) The client and server machines are connected by a network, as illustrated in Fig. 1-1. Note that we have shown the network as a simple oval, without any detail. We will use this form when we mean a network in the most abstract sense. When more detail is required, it will be provided.

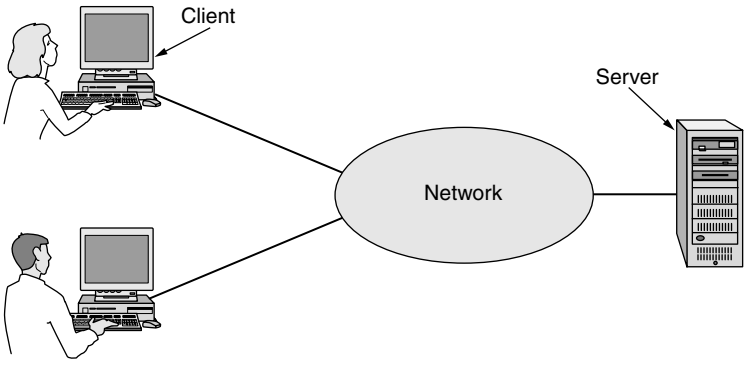


Figure 1-1. A network with two clients and one server.

This whole arrangement is called the **client-server model**. It is widely used and forms the basis of much network usage. The most popular realization is that of a **Web application**, in which the server generates Web pages based on its database in response to client requests that may update the database. The client-server model is applicable when the client and server are both in the same building (and belong to the same company), but also when they are far apart. For example, when a person at home accesses a page on the World Wide Web, the same model is employed, with the remote Web server being the server and the user's personal

computer being the client. Under most conditions, one server can handle a large number (hundreds or thousands) of clients simultaneously.

If we look at the client-server model in detail, we see that two processes (i.e., running programs) are involved, one on the client machine and one on the server machine. Communication takes the form of the client process sending a message over the network to the server process. The client process then waits for a reply message. When the server process gets the request, it performs the requested work or looks up the requested data and sends back a reply. These messages are shown in Fig. 1-2.

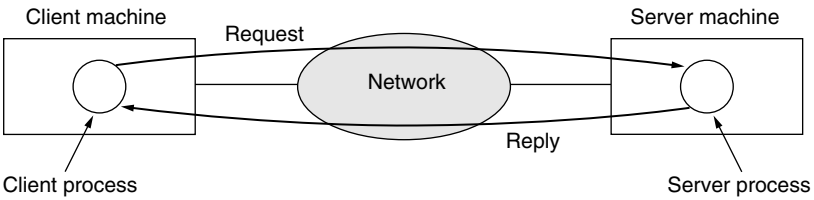


Figure 1-2. The client-server model involves requests and replies.

A second goal of setting up a computer network has to do with people rather than information or even computers. A computer network can provide a powerful **communication medium** among employees. Virtually every company that has two or more computers now has **email** (**electronic mail**), which employees generally use for a great deal of daily communication. In fact, a common gripe around the water cooler is how much email everyone has to deal with, much of it quite meaningless because bosses have discovered that they can send the same (often content-free) message to all their subordinates at the push of a button.

Telephone calls between employees may be carried by the computer network instead of by the phone company. This technology is called **IP telephony** or **Voice over IP** (**VoIP**) when Internet technology is used. The microphone and speaker at each end may belong to a VoIP-enabled phone or the employee's computer. Companies find this a wonderful way to save on their telephone bills.

Other, richer forms of communication are made possible by computer networks. Video can be added to audio so that employees at distant locations can see and hear each other as they hold a meeting. This technique is a powerful tool for eliminating the cost and time previously devoted to travel. **Desktop sharing** lets remote workers see and interact with a graphical computer screen. This makes it easy for two or more people who work far apart to read and write a shared blackboard or write a report together. When one worker makes a change to an online document, the others can see the change immediately, instead of waiting several days for a letter. Such a speedup makes cooperation among far-flung groups of people easy where it previously had been impossible. More ambitious forms of remote coordination such as telemedicine are only now starting to be used (e.g.,

remote patient monitoring) but may become much more important. It is sometimes said that communication and transportation are having a race, and whichever wins will make the other obsolete.

A third goal for many companies is doing business electronically, especially with customers and suppliers. This new model is called **e-commerce** (**electronic commerce**) and it has grown rapidly in recent years. Airlines, bookstores, and other retailers have discovered that many customers like the convenience of shopping from home. Consequently, many companies provide catalogs of their goods and services online and take orders online. Manufacturers of automobiles, aircraft, and computers, among others, buy subsystems from a variety of suppliers and then assemble the parts. Using computer networks, manufacturers can place orders electronically as needed. This reduces the need for large inventories and enhances efficiency.

1.1.2 Home Applications

In 1977, Ken Olsen was president of the Digital Equipment Corporation, then the number two computer vendor in the world (after IBM). When asked why Digital was not going after the personal computer market in a big way, he said: ''There is no reason for any individual to have a computer in his home.'' History showed otherwise and Digital no longer exists. People initially bought computers for word processing and games. Recently, the biggest reason to buy a home computer was probably for Internet access. Now, many consumer electronic devices, such as set-top boxes, game consoles, and clock radios, come with embedded computers and computer networks, especially wireless networks, and home networks are broadly used for entertainment, including listening to, looking at, and creating music, photos, and videos.

Internet access provides home users with **connectivity** to remote computers. As with companies, home users can access information, communicate with other people, and buy products and services with e-commerce. The main benefit now comes from connecting outside of the home. Bob Metcalfe, the inventor of Ethernet, hypothesized that the value of a network is proportional to the square of the number of users because this is roughly the number of different connections that may be made (Gilder, 1993). This hypothesis is known as ''Metcalfe's law.'' It helps to explain how the tremendous popularity of the Internet comes from its size.

Access to remote information comes in many forms. It can be surfing the World Wide Web for information or just for fun. Information available includes the arts, business, cooking, government, health, history, hobbies, recreation, science, sports, travel, and many others. Fun comes in too many ways to mention, plus some ways that are better left unmentioned.

Many newspapers have gone online and can be personalized. For example, it is sometimes possible to tell a newspaper that you want everything about corrupt

politicians, big fires, scandals involving celebrities, and epidemics, but no football, thank you. Sometimes it is possible to have the selected articles downloaded to your computer while you sleep. As this trend continues, it will cause massive unemployment among 12-year-old paperboys, but newspapers like it because distribution has always been the weakest link in the whole production chain. Of course, to make this model work, they will first have to figure out how to make money in this new world, something not entirely obvious since Internet users expect everything to be free.

The next step beyond newspapers (plus magazines and scientific journals) is the online digital library. Many professional organizations, such as the ACM (*www.acm.org*) and the IEEE Computer Society (*www.computer.org*), already have all their journals and conference proceedings online. Electronic book readers and online libraries may make printed books obsolete. Skeptics should take note of the effect the printing press had on the medieval illuminated manuscript.

Much of this information is accessed using the client-server model, but there is different, popular model for accessing information that goes by the name of **peer-to-peer** communication (Parameswaran et al., 2001). In this form, individuals who form a loose group can communicate with others in the group, as shown in Fig. 1-3. Every person can, in principle, communicate with one or more other people; there is no fixed division into clients and servers.

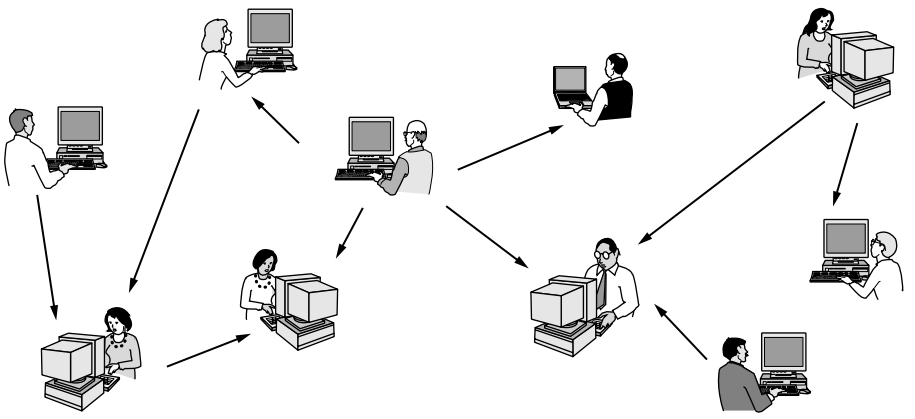


Figure 1-3. In a peer-to-peer system there are no fixed clients and servers.

Many peer-to-peer systems, such BitTorrent (Cohen, 2003), do not have any central database of content. Instead, each user maintains his own database locally and provides a list of other nearby people who are members of the system. A new user can then go to any existing member to see what he has and get the names of other members to inspect for more content and more names. This lookup process can be repeated indefinitely to build up a large local database of what is out there. It is an activity that would get tedious for people but computers excel at it.

Peer-to-peer communication is often used to share music and videos. It really hit the big time around 2000 with a music sharing service called Napster that was shut down after what was probably the biggest copyright infringement case in all of recorded history (Lam and Tan, 2001; and Macedonia, 2000). Legal applications for peer-to-peer communication also exist. These include fans sharing public domain music, families sharing photos and movies, and users downloading public software packages. In fact, one of the most popular Internet applications of all, email, is inherently peer-to-peer. This form of communication is likely to grow considerably in the future.

All of the above applications involve interactions between a person and a remote database full of information. The second broad category of network use is person-to-person communication, basically the 21st century's answer to the 19th century's telephone. E-mail is already used on a daily basis by millions of people all over the world and its use is growing rapidly. It already routinely contains audio and video as well as text and pictures. Smell may take a while.

Any teenager worth his or her salt is addicted to **instant messaging**. This facility, derived from the UNIX *talk* program in use since around 1970, allows two people to type messages at each other in real time. There are multi-person messaging services too, such as the **Twitter** service that lets people send short text messages called "tweets" to their circle of friends or other willing audiences.

The Internet can be used by applications to carry audio (e.g., Internet radio stations) and video (e.g., YouTube). Besides being a cheap way to call to distant friends, these applications can provide rich experiences such as telelearning, meaning attending 8 A.M. classes without the inconvenience of having to get out of bed first. In the long run, the use of networks to enhance human-to-human communication may prove more important than any of the others. It may become hugely important to people who are geographically challenged, giving them the same access to services as people living in the middle of a big city.

Between person-to-person communications and accessing information are **social network** applications. Here, the flow of information is driven by the relationships that people declare between each other. One of the most popular social networking sites is **Facebook**. It lets people update their personal profiles and shares the updates with other people who they have declared to be their friends. Other social networking applications can make introductions via friends of friends, send news messages to friends such as Twitter above, and much more.

Even more loosely, groups of people can work together to create content. A **wiki**, for example, is a collaborative Web site that the members of a community edit. The most famous wiki is the **Wikipedia**, an encyclopedia anyone can edit, but there are thousands of other wikis.

Our third category is electronic commerce in the broadest sense of the term. Home shopping is already popular and enables users to inspect the online catalogs of thousands of companies. Some of these catalogs are interactive, showing products from different viewpoints and in configurations that can be personalized.

After the customer buys a product electronically but cannot figure out how to use it, online technical support may be consulted.

Another area in which e-commerce is widely used is access to financial institutions. Many people already pay their bills, manage their bank accounts, and handle their investments electronically. This trend will surely continue as networks become more secure.

One area that virtually nobody foresaw is electronic flea markets (e-flea?). Online auctions of second-hand goods have become a massive industry. Unlike traditional e-commerce, which follows the client-server model, online auctions are peer-to-peer in the sense that consumers can act as both buyers and sellers.

Some of these forms of e-commerce have acquired cute little tags based on the fact that "to" and "2" are pronounced the same. The most popular ones are listed in Fig. 1-4.

Tag	Full name	Example
B2C	Business-to-consumer	Ordering books online
B2B	Business-to-business	Car manufacturer ordering tires from supplier
G2C	Government-to-consumer	Government distributing tax forms electronically
C2C	Consumer-to-consumer	Auctioning second-hand products online
P2P	Peer-to-peer	Music sharing

Figure 1-4. Some forms of e-commerce.

Our fourth category is entertainment. This has made huge strides in the home in recent years, with the distribution of music, radio and television programs, and movies over the Internet beginning to rival that of traditional mechanisms. Users can find, buy, and download MP3 songs and DVD-quality movies and add them to their personal collection. TV shows now reach many homes via **IPTV** (**IP TeleVision**) systems that are based on IP technology instead of cable TV or radio transmissions. Media streaming applications let users tune into Internet radio stations or watch recent episodes of their favorite TV shows. Naturally, all of this content can be moved around your house between different devices, displays and speakers, usually with a wireless network.

Soon, it may be possible to search for any movie or television program ever made, in any country, and have it displayed on your screen instantly. New films may become interactive, where the user is occasionally prompted for the story direction (should Macbeth murder Duncan or just bide his time?) with alternative scenarios provided for all cases. Live television may also become interactive, with the audience participating in quiz shows, choosing among contestants, and so on.

Another form of entertainment is game playing. Already we have multiperson real-time simulation games, like hide-and-seek in a virtual dungeon, and flight

simulators with the players on one team trying to shoot down the players on the opposing team. Virtual worlds provide a persistent setting in which thousands of users can experience a shared reality with three-dimensional graphics.

Our last category is **ubiquitous computing**, in which computing is embedded into everyday life, as in the vision of Mark Weiser (1991). Many homes are already wired with security systems that include door and window sensors, and there are many more sensors that can be folded in to a smart home monitor, such as energy consumption. Your electricity, gas and water meters could also report usage over the network. This would save money as there would be no need to send out meter readers. And your smoke detectors could call the fire department instead of making a big noise (which has little value if no one is home). As the cost of sensing and communication drops, more and more measurement and reporting will be done with networks.

Increasingly, consumer electronic devices are networked. For example, some high-end cameras already have a wireless network capability and use it to send photos to a nearby display for viewing. Professional sports photographers can also send their photos to their editors in real-time, first wirelessly to an access point then over the Internet. Devices such as televisions that plug into the wall can use **power-line networks** to send information throughout the house over the wires that carry electricity. It may not be very surprising to have these objects on the network, but objects that we do not think of as computers may sense and communicate information too. For example, your shower may record water usage, give you visual feedback while you lather up, and report to a home environmental monitoring application when you are done to help save on your water bill.

A technology called **RFID** (**Radio Frequency IDentification**) will push this idea even further in the future. RFID tags are passive (i.e., have no battery) chips the size of stamps and they can already be affixed to books, passports, pets, credit cards, and other items in the home and out. This lets RFID readers locate and communicate with the items over a distance of up to several meters, depending on the kind of RFID. Originally, RFID was commercialized to replace barcodes. It has not succeeded yet because barcodes are free and RFID tags cost a few cents. Of course, RFID tags offer much more and their price is rapidly declining. They may turn the real world into the Internet of things (ITU, 2005).

1.1.3 Mobile Users

Mobile computers, such as laptop and handheld computers, are one of the fastest-growing segments of the computer industry. Their sales have already overtaken those of desktop computers. Why would anyone want one? People on the go often want to use their mobile devices to read and send email, tweet, watch movies, download music, play games, or simply to surf the Web for information. They want to do all of the things they do at home and in the office. Naturally, they want to do them from anywhere on land, sea or in the air.

Connectivity to the Internet enables many of these mobile uses. Since having a wired connection is impossible in cars, boats, and airplanes, there is a lot of interest in wireless networks. Cellular networks operated by the telephone companies are one familiar kind of wireless network that blankets us with coverage for mobile phones. Wireless **hotspots** based on the 802.11 standard are another kind of wireless network for mobile computers. They have sprung up everywhere that people go, resulting in a patchwork of coverage at cafes, hotels, airports, schools, trains and planes. Anyone with a laptop computer and a wireless modem can just turn on their computer on and be connected to the Internet through the hotspot, as though the computer were plugged into a wired network.

Wireless networks are of great value to fleets of trucks, taxis, delivery vehicles, and repairpersons for keeping in contact with their home base. For example, in many cities, taxi drivers are independent businessmen, rather than being employees of a taxi company. In some of these cities, the taxis have a display the driver can see. When a customer calls up, a central dispatcher types in the pickup and destination points. This information is displayed on the drivers' displays and a beep sounds. The first driver to hit a button on the display gets the call.

Wireless networks are also important to the military. If you have to be able to fight a war anywhere on Earth at short notice, counting on using the local networking infrastructure is probably not a good idea. It is better to bring your own.

Although wireless networking and mobile computing are often related, they are not identical, as Fig. 1-5 shows. Here we see a distinction between **fixed wireless** and **mobile wireless** networks. Even notebook computers are sometimes wired. For example, if a traveler plugs a notebook computer into the wired network jack in a hotel room, he has mobility without a wireless network.

Wireless	Mobile	Typical applications
No	No	Desktop computers in offices
No	Yes	A notebook computer used in a hotel room
Yes	No	Networks in unwired buildings
Yes	Yes	Store inventory with a handheld computer

Figure 1-5. Combinations of wireless networks and mobile computing.

Conversely, some wireless computers are not mobile. In the home, and in offices or hotels that lack suitable cabling, it can be more convenient to connect desktop computers or media players wirelessly than to install wires. Installing a wireless network may require little more than buying a small box with some electronics in it, unpacking it, and plugging it in. This solution may be far cheaper than having workmen put in cable ducts to wire the building.

Finally, there are also true mobile, wireless applications, such as people walking around stores with a handheld computers recording inventory. At many busy

airports, car rental return clerks work in the parking lot with wireless mobile computers. They scan the barcodes or RFID chips of returning cars, and their mobile device, which has a built-in printer, calls the main computer, gets the rental information, and prints out the bill on the spot.

Perhaps the key driver of mobile, wireless applications is the mobile phone. **Text messaging** or **texting** is tremendously popular. It lets a mobile phone user type a short message that is then delivered by the cellular network to another mobile subscriber. Few people would have predicted ten years ago that having teenagers tediously typing short text messages on mobile phones would be an immense money maker for telephone companies. But texting (or **Short Message Service** as it is known outside the U.S.) is very profitable since it costs the carrier but a tiny fraction of one cent to relay a text message, a service for which they charge far more.

The long-awaited convergence of telephones and the Internet has finally arrived, and it will accelerate the growth of mobile applications. **Smart phones**, such as the popular iPhone, combine aspects of mobile phones and mobile computers. The (3G and 4G) cellular networks to which they connect can provide fast data services for using the Internet as well as handling phone calls. Many advanced phones connect to wireless hotspots too, and automatically switch between networks to choose the best option for the user.

Other consumer electronics devices can also use cellular and hotspot networks to stay connected to remote computers. Electronic book readers can download a newly purchased book or the next edition of a magazine or today's newspaper wherever they roam. Electronic picture frames can update their displays on cue with fresh images.

Since mobile phones know their locations, often because they are equipped with **GPS** (**Global Positioning System**) receivers, some services are intentionally location dependent. Mobile maps and directions are an obvious candidate as your GPS-enabled phone and car probably have a better idea of where you are than you do. So, too, are searches for a nearby bookstore or Chinese restaurant, or a local weather forecast. Other services may record location, such as annotating photos and videos with the place at which they were made. This annotation is known as "geo-tagging."

An area in which mobile phones are now starting to be used is **m-commerce** (**mobile-commerce**) (Senn, 2000). Short text messages from the mobile are used to authorize payments for food in vending machines, movie tickets, and other small items instead of cash and credit cards. The charge then appears on the mobile phone bill. When equipped with **NFC** (**Near Field Communication**) technology the mobile can act as an RFID smartcard and interact with a nearby reader for payment. The driving forces behind this phenomenon are the mobile device makers and network operators, who are trying hard to figure out how to get a piece of the e-commerce pie. From the store's point of view, this scheme may save them most of the credit card company's fee, which can be several percent.

Of course, this plan may backfire, since customers in a store might use the RFID or barcode readers on their mobile devices to check out competitors' prices before buying and use them to get a detailed report on where else an item can be purchased nearby and at what price.

One huge thing that m-commerce has going for it is that mobile phone users are accustomed to paying for everything (in contrast to Internet users, who expect everything to be free). If an Internet Web site charged a fee to allow its customers to pay by credit card, there would be an immense howling noise from the users. If, however, a mobile phone operator its customers to pay for items in a store by waving the phone at the cash register and then tacked on a fee for this convenience, it would probably be accepted as normal. Time will tell.

No doubt the uses of mobile and wireless computers will grow rapidly in the future as the size of computers shrinks, probably in ways no one can now foresee. Let us take a quick look at some possibilities. **Sensor networks** are made up of nodes that gather and wirelessly relay information they sense about the state of the physical world. The nodes may be part of familiar items such as cars or phones, or they may be small separate devices. For example, your car might gather data on its location, speed, vibration, and fuel efficiency from its on-board diagnostic system and upload this information to a database (Hull et al., 2006). Those data can help find potholes, plan trips around congested roads, and tell you if you are a ''gas guzzler'' compared to other drivers on the same stretch of road.

Sensor networks are revolutionizing science by providing a wealth of data on behavior that could not previously be observed. One example is tracking the migration of individual zebras by placing a small sensor on each animal (Juang et al., 2002). Researchers have packed a wireless computer into a cube 1 mm on edge (Warneke et al., 2001). With mobile computers this small, even small birds, rodents, and insects can be tracked.

Even mundane uses, such as in parking meters, can be significant because they make use of data that were not previously available. Wireless parking meters can accept credit or debit card payments with instant verification over the wireless link. They can also report when they are in use over the wireless network. This would let drivers download a recent parking map to their car so they can find an available spot more easily. Of course, when a meter expires, it might also check for the presence of a car (by bouncing a signal off it) and report the expiration to parking enforcement. It has been estimated that city governments in the U.S. alone could collect an additional $10 billion this way (Harte et al., 2000).

Wearable computers are another promising application. Smart watches with radios have been part of our mental space since their appearance in the Dick Tracy comic strip in 1946; now you can buy them. Other such devices may be implanted, such as pacemakers and insulin pumps. Some of these can be controlled over a wireless network. This lets doctors test and reconfigure them more easily. It could also lead to some nasty problems if the devices are as insecure as the average PC and can be hacked easily (Halperin et al., 2008).

1.1.4 Social Issues

Computer networks, like the printing press 500 years ago, allow ordinary citizens to distribute and view content in ways that were not previously possible. But along with the good comes the bad, as this new-found freedom brings with it many unsolved social, political, and ethical issues. Let us just briefly mention a few of them; a thorough study would require a full book, at least.

Social networks, message boards, content sharing sites, and a host of other applications allow people to share their views with like-minded individuals. As long as the subjects are restricted to technical topics or hobbies like gardening, not too many problems will arise.

The trouble comes with topics that people actually care about, like politics, religion, or sex. Views that are publicly posted may be deeply offensive to some people. Worse yet, they may not be politically correct. Furthermore, opinions need not be limited to text; high-resolution color photographs and video clips are easily shared over computer networks. Some people take a live-and-let-live view, but others feel that posting certain material (e.g., verbal attacks on particular countries or religions, pornography, etc.) is simply unacceptable and that such content must be censored. Different countries have different and conflicting laws in this area. Thus, the debate rages.

In the past, people have sued network operators, claiming that they are responsible for the contents of what they carry, just as newspapers and magazines are. The inevitable response is that a network is like a telephone company or the post office and cannot be expected to police what its users say.

It should now come only as a slight surprise to learn that some network operators block content for their own reasons. Some users of peer-to-peer applications had their network service cut off because the network operators did not find it profitable to carry the large amounts of traffic sent by those applications. Those same operators would probably like to treat different companies differently. If you are a big company and pay well then you get good service, but if you are a small-time player, you get poor service. Opponents of this practice argue that peer-to-peer and other content should be treated in the same way because they are all just bits to the network. This argument for communications that are not differentiated by their content or source or who is providing the content is known as **network neutrality** (Wu, 2003). It is probably safe to say that this debate will go on for a while.

Many other parties are involved in the tussle over content. For instance, pirated music and movies fueled the massive growth of peer-to-peer networks, which did not please the copyright holders, who have threatened (and sometimes taken) legal action. There are now automated systems that search peer-to-peer networks and fire off warnings to network operators and users who are suspected of infringing copyright. In the United States, these warnings are known as **DMCA takedown notices** after the **Digital Millennium Copyright Act**. This

search is an arms' race because it is hard to reliably catch copyright infringement. Even your printer might be mistaken for a culprit (Piatek et al., 2008).

Computer networks make it very easy to communicate. They also make it easy for the people who run the network to snoop on the traffic. This sets up conflicts over issues such as employee rights versus employer rights. Many people read and write email at work. Many employers have claimed the right to read and possibly censor employee messages, including messages sent from a home computer outside working hours. Not all employees agree with this, especially the latter part.

Another conflict is centered around government versus citizen's rights. The FBI has installed systems at many Internet service providers to snoop on all incoming and outgoing email for nuggets of interest. One early system was originally called Carnivore, but bad publicity caused it to be renamed to the more innocent-sounding DCS1000 (Blaze and Bellovin, 2000; Sobel, 2001; and Zacks, 2001). The goal of such systems is to spy on millions of people in the hope of perhaps finding information about illegal activities. Unfortunately for the spies, the Fourth Amendment to the U.S. Constitution prohibits government searches without a search warrant, but the government often ignores it.

Of course, the government does not have a monopoly on threatening people's privacy. The private sector does its bit too by **profiling** users. For example, small files called **cookies** that Web browsers store on users' computers allow companies to track users' activities in cyberspace and may also allow credit card numbers, social security numbers, and other confidential information to leak all over the Internet (Berghel, 2001). Companies that provide Web-based services may maintain large amounts of personal information about their users that allows them to study user activities directly. For example, Google can read your email and show you advertisements based on your interests if you use its email service, **Gmail**.

A new twist with mobile devices is location privacy (Beresford and Stajano, 2003). As part of the process of providing service to your mobile device the network operators learn where you are at different times of day. This allows them to track your movements. They may know which nightclub you frequent and which medical center you visit.

Computer networks also offer the potential to increase privacy by sending anonymous messages. In some situations, this capability may be desirable. Beyond preventing companies from learning your habits, it provides, for example, a way for students, soldiers, employees, and citizens to blow the whistle on illegal behavior on the part of professors, officers, superiors, and politicians without fear of reprisals. On the other hand, in the United States and most other democracies, the law specifically permits an accused person the right to confront and challenge his accuser in court so anonymous accusations cannot be used as evidence.

The Internet makes it possible to find information quickly, but a great deal of it is ill considered, misleading, or downright wrong. That medical advice you

plucked from the Internet about the pain in your chest may have come from a Nobel Prize winner or from a high-school dropout.

Other information is frequently unwanted. Electronic junk mail (spam) has become a part of life because spammers have collected millions of email addresses and would-be marketers can cheaply send computer-generated messages to them. The resulting flood of spam rivals the flow messages from real people. Fortunately, filtering software is able to read and discard the spam generated by other computers, with lesser or greater degrees of success.

Still other content is intended for criminal behavior. Web pages and email messages containing active content (basically, programs or macros that execute on the receiver's machine) can contain viruses that take over your computer. They might be used to steal your bank account passwords, or to have your computer send spam as part of a **botnet** or pool of compromised machines.

Phishing messages masquerade as originating from a trustworthy party, for example, your bank, to try to trick you into revealing sensitive information, for example, credit card numbers. Identity theft is becoming a serious problem as thieves collect enough information about a victim to obtain credit cards and other documents in the victim's name.

It can be difficult to prevent computers from impersonating people on the Internet. This problem has led to the development of **CAPTCHAs**, in which a computer asks a person to solve a short recognition task, for example, typing in the letters shown in a distorted image, to show that they are human (von Ahn, 2001). This process is a variation on the famous Turing test in which a person asks questions over a network to judge whether the entity responding is human.

A lot of these problems could be solved if the computer industry took computer security seriously. If all messages were encrypted and authenticated, it would be harder to commit mischief. Such technology is well established and we will study it in detail in Chap. 8. The problem is that hardware and software vendors know that putting in security features costs money and their customers are not demanding such features. In addition, a substantial number of the problems are caused by buggy software, which occurs because vendors keep adding more and more features to their programs, which inevitably means more code and thus more bugs. A tax on new features might help, but that might be a tough sell in some quarters. A refund for defective software might be nice, except it would bankrupt the entire software industry in the first year.

Computer networks raise new legal problems when they interact with old laws. Electronic gambling provides an example. Computers have been simulating things for decades, so why not simulate slot machines, roulette wheels, blackjack dealers, and more gambling equipment? Well, because it is illegal in a lot of places. The trouble is, gambling is legal in a lot of other places (England, for example) and casino owners there have grasped the potential for Internet gambling. What happens if the gambler, the casino, and the server are all in different countries, with conflicting laws? Good question.

1.2 NETWORK HARDWARE

It is now time to turn our attention from the applications and social aspects of networking (the dessert) to the technical issues involved in network design (the spinach). There is no generally accepted taxonomy into which all computer networks fit, but two dimensions stand out as important: transmission technology and scale. We will now examine each of these in turn.

Broadly speaking, there are two types of transmission technology that are in widespread use: **broadcast** links and **point-to-point** links.

Point-to-point links connect individual pairs of machines. To go from the source to the destination on a network made up of point-to-point links, short messages, called **packets** in certain contexts, may have to first visit one or more intermediate machines. Often multiple routes, of different lengths, are possible, so finding good ones is important in point-to-point networks. Point-to-point transmission with exactly one sender and exactly one receiver is sometimes called **unicasting**.

In contrast, on a broadcast network, the communication channel is shared by all the machines on the network; packets sent by any machine are received by all the others. An address field within each packet specifies the intended recipient. Upon receiving a packet, a machine checks the address field. If the packet is intended for the receiving machine, that machine processes the packet; if the packet is intended for some other machine, it is just ignored.

A wireless network is a common example of a broadcast link, with communication shared over a coverage region that depends on the wireless channel and the transmitting machine. As an analogy, consider someone standing in a meeting room and shouting "Watson, come here. I want you." Although the packet may actually be received (heard) by many people, only Watson will respond; the others just ignore it.

Broadcast systems usually also allow the possibility of addressing a packet to *all* destinations by using a special code in the address field. When a packet with this code is transmitted, it is received and processed by every machine on the network. This mode of operation is called **broadcasting**. Some broadcast systems also support transmission to a subset of the machines, which known as **multicasting**.

An alternative criterion for classifying networks is by scale. Distance is important as a classification metric because different technologies are used at different scales.

In Fig. 1-6 we classify multiple processor systems by their rough physical size. At the top are the personal area networks, networks that are meant for one person. Beyond these come longer-range networks. These can be divided into local, metropolitan, and wide area networks, each with increasing scale. Finally, the connection of two or more networks is called an internetwork. The worldwide Internet is certainly the best-known (but not the only) example of an internetwork.

Patent Owner's Ex. 2010
Page 033

Soon we will have even larger internetworks with the **Interplanetary Internet** that connects networks across space (Burleigh et al., 2003).

Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

Figure 1-6. Classification of interconnected processors by scale.

In this book we will be concerned with networks at all these scales. In the following sections, we give a brief introduction to network hardware by scale.

1.2.1 Personal Area Networks

PANs (**Personal Area Networks**) let devices communicate over the range of a person. A common example is a wireless network that connects a computer with its peripherals. Almost every computer has an attached monitor, keyboard, mouse, and printer. Without using wireless, this connection must be done with cables. So many new users have a hard time finding the right cables and plugging them into the right little holes (even though they are usually color coded) that most computer vendors offer the option of sending a technician to the user's home to do it. To help these users, some companies got together to design a short-range wireless network called **Bluetooth** to connect these components without wires. The idea is that if your devices have Bluetooth, then you need no cables. You just put them down, turn them on, and they work together. For many people, this ease of operation is a big plus.

In the simplest form, Bluetooth networks use the master-slave paradigm of Fig. 1-7. The system unit (the PC) is normally the master, talking to the mouse, keyboard, etc., as slaves. The master tells the slaves what addresses to use, when they can broadcast, how long they can transmit, what frequencies they can use, and so on.

Bluetooth can be used in other settings, too. It is often used to connect a headset to a mobile phone without cords and it can allow your digital music player

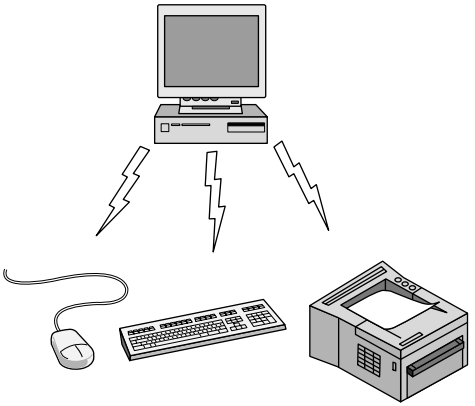


Figure 1-7. Bluetooth PAN configuration.

to connect to your car merely being brought within range. A completely different kind of PAN is formed when an embedded medical device such as a pacemaker, insulin pump, or hearing aid talks to a user-operated remote control. We will discuss Bluetooth in more detail in Chap. 4.

PANs can also be built with other technologies that communicate over short ranges, such as RFID on smartcards and library books. We will study RFID in Chap. 4.

1.2.2 Local Area Networks

The next step up is the **LAN** (**Local Area Network**). A LAN is a privately owned network that operates within and nearby a single building like a home, office or factory. LANs are widely used to connect personal computers and consumer electronics to let them share resources (e.g., printers) and exchange information. When LANs are used by companies, they are called **enterprise networks**.

Wireless LANs are very popular these days, especially in homes, older office buildings, cafeterias, and other places where it is too much trouble to install cables. In these systems, every computer has a radio modem and an antenna that it uses to communicate with other computers. In most cases, each computer talks to a device in the ceiling as shown in Fig. 1-8(a). This device, called an **AP** (**Access Point**), **wireless router**, or **base station**, relays packets between the wireless computers and also between them and the Internet. Being the AP is like being the popular kid as school because everyone wants to talk to you. However, if other computers are close enough, they can communicate directly with one another in a peer-to-peer configuration.

There is a standard for wireless LANs called **IEEE 802.11**, popularly known as **WiFi**, which has become very widespread. It runs at speeds anywhere from 11

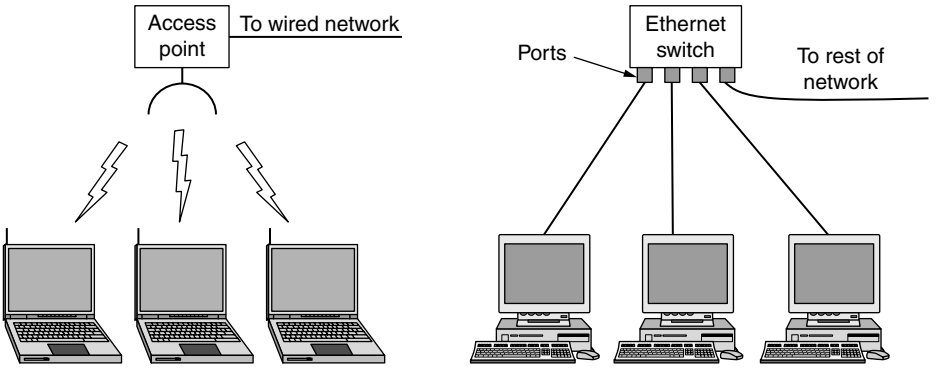


Figure 1-8. Wireless and wired LANs. (a) 802.11. (b) Switched Ethernet.

to hundreds of Mbps. (In this book we will adhere to tradition and measure line speeds in megabits/sec, where 1 Mbps is 1,000,000 bits/sec, and gigabits/sec, where 1 Gbps is 1,000,000,000 bits/sec.) We will discuss 802.11 in Chap. 4.

Wired LANs use a range of different transmission technologies. Most of them use copper wires, but some use optical fiber. LANs are restricted in size, which means that the worst-case transmission time is bounded and known in advance. Knowing these bounds helps with the task of designing network protocols. Typically, wired LANs run at speeds of 100 Mbps to 1 Gbps, have low delay (microseconds or nanoseconds), and make very few errors. Newer LANs can operate at up to 10 Gbps. Compared to wireless networks, wired LANs exceed them in all dimensions of performance. It is just easier to send signals over a wire or through a fiber than through the air.

The topology of many wired LANs is built from point-to-point links. IEEE 802.3, popularly called **Ethernet**, is, by far, the most common type of wired LAN. Fig. 1-8(b) shows a sample topology of **switched Ethernet**. Each computer speaks the Ethernet protocol and connects to a box called a **switch** with a point-to-point link. Hence the name. A switch has multiple **ports**, each of which can connect to one computer. The job of the switch is to relay packets between computers that are attached to it, using the address in each packet to determine which computer to send it to.

To build larger LANs, switches can be plugged into each other using their ports. What happens if you plug them together in a loop? Will the network still work? Luckily, the designers thought of this case. It is the job of the protocol to sort out what paths packets should travel to safely reach the intended computer. We will see how this works in Chap. 4.

It is also possible to divide one large physical LAN into two smaller logical LANs. You might wonder why this would be useful. Sometimes, the layout of the network equipment does not match the organization's structure. For example, the

engineering and finance departments of a company might have computers on the same physical LAN because they are in the same wing of the building but it might be easier to manage the system if engineering and finance logically each had its own network **Virtual LAN** or **VLAN**. In this design each port is tagged with a "color," say green for engineering and red for finance. The switch then forwards packets so that computers attached to the green ports are separated from the computers attached to the red ports. Broadcast packets sent on a red port, for example, will not be received on a green port, just as though there were two different LANs. We will cover VLANs at the end of Chap. 4.

There are other wired LAN topologies too. In fact, switched Ethernet is a modern version of the original Ethernet design that broadcast all the packets over a single linear cable. At most one machine could successfully transmit at a time, and a distributed arbitration mechanism was used to resolve conflicts. It used a simple algorithm: computers could transmit whenever the cable was idle. If two or more packets collided, each computer just waited a random time and tried later. We will call that version **classic Ethernet** for clarity, and as you suspected, you will learn about it in Chap. 4.

Both wireless and wired broadcast networks can be divided into static and dynamic designs, depending on how the channel is allocated. A typical static allocation would be to divide time into discrete intervals and use a round-robin algorithm, allowing each machine to broadcast only when its time slot comes up. Static allocation wastes channel capacity when a machine has nothing to say during its allocated slot, so most systems attempt to allocate the channel dynamically (i.e., on demand).

Dynamic allocation methods for a common channel are either centralized or decentralized. In the centralized channel allocation method, there is a single entity, for example, the base station in cellular networks, which determines who goes next. It might do this by accepting multiple packets and prioritizing them according to some internal algorithm. In the decentralized channel allocation method, there is no central entity; each machine must decide for itself whether to transmit. You might think that this approach would lead to chaos, but it does not. Later we will study many algorithms designed to bring order out of the potential chaos.

It is worth spending a little more time discussing LANs in the home. In the future, it is likely that every appliance in the home will be capable of communicating with every other appliance, and all of them will be accessible over the Internet. This development is likely to be one of those visionary concepts that nobody asked for (like TV remote controls or mobile phones), but once they arrived nobody can imagine how they lived without them.

Many devices are already capable of being networked. These include computers, entertainment devices such as TVs and DVDs, phones and other consumer electronics such as cameras, appliances like clock radios, and infrastructure like utility meters and thermostats. This trend will only continue. For instance, the average home probably has a dozen clocks (e.g., in appliances), all of which could

adjust to daylight savings time automatically if the clocks were on the Internet. Remote monitoring of the home is a likely winner, as many grown children would be willing to spend some money to help their aging parents live safely in their own homes.

While we could think of the home network as just another LAN, it is more likely to have different properties than other networks. First, the networked devices have to be very easy to install. Wireless routers are the most returned consumer electronic item. People buy one because they want a wireless network at home, find that it does not work "out of the box," and then return it rather than listen to elevator music while on hold on the technical helpline.

Second, the network and devices have to be foolproof in operation. Air conditioners used to have one knob with four settings: OFF, LOW, MEDIUM, and HIGH. Now they have 30-page manuals. Once they are networked, expect the chapter on security alone to be 30 pages. This is a problem because only computer users are accustomed to putting up with products that do not work; the car-, television-, and refrigerator-buying public is far less tolerant. They expect products to work 100% without the need to hire a geek.

Third, low price is essential for success. People will not pay a $50 premium for an Internet thermostat because few people regard monitoring their home temperature from work that important. For $5 extra, though, it might sell.

Fourth, it must be possible to start out with one or two devices and expand the reach of the network gradually. This means no format wars. Telling consumers to buy peripherals with IEEE 1394 (FireWire) interfaces and a few years later retracting that and saying USB 2.0 is the interface-of-the-month and then switching that to 802.11g—oops, no, make that 802.11n—I mean 802.16 (different wireless networks)—is going to make consumers very skittish. The network interface will have to remain stable for decades, like the television broadcasting standards.

Fifth, security and reliability will be very important. Losing a few files to an email virus is one thing; having a burglar disarm your security system from his mobile computer and then plunder your house is something quite different.

An interesting question is whether home networks will be wired or wireless. Convenience and cost favors wireless networking because there are no wires to fit, or worse, retrofit. Security favors wired networking because the radio waves that wireless networks use are quite good at going through walls. Not everyone is overjoyed at the thought of having the neighbors piggybacking on their Internet connection and reading their email. In Chap. 8 we will study how encryption can be used to provide security, but it is easier said than done with inexperienced users.

A third option that may be appealing is to reuse the networks that are already in the home. The obvious candidate is the electric wires that are installed throughout the house. **Power-line networks** let devices that plug into outlets broadcast information throughout the house. You have to plug in the TV anyway, and this way it can get Internet connectivity at the same time. The difficulty is

how to carry both power and data signals at the same time. Part of the answer is that they use different frequency bands.

In short, home LANs offer many opportunities and challenges. Most of the latter relate to the need for the networks to be easy to manage, dependable, and secure, especially in the hands of nontechnical users, as well as low cost.

1.2.3 Metropolitan Area Networks

A **MAN** (**Metropolitan Area Network**) covers a city. The best-known examples of MANs are the cable television networks available in many cities. These systems grew from earlier community antenna systems used in areas with poor over-the-air television reception. In those early systems, a large antenna was placed on top of a nearby hill and a signal was then piped to the subscribers' houses.

At first, these were locally designed, ad hoc systems. Then companies began jumping into the business, getting contracts from local governments to wire up entire cities. The next step was television programming and even entire channels designed for cable only. Often these channels were highly specialized, such as all news, all sports, all cooking, all gardening, and so on. But from their inception until the late 1990s, they were intended for television reception only.

When the Internet began attracting a mass audience, the cable TV network operators began to realize that with some changes to the system, they could provide two-way Internet service in unused parts of the spectrum. At that point, the cable TV system began to morph from simply a way to distribute television to a metropolitan area network. To a first approximation, a MAN might look something like the system shown in Fig. 1-9. In this figure we see both television signals and Internet being fed into the centralized **cable headend** for subsequent distribution to people's homes. We will come back to this subject in detail in Chap. 2.

Cable television is not the only MAN, though. Recent developments in high-speed wireless Internet access have resulted in another MAN, which has been standardized as IEEE 802.16 and is popularly known as **WiMAX**. We will look at it in Chap. 4.

1.2.4 Wide Area Networks

A **WAN** (**Wide Area Network**) spans a large geographical area, often a country or continent. We will begin our discussion with wired WANs, using the example of a company with branch offices in different cities.

The WAN in Fig. 1-10 is a network that connects offices in Perth, Melbourne, and Brisbane. Each of these offices contains computers intended for running user (i.e., application) programs. We will follow traditional usage and call these machines **hosts**. The rest of the network that connects these hosts is then called the

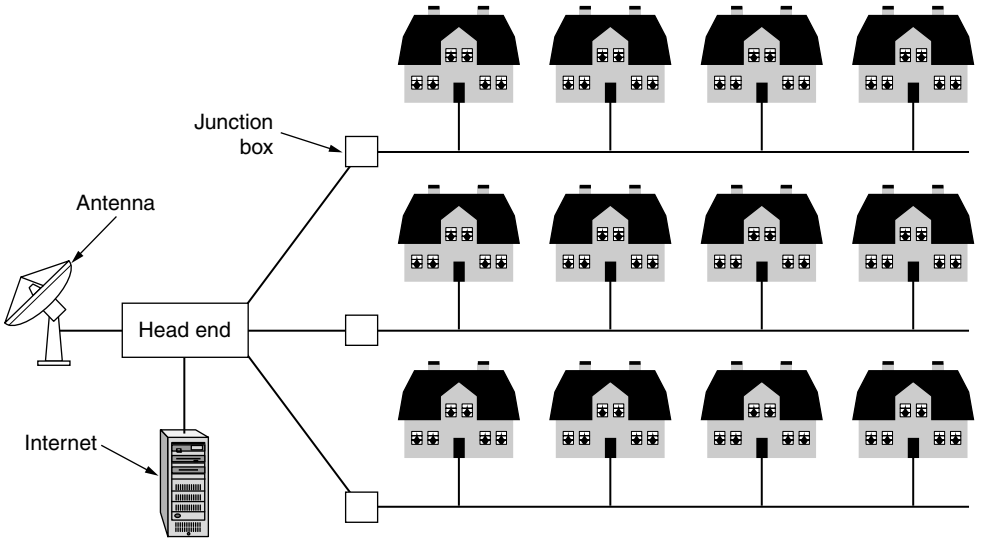


Figure 1-9. A metropolitan area network based on cable TV.

communication subnet, or just **subnet** for short. The job of the subnet is to carry messages from host to host, just as the telephone system carries words (really just sounds) from speaker to listener.

In most WANs, the subnet consists of two distinct components: transmission lines and switching elements. **Transmission lines** move bits between machines. They can be made of copper wire, optical fiber, or even radio links. Most companies do not have transmission lines lying about, so instead they lease the lines from a telecommunications company. **Switching elements**, or just **switches**, are specialized computers that connect two or more transmission lines. When data arrive on an incoming line, the switching element must choose an outgoing line on which to forward them. These switching computers have been called by various names in the past; the name **router** is now most commonly used. Unfortunately, some people pronounce it ''rooter'' while others have it rhyme with ''doubter.'' Determining the correct pronunciation will be left as an exercise for the reader. (Note: the perceived correct answer may depend on where you live.)

A short comment about the term ''subnet'' is in order here. Originally, its **only** meaning was the collection of routers and communication lines that moved packets from the source host to the destination host. Readers should be aware that it has acquired a second, more recent meaning in conjunction with network addressing. We will discuss that meaning in Chap. 5 and stick with the original meaning (a collection of lines and routers) until then.

The WAN as we have described it looks similar to a large wired LAN, but there are some important differences that go beyond long wires. Usually in a WAN, the hosts and subnet are owned and operated by different people. In our

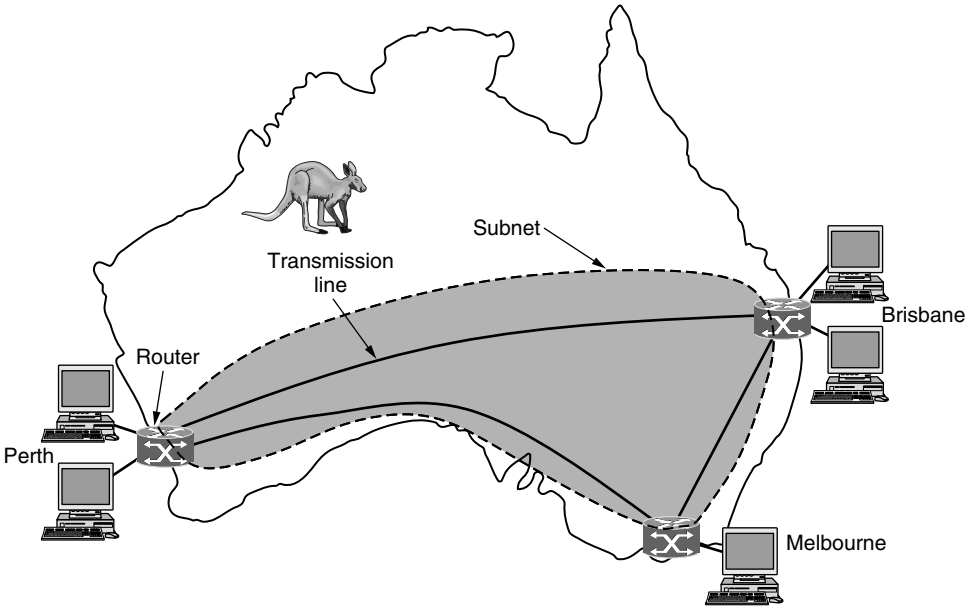


Figure 1-10. WAN that connects three branch offices in Australia.

example, the employees might be responsible for their own computers, while the company's IT department is in charge of the rest of the network. We will see clearer boundaries in the coming examples, in which the network provider or telephone company operates the subnet. Separation of the pure communication aspects of the network (the subnet) from the application aspects (the hosts) greatly simplifies the overall network design.

A second difference is that the routers will usually connect different kinds of networking technology. The networks inside the offices may be switched Ethernet, for example, while the long-distance transmission lines may be SONET links (which we will cover in Chap. 2). Some device needs to join them. The astute reader will notice that this goes beyond our definition of a network. This means that many WANs will in fact be **internetworks**, or composite networks that are made up of more than one network. We will have more to say about internetworks in the next section.

A final difference is in what is connected to the subnet. This could be individual computers, as was the case for connecting to LANs, or it could be entire LANs. This is how larger networks are built from smaller ones. As far as the subnet is concerned, it does the same job.

We are now in a position to look at two other varieties of WANs. First, rather than lease dedicated transmission lines, a company might connect its offices to the Internet This allows connections to be made between the offices as virtual links

Patent Owner's Ex. 2010
Page 041

that use the underlying capacity of the Internet. This arrangement, shown in Fig. 1-11, is called a **VPN** (**Virtual Private Network**). Compared to the dedicated arrangement, a VPN has the usual advantage of virtualization, which is that it provides flexible reuse of a resource (Internet connectivity). Consider how easy it is to add a fourth office to see this. A VPN also has the usual disadvantage of virtualization, which is a lack of control over the underlying resources. With a dedicated line, the capacity is clear. With a VPN your mileage may vary with your Internet service.

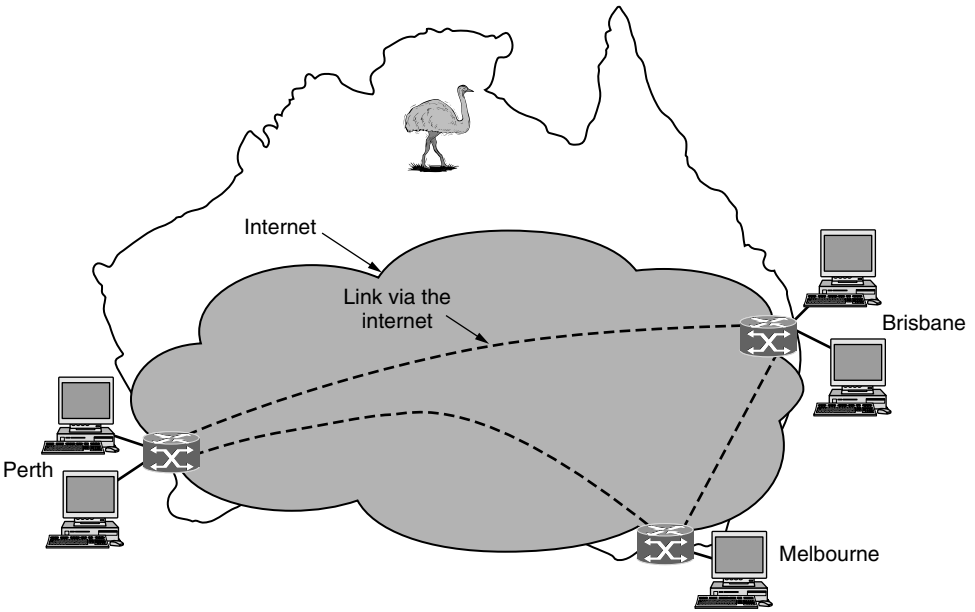


Figure 1-11. WAN using a virtual private network.

The second variation is that the subnet may be run by a different company. The subnet operator is known as a **network service provider** and the offices are its customers. This structure is shown in Fig. 1-12. The subnet operator will connect to other customers too, as long as they can pay and it can provide service. Since it would be a disappointing network service if the customers could only send packets to each other, the subnet operator will also connect to other networks that are part of the Internet. Such a subnet operator is called an **ISP** (**Internet Service Provider**) and the subnet is an **ISP network**. Its customers who connect to the ISP receive Internet service.

We can use the ISP network to preview some key issues that we will study in later chapters. In most WANs, the network contains many transmission lines, each connecting a pair of routers. If two routers that do not share a transmission line wish to communicate, they must do this indirectly, via other routers. There

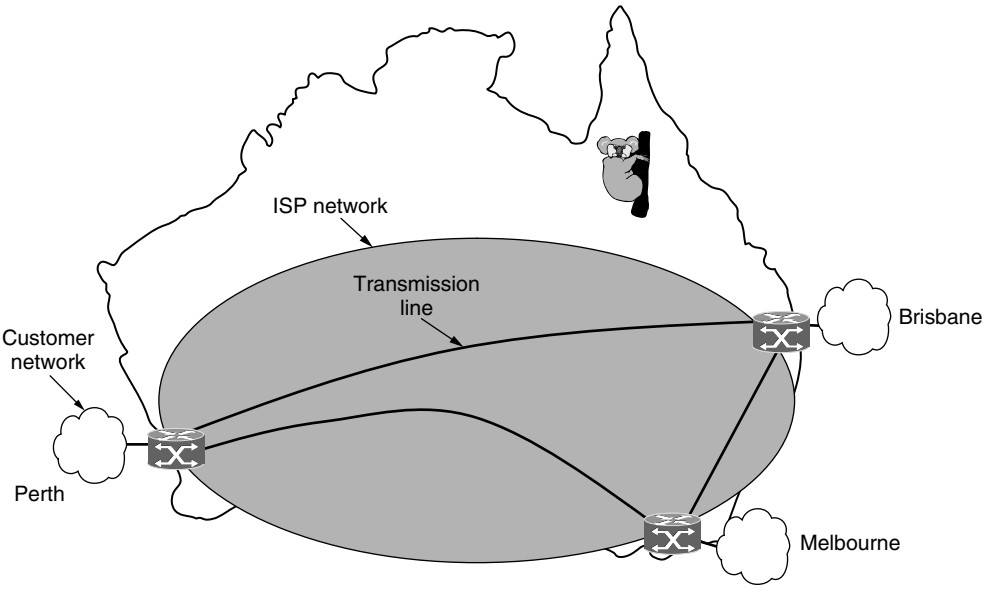


Figure 1-12. WAN using an ISP network.

may be many paths in the network that connect these two routers. How the network makes the decision as to which path to use is called the **routing algorithm**. Many such algorithms exist. How each router makes the decision as to where to send a packet next is called the **forwarding algorithm**. Many of them exist too. We will study some of both types in detail in Chap. 5.

Other kinds of WANs make heavy use of wireless technologies. In satellite systems, each computer on the ground has an antenna through which it can send data to and receive data from to a satellite in orbit. All computers can hear the output *from* the satellite, and in some cases they can also hear the upward transmissions of their fellow computers *to* the satellite as well. Satellite networks are inherently broadcast and are most useful when the broadcast property is important.

The cellular telephone network is another example of a WAN that uses wireless technology. This system has already gone through three generations and a fourth one is on the horizon. The first generation was analog and for voice only. The second generation was digital and for voice only. The third generation is digital and is for both voice and data. Each cellular base station covers a distance much larger than a wireless LAN, with a range measured in kilometers rather than tens of meters. The base stations are connected to each other by a backbone network that is usually wired. The data rates of cellular networks are often on the order of 1 Mbps, much smaller than a wireless LAN that can range up to on the order of 100 Mbps. We will have a lot to say about these networks in Chap. 2.

Patent Owner's Ex. 2010
Page 043

1.2.5 Internetworks

Many networks exist in the world, often with different hardware and software. People connected to one network often want to communicate with people attached to a different one. The fulfillment of this desire requires that different, and frequently incompatible, networks be connected. A collection of interconnected networks is called an **internetwork** or **internet**. These terms will be used in a generic sense, in contrast to the worldwide Internet (which is one specific internet), which we will always capitalize. The Internet uses ISP networks to connect enterprise networks, home networks, and many other networks. We will look at the Internet in great detail later in this book.

Subnets, networks, and internetworks are often confused. The term "subnet" makes the most sense in the context of a wide area network, where it refers to the collection of routers and communication lines owned by the network operator. As an analogy, the telephone system consists of telephone switching offices connected to one another by high-speed lines, and to houses and businesses by low-speed lines. These lines and equipment, owned and managed by the telephone company, form the subnet of the telephone system. The telephones themselves (the hosts in this analogy) are not part of the subnet.

A network is formed by the combination of a subnet and its hosts. However, the word "network" is often used in a loose sense as well. A subnet might be described as a network, as in the case of the "ISP network" of Fig. 1-12. An internetwork might also be described as a network, as in the case of the WAN in Fig. 1-10. We will follow similar practice, and if we are distinguishing a network from other arrangements, we will stick with our original definition of a collection of computers interconnected by a single technology.

Let us say more about what constitutes an internetwork. We know that an internet is formed when distinct networks are interconnected. In our view, connecting a LAN and a WAN or connecting two LANs is the usual way to form an internetwork, but there is little agreement in the industry over terminology in this area. There are two rules of thumb that are useful. First, if different organizations have paid to construct different parts of the network and each maintains its part, we have an internetwork rather than a single network. Second, if the underlying technology is different in different parts (e.g., broadcast versus point-to-point and wired versus wireless), we probably have an internetwork.

To go deeper, we need to talk about how two different networks can be connected. The general name for a machine that makes a connection between two or more networks and provides the necessary translation, both in terms of hardware and software, is a **gateway**. Gateways are distinguished by the layer at which they operate in the protocol hierarchy. We will have much more to say about layers and protocol hierarchies starting in the next section, but for now imagine that higher layers are more tied to applications, such as the Web, and lower layers are more tied to transmission links, such as Ethernet.

Patent Owner's Ex. 2010
Page 044

Since the benefit of forming an internet is to connect computers across networks, we do not want to use too low-level a gateway or we will be unable to make connections between different kinds of networks. We do not want to use too high-level a gateway either, or the connection will only work for particular applications. The level in the middle that is "just right" is often called the network layer, and a router is a gateway that switches packets at the network layer. We can now spot an internet by finding a network that has routers.

1.3 NETWORK SOFTWARE

The first computer networks were designed with the hardware as the main concern and the software as an afterthought. This strategy no longer works. Network software is now highly structured. In the following sections we examine the software structuring technique in some detail. The approach described here forms the keystone of the entire book and will occur repeatedly later on.

1.3.1 Protocol Hierarchies

To reduce their design complexity, most networks are organized as a stack of **layers** or **levels**, each one built upon the one below it. The number of layers, the name of each layer, the contents of each layer, and the function of each layer differ from network to network. The purpose of each layer is to offer certain services to the higher layers while shielding those layers from the details of how the offered services are actually implemented. In a sense, each layer is a kind of virtual machine, offering certain services to the layer above it.

This concept is actually a familiar one and is used throughout computer science, where it is variously known as information hiding, abstract data types, data encapsulation, and object-oriented programming. The fundamental idea is that a particular piece of software (or hardware) provides a service to its users but keeps the details of its internal state and algorithms hidden from them.

When layer *n* on one machine carries on a conversation with layer *n* on another machine, the rules and conventions used in this conversation are collectively known as the layer *n* protocol. Basically, a **protocol** is an agreement between the communicating parties on how communication is to proceed. As an analogy, when a woman is introduced to a man, she may choose to stick out her hand. He, in turn, may decide to either shake it or kiss it, depending, for example, on whether she is an American lawyer at a business meeting or a European princess at a formal ball. Violating the protocol will make communication more difficult, if not completely impossible.

A five-layer network is illustrated in Fig. 1-13. The entities comprising the corresponding layers on different machines are called **peers**. The peers may be

software processes, hardware devices, or even human beings. In other words, it is the peers that communicate by using the protocol to talk to each other.

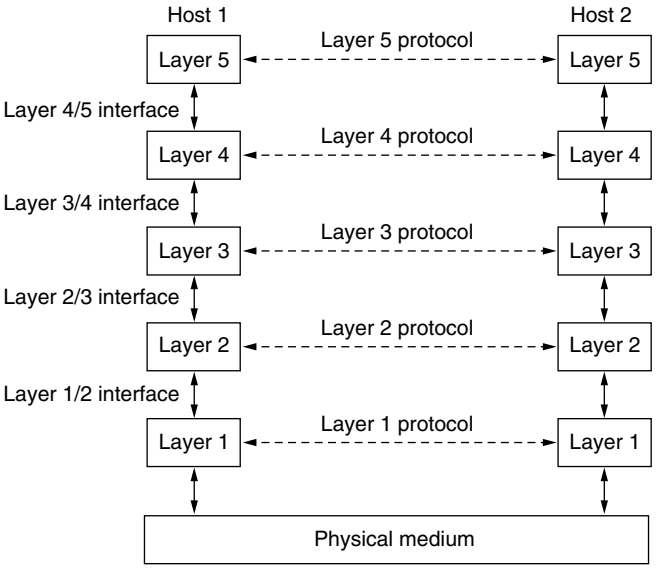


Figure 1-13. Layers, protocols, and interfaces.

In reality, no data are directly transferred from layer *n* on one machine to layer *n* on another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer 1 is the **physical medium** through which actual communication occurs. In Fig. 1-13, virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers is an **interface**. The interface defines which primitive operations and services the lower layer makes available to the upper one. When network designers decide how many layers to include in a network and what each one should do, one of the most important considerations is defining clean interfaces between the layers. Doing so, in turn, requires that each layer perform a specific collection of well-understood functions. In addition to minimizing the amount of information that must be passed between layers, clear-cut interfaces also make it simpler to replace one layer with a completely different protocol or implementation (e.g., replacing all the telephone lines by satellite channels) because all that is required of the new protocol or implementation is that it offer exactly the same set of services to its upstairs neighbor as the old one did. It is common that different hosts use different implementations of the same protocol (often written by different companies). In fact, the protocol itself can change in some layer without the layers above and below it even noticing.

Patent Owner's Ex. 2010
Page 046

A set of layers and protocols is called a **network architecture**. The specification of an architecture must contain enough information to allow an implementer to write the program or build the hardware for each layer so that it will correctly obey the appropriate protocol. Neither the details of the implementation nor the specification of the interfaces is part of the architecture because these are hidden away inside the machines and not visible from the outside. It is not even necessary that the interfaces on all machines in a network be the same, provided that each machine can correctly use all the protocols. A list of the protocols used by a certain system, one protocol per layer, is called a **protocol stack**. Network architectures, protocol stacks, and the protocols themselves are the principal subjects of this book.

An analogy may help explain the idea of multilayer communication. Imagine two philosophers (peer processes in layer 3), one of whom speaks Urdu and English and one of whom speaks Chinese and French. Since they have no common language, they each engage a translator (peer processes at layer 2), each of whom in turn contacts a secretary (peer processes in layer 1). Philosopher 1 wishes to convey his affection for *oryctolagus cuniculus* to his peer. To do so, he passes a message (in English) across the 2/3 interface to his translator, saying "I like rabbits," as illustrated in Fig. 1-14. The translators have agreed on a neutral language known to both of them, Dutch, so the message is converted to "Ik vind konijnen leuk." The choice of the language is the layer 2 protocol and is up to the layer 2 peer processes.

The translator then gives the message to a secretary for transmission, for example, by email (the layer 1 protocol). When the message arrives at the other secretary, it is passed to the local translator, who translates it into French and passes it across the 2/3 interface to the second philosopher. Note that each protocol is completely independent of the other ones as long as the interfaces are not changed. The translators can switch from Dutch to, say, Finnish, at will, provided that they both agree and neither changes his interface with either layer 1 or layer 3. Similarly, the secretaries can switch from email to telephone without disturbing (or even informing) the other layers. Each process may add some information intended only for its peer. This information is not passed up to the layer above.

Now consider a more technical example: how to provide communication to the top layer of the five-layer network in Fig. 1-15. A message, *M*, is produced by an application process running in layer 5 and given to layer 4 for transmission. Layer 4 puts a **header** in front of the message to identify the message and passes the result to layer 3. The header includes control information, such as addresses, to allow layer 4 on the destination machine to deliver the message. Other examples of control information used in some layers are sequence numbers (in case the lower layer does not preserve message order), sizes, and times.

In many networks, no limit is placed on the size of messages transmitted in the layer 4 protocol but there is nearly always a limit imposed by the layer 3 protocol. Consequently, layer 3 must break up the incoming messages into smaller

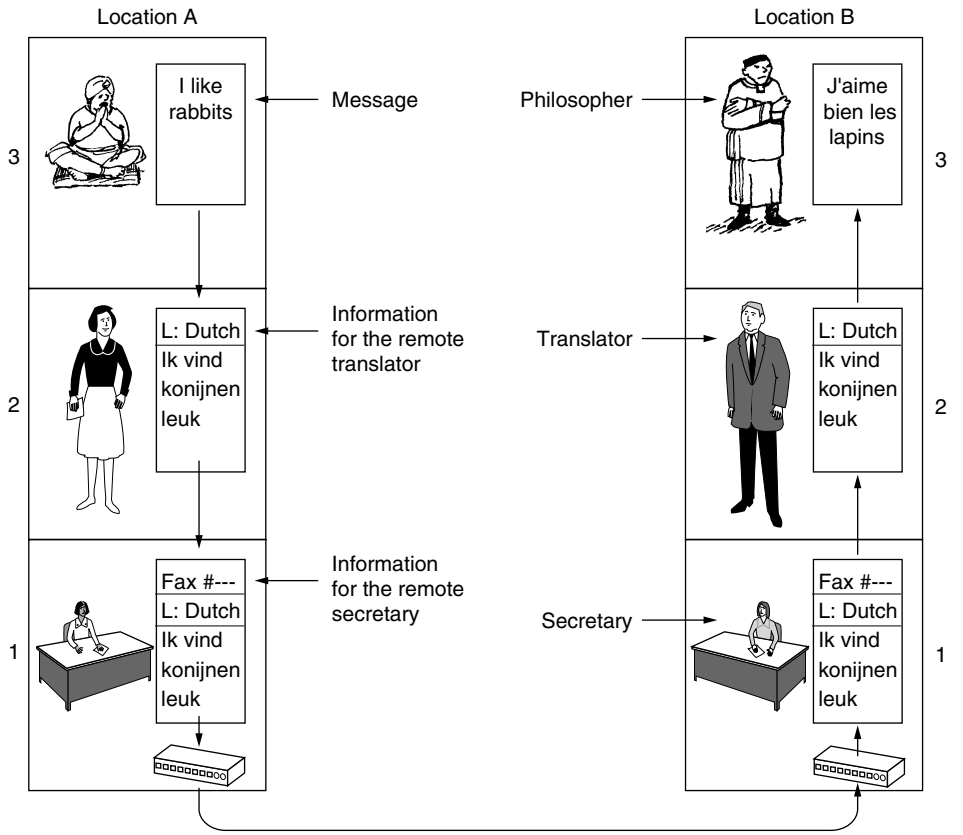


Figure 1-14. The philosopher-translator-secretary architecture.

units, packets, prepending a layer 3 header to each packet. In this example, M is split into two parts, M_1 and M_2, that will be transmitted separately.

Layer 3 decides which of the outgoing lines to use and passes the packets to layer 2. Layer 2 adds to each piece not only a header but also a trailer, and gives the resulting unit to layer 1 for physical transmission. At the receiving machine the message moves upward, from layer to layer, with headers being stripped off as it progresses. None of the headers for layers below n are passed up to layer n.

The important thing to understand about Fig. 1-15 is the relation between the virtual and actual communication and the difference between protocols and interfaces. The peer processes in layer 4, for example, conceptually think of their communication as being "horizontal," using the layer 4 protocol. Each one is likely to have procedures called something like *SendToOtherSide* and *GetFromOtherSide*, even though these procedures actually communicate with lower layers across the 3/4 interface, and not with the other side.

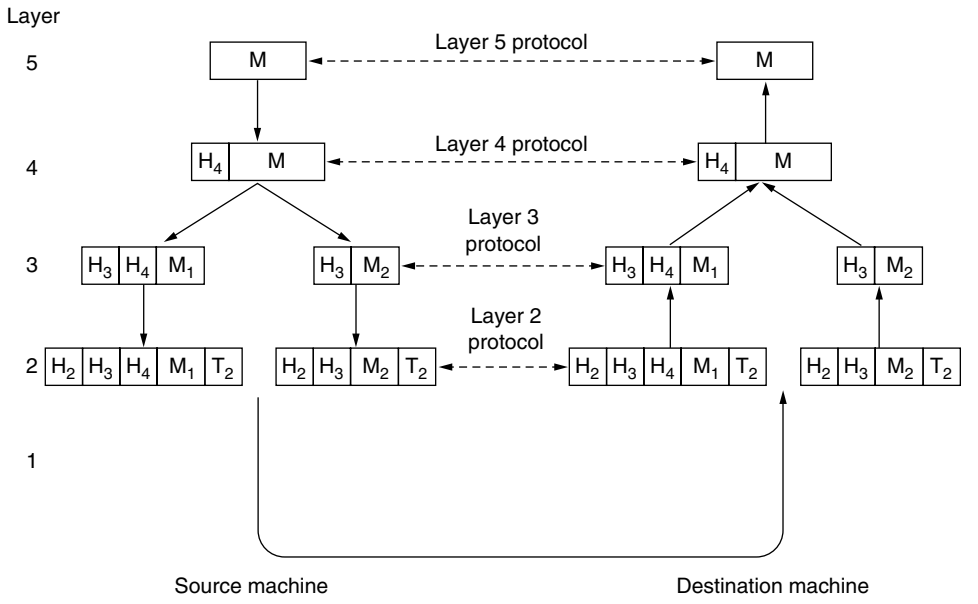


Figure 1-15. Example information flow supporting virtual communication in layer 5.

The peer process abstraction is crucial to all network design. Using it, the unmanageable task of designing the complete network can be broken into several smaller, manageable design problems, namely, the design of the individual layers.

Although Sec. 1.3 is called "Network Software," it is worth pointing out that the lower layers of a protocol hierarchy are frequently implemented in hardware or firmware. Nevertheless, complex protocol algorithms are involved, even if they are embedded (in whole or in part) in hardware.

1.3.2 Design Issues for the Layers

Some of the key design issues that occur in computer networks will come up in layer after layer. Below, we will briefly mention the more important ones.

Reliability is the design issue of making a network that operates correctly even though it is made up of a collection of components that are themselves unreliable. Think about the bits of a packet traveling through the network. There is a chance that some of these bits will be received damaged (inverted) due to fluke electrical noise, random wireless signals, hardware flaws, software bugs and so on. How is it possible that we find and fix these errors?

One mechanism for finding errors in received information uses codes for **error detection**. Information that is incorrectly received can then be retransmitted

until it is received correctly. More powerful codes allow for **error correction**, where the correct message is recovered from the possibly incorrect bits that were originally received. Both of these mechanisms work by adding redundant information. They are used at low layers, to protect packets sent over individual links, and high layers, to check that the right contents were received.

Another reliability issue is finding a working path through a network. Often there are multiple paths between a source and destination, and in a large network, there may be some links or routers that are broken. Suppose that the network is down in Germany. Packets sent from London to Rome via Germany will not get through, but we could instead send packets from London to Rome via Paris. The network should automatically make this decision. This topic is called **routing**.

A second design issue concerns the evolution of the network. Over time, networks grow larger and new designs emerge that need to be connected to the existing network. We have recently seen the key structuring mechanism used to support change by dividing the overall problem and hiding implementation details: **protocol layering**. There are many other strategies as well.

Since there are many computers on the network, every layer needs a mechanism for identifying the senders and receivers that are involved in a particular message. This mechanism is called **addressing** or **naming**, in the low and high layers, respectively.

An aspect of growth is that different network technologies often have different limitations. For example, not all communication channels preserve the order of messages sent on them, leading to solutions that number messages. Another example is differences in the maximum size of a message that the networks can transmit. This leads to mechanisms for disassembling, transmitting, and then reassembling messages. This overall topic is called **internetworking**.

When networks get large, new problems arise. Cities can have traffic jams, a shortage of telephone numbers, and it is easy to get lost. Not many people have these problems in their own neighborhood, but citywide they may be a big issue. Designs that continue to work well when the network gets large are said to be **scalable**.

A third design issue is resource allocation. Networks provide a service to hosts from their underlying resources, such as the capacity of transmission lines. To do this well, they need mechanisms that divide their resources so that one host does not interfere with another too much.

Many designs share network bandwidth dynamically, according to the short-term needs of hosts, rather than by giving each host a fixed fraction of the bandwidth that it may or may not use. This design is called **statistical multiplexing**, meaning sharing based on the statistics of demand. It can be applied at low layers for a single link, or at high layers for a network or even applications that use the network.

An allocation problem that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Feedback from the receiver to the

sender is often used. This subject is called **flow control**. Sometimes the problem is that the network is oversubscribed because too many computers want to send too much traffic, and the network cannot deliver it all. This overloading of the network is called **congestion**. One strategy is for each computer to reduce its demand when it experiences congestion. It, too, can be used in all layers.

It is interesting to observe that the network has more resources to offer than simply bandwidth. For uses such as carrying live video, the timeliness of delivery matters a great deal. Most networks must provide service to applications that want this **real-time** delivery at the same time that they provide service to applications that want high throughput. **Quality of service** is the name given to mechanisms that reconcile these competing demands.

The last major design issue is to secure the network by defending it against different kinds of threats. One of the threats we have mentioned previously is that of eavesdropping on communications. Mechanisms that provide **confidentiality** defend against this threat, and they are used in multiple layers. Mechanisms for **authentication** prevent someone from impersonating someone else. They might be used to tell fake banking Web sites from the real one, or to let the cellular network check that a call is really coming from your phone so that you will pay the bill. Other mechanisms for **integrity** prevent surreptitious changes to messages, such as altering "debit my account $10" to "debit my account $1000." All of these designs are based on cryptography, which we shall study in Chap. 8.

1.3.3 Connection-Oriented Versus Connectionless Service

Layers can offer two different types of service to the layers above them: connection-oriented and connectionless. In this section we will look at these two types and examine the differences between them.

Connection-oriented service is modeled after the telephone system. To talk to someone, you pick up the phone, dial the number, talk, and then hang up. Similarly, to use a connection-oriented network service, the service user first establishes a connection, uses the connection, and then releases the connection. The essential aspect of a connection is that it acts like a tube: the sender pushes objects (bits) in at one end, and the receiver takes them out at the other end. In most cases the order is preserved so that the bits arrive in the order they were sent.

In some cases when a connection is established, the sender, receiver, and subnet conduct a **negotiation** about the parameters to be used, such as maximum message size, quality of service required, and other issues. Typically, one side makes a proposal and the other side can accept it, reject it, or make a counterproposal. A **circuit** is another name for a connection with associated resources, such as a fixed bandwidth. This dates from the telephone network in which a circuit was a path over copper wire that carried a phone conversation.

In contrast to connection-oriented service, **connectionless** service is modeled after the postal system. Each message (letter) carries the full destination address,

and each one is routed through the intermediate nodes inside the system independent of all the subsequent messages. There are different names for messages in different contexts; a **packet** is a message at the network layer. When the intermediate nodes receive a message in full before sending it on to the next node, this is called **store-and-forward switching**. The alternative, in which the onward transmission of a message at a node starts before it is completely received by the node, is called **cut-through switching**. Normally, when two messages are sent to the same destination, the first one sent will be the first one to arrive. However, it is possible that the first one sent can be delayed so that the second one arrives first.

Each kind of service can further be characterized by its reliability. Some services are reliable in the sense that they never lose data. Usually, a reliable service is implemented by having the receiver acknowledge the receipt of each message so the sender is sure that it arrived. The acknowledgement process introduces overhead and delays, which are often worth it but are sometimes undesirable.

A typical situation in which a reliable connection-oriented service is appropriate is file transfer. The owner of the file wants to be sure that all the bits arrive correctly and in the same order they were sent. Very few file transfer customers would prefer a service that occasionally scrambles or loses a few bits, even if it is much faster.

Reliable connection-oriented service has two minor variations: message sequences and byte streams. In the former variant, the message boundaries are preserved. When two 1024-byte messages are sent, they arrive as two distinct 1024-byte messages, never as one 2048-byte message. In the latter, the connection is simply a stream of bytes, with no message boundaries. When 2048 bytes arrive at the receiver, there is no way to tell if they were sent as one 2048-byte message, two 1024-byte messages, or 2048 1-byte messages. If the pages of a book are sent over a network to a phototypesetter as separate messages, it might be important to preserve the message boundaries. On the other hand, to download a DVD movie, a byte stream from the server to the user's computer is all that is needed. Message boundaries within the movie are not relevant.

For some applications, the transit delays introduced by acknowledgements are unacceptable. One such application is digitized voice traffic for **voice over IP**. It is less disruptive for telephone users to hear a bit of noise on the line from time to time than to experience a delay waiting for acknowledgements. Similarly, when transmitting a video conference, having a few pixels wrong is no problem, but having the image jerk along as the flow stops and starts to correct errors is irritating.

Not all applications require connections. For example, spammers send electronic junk-mail to many recipients. The spammer probably does not want to go to the trouble of setting up and later tearing down a connection to a recipient just to send them one item. Nor is 100 percent reliable delivery essential, especially if it costs more. All that is needed is a way to send a single message that has a high

probability of arrival, but no guarantee. Unreliable (meaning not acknowledged) connectionless service is often called **datagram** service, in analogy with telegram service, which also does not return an acknowledgement to the sender. Despite it being unreliable, it is the dominant form in most networks for reasons that will become clear later

In other situations, the convenience of not having to establish a connection to send one message is desired, but reliability is essential. The **acknowledged datagram** service can be provided for these applications. It is like sending a registered letter and requesting a return receipt. When the receipt comes back, the sender is absolutely sure that the letter was delivered to the intended party and not lost along the way. Text messaging on mobile phones is an example.

Still another service is the **request-reply** service. In this service the sender transmits a single datagram containing a request; the reply contains the answer. Request-reply is commonly used to implement communication in the client-server model: the client issues a request and the server responds to it. For example, a mobile phone client might send a query to a map server to retrieve the map data for the current location. Figure 1-16 summarizes the types of services discussed above.

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Movie download
	Unreliable connection	Voice over IP
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Text messaging
	Request-reply	Database query

Figure 1-16. Six different types of service.

The concept of using unreliable communication may be confusing at first. After all, why would anyone actually prefer unreliable communication to reliable communication? First of all, reliable communication (in our sense, that is, acknowledged) may not be available in a given layer. For example, Ethernet does not provide reliable communication. Packets can occasionally be damaged in transit. It is up to higher protocol levels to recover from this problem. In particular, many reliable services are built on top of an unreliable datagram service. Second, the delays inherent in providing a reliable service may be unacceptable, especially in real-time applications such as multimedia. For these reasons, both reliable and unreliable communication coexist.

1.3.4 Service Primitives

A service is formally specified by a set of **primitives** (operations) available to user processes to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. If the protocol stack is located in the operating system, as it often is, the primitives are normally system calls. These calls cause a trap to kernel mode, which then turns control of the machine over to the operating system to send the necessary packets.

The set of primitives available depends on the nature of the service being provided. The primitives for connection-oriented service are different from those of connectionless service. As a minimal example of the service primitives that might provide a reliable byte stream, consider the primitives listed in Fig. 1-17. They will be familiar to fans of the Berkeley socket interface, as the primitives are a simplified version of that interface.

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
ACCEPT	Accept an incoming connection from a peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

Figure 1-17. Six service primitives that provide a simple connection-oriented service.

These primitives might be used for a request-reply interaction in a client-server environment. To illustrate how, We sketch a simple protocol that implements the service using acknowledged datagrams.

First, the server executes LISTEN to indicate that it is prepared to accept incoming connections. A common way to implement LISTEN is to make it a blocking system call. After executing the primitive, the server process is blocked until a request for connection appears.

Next, the client process executes CONNECT to establish a connection with the server. The CONNECT call needs to specify who to connect to, so it might have a parameter giving the server's address. The operating system then typically sends a packet to the peer asking it to connect, as shown by (1) in Fig. 1-18. The client process is suspended until there is a response.

When the packet arrives at the server, the operating system sees that the packet is requesting a connection. It checks to see if there is a listener, and if so it unblocks the listener. The server process can then establish the connection with the ACCEPT call. This sends a response (2) back to the client process to accept the

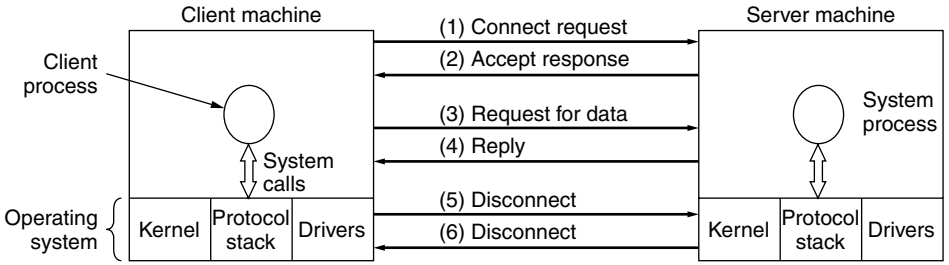


Figure 1-18. A simple client-server interaction using acknowledged datagrams.

connection. The arrival of this response then releases the client. At this point the client and server are both running and they have a connection established.

The obvious analogy between this protocol and real life is a customer (client) calling a company's customer service manager. At the start of the day, the service manager sits next to his telephone in case it rings. Later, a client places a call. When the manager picks up the phone, the connection is established.

The next step is for the server to execute RECEIVE to prepare to accept the first request. Normally, the server does this immediately upon being released from the LISTEN, before the acknowledgement can get back to the client. The RECEIVE call blocks the server.

Then the client executes SEND to transmit its request (3) followed by the execution of RECEIVE to get the reply. The arrival of the request packet at the server machine unblocks the server so it can handle the request. After it has done the work, the server uses SEND to return the answer to the client (4). The arrival of this packet unblocks the client, which can now inspect the answer. If the client has additional requests, it can make them now.

When the client is done, it executes DISCONNECT to terminate the connection (5). Usually, an initial DISCONNECT is a blocking call, suspending the client and sending a packet to the server saying that the connection is no longer needed. When the server gets the packet, it also issues a DISCONNECT of its own, acknowledging the client and releasing the connection (6). When the server's packet gets back to the client machine, the client process is released and the connection is broken. In a nutshell, this is how connection-oriented communication works.

Of course, life is not so simple. Many things can go wrong here. The timing can be wrong (e.g., the CONNECT is done before the LISTEN), packets can get lost, and much more. We will look at these issues in great detail later, but for the moment, Fig. 1-18 briefly summarizes how client-server communication might work with acknowledged datagrams so that we can ignore lost packets.

Given that six packets are required to complete this protocol, one might wonder why a connectionless protocol is not used instead. The answer is that in a perfect world it could be, in which case only two packets would be needed: one

for the request and one for the reply. However, in the face of large messages in either direction (e.g., a megabyte file), transmission errors, and lost packets, the situation changes. If the reply consisted of hundreds of packets, some of which could be lost during transmission, how would the client know if some pieces were missing? How would the client know whether the last packet actually received was really the last packet sent? Suppose the client wanted a second file. How could it tell packet 1 from the second file from a lost packet 1 from the first file that suddenly found its way to the client? In short, in the real world, a simple request-reply protocol over an unreliable network is often inadequate. In Chap. 3 we will study a variety of protocols in detail that overcome these and other problems. For the moment, suffice it to say that having a reliable, ordered byte stream between processes is sometimes very convenient.

1.3.5 The Relationship of Services to Protocols

Services and protocols are distinct concepts. This distinction is so important that we emphasize it again here. A *service* is a set of primitives (operations) that a layer provides to the layer above it. The service defines what operations the layer is prepared to perform on behalf of its users, but it says nothing at all about how these operations are implemented. A service relates to an interface between two layers, with the lower layer being the service provider and the upper layer being the service user.

A *protocol*, in contrast, is a set of rules governing the format and meaning of the packets, or messages that are exchanged by the peer entities within a layer. Entities use protocols to implement their service definitions. They are free to change their protocols at will, provided they do not change the service visible to their users. In this way, the service and the protocol are completely decoupled. This is a key concept that any network designer should understand well.

To repeat this crucial point, services relate to the interfaces between layers, as illustrated in Fig. 1-19. In contrast, protocols relate to the packets sent between peer entities on different machines. It is very important not to confuse the two concepts.

An analogy with programming languages is worth making. A service is like an abstract data type or an object in an object-oriented language. It defines operations that can be performed on an object but does not specify how these operations are implemented. In contrast, a protocol relates to the *implementation* of the service and as such is not visible to the user of the service.

Many older protocols did not distinguish the service from the protocol. In effect, a typical layer might have had a service primitive SEND PACKET with the user providing a pointer to a fully assembled packet. This arrangement meant that all changes to the protocol were immediately visible to the users. Most network designers now regard such a design as a serious blunder.

Patent Owner's Ex. 2010
Page 056

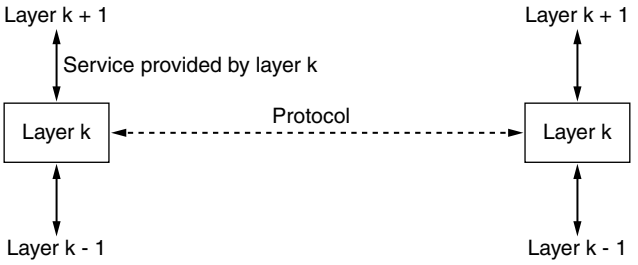


Figure 1-19. The relationship between a service and a protocol.

1.4 REFERENCE MODELS

Now that we have discussed layered networks in the abstract, it is time to look at some examples. We will discuss two important network architectures: the OSI reference model and the TCP/IP reference model. Although the *protocols* associated with the OSI model are not used any more, the *model* itself is actually quite general and still valid, and the features discussed at each layer are still very important. The TCP/IP model has the opposite properties: the model itself is not of much use but the protocols are widely used. For this reason we will look at both of them in detail. Also, sometimes you can learn more from failures than from successes.

1.4.1 The OSI Reference Model

The OSI model (minus the physical medium) is shown in Fig. 1-20. This model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the protocols used in the various layers (Day and Zimmermann, 1983). It was revised in 1995 (Day, 1995). The model is called the ISO **OSI** (**Open Systems Interconnection**) Reference Model because it deals with connecting open systems—that is, systems that are open for communication with other systems. We will just call it the **OSI model** for short.

The OSI model has seven layers. The principles that were applied to arrive at the seven layers can be briefly summarized as follows:

1. A layer should be created where a different abstraction is needed.
2. Each layer should perform a well-defined function.
3. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.

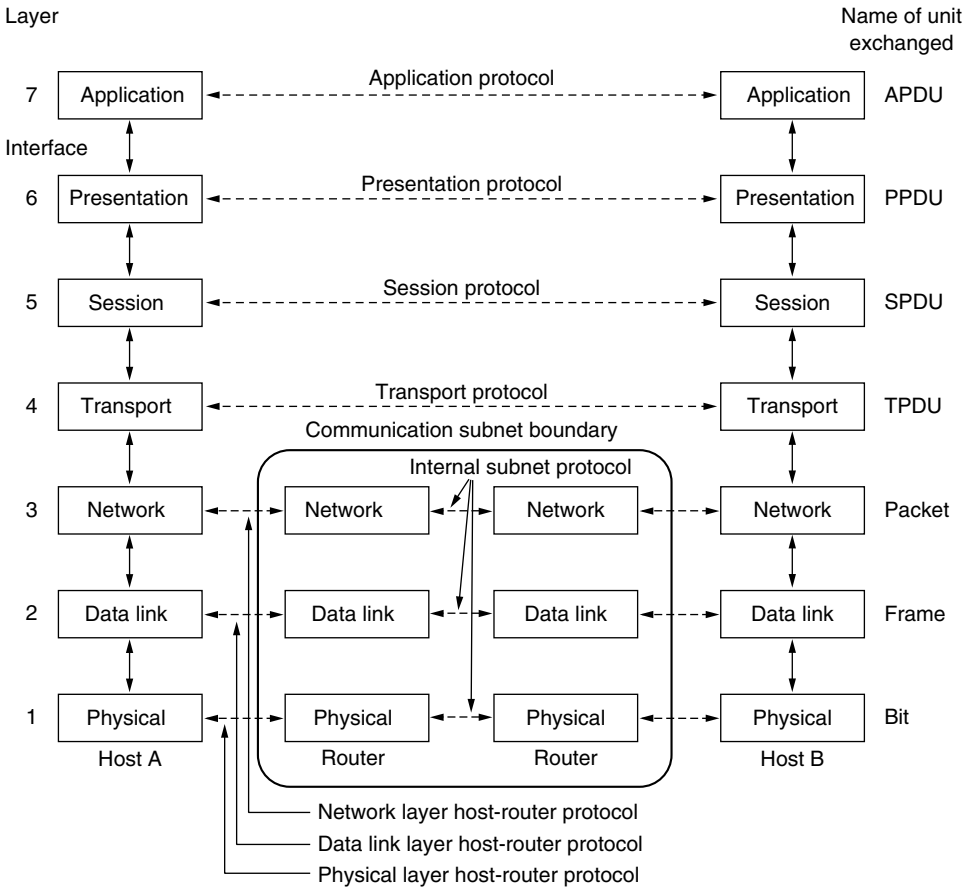


Figure 1-20. The OSI reference model.

4. The layer boundaries should be chosen to minimize the information flow across the interfaces.
5. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity and small enough that the architecture does not become unwieldy.

Below we will discuss each layer of the model in turn, starting at the bottom layer. Note that the OSI model itself is not a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do. However, ISO has also produced standards for all the layers, although these are not part of the reference model itself. Each one has been published as a separate international standard. The *model* (in part) is widely used although the associated protocols have been long forgotten.

The Physical Layer

The **physical layer** is concerned with transmitting raw bits over a communication channel. The design issues have to do with making sure that when one side sends a 1 bit it is received by the other side as a 1 bit, not as a 0 bit. Typical questions here are what electrical signals should be used to represent a 1 and a 0, how many nanoseconds a bit lasts, whether transmission may proceed simultaneously in both directions, how the initial connection is established, how it is torn down when both sides are finished, how many pins the network connector has, and what each pin is used for. These design issues largely deal with mechanical, electrical, and timing interfaces, as well as the physical transmission medium, which lies below the physical layer.

The Data Link Layer

The main task of the **data link layer** is to transform a raw transmission facility into a line that appears free of undetected transmission errors. It does so by masking the real errors so the network layer does not see them. It accomplishes this task by having the sender break up the input data into **data frames** (typically a few hundred or a few thousand bytes) and transmit the frames sequentially. If the service is reliable, the receiver confirms correct receipt of each frame by sending back an **acknowledgement frame**.

Another issue that arises in the data link layer (and most of the higher layers as well) is how to keep a fast transmitter from drowning a slow receiver in data. Some traffic regulation mechanism may be needed to let the transmitter know when the receiver can accept more data.

Broadcast networks have an additional issue in the data link layer: how to control access to the shared channel. A special sublayer of the data link layer, the **medium access control** sublayer, deals with this problem.

The Network Layer

The **network layer** controls the operation of the subnet. A key design issue is determining how packets are routed from source to destination. Routes can be based on static tables that are "wired into" the network and rarely changed, or more often they can be updated automatically to avoid failed components. They can also be determined at the start of each conversation, for example, a terminal session, such as a login to a remote machine. Finally, they can be highly dynamic, being determined anew for each packet to reflect the current network load.

If too many packets are present in the subnet at the same time, they will get in one another's way, forming bottlenecks. Handling congestion is also a responsibility of the network layer, in conjunction with higher layers that adapt the load

they place on the network. More generally, the quality of service provided (delay, transit time, jitter, etc.) is also a network layer issue.

When a packet has to travel from one network to another to get to its destination, many problems can arise. The addressing used by the second network may be different from that used by the first one. The second one may not accept the packet at all because it is too large. The protocols may differ, and so on. It is up to the network layer to overcome all these problems to allow heterogeneous networks to be interconnected.

In broadcast networks, the routing problem is simple, so the network layer is often thin or even nonexistent.

The Transport Layer

The basic function of the **transport layer** is to accept data from above it, split it up into smaller units if need be, pass these to the network layer, and ensure that the pieces all arrive correctly at the other end. Furthermore, all this must be done efficiently and in a way that isolates the upper layers from the inevitable changes in the hardware technology over the course of time.

The transport layer also determines what type of service to provide to the session layer, and, ultimately, to the users of the network. The most popular type of transport connection is an error-free point-to-point channel that delivers messages or bytes in the order in which they were sent. However, other possible kinds of transport service exist, such as the transporting of isolated messages with no guarantee about the order of delivery, and the broadcasting of messages to multiple destinations. The type of service is determined when the connection is established. (As an aside, an error-free channel is completely impossible to achieve; what people really mean by this term is that the error rate is low enough to ignore in practice.)

The transport layer is a true end-to-end layer; it carries data all the way from the source to the destination. In other words, a program on the source machine carries on a conversation with a similar program on the destination machine, using the message headers and control messages. In the lower layers, each protocols is between a machine and its immediate neighbors, and not between the ultimate source and destination machines, which may be separated by many routers. The difference between layers 1 through 3, which are chained, and layers 4 through 7, which are end-to-end, is illustrated in Fig. 1-20.

The Session Layer

The session layer allows users on different machines to establish **sessions** between them. Sessions offer various services, including **dialog control** (keeping track of whose turn it is to transmit), **token management** (preventing two parties from attempting the same critical operation simultaneously), and **synchronization**

(checkpointing long transmissions to allow them to pick up from where they left off in the event of a crash and subsequent recovery).

The Presentation Layer

Unlike the lower layers, which are mostly concerned with moving bits around, the **presentation layer** is concerned with the syntax and semantics of the information transmitted. In order to make it possible for computers with different internal data representations to communicate, the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire." The presentation layer manages these abstract data structures and allows higher-level data structures (e.g., banking records) to be defined and exchanged.

The Application Layer

The **application layer** contains a variety of protocols that are commonly needed by users. One widely used application protocol is **HTTP** (**HyperText Transfer Protocol**), which is the basis for the World Wide Web. When a browser wants a Web page, it sends the name of the page it wants to the server hosting the page using HTTP. The server then sends the page back. Other application protocols are used for file transfer, electronic mail, and network news.

1.4.2 The TCP/IP Reference Model

Let us now turn from the OSI reference model to the reference model used in the grandparent of all wide area computer networks, the ARPANET, and its successor, the worldwide Internet. Although we will give a brief history of the ARPANET later, it is useful to mention a few key aspects of it now. The ARPANET was a research network sponsored by the DoD (U.S. Department of Defense). It eventually connected hundreds of universities and government installations, using leased telephone lines. When satellite and radio networks were added later, the existing protocols had trouble interworking with them, so a new reference architecture was needed. Thus, from nearly the beginning, the ability to connect multiple networks in a seamless way was one of the major design goals. This architecture later became known as the **TCP/IP Reference Model**, after its two primary protocols. It was first described by Cerf and Kahn (1974), and later refined and defined as a standard in the Internet community (Braden, 1989). The design philosophy behind the model is discussed by Clark (1988).

Given the DoD's worry that some of its precious hosts, routers, and internetwork gateways might get blown to pieces at a moment's notice by an attack from the Soviet Union, another major goal was that the network be able to survive loss of subnet hardware, without existing conversations being broken off. In other

words, the DoD wanted connections to remain intact as long as the source and destination machines were functioning, even if some of the machines or transmission lines in between were suddenly put out of operation. Furthermore, since applications with divergent requirements were envisioned, ranging from transferring files to real-time speech transmission, a flexible architecture was needed.

The Link Layer

All these requirements led to the choice of a packet-switching network based on a connectionless layer that runs across different networks. The lowest layer in the model, the **link layer** describes what links such as serial lines and classic Ethernet must do to meet the needs of this connectionless internet layer. It is not really a layer at all, in the normal sense of the term, but rather an interface between hosts and transmission links. Early material on the TCP/IP model has little to say about it.

The Internet Layer

The **internet layer** is the linchpin that holds the whole architecture together. It is shown in Fig. 1-21 as corresponding roughly to the OSI network layer. Its job is to permit hosts to inject packets into any network and have them travel independently to the destination (potentially on a different network). They may even arrive in a completely different order than they were sent, in which case it is the job of higher layers to rearrange them, if in-order delivery is desired. Note that "internet" is used here in a generic sense, even though this layer is present in the Internet.

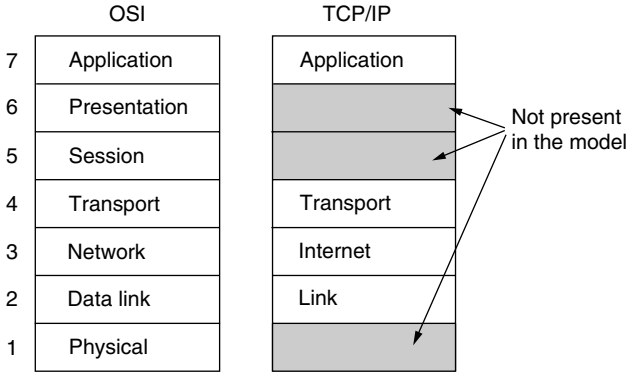


Figure 1-21. The TCP/IP reference model.

The analogy here is with the (snail) mail system. A person can drop a sequence of international letters into a mailbox in one country, and with a little luck,

most of them will be delivered to the correct address in the destination country. The letters will probably travel through one or more international mail gateways along the way, but this is transparent to the users. Furthermore, that each country (i.e., each network) has its own stamps, preferred envelope sizes, and delivery rules is hidden from the users.

The internet layer defines an official packet format and protocol called **IP** (**Internet Protocol**), plus a companion protocol called **ICMP** (**Internet Control Message Protocol**) that helps it function. The job of the internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly a major issue here, as is congestion (though IP has not proven effective at avoiding congestion).

The Transport Layer

The layer above the internet layer in the TCP/IP model is now usually called the **transport layer**. It is designed to allow peer entities on the source and destination hosts to carry on a conversation, just as in the OSI transport layer. Two end-to-end transport protocols have been defined here. The first one, **TCP** (**Transmission Control Protocol**), is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the internet. It segments the incoming byte stream into discrete messages and passes each one on to the internet layer. At the destination, the receiving TCP process reassembles the received messages into the output stream. TCP also handles flow control to make sure a fast sender cannot swamp a slow receiver with more messages than it can handle.

The second protocol in this layer, **UDP** (**User Datagram Protocol**), is an unreliable, connectionless protocol for applications that do not want TCP's sequencing or flow control and wish to provide their own. It is also widely used for one-shot, client-server-type request-reply queries and applications in which prompt delivery is more important than accurate delivery, such as transmitting speech or video. The relation of IP, TCP, and UDP is shown in Fig. 1-22. Since the model was developed, IP has been implemented on many other networks.

The Application Layer

The TCP/IP model does not have session or presentation layers. No need for them was perceived. Instead, applications simply include any session and presentation functions that they require. Experience with the OSI model has proven this view correct: these layers are of little use to most applications.

On top of the transport layer is the **application layer**. It contains all the higher-level protocols. The early ones included virtual terminal (TELNET), file transfer (FTP), and electronic mail (SMTP). Many other protocols have been added to these over the years. Some important ones that we will study, shown in Fig. 1-22,

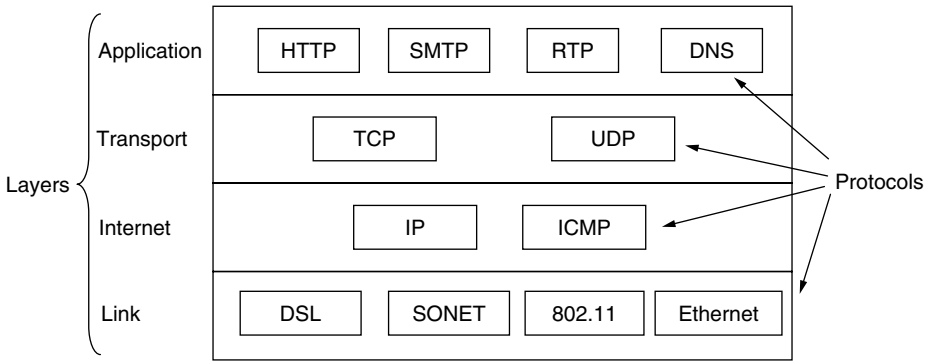


Figure 1-22. The TCP/IP model with some protocols we will study.

include the Domain Name System (DNS), for mapping host names onto their network addresses, HTTP, the protocol for fetching pages on the World Wide Web, and RTP, the protocol for delivering real-time media such as voice or movies.

1.4.3 The Model Used in This Book

As mentioned earlier, the strength of the OSI reference model is the *model* itself (minus the presentation and session layers), which has proven to be exceptionally useful for discussing computer networks. In contrast, the strength of the TCP/IP reference model is the *protocols*, which have been widely used for many years. Since computer scientists like to have their cake and eat it, too, we will use the hybrid model of Fig. 1-23 as the framework for this book.

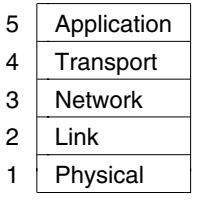


Figure 1-23. The reference model used in this book.

This model has five layers, running from the physical layer up through the link, network and transport layers to the application layer. The physical layer specifies how to transmit bits across different kinds of media as electrical (or other analog) signals. The link layer is concerned with how to send finite-length messages between directly connected computers with specified levels of reliability. Ethernet and 802.11 are examples of link layer protocols.

The network layer deals with how to combine multiple links into networks, and networks of networks, into internetworks so that we can send packets between distant computers. This includes the task of finding the path along which to send the packets. IP is the main example protocol we will study for this layer. The transport layer strengthens the delivery guarantees of the Network layer, usually with increased reliability, and provide delivery abstractions, such as a reliable byte stream, that match the needs of different applications. TCP is an important example of a transport layer protocol.

Finally, the application layer contains programs that make use of the network. Many, but not all, networked applications have user interfaces, such as a Web browser. Our concern, however, is with the portion of the program that uses the network. This is the HTTP protocol in the case of the Web browser. There are also important support programs in the application layer, such as the DNS, that are used by many applications.

Our chapter sequence is based on this model. In this way, we retain the value of the OSI model for understanding network architectures, but concentrate primarily on protocols that are important in practice, from TCP/IP and related protocols to newer ones such as 802.11, SONET, and Bluetooth.

1.4.4 A Comparison of the OSI and TCP/IP Reference Models

The OSI and TCP/IP reference models have much in common. Both are based on the concept of a stack of independent protocols. Also, the functionality of the layers is roughly similar. For example, in both models the layers up through and including the transport layer are there to provide an end-to-end, network-independent transport service to processes wishing to communicate. These layers form the transport provider. Again in both models, the layers above transport are application-oriented users of the transport service.

Despite these fundamental similarities, the two models also have many differences. In this section we will focus on the key differences between the two reference models. It is important to note that we are comparing the *reference models* here, not the corresponding *protocol stacks*. The protocols themselves will be discussed later. For an entire book comparing and contrasting TCP/IP and OSI, see Piscitello and Chapin (1993).

Three concepts are central to the OSI model:

1. Services.
2. Interfaces.
3. Protocols.

Probably the biggest contribution of the OSI model is that it makes the distinction between these three concepts explicit. Each layer performs some *services* for the

layer above it. The service definition tells what the layer does, not how entities above it access it or how the layer works. It defines the layer's semantics.

A layer's *interface* tells the processes above it how to access it. It specifies what the parameters are and what results to expect. It, too, says nothing about how the layer works inside.

Finally, the peer *protocols* used in a layer are the layer's own business. It can use any protocols it wants to, as long as it gets the job done (i.e., provides the offered services). It can also change them at will without affecting software in higher layers.

These ideas fit very nicely with modern ideas about object-oriented programming. An object, like a layer, has a set of methods (operations) that processes outside the object can invoke. The semantics of these methods define the set of services that the object offers. The methods' parameters and results form the object's interface. The code internal to the object is its protocol and is not visible or of any concern outside the object.

The TCP/IP model did not originally clearly distinguish between services, interfaces, and protocols, although people have tried to retrofit it after the fact to make it more OSI-like. For example, the only real services offered by the internet layer are SEND IP PACKET and RECEIVE IP PACKET. As a consequence, the protocols in the OSI model are better hidden than in the TCP/IP model and can be replaced relatively easily as the technology changes. Being able to make such changes transparently is one of the main purposes of having layered protocols in the first place.

The OSI reference model was devised *before* the corresponding protocols were invented. This ordering meant that the model was not biased toward one particular set of protocols, a fact that made it quite general. The downside of this ordering was that the designers did not have much experience with the subject and did not have a good idea of which functionality to put in which layer.

For example, the data link layer originally dealt only with point-to-point networks. When broadcast networks came around, a new sublayer had to be hacked into the model. Furthermore, when people started to build real networks using the OSI model and existing protocols, it was discovered that these networks did not match the required service specifications (wonder of wonders), so convergence sublayers had to be grafted onto the model to provide a place for papering over the differences. Finally, the committee originally expected that each country would have one network, run by the government and using the OSI protocols, so no thought was given to internetworking. To make a long story short, things did not turn out that way.

With TCP/IP the reverse was true: the protocols came first, and the model was really just a description of the existing protocols. There was no problem with the protocols fitting the model. They fit perfectly. The only trouble was that the *model* did not fit any other protocol stacks. Consequently, it was not especially useful for describing other, non-TCP/IP networks.

Turning from philosophical matters to more specific ones, an obvious difference between the two models is the number of layers: the OSI model has seven layers and the TCP/IP model has four. Both have (inter)network, transport, and application layers, but the other layers are different.

Another difference is in the area of connectionless versus connection-oriented communication. The OSI model supports both connectionless and connection-oriented communication in the network layer, but only connection-oriented communication in the transport layer, where it counts (because the transport service is visible to the users). The TCP/IP model supports only one mode in the network layer (connectionless) but both in the transport layer, giving the users a choice. This choice is especially important for simple request-response protocols.

1.4.5 A Critique of the OSI Model and Protocols

Neither the OSI model and its protocols nor the TCP/IP model and its protocols are perfect. Quite a bit of criticism can be, and has been, directed at both of them. In this section and the next one, we will look at some of these criticisms. We will begin with OSI and examine TCP/IP afterward.

At the time the second edition of this book was published (1989), it appeared to many experts in the field that the OSI model and its protocols were going to take over the world and push everything else out of their way. This did not happen. Why? A look back at some of the reasons may be useful. They can be summarized as:

1. Bad timing.
2. Bad technology.
3. Bad implementations.
4. Bad politics.

Bad Timing

First let us look at reason one: bad timing. The time at which a standard is established is absolutely critical to its success. David Clark of M.I.T. has a theory of standards that he calls the *apocalypse of the two elephants*, which is illustrated in Fig. 1-24.

This figure shows the amount of activity surrounding a new subject. When the subject is first discovered, there is a burst of research activity in the form of discussions, papers, and meetings. After a while this activity subsides, corporations discover the subject, and the billion-dollar wave of investment hits.

It is essential that the standards be written in the trough in between the two ''elephants.'' If they are written too early (before the research results are well

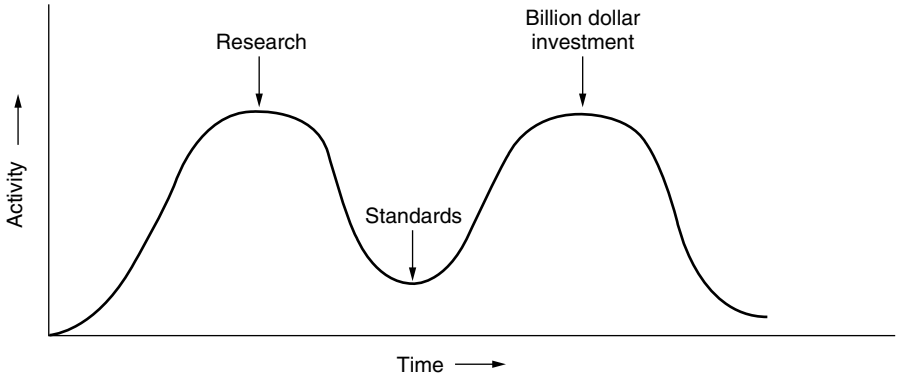


Figure 1-24. The apocalypse of the two elephants.

established), the subject may still be poorly understood; the result is a bad standard. If they are written too late, so many companies may have already made major investments in different ways of doing things that the standards are effectively ignored. If the interval between the two elephants is very short (because everyone is in a hurry to get started), the people developing the standards may get crushed.

It now appears that the standard OSI protocols got crushed. The competing TCP/IP protocols were already in widespread use by research universities by the time the OSI protocols appeared. While the billion-dollar wave of investment had not yet hit, the academic market was large enough that many vendors had begun cautiously offering TCP/IP products. When OSI came around, they did not want to support a second protocol stack until they were forced to, so there were no initial offerings. With every company waiting for every other company to go first, no company went first and OSI never happened.

Bad Technology

The second reason that OSI never caught on is that both the model and the protocols are flawed. The choice of seven layers was more political than technical, and two of the layers (session and presentation) are nearly empty, whereas two other ones (data link and network) are overfull.

The OSI model, along with its associated service definitions and protocols, is extraordinarily complex. When piled up, the printed standards occupy a significant fraction of a meter of paper. They are also difficult to implement and inefficient in operation. In this context, a riddle posed by Paul Mockapetris and cited by Rose (1993) comes to mind:

Q: What do you get when you cross a mobster with an international standard?
A: Someone who makes you an offer you can't understand.

In addition to being incomprehensible, another problem with OSI is that some functions, such as addressing, flow control, and error control, reappear again and again in each layer. Saltzer et al. (1984), for example, have pointed out that to be effective, error control must be done in the highest layer, so that repeating it over and over in each of the lower layers is often unnecessary and inefficient.

Bad Implementations

Given the enormous complexity of the model and the protocols, it will come as no surprise that the initial implementations were huge, unwieldy, and slow. Everyone who tried them got burned. It did not take long for people to associate "OSI" with "poor quality." Although the products improved in the course of time, the image stuck.

In contrast, one of the first implementations of TCP/IP was part of Berkeley UNIX and was quite good (not to mention, free). People began using it quickly, which led to a large user community, which led to improvements, which led to an even larger community. Here the spiral was upward instead of downward.

Bad Politics

On account of the initial implementation, many people, especially in academia, thought of TCP/IP as part of UNIX, and UNIX in the 1980s in academia was not unlike parenthood (then incorrectly called motherhood) and apple pie.

OSI, on the other hand, was widely thought to be the creature of the European telecommunication ministries, the European Community, and later the U.S. Government. This belief was only partly true, but the very idea of a bunch of government bureaucrats trying to shove a technically inferior standard down the throats of the poor researchers and programmers down in the trenches actually developing computer networks did not aid OSI's cause. Some people viewed this development in the same light as IBM announcing in the 1960s that PL/I was the language of the future, or the DoD correcting this later by announcing that it was actually Ada.

1.4.6 A Critique of the TCP/IP Reference Model

The TCP/IP model and protocols have their problems too. First, the model does not clearly distinguish the concepts of services, interfaces, and protocols. Good software engineering practice requires differentiating between the specification and the implementation, something that OSI does very carefully, but TCP/IP does not. Consequently, the TCP/IP model is not much of a guide for designing new networks using new technologies.

Second, the TCP/IP model is not at all general and is poorly suited to describing any protocol stack other than TCP/IP. Trying to use the TCP/IP model to describe Bluetooth, for example, is completely impossible.

Third, the link layer is not really a layer at all in the normal sense of the term as used in the context of layered protocols. It is an interface (between the network and data link layers). The distinction between an interface and a layer is crucial, and one should not be sloppy about it.

Fourth, the TCP/IP model does not distinguish between the physical and data link layers. These are completely different. The physical layer has to do with the transmission characteristics of copper wire, fiber optics, and wireless communication. The data link layer's job is to delimit the start and end of frames and get them from one side to the other with the desired degree of reliability. A proper model should include both as separate layers. The TCP/IP model does not do this.

Finally, although the IP and TCP protocols were carefully thought out and well implemented, many of the other protocols were ad hoc, generally produced by a couple of graduate students hacking away until they got tired. The protocol implementations were then distributed free, which resulted in their becoming widely used, deeply entrenched, and thus hard to replace. Some of them are a bit of an embarrassment now. The virtual terminal protocol, TELNET, for example, was designed for a ten-character-per-second mechanical Teletype terminal. It knows nothing of graphical user interfaces and mice. Nevertheless, it is still in use some 30 years later.

1.5 EXAMPLE NETWORKS

The subject of computer networking covers many different kinds of networks, large and small, well known and less well known. They have different goals, scales, and technologies. In the following sections, we will look at some examples, to get an idea of the variety one finds in the area of computer networking.

We will start with the Internet, probably the best known network, and look at its history, evolution, and technology. Then we will consider the mobile phone network. Technically, it is quite different from the Internet, contrasting nicely with it. Next we will introduce IEEE 802.11, the dominant standard for wireless LANs. Finally, we will look at RFID and sensor networks, technologies that extend the reach of the network to include the physical world and everyday objects.

1.5.1 The Internet

The Internet is not really a network at all, but a vast collection of different networks that use certain common protocols and provide certain common services. It is an unusual system in that it was not planned by anyone and is not controlled by anyone. To better understand it, let us start from the beginning and see how it has developed and why. For a wonderful history of the Internet, John Naughton's (2000) book is highly recommended. It is one of those rare books that is not only fun to read, but also has 20 pages of *ibid.*'s and *op. cit.*'s for the serious historian. Some of the material in this section is based on this book.

Of course, countless technical books have been written about the Internet and its protocols as well. For more information, see, for example, Maufer (1999).

The ARPANET

The story begins in the late 1950s. At the height of the Cold War, the U.S. DoD wanted a command-and-control network that could survive a nuclear war. At that time, all military communications used the public telephone network, which was considered vulnerable. The reason for this belief can be gleaned from Fig. 1-25(a). Here the black dots represent telephone switching offices, each of which was connected to thousands of telephones. These switching offices were, in turn, connected to higher-level switching offices (toll offices), to form a national hierarchy with only a small amount of redundancy. The vulnerability of the system was that the destruction of a few key toll offices could fragment it into many isolated islands.

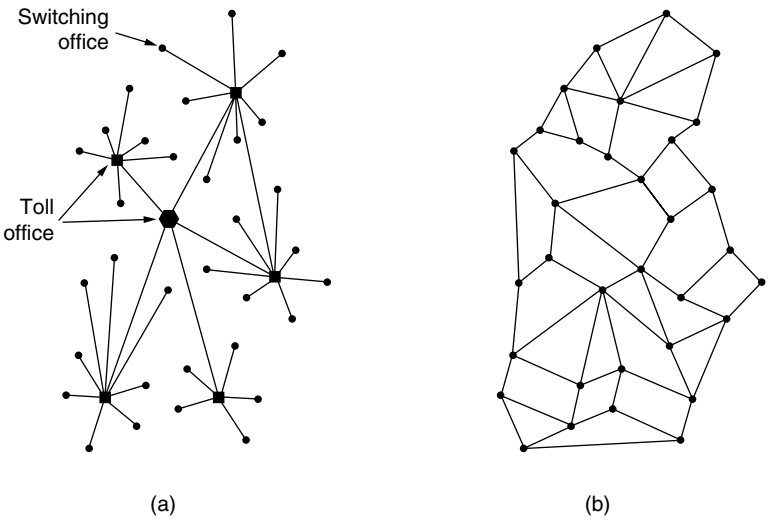


Figure 1-25. (a) Structure of the telephone system. (b) Baran's proposed distributed switching system.

Around 1960, the DoD awarded a contract to the RAND Corporation to find a solution. One of its employees, Paul Baran, came up with the highly distributed and fault-tolerant design of Fig. 1-25(b). Since the paths between any two switching offices were now much longer than analog signals could travel without distortion, Baran proposed using digital packet-switching technology. Baran wrote several reports for the DoD describing his ideas in detail (Baran, 1964). Officials at the Pentagon liked the concept and asked AT&T, then the U.S.' national telephone monopoly, to build a prototype. AT&T dismissed Baran's ideas out of hand. The biggest and richest corporation in the world was not about to allow

some young whippersnapper tell it how to build a telephone system. They said Baran's network could not be built and the idea was killed.

Several years went by and still the DoD did not have a better command-and-control system. To understand what happened next, we have to go back all the way to October 1957, when the Soviet Union beat the U.S. into space with the launch of the first artificial satellite, Sputnik. When President Eisenhower tried to find out who was asleep at the switch, he was appalled to find the Army, Navy, and Air Force squabbling over the Pentagon's research budget. His immediate response was to create a single defense research organization, **ARPA**, the **Advanced Research Projects Agency**. ARPA had no scientists or laboratories; in fact, it had nothing more than an office and a small (by Pentagon standards) budget. It did its work by issuing grants and contracts to universities and companies whose ideas looked promising to it.

For the first few years, ARPA tried to figure out what its mission should be. In 1967, the attention of Larry Roberts, a program manager at ARPA who was trying to figure out how to provide remote access to computers, turned to networking. He contacted various experts to decide what to do. One of them, Wesley Clark, suggested building a packet-switched subnet, connecting each host to its own router.

After some initial skepticism, Roberts bought the idea and presented a somewhat vague paper about it at the ACM SIGOPS Symposium on Operating System Principles held in Gatlinburg, Tennessee in late 1967 (Roberts, 1967). Much to Roberts' surprise, another paper at the conference described a similar system that had not only been designed but actually fully implemented under the direction of Donald Davies at the National Physical Laboratory in England. The NPL system was not a national system (it just connected several computers on the NPL campus), but it demonstrated that packet switching could be made to work. Furthermore, it cited Baran's now discarded earlier work. Roberts came away from Gatlinburg determined to build what later became known as the **ARPANET**.

The subnet would consist of minicomputers called **IMPs** (**Interface Message Processors**) connected by 56-kbps transmission lines. For high reliability, each IMP would be connected to at least two other IMPs. The subnet was to be a datagram subnet, so if some lines and IMPs were destroyed, messages could be automatically rerouted along alternative paths.

Each node of the network was to consist of an IMP and a host, in the same room, connected by a short wire. A host could send messages of up to 8063 bits to its IMP, which would then break these up into packets of at most 1008 bits and forward them independently toward the destination. Each packet was received in its entirety before being forwarded, so the subnet was the first electronic store-and-forward packet-switching network.

ARPA then put out a tender for building the subnet. Twelve companies bid for it. After evaluating all the proposals, ARPA selected BBN, a consulting firm based in Cambridge, Massachusetts, and in December 1968 awarded it a contract

to build the subnet and write the subnet software. BBN chose to use specially modified Honeywell DDP-316 minicomputers with 12K 16-bit words of core memory as the IMPs. The IMPs did not have disks, since moving parts were considered unreliable. The IMPs were interconnected by 56-kbps lines leased from telephone companies. Although 56 kbps is now the choice of teenagers who cannot afford DSL or cable, it was then the best money could buy.

The software was split into two parts: subnet and host. The subnet software consisted of the IMP end of the host-IMP connection, the IMP-IMP protocol, and a source IMP to destination IMP protocol designed to improve reliability. The original ARPANET design is shown in Fig. 1-26.

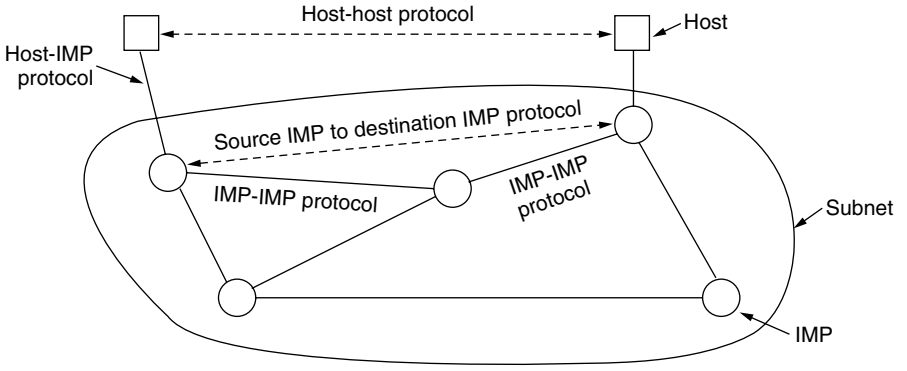


Figure 1-26. The original ARPANET design.

Outside the subnet, software was also needed, namely, the host end of the host-IMP connection, the host-host protocol, and the application software. It soon became clear that BBN was of the opinion that when it had accepted a message on a host-IMP wire and placed it on the host-IMP wire at the destination, its job was done.

Roberts had a problem, though: the hosts needed software too. To deal with it, he convened a meeting of network researchers, mostly graduate students, at Snowbird, Utah, in the summer of 1969. The graduate students expected some network expert to explain the grand design of the network and its software to them and then assign each of them the job of writing part of it. They were astounded when there was no network expert and no grand design. They had to figure out what to do on their own.

Nevertheless, somehow an experimental network went online in December 1969 with four nodes: at UCLA, UCSB, SRI, and the University of Utah. These four were chosen because all had a large number of ARPA contracts, and all had different and completely incompatible host computers (just to make it more fun). The first host-to-host message had been sent two months earlier from the UCLA

node by a team led by Len Kleinrock (a pioneer of the theory of packet switching) to the SRI node. The network grew quickly as more IMPs were delivered and installed; it soon spanned the United States. Figure 1-27 shows how rapidly the ARPANET grew in the first 3 years.

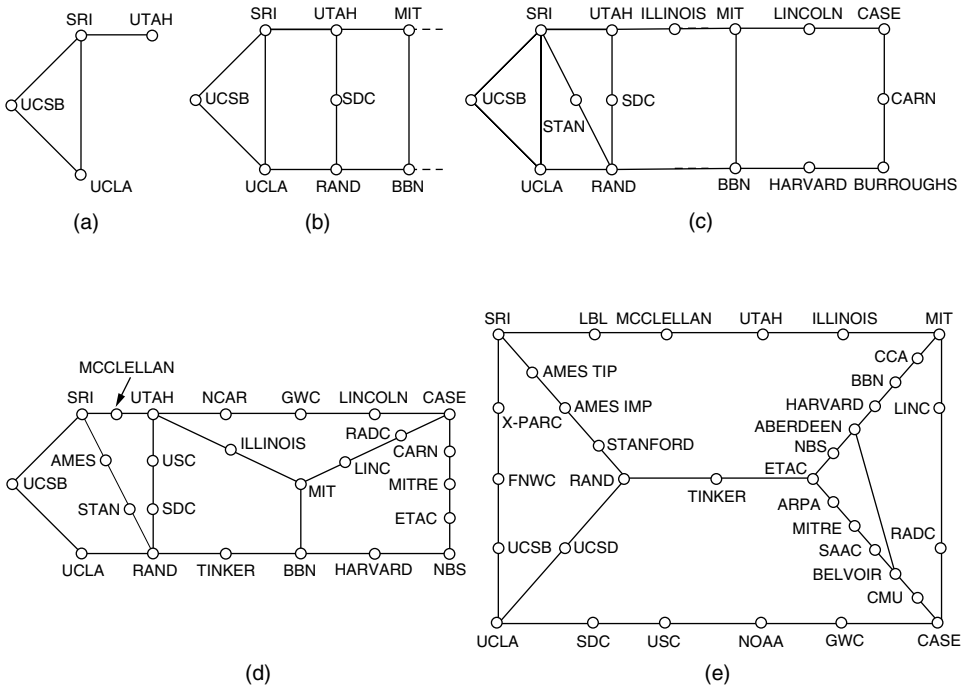


Figure 1-27. Growth of the ARPANET. (a) December 1969. (b) July 1970. (c) March 1971. (d) April 1972. (e) September 1972.

In addition to helping the fledgling ARPANET grow, ARPA also funded research on the use of satellite networks and mobile packet radio networks. In one now famous demonstration, a truck driving around in California used the packet radio network to send messages to SRI, which were then forwarded over the ARPANET to the East Coast, where they were shipped to University College in London over the satellite network. This allowed a researcher in the truck to use a computer in London while driving around in California.

This experiment also demonstrated that the existing ARPANET protocols were not suitable for running over different networks. This observation led to more research on protocols, culminating with the invention of the TCP/IP model and protocols (Cerf and Kahn, 1974). TCP/IP was specifically designed to handle communication over internetworks, something becoming increasingly important as more and more networks were hooked up to the ARPANET.

To encourage adoption of these new protocols, ARPA awarded several contracts to implement TCP/IP on different computer platforms, including IBM, DEC, and HP systems, as well as for Berkeley UNIX. Researchers at the University of California at Berkeley rewrote TCP/IP with a new programming interface called **sockets** for the upcoming 4.2BSD release of Berkeley UNIX. They also wrote many application, utility, and management programs to show how convenient it was to use the network with sockets.

The timing was perfect. Many universities had just acquired a second or third VAX computer and a LAN to connect them, but they had no networking software. When 4.2BSD came along, with TCP/IP, sockets, and many network utilities, the complete package was adopted immediately. Furthermore, with TCP/IP, it was easy for the LANs to connect to the ARPANET, and many did.

During the 1980s, additional networks, especially LANs, were connected to the ARPANET. As the scale increased, finding hosts became increasingly expensive, so **DNS** (**Domain Name System**) was created to organize machines into domains and map host names onto IP addresses. Since then, DNS has become a generalized, distributed database system for storing a variety of information related to naming. We will study it in detail in Chap. 7.

NSFNET

By the late 1970s, NSF (the U.S. National Science Foundation) saw the enormous impact the ARPANET was having on university research, allowing scientists across the country to share data and collaborate on research projects. However, to get on the ARPANET a university had to have a research contract with the DoD. Many did not have a contract. NSF's initial response was to fund the Computer Science Network (**CSNET**) in 1981. It connected computer science departments and industrial research labs to the ARPANET via dial-up and leased lines. In the late 1980s, the NSF went further and decided to design a successor to the ARPANET that would be open to all university research groups.

To have something concrete to start with, NSF decided to build a backbone network to connect its six supercomputer centers, in San Diego, Boulder, Champaign, Pittsburgh, Ithaca, and Princeton. Each supercomputer was given a little brother, consisting of an LSI-11 microcomputer called a **fuzzball**. The fuzzballs were connected with 56-kbps leased lines and formed the subnet, the same hardware technology the ARPANET used. The software technology was different however: the fuzzballs spoke TCP/IP right from the start, making it the first TCP/IP WAN.

NSF also funded some (eventually about 20) regional networks that connected to the backbone to allow users at thousands of universities, research labs, libraries, and museums to access any of the supercomputers and to communicate with one another. The complete network, including backbone and the regional networks, was called **NSFNET**. It connected to the ARPANET through a link between an

IMP and a fuzzball in the Carnegie-Mellon machine room. The first NSFNET backbone is illustrated in Fig. 1-28 superimposed on a map of the U.S.

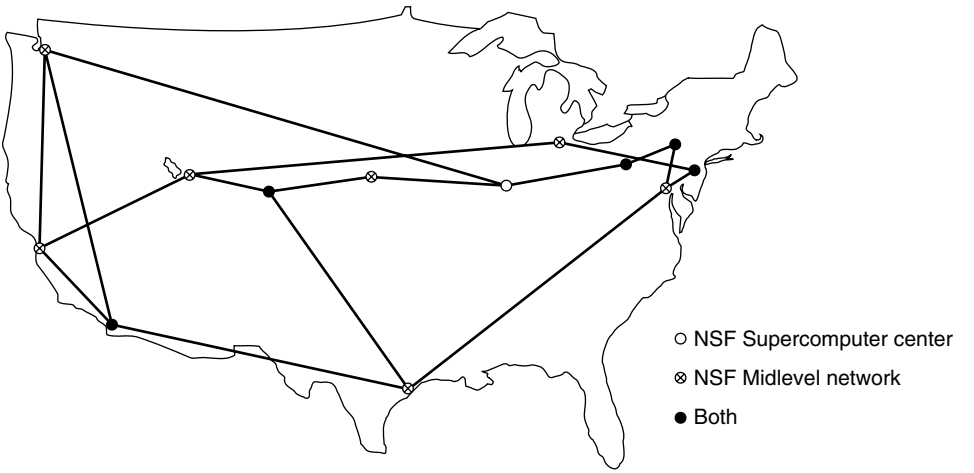


Figure 1-28. The NSFNET backbone in 1988.

NSFNET was an instantaneous success and was overloaded from the word go. NSF immediately began planning its successor and awarded a contract to the Michigan-based MERIT consortium to run it. Fiber optic channels at 448 kbps were leased from MCI (since merged with WorldCom) to provide the version 2 backbone. IBM PC-RTs were used as routers. This, too, was soon overwhelmed, and by 1990, the second backbone was upgraded to 1.5 Mbps.

As growth continued, NSF realized that the government could not continue financing networking forever. Furthermore, commercial organizations wanted to join but were forbidden by NSF's charter from using networks NSF paid for. Consequently, NSF encouraged MERIT, MCI, and IBM to form a nonprofit corporation, **ANS** (**Advanced Networks and Services**), as the first step along the road to commercialization. In 1990, ANS took over NSFNET and upgraded the 1.5-Mbps links to 45 Mbps to form **ANSNET**. This network operated for 5 years and was then sold to America Online. But by then, various companies were offering commercial IP service and it was clear the government should now get out of the networking business.

To ease the transition and make sure every regional network could communicate with every other regional network, NSF awarded contracts to four different network operators to establish a **NAP** (**Network Access Point**). These operators were PacBell (San Francisco), Ameritech (Chicago), MFS (Washington, D.C.), and Sprint (New York City, where for NAP purposes, Pennsauken, New Jersey counts as New York City). Every network operator that wanted to provide backbone service to the NSF regional networks had to connect to all the NAPs.

This arrangement meant that a packet originating on any regional network had a choice of backbone carriers to get from its NAP to the destination's NAP. Consequently, the backbone carriers were forced to compete for the regional networks' business on the basis of service and price, which was the idea, of course. As a result, the concept of a single default backbone was replaced by a commercially driven competitive infrastructure. Many people like to criticize the Federal Government for not being innovative, but in the area of networking, it was DoD and NSF that created the infrastructure that formed the basis for the Internet and then handed it over to industry to operate.

During the 1990s, many other countries and regions also built national research networks, often patterned on the ARPANET and NSFNET. These included EuropaNET and EBONE in Europe, which started out with 2-Mbps lines and then upgraded to 34-Mbps lines. Eventually, the network infrastructure in Europe was handed over to industry as well.

The Internet has changed a great deal since those early days. It exploded in size with the emergence of the World Wide Web (WWW) in the early 1990s. Recent data from the Internet Systems Consortium puts the number of visible Internet hosts at over 600 million. This guess is only a low-ball estimate, but it far exceeds the few million hosts that were around when the first conference on the WWW was held at CERN in 1994.

The way we use the Internet has also changed radically. Initially, applications such as email-for-academics, newsgroups, remote login, and file transfer dominated. Later it switched to email-for-everyman, then the Web and peer-to-peer content distribution, such as the now-shuttered Napster. Now real-time media distribution, social networks (e.g., Facebook), and microblogging (e.g., Twitter) are taking off. These switches brought richer kinds of media to the Internet and hence much more traffic. In fact, the dominant traffic on the Internet seems to change with some regularity as, for example, new and better ways to work with music or movies can become very popular very quickly.

Architecture of the Internet

The architecture of the Internet has also changed a great deal as it has grown explosively. In this section, we will attempt to give a brief overview of what it looks like today. The picture is complicated by continuous upheavals in the businesses of telephone companies (telcos), cable companies and ISPs that often make it hard to tell who is doing what. One driver of these upheavals is telecommunications convergence, in which one network is used for previously different uses. For example, in a "triple play" one company sells you telephony, TV, and Internet service over the same network connection on the assumption that this will save you money. Consequently, the description given here will be of necessity somewhat simpler than reality. And what is true today may not be true tomorrow.

The big picture is shown in Fig. 1-29. Let us examine this figure piece by piece, starting with a computer at home (at the edges of the figure). To join the Internet, the computer is connected to an **Internet Service Provider**, or simply **ISP**, from who the user purchases **Internet access** or **connectivity**. This lets the computer exchange packets with all of the other accessible hosts on the Internet. The user might send packets to surf the Web or for any of a thousand other uses, it does not matter. There are many kinds of Internet access, and they are usually distinguished by how much bandwidth they provide and how much they cost, but the most important attribute is connectivity.

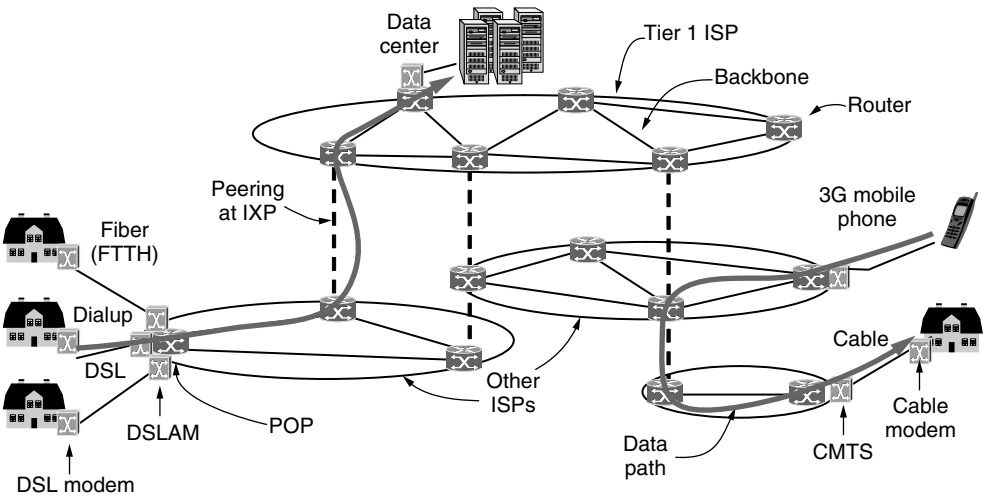


Figure 1-29. Overview of the Internet architecture.

A common way to connect to an ISP is to use the phone line to your house, in which case your phone company is your ISP. **DSL**, short for **Digital Subscriber Line**, reuses the telephone line that connects to your house for digital data transmission. The computer is connected to a device called a **DSL modem** that converts between digital packets and analog signals that can pass unhindered over the telephone line. At the other end, a device called a **DSLAM** (**Digital Subscriber Line Access Multiplexer**) converts between signals and packets.

Several other popular ways to connect to an ISP are shown in Fig. 1-29. DSL is a higher-bandwidth way to use the local telephone line than to send bits over a traditional telephone call instead of a voice conversation. That is called **dial-up** and done with a different kind of modem at both ends. The word **modem** is short for "*mo*dulator *dem*odulator" and refers to any device that converts between digital bits and analog signals.

Another method is to send signals over the cable TV system. Like DSL, this is a way to reuse existing infrastructure, in this case otherwise unused cable TV

channels. The device at the home end is called a **cable modem** and the device at the **cable headend** is called the **CMTS** (**Cable Modem Termination System**).

DSL and cable provide Internet access at rates from a small fraction of a megabit/sec to multiple megabit/sec, depending on the system. These rates are much greater than dial-up rates, which are limited to 56 kbps because of the narrow bandwidth used for voice calls. Internet access at much greater than dial-up speeds is called **broadband**. The name refers to the broader bandwidth that is used for faster networks, rather than any particular speed.

The access methods mentioned so far are limited by the bandwidth of the ''last mile'' or last leg of transmission. By running optical fiber to residences, faster Internet access can be provided at rates on the order of 10 to 100 Mbps. This design is called **FTTH** (**Fiber to the Home**). For businesses in commercial areas, it may make sense to lease a high-speed transmission line from the offices to the nearest ISP. For example, in North America, a T3 line runs at roughly 45 Mbps.

Wireless is used for Internet access too. An example we will explore shortly is that of 3G mobile phone networks. They can provide data delivery at rates of 1 Mbps or higher to mobile phones and fixed subscribers in the coverage area.

We can now move packets between the home and the ISP. We call the location at which customer packets enter the ISP network for service the ISP's **POP** (**Point of Presence**). We will next explain how packets are moved between the POPs of different ISPs. From this point on, the system is fully digital and packet switched.

ISP networks may be regional, national, or international in scope. We have already seen that their architecture is made up of long-distance transmission lines that interconnect routers at POPs in the different cities that the ISPs serve. This equipment is called the **backbone** of the ISP. If a packet is destined for a host served directly by the ISP, that packet is routed over the backbone and delivered to the host. Otherwise, it must be handed over to another ISP.

ISPs connect their networks to exchange traffic at **IXPs** (**Internet eXchange Points**). The connected ISPs are said to **peer** with each other. There are many IXPs in cities around the world. They are drawn vertically in Fig. 1-29 because ISP networks overlap geographically. Basically, an IXP is a room full of routers, at least one per ISP. A LAN in the room connects all the routers, so packets can be forwarded from any ISP backbone to any other ISP backbone. IXPs can be large and independently owned facilities. One of the largest is the Amsterdam Internet Exchange, to which hundreds of ISPs connect and through which they exchange hundreds of gigabits/sec of traffic.

The peering that happens at IXPs depends on the business relationships between ISPs. There are many possible relationships. For example, a small ISP might pay a larger ISP for Internet connectivity to reach distant hosts, much as a customer purchases service from an Internet provider. In this case, the small ISP is said to pay for **transit**. Alternatively, two large ISPs might decide to exchange

traffic so that each ISP can deliver some traffic to the other ISP without having to pay for transit. One of the many paradoxes of the Internet is that ISPs who publicly compete with one another for customers often privately cooperate to do peering (Metz, 2001).

The path a packet takes through the Internet depends on the peering choices of the ISPs. If the ISP delivering a packet peers with the destination ISP, it might deliver the packet directly to its peer. Otherwise, it might route the packet to the nearest place at which it connects to a paid transit provider so that provider can deliver the packet. Two example paths across ISPs are drawn in Fig. 1-29. Often, the path a packet takes will not be the shortest path through the Internet.

At the top of the food chain are a small handful of companies, like AT&T and Sprint, that operate large international backbone networks with thousands of routers connected by high-bandwidth fiber optic links. These ISPs do not pay for transit. They are usually called **tier 1** ISPs and are said to form the backbone of the Internet, since everyone else must connect to them to be able to reach the entire Internet.

Companies that provide lots of content, such as Google and Yahoo!, locate their computers in **data centers** that are well connected to the rest of the Internet. These data centers are designed for computers, not humans, and may be filled with rack upon rack of machines called a **server farm**. **Colocation** or **hosting** data centers let customers put equipment such as servers at ISP POPs so that short, fast connections can be made between the servers and the ISP backbones. The Internet hosting industry has become increasingly virtualized so that it is now common to rent a virtual machine that is run on a server farm instead of installing a physical computer. These data centers are so large (tens or hundreds of thousands of machines) that electricity is a major cost, so data centers are sometimes built in areas where electricity is cheap.

This ends our quick tour of the Internet. We will have a great deal to say about the individual components and their design, algorithms, and protocols in subsequent chapters. One further point worth mentioning here is that what it means to be on the Internet is changing. It used to be that a machine was on the Internet if it: (1) ran the TCP/IP protocol stack; (2) had an IP address; and (3) could send IP packets to all the other machines on the Internet. However, ISPs often reuse IP addresses depending on which computers are in use at the moment, and home networks often share one IP address between multiple computers. This practice undermines the second condition. Security measures such as firewalls can also partly block computers from receiving packets, undermining the third condition. Despite these difficulties, it makes sense to regard such machines as being on the Internet while they are connected to their ISPs.

Also worth mentioning in passing is that some companies have interconnected all their existing internal networks, often using the same technology as the Internet. These **intranets** are typically accessible only on company premises or from company notebooks but otherwise work the same way as the Internet.

1.5.2 Third-Generation Mobile Phone Networks

People love to talk on the phone even more than they like to surf the Internet, and this has made the mobile phone network the most successful network in the world. It has more than four billion subscribers worldwide. To put this number in perspective, it is roughly 60% of the world's population and more than the number of Internet hosts and fixed telephone lines combined (ITU, 2009).

The architecture of the mobile phone network has changed greatly over the past 40 years along with its tremendous growth. First-generation mobile phone systems transmitted voice calls as continuously varying (analog) signals rather than sequences of (digital) bits. **AMPS** (**Advanced Mobile Phone System**), which was deployed in the United States in 1982, was a widely used first-generation system. Second-generation mobile phone systems switched to transmitting voice calls in digital form to increase capacity, improve security, and offer text messaging. **GSM** (**Global System for Mobile communications**), which was deployed starting in 1991 and has become the most widely used mobile phone system in the world, is a 2G system.

The third generation, or 3G, systems were initially deployed in 2001 and offer both digital voice and broadband digital data services. They also come with a lot of jargon and many different standards to choose from. 3G is loosely defined by the ITU (an international standards body we will discuss in the next section) as providing rates of at least 2 Mbps for stationary or walking users and 384 kbps in a moving vehicle. **UMTS** (**Universal Mobile Telecommunications System**), also called **WCDMA** (**Wideband Code Division Multiple Access**), is the main 3G system that is being rapidly deployed worldwide. It can provide up to 14 Mbps on the downlink and almost 6 Mbps on the uplink. Future releases will use multiple antennas and radios to provide even greater speeds for users.

The scarce resource in 3G systems, as in 2G and 1G systems before them, is radio spectrum. Governments license the right to use parts of the spectrum to the mobile phone network operators, often using a spectrum auction in which network operators submit bids. Having a piece of licensed spectrum makes it easier to design and operate systems, since no one else is allowed transmit on that spectrum, but it often costs a serious amount of money. In the UK in 2000, for example, five 3G licenses were auctioned for a total of about $40 billion.

It is the scarcity of spectrum that led to the **cellular network** design shown in Fig. 1-30 that is now used for mobile phone networks. To manage the radio interference between users, the coverage area is divided into cells. Within a cell, users are assigned channels that do not interfere with each other and do not cause too much interference for adjacent cells. This allows for good reuse of the spectrum, or **frequency reuse**, in the neighboring cells, which increases the capacity of the network. In 1G systems, which carried each voice call on a specific frequency band, the frequencies were carefully chosen so that they did not conflict with neighboring cells. In this way, a given frequency might only be reused once

in several cells. Modern 3G systems allow each cell to use all frequencies, but in a way that results in a tolerable level of interference to the neighboring cells. There are variations on the cellular design, including the use of directional or sectored antennas on cell towers to further reduce interference, but the basic idea is the same.

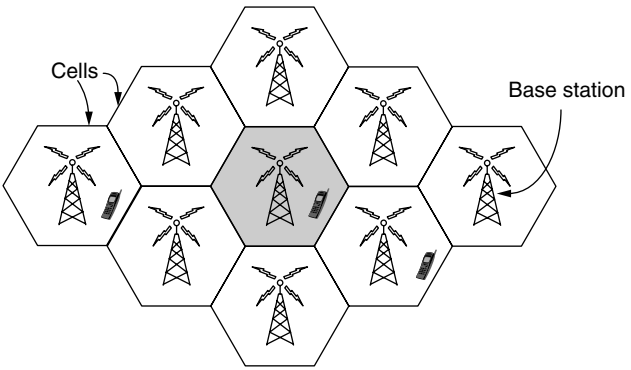


Figure 1-30. Cellular design of mobile phone networks.

The architecture of the mobile phone network is very different than that of the Internet. It has several parts, as shown in the simplified version of the UMTS architecture in Fig. 1-31. First, there is the **air interface**. This term is a fancy name for the radio communication protocol that is used over the air between the mobile device (e.g., the cell phone) and the **cellular base station**. Advances in the air interface over the past decades have greatly increased wireless data rates. The UMTS air interface is based on **Code Division Multiple Access** (**CDMA**), a technique that we will study in Chap. 2.

The cellular base station together with its controller forms the **radio access network**. This part is the wireless side of the mobile phone network. The controller node or **RNC** (**Radio Network Controller**) controls how the spectrum is used. The base station implements the air interface. It is called **Node B**, a temporary label that stuck.

The rest of the mobile phone network carries the traffic for the radio access network. It is called the **core network**. The UMTS core network evolved from the core network used for the 2G GSM system that came before it. However, something surprising is happening in the UMTS core network.

Since the beginning of networking, a war has been going on between the people who support packet networks (i.e., connectionless subnets) and the people who support circuit networks (i.e., connection-oriented subnets). The main proponents of packets come from the Internet community. In a connectionless design, every packet is routed independently of every other packet. As a consequence, if some routers go down during a session, no harm will be done as long as the system can

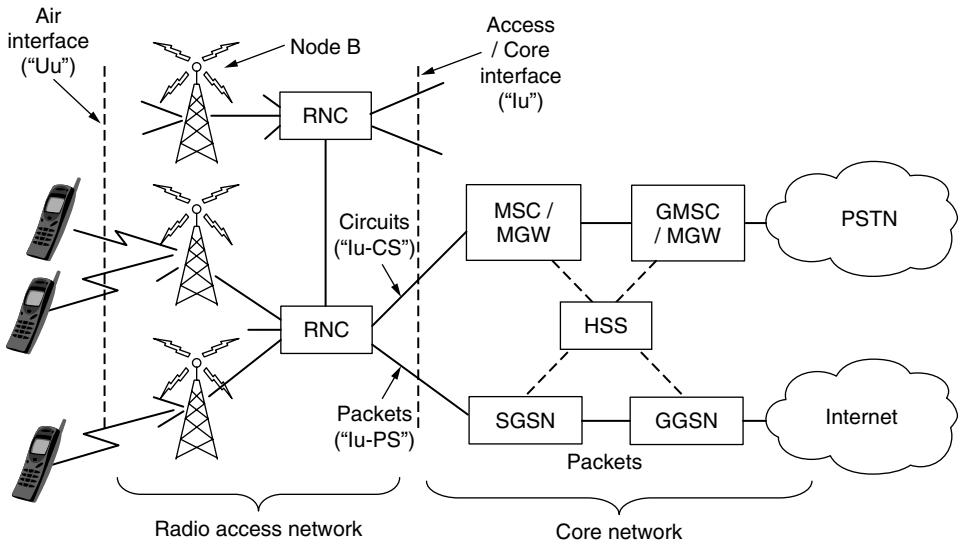


Figure 1-31. Architecture of the UMTS 3G mobile phone network.

dynamically reconfigure itself so that subsequent packets can find some route to the destination, even if it is different from that which previous packets used.

The circuit camp comes from the world of telephone companies. In the telephone system, a caller must dial the called party's number and wait for a connection before talking or sending data. This connection setup establishes a route through the telephone system that is maintained until the call is terminated. All words or packets follow the same route. If a line or switch on the path goes down, the call is aborted, making it less fault tolerant than a connectionless design.

The advantage of circuits is that they can support quality of service more easily. By setting up a connection in advance, the subnet can reserve resources such as link bandwidth, switch buffer space, and CPU. If an attempt is made to set up a call and insufficient resources are available, the call is rejected and the caller gets a kind of busy signal. In this way, once a connection has been set up, the connection will get good service.

With a connectionless network, if too many packets arrive at the same router at the same moment, the router will choke and probably lose packets. The sender will eventually notice this and resend them, but the quality of service will be jerky and unsuitable for audio or video unless the network is lightly loaded. Needless to say, providing adequate audio quality is something telephone companies care about very much, hence their preference for connections.

The surprise in Fig. 1-31 is that there is both packet and circuit switched equipment in the core network. This shows the mobile phone network in transition, with mobile phone companies able to implement one or sometimes both of

the alternatives. Older mobile phone networks used a circuit-switched core in the style of the traditional phone network to carry voice calls. This legacy is seen in the UMTS network with the **MSC** (**Mobile Switching Center**), **GMSC** (**Gateway Mobile Switching Center**), and **MGW** (**Media Gateway**) elements that set up connections over a circuit-switched core network such as the **PSTN** (**Public Switched Telephone Network**).

Data services have become a much more important part of the mobile phone network than they used to be, starting with text messaging and early packet data services such as **GPRS** (**General Packet Radio Service**) in the GSM system. These older data services ran at tens of kbps, but users wanted more. Newer mobile phone networks carry packet data at rates of multiple Mbps. For comparison, a voice call is carried at a rate of 64 kbps, typically 3–4x less with compression.

To carry all this data, the UMTS core network nodes connect directly to a packet-switched network. The **SGSN** (**Serving GPRS Support Node**) and the **GGSN** (**Gateway GPRS Support Node**) deliver data packets to and from mobiles and interface to external packet networks such as the Internet.

This transition is set to continue in the mobile phone networks that are now being planned and deployed. Internet protocols are even used on mobiles to set up connections for voice calls over a packet data network, in the manner of voice-over-IP. IP and packets are used all the way from the radio access through to the core network. Of course, the way that IP networks are designed is also changing to support better quality of service. If it did not, then problems with chopped-up audio and jerky video would not impress paying customers. We will return to this subject in Chap. 5.

Another difference between mobile phone networks and the traditional Internet is mobility. When a user moves out of the range of one cellular base station and into the range of another one, the flow of data must be re-routed from the old to the new cell base station. This technique is known as **handover** or **handoff**, and it is illustrated in Fig. 1-32.

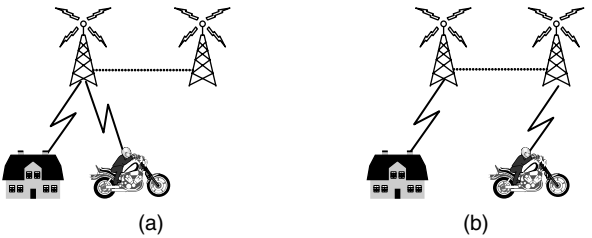


Figure 1-32. Mobile phone handover (a) before, (b) after.

Either the mobile device or the base station may request a handover when the quality of the signal drops. In some cell networks, usually those based on CDMA

technology, it is possible to connect to the new base station before disconnecting from the old base station. This improves the connection quality for the mobile because there is no break in service; the mobile is actually connected to two base stations for a short while. This way of doing a handover is called a **soft handover** to distinguish it from a **hard handover**, in which the mobile disconnects from the old base station before connecting to the new one.

A related issue is how to find a mobile in the first place when there is an incoming call. Each mobile phone network has a **HSS** (**Home Subscriber Server**) in the core network that knows the location of each subscriber, as well as other profile information that is used for authentication and authorization. In this way, each mobile can be found by contacting the HSS.

A final area to discuss is security. Historically, phone companies have taken security much more seriously than Internet companies for a long time because of the need to bill for service and avoid (payment) fraud. Unfortunately that is not saying much. Nevertheless, in the evolution from 1G through 3G technologies, mobile phone companies have been able to roll out some basic security mechanisms for mobiles.

Starting with the 2G GSM system, the mobile phone was divided into a handset and a removable chip containing the subscriber's identity and account information. The chip is informally called a **SIM card**, short for **Subscriber Identity Module**. SIM cards can be switched to different handsets to activate them, and they provide a basis for security. When GSM customers travel to other countries on vacation or business, they often bring their handsets but buy a new SIM card for few dollars upon arrival in order to make local calls with no roaming charges.

To reduce fraud, information on SIM cards is also used by the mobile phone network to authenticate subscribers and check that they are allowed to use the network. With UMTS, the mobile also uses the information on the SIM card to check that it is talking to a legitimate network.

Another aspect of security is privacy. Wireless signals are broadcast to all nearby receivers, so to make it difficult to eavesdrop on conversations, cryptographic keys on the SIM card are used to encrypt transmissions. This approach provides much better privacy than in 1G systems, which were easily tapped, but is not a panacea due to weaknesses in the encryption schemes.

Mobile phone networks are destined to play a central role in future networks. They are now more about mobile broadband applications than voice calls, and this has major implications for the air interfaces, core network architecture, and security of future networks. 4G technologies that are faster and better are on the drawing board under the name of **LTE** (**Long Term Evolution**), even as 3G design and deployment continues. Other wireless technologies also offer broadband Internet access to fixed and mobile clients, notably 802.16 networks under the common name of **WiMAX**. It is entirely possible that LTE and WiMAX are on a collision course with each other and it is hard to predict what will happen to them.

1.5.3 Wireless LANs: 802.11

Almost as soon as laptop computers appeared, many people had a dream of walking into an office and magically having their laptop computer be connected to the Internet. Consequently, various groups began working on ways to accomplish this goal. The most practical approach is to equip both the office and the laptop computers with short-range radio transmitters and receivers to allow them to talk.

Work in this field rapidly led to wireless LANs being marketed by a variety of companies. The trouble was that no two of them were compatible. The proliferation of standards meant that a computer equipped with a brand *X* radio would not work in a room equipped with a brand *Y* base station. In the mid 1990s, the industry decided that a wireless LAN standard might be a good idea, so the IEEE committee that had standardized wired LANs was given the task of drawing up a wireless LAN standard.

The first decision was the easiest: what to call it. All the other LAN standards had numbers like 802.1, 802.2, and 802.3, up to 802.10, so the wireless LAN standard was dubbed 802.11. A common slang name for it is **WiFi** but it is an important standard and deserves respect, so we will call it by its proper name, 802.11.

The rest was harder. The first problem was to find a suitable frequency band that was available, preferably worldwide. The approach taken was the opposite of that used in mobile phone networks. Instead of expensive, licensed spectrum, 802.11 systems operate in unlicensed bands such as the **ISM** (**Industrial, Scientific, and Medical**) bands defined by ITU-R (e.g., 902-928 MHz, 2.4-2.5 GHz, 5.725-5.825 GHz). All devices are allowed to use this spectrum provided that they limit their transmit power to let different devices coexist. Of course, this means that 802.11 radios may find themselves competing with cordless phones, garage door openers, and microwave ovens.

802.11 networks are made up of clients, such as laptops and mobile phones, and infrastructure called **APs** (**access points**) that is installed in buildings. Access points are sometimes called **base stations**. The access points connect to the wired network, and all communication between clients goes through an access point. It is also possible for clients that are in radio range to talk directly, such as two computers in an office without an access point. This arrangement is called an **ad hoc network**. It is used much less often than the access point mode. Both modes are shown in Fig. 1-33.

802.11 transmission is complicated by wireless conditions that vary with even small changes in the environment. At the frequencies used for 802.11, radio signals can be reflected off solid objects so that multiple echoes of a transmission may reach a receiver along different paths. The echoes can cancel or reinforce each other, causing the received signal to fluctuate greatly. This phenomenon is called **multipath fading**, and it is shown in Fig. 1-34.

The key idea for overcoming variable wireless conditions is **path diversity**, or the sending of information along multiple, independent paths. In this way, the

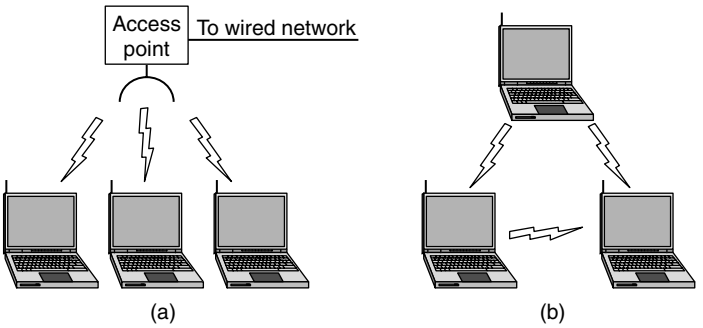


Figure 1-33. (a) Wireless network with an access point. (b) Ad hoc network.

information is likely to be received even if one of the paths happens to be poor due to a fade. These independent paths are typically built into the digital modulation scheme at the physical layer. Options include using different frequencies across the allowed band, following different spatial paths between different pairs of antennas, or repeating bits over different periods of time.

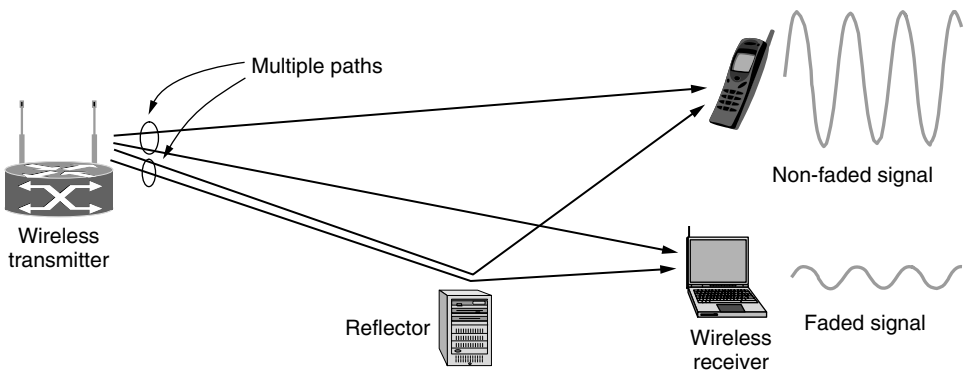


Figure 1-34. Multipath fading.

Different versions of 802.11 have used all of these techniques. The initial (1997) standard defined a wireless LAN that ran at either 1 Mbps or 2 Mbps by hopping between frequencies or spreading the signal across the allowed spectrum. Almost immediately, people complained that it was too slow, so work began on faster standards. The spread spectrum design was extended and became the (1999) 802.11b standard running at rates up to 11 Mbps. The 802.11a (1999) and 802.11g (2003) standards switched to a different modulation scheme called **OFDM** (**Orthogonal Frequency Division Multiplexing**). It divides a wide band of spectrum into many narrow slices over which different bits are sent in parallel. This improved scheme, which we will study in Chap. 2, boosted the 802.11a/g bit

rates up to 54 Mbps. That is a significant increase, but people still wanted more throughput to support more demanding uses. The latest version is 802.11n (2009). It uses wider frequency bands and up to four antennas per computer to achieve rates up to 450 Mbps.

Since wireless is inherently a broadcast medium, 802.11 radios also have to deal with the problem that multiple transmissions that are sent at the same time will collide, which may interfere with reception. To handle this problem, 802.11 uses a **CSMA** (**Carrier Sense Multiple Access**) scheme that draws on ideas from classic wired Ethernet, which, ironically, drew from an early wireless network developed in Hawaii and called **ALOHA**. Computers wait for a short random interval before transmitting, and defer their transmissions if they hear that someone else is already transmitting. This scheme makes it less likely that two computers will send at the same time. It does not work as well as in the case of wired networks, though. To see why, examine Fig. 1-35. Suppose that computer *A* is transmitting to computer *B*, but the radio range of *A*'s transmitter is too short to reach computer *C*. If *C* wants to transmit to *B* it can listen before starting, but the fact that it does not hear anything does not mean that its transmission will succeed. The inability of *C* to hear *A* before starting causes some collisions to occur. After any collision, the sender then waits another, longer, random delay and retransmits the packet. Despite this and some other issues, the scheme works well enough in practice.

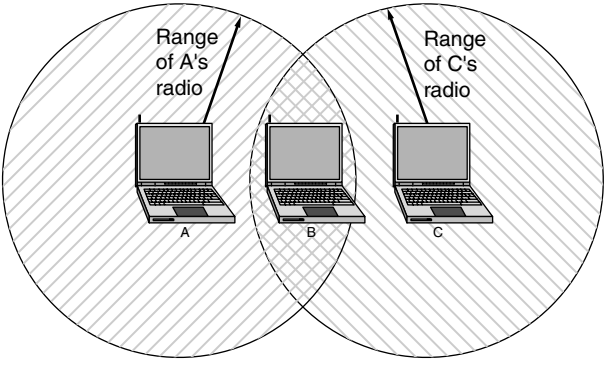


Figure 1-35. The range of a single radio may not cover the entire system.

Another problem is that of mobility. If a mobile client is moved away from the access point it is using and into the range of a different access point, some way of handing it off is needed. The solution is that an 802.11 network can consist of multiple cells, each with its own access point, and a distribution system that connects the cells. The distribution system is often switched Ethernet, but it can use any technology. As the clients move, they may find another access point with a better signal than the one they are currently using and change their association. From the outside, the entire system looks like a single wired LAN.

That said, mobility in 802.11 has been of limited value so far compared to mobility in the mobile phone network. Typically, 802.11 is used by nomadic clients that go from one fixed location to another, rather than being used on-the-go. Mobility is not really needed for nomadic usage. Even when 802.11 mobility is used, it extends over a single 802.11 network, which might cover at most a large building. Future schemes will need to provide mobility across different networks and across different technologies (e.g., 802.21).

Finally, there is the problem of security. Since wireless transmissions are broadcast, it is easy for nearby computers to receive packets of information that were not intended for them. To prevent this, the 802.11 standard included an encryption scheme known as **WEP** (**Wired Equivalent Privacy**). The idea was to make wireless security like that of wired security. It is a good idea, but unfortunately the scheme was flawed and soon broken (Borisov et al., 2001). It has since been replaced with newer schemes that have different cryptographic details in the 802.11i standard, also called **WiFi Protected Access**, initially called **WPA** but now replaced by **WPA2**.

802.11 has caused a revolution in wireless networking that is set to continue. Beyond buildings, it is starting to be installed in trains, planes, boats, and automobiles so that people can surf the Internet wherever they go. Mobile phones and all manner of consumer electronics, from game consoles to digital cameras, can communicate with it. We will come back to it in detail in Chap. 4.

1.5.4 RFID and Sensor Networks

The networks we have studied so far are made up of computing devices that are easy to recognize, from computers to mobile phones. With **Radio Frequency IDentification** (**RFID**), everyday objects can also be part of a computer network.

An RFID tag looks like a postage stamp-sized sticker that can be affixed to (or embedded in) an object so that it can be tracked. The object might be a cow, a passport, a book or a shipping pallet. The tag consists of a small microchip with a unique identifier and an antenna that receives radio transmissions. RFID readers installed at tracking points find tags when they come into range and interrogate them for their information as shown in Fig. 1-36. Applications include checking identities, managing the supply chain, timing races, and replacing barcodes.

There are many kinds of RFID, each with different properties, but perhaps the most fascinating aspect of RFID technology is that most RFID tags have neither an electric plug nor a battery. Instead, all of the energy needed to operate them is supplied in the form of radio waves by RFID readers. This technology is called **passive RFID** to distinguish it from the (less common) **active RFID** in which there is a power source on the tag.

One common form of RFID is **UHF RFID** (**Ultra-High Frequency RFID**). It is used on shipping pallets and some drivers licenses. Readers send signals in

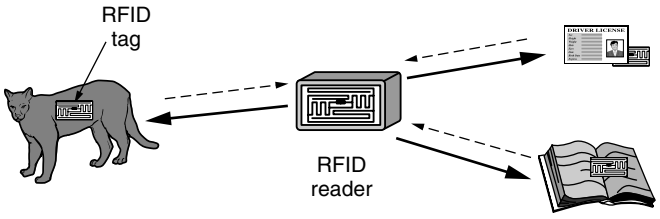


Figure 1-36. RFID used to network everyday objects.

the 902-928 MHz band in the United States. Tags communicate at distances of several meters by changing the way they reflect the reader signals; the reader is able to pick up these reflections. This way of operating is called **backscatter**.

Another popular kind of RFID is **HF RFID** (**High Frequency RFID**). It operates at 13.56 MHz and is likely to be in your passport, credit cards, books, and noncontact payment systems. HF RFID has a short range, typically a meter or less, because the physical mechanism is based on induction rather than backscatter. There are also other forms of RFID using other frequencies, such as **LF RFID** (**Low Frequency RFID**), which was developed before HF RFID and used for animal tracking. It is the kind of RFID likely to be in your cat.

RFID readers must somehow solve the problem of dealing with multiple tags within reading range. This means that a tag cannot simply respond when it hears a reader, or the signals from multiple tags may collide. The solution is similar to the approach taken in 802.11: tags wait for a short random interval before responding with their identification, which allows the reader to narrow down individual tags and interrogate them further.

Security is another problem. The ability of RFID readers to easily track an object, and hence the person who uses it, can be an invasion of privacy. Unfortunately, it is difficult to secure RFID tags because they lack the computation and communication power to run strong cryptographic algorithms. Instead, weak measures like passwords (which can easily be cracked) are used. If an identity card can be remotely read by an official at a border, what is to stop the same card from being tracked by other people without your knowledge? Not much.

RFID tags started as identification chips, but are rapidly turning into full-fledged computers. For example, many tags have memory that can be updated and later queried, so that information about what has happened to the tagged object can be stored with it. Rieback et al. (2006) demonstrated that this means that all of the usual problems of computer malware apply, only now your cat or your passport might be used to spread an RFID virus.

A step up in capability from RFID is the **sensor network**. Sensor networks are deployed to monitor aspects of the physical world. So far, they have mostly been used for scientific experimentation, such as monitoring bird habitats, volcanic activity, and zebra migration, but business applications including healthcare,

monitoring equipment for vibration, and tracking of frozen, refrigerated, or otherwise perishable goods cannot be too far behind.

Sensor nodes are small computers, often the size of a key fob, that have temperature, vibration, and other sensors. Many nodes are placed in the environment that is to be monitored. Typically, they have batteries, though they may scavenge energy from vibrations or the sun. As with RFID, having enough energy is a key challenge, and the nodes must communicate carefully to be able to deliver their sensor information to an external collection point. A common strategy is for the nodes to self-organize to relay messages for each other, as shown in Fig. 1-37. This design is called a **multihop network**.

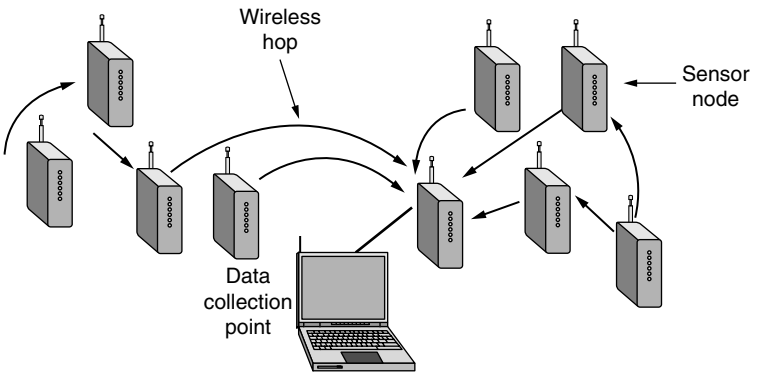


Figure 1-37. Multihop topology of a sensor network.

RFID and sensor networks are likely to become much more capable and pervasive in the future. Researchers have already combined the best of both technologies by prototyping programmable RFID tags with light, movement, and other sensors (Sample et al., 2008).

1.6 NETWORK STANDARDIZATION

Many network vendors and suppliers exist, each with its own ideas of how things should be done. Without coordination, there would be complete chaos, and users would get nothing done. The only way out is to agree on some network standards. Not only do good standards allow different computers to communicate, but they also increase the market for products adhering to the standards. A larger market leads to mass production, economies of scale in manufacturing, better implementations, and other benefits that decrease price and further increase acceptance.

In this section we will take a quick look at the important but little-known, world of international standardization. But let us first discuss what belongs in a

standard. A reasonable person might assume that a standard tells you how a protocol should work so that you can do a good job of implementing it. That person would be wrong.

Standards define what is needed for interoperability: no more, no less. That lets the larger market emerge and also lets companies compete on the basis of how good their products are. For example, the 802.11 standard defines many transmission rates but does not say when a sender should use which rate, which is a key factor in good performance. That is up to whoever makes the product. Often getting to interoperability this way is difficult, since there are many implementation choices and standards usually define many options. For 802.11, there were so many problems that, in a strategy that has become common practice, a trade group called the **WiFi Alliance** was started to work on interoperability within the 802.11 standard.

Similarly, a protocol standard defines the protocol over the wire but not the service interface inside the box, except to help explain the protocol. Real service interfaces are often proprietary. For example, the way TCP interfaces to IP within a computer does not matter for talking to a remote host. It only matters that the remote host speaks TCP/IP. In fact, TCP and IP are commonly implemented together without any distinct interface. That said, good service interfaces, like good APIs, are valuable for getting protocols used, and the best ones (such as Berkeley sockets) can become very popular.

Standards fall into two categories: de facto and de jure. **De facto** (Latin for "from the fact") standards are those that have just happened, without any formal plan. HTTP, the protocol on which the Web runs, started life as a de facto standard. It was part of early WWW browsers developed by Tim Berners-Lee at CERN, and its use took off with the growth of the Web. Bluetooth is another example. It was originally developed by Ericsson but now everyone is using it.

De jure (Latin for "by law") standards, in contrast, are adopted through the rules of some formal standardization body. International standardization authorities are generally divided into two classes: those established by treaty among national governments, and those comprising voluntary, nontreaty organizations. In the area of computer network standards, there are several organizations of each type, notably ITU, ISO, IETF and IEEE, all of which we will discuss below.

In practice, the relationships between standards, companies, and standardization bodies are complicated. De facto standards often evolve into de jure standards, especially if they are successful. This happened in the case of HTTP, which was quickly picked up by IETF. Standards bodies often ratify each others' standards, in what looks like patting one another on the back, to increase the market for a technology. These days, many ad hoc business alliances that are formed around particular technologies also play a significant role in developing and refining network standards. For example, **3GPP** (**Third Generation Partnership Project**) is a collaboration between telecommunications associations that drives the UMTS 3G mobile phone standards.

1.6.1 Who's Who in the Telecommunications World

The legal status of the world's telephone companies varies considerably from country to country. At one extreme is the United States, which has over 2000 separate, (mostly very small) privately owned telephone companies. A few more were added with the breakup of AT&T in 1984 (which was then the world's largest corporation, providing telephone service to about 80 percent of America's telephones), and the Telecommunications Act of 1996 that overhauled regulation to foster competition.

At the other extreme are countries in which the national government has a complete monopoly on all communication, including the mail, telegraph, telephone, and often radio and television. Much of the world falls into this category. In some cases the telecommunication authority is a nationalized company, and in others it is simply a branch of the government, usually known as the **PTT** (**Post, Telegraph & Telephone** administration). Worldwide, the trend is toward liberalization and competition and away from government monopoly. Most European countries have now (partially) privatized their PTTs, but elsewhere the process is still only slowly gaining steam.

With all these different suppliers of services, there is clearly a need to provide compatibility on a worldwide scale to ensure that people (and computers) in one country can call their counterparts in another one. Actually, this need has existed for a long time. In 1865, representatives from many European governments met to form the predecessor to today's **ITU** (**International Telecommunication Union**). Its job was to standardize international telecommunications, which in those days meant telegraphy. Even then it was clear that if half the countries used Morse code and the other half used some other code, there was going to be a problem. When the telephone was put into international service, ITU took over the job of standardizing telephony (pronounced te-LEF-ony) as well. In 1947, ITU became an agency of the United Nations.

ITU has about 200 governmental members, including almost every member of the United Nations. Since the United States does not have a PTT, somebody else had to represent it in ITU. This task fell to the State Department, probably on the grounds that ITU had to do with foreign countries, the State Department's specialty. ITU also has more than 700 sector and associate members. They include telephone companies (e.g., AT&T, Vodafone, Sprint), telecom equipment manufacturers (e.g., Cisco, Nokia, Nortel), computer vendors (e.g., Microsoft, Agilent, Toshiba), chip manufacturers (e.g., Intel, Motorola, TI), and other interested companies (e.g., Boeing, CBS, VeriSign).

ITU has three main sectors. We will focus primarily on **ITU-T**, the Telecommunications Standardization Sector, which is concerned with telephone and data communication systems. Before 1993, this sector was called **CCITT**, which is an acronym for its French name, Comité Consultatif International Télégraphique et Téléphonique. **ITU-R**, the Radiocommunications Sector, is concerned with

coordinating the use by competing interest groups of radio frequencies worldwide. The other sector is ITU-D, the Development Sector. It promotes the development of information and communication technologies to narrow the "digital divide" between countries with effective access to the information technologies and countries with limited access.

ITU-T's task is to make technical recommendations about telephone, telegraph, and data communication interfaces. These often become internationally recognized standards, though technically the recommendations are only suggestions that governments can adopt or ignore, as they wish (because governments are like 13-year-old boys—they do not take kindly to being given orders). In practice, a country that wishes to adopt a telephone standard different from that used by the rest of the world is free to do so, but at the price of cutting itself off from everyone else. This might work for North Korea, but elsewhere it would be a real problem.

The real work of ITU-T is done in its Study Groups. There are currently 10 Study Groups, often as large as 400 people, that cover topics ranging from telephone billing to multimedia services to security. SG 15, for example, standardizes the DSL technologies popularly used to connect to the Internet. In order to make it possible to get anything at all done, the Study Groups are divided into Working Parties, which are in turn divided into Expert Teams, which are in turn divided into ad hoc groups. Once a bureaucracy, always a bureaucracy.

Despite all this, ITU-T actually does get things done. Since its inception, it has produced more than 3000 recommendations, many of which are widely used in practice. For example, Recommendation H.264 (also an ISO standard known as MPEG-4 AVC) is widely used for video compression, and X.509 public key certificates are used for secure Web browsing and digitally signed email.

As the field of telecommunications completes the transition started in the 1980s from being entirely national to being entirely global, standards will become increasingly important, and more and more organizations will want to become involved in setting them. For more information about ITU, see Irmer (1994).

1.6.2 Who's Who in the International Standards World

International standards are produced and published by **ISO** (**International Standards Organization**†), a voluntary nontreaty organization founded in 1946. Its members are the national standards organizations of the 157 member countries. These members include ANSI (U.S.), BSI (Great Britain), AFNOR (France), DIN (Germany), and 153 others.

ISO issues standards on a truly vast number of subjects, ranging from nuts and bolts (literally) to telephone pole coatings [not to mention cocoa beans (ISO 2451), fishing nets (ISO 1530), women's underwear (ISO 4416) and quite a few

† For the purist, ISO's true name is the International Organization for Standardization.

other subjects one might not think were subject to standardization]. On issues of telecommunication standards, ISO and ITU-T often cooperate (ISO is a member of ITU-T) to avoid the irony of two official and mutually incompatible international standards.

Over 17,000 standards have been issued, including the OSI standards. ISO has over 200 Technical Committees (TCs), numbered in the order of their creation, each dealing with a specific subject. TC1 deals with the nuts and bolts (standardizing screw thread pitches). JTC1 deals with information technology, including networks, computers, and software. It is the first (and so far only) Joint Technical Committee, created in 1987 by merging TC97 with activities in IEC, yet another standardization body. Each TC has subcommittees (SCs) divided into working groups (WGs).

The real work is done largely in the WGs by over 100,000 volunteers worldwide. Many of these "volunteers" are assigned to work on ISO matters by their employers, whose products are being standardized. Others are government officials keen on having their country's way of doing things become the international standard. Academic experts also are active in many of the WGs.

The procedure used by ISO for adopting standards has been designed to achieve as broad a consensus as possible. The process begins when one of the national standards organizations feels the need for an international standard in some area. A working group is then formed to come up with a **CD** (**Committee Draft**). The CD is then circulated to all the member bodies, which get 6 months to criticize it. If a substantial majority approves, a revised document, called a **DIS** (**Draft International Standard**) is produced and circulated for comments and voting. Based on the results of this round, the final text of the **IS** (**International Standard**) is prepared, approved, and published. In areas of great controversy, a CD or DIS may have to go through several versions before acquiring enough votes, and the whole process can take years.

NIST (**National Institute of Standards and Technology**) is part of the U.S. Department of Commerce. It used to be called the National Bureau of Standards. It issues standards that are mandatory for purchases made by the U.S. Government, except for those of the Department of Defense, which defines its own standards.

Another major player in the standards world is **IEEE** (**Institute of Electrical and Electronics Engineers**), the largest professional organization in the world. In addition to publishing scores of journals and running hundreds of conferences each year, IEEE has a standardization group that develops standards in the area of electrical engineering and computing. IEEE's 802 committee has standardized many kinds of LANs. We will study some of its output later in this book. The actual work is done by a collection of working groups, which are listed in Fig. 1-38. The success rate of the various 802 working groups has been low; having an 802.x number is no guarantee of success. Still, the impact of the success stories (especially 802.3 and 802.11) on the industry and the world has been enormous.

Number	Topic
802.1	Overview and architecture of LANs
802.2 ↓	Logical link control
802.3 *	Ethernet
802.4 ↓	Token bus (was briefly used in manufacturing plants)
802.5	Token ring (IBM's entry into the LAN world)
802.6 ↓	Dual queue dual bus (early metropolitan area network)
802.7 ↓	Technical advisory group on broadband technologies
802.8 †	Technical advisory group on fiber optic technologies
802.9 ↓	Isochronous LANs (for real-time applications)
802.10 ↓	Virtual LANs and security
802.11 *	Wireless LANs (WiFi)
802.12 ↓	Demand priority (Hewlett-Packard's AnyLAN)
802.13	Unlucky number; nobody wanted it
802.14 ↓	Cable modems (defunct: an industry consortium got there first)
802.15 *	Personal area networks (Bluetooth, Zigbee)
802.16 *	Broadband wireless (WiMAX)
802.17	Resilient packet ring
802.18	Technical advisory group on radio regulatory issues
802.19	Technical advisory group on coexistence of all these standards
802.20	Mobile broadband wireless (similar to 802.16e)
802.21	Media independent handoff (for roaming over technologies)
802.22	Wireless regional area network

Figure 1-38. The 802 working groups. The important ones are marked with *. The ones marked with ↓ are hibernating. The one marked with † gave up and disbanded itself.

1.6.3 Who's Who in the Internet Standards World

The worldwide Internet has its own standardization mechanisms, very different from those of ITU-T and ISO. The difference can be crudely summed up by saying that the people who come to ITU or ISO standardization meetings wear suits, while the people who come to Internet standardization meetings wear jeans (except when they meet in San Diego, when they wear shorts and T-shirts).

ITU-T and ISO meetings are populated by corporate officials and government civil servants for whom standardization is their job. They regard standardization as a Good Thing and devote their lives to it. Internet people, on the other hand, prefer anarchy as a matter of principle. However, with hundreds of millions of

people all doing their own thing, little communication can occur. Thus, standards, however regrettable, are sometimes needed. In this context, David Clark of M.I.T. once made a now-famous remark about Internet standardization consisting of "rough consensus and running code."

When the ARPANET was set up, DoD created an informal committee to oversee it. In 1983, the committee was renamed the **IAB** (**Internet Activities Board**) and was given a slighter broader mission, namely, to keep the researchers involved with the ARPANET and the Internet pointed more or less in the same direction, an activity not unlike herding cats. The meaning of the acronym "IAB" was later changed to **Internet Architecture Board**.

Each of the approximately ten members of the IAB headed a task force on some issue of importance. The IAB met several times a year to discuss results and to give feedback to the DoD and NSF, which were providing most of the funding at this time. When a standard was needed (e.g., a new routing algorithm), the IAB members would thrash it out and then announce the change so the graduate students who were the heart of the software effort could implement it. Communication was done by a series of technical reports called **RFCs** (**Request For Comments**). RFCs are stored online and can be fetched by anyone interested in them from *www.ietf.org/rfc*. They are numbered in chronological order of creation. Over 5000 now exist. We will refer to many RFCs in this book.

By 1989, the Internet had grown so large that this highly informal style no longer worked. Many vendors by then offered TCP/IP products and did not want to change them just because ten researchers had thought of a better idea. In the summer of 1989, the IAB was reorganized again. The researchers were moved to the **IRTF** (**Internet Research Task Force**), which was made subsidiary to IAB, along with the **IETF** (**Internet Engineering Task Force**). The IAB was repopulated with people representing a broader range of organizations than just the research community. It was initially a self-perpetuating group, with members serving for a 2-year term and new members being appointed by the old ones. Later, the **Internet Society** was created, populated by people interested in the Internet. The Internet Society is thus in a sense comparable to ACM or IEEE. It is governed by elected trustees who appoint the IAB's members.

The idea of this split was to have the IRTF concentrate on long-term research while the IETF dealt with short-term engineering issues. The IETF was divided up into working groups, each with a specific problem to solve. The chairmen of these working groups initially met as a steering committee to direct the engineering effort. The working group topics include new applications, user information, OSI integration, routing and addressing, security, network management, and standards. Eventually, so many working groups were formed (more than 70) that they were grouped into areas and the area chairmen met as the steering committee.

In addition, a more formal standardization process was adopted, patterned after ISOs. To become a **Proposed Standard**, the basic idea must be explained in an RFC and have sufficient interest in the community to warrant consideration.

To advance to the **Draft Standard** stage, a working implementation must have been rigorously tested by at least two independent sites for at least 4 months. If the IAB is convinced that the idea is sound and the software works, it can declare the RFC to be an **Internet Standard**. Some Internet Standards have become DoD standards (MIL-STD), making them mandatory for DoD suppliers.

For Web standards, the **World Wide Web Consortium** (**W3C**) develops protocols and guidelines to facilitate the long-term growth of the Web. It is an industry consortium led by Tim Berners-Lee and set up in 1994 as the Web really begun to take off. W3C now has more than 300 members from around the world and has produced more than 100 W3C Recommendations, as its standards are called, covering topics such as HTML and Web privacy.

1.7 METRIC UNITS

To avoid any confusion, it is worth stating explicitly that in this book, as in computer science in general, metric units are used instead of traditional English units (the furlong-stone-fortnight system). The principal metric prefixes are listed in Fig. 1-39. The prefixes are typically abbreviated by their first letters, with the units greater than 1 capitalized (KB, MB, etc.). One exception (for historical reasons) is kbps for kilobits/sec. Thus, a 1-Mbps communication line transmits 10^6 bits/sec and a 100-psec (or 100-ps) clock ticks every 10^{-10} seconds. Since milli and micro both begin with the letter "m," a choice had to be made. Normally, "m" is used for milli and "μ" (the Greek letter mu) is used for micro.

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

Figure 1-39. The principal metric prefixes.

It is also worth pointing out that for measuring memory, disk, file, and database sizes, in common industry practice, the units have slightly different meanings. There, kilo means 2^{10} (1024) rather than 10^3 (1000) because memories are always a power of two. Thus, a 1-KB memory contains 1024 bytes, not 1000 bytes. Note also the capital "B" in that usage to mean "bytes" (units of eight

bits), instead of a lowercase "b" that means "bits." Similarly, a 1-MB memory contains 2^{20} (1,048,576) bytes, a 1-GB memory contains 2^{30} (1,073,741,824) bytes, and a 1-TB database contains 2^{40} (1,099,511,627,776) bytes. However, a 1-kbps communication line transmits 1000 bits per second and a 10-Mbps LAN runs at 10,000,000 bits/sec because these speeds are not powers of two. Unfortunately, many people tend to mix up these two systems, especially for disk sizes. To avoid ambiguity, in this book, we will use the symbols KB, MB, GB, and TB for 2^{10}, 2^{20}, 2^{30}, and 2^{40} bytes, respectively, and the symbols kbps, Mbps, Gbps, and Tbps for 10^3, 10^6, 10^9, and 10^{12} bits/sec, respectively.

1.8 OUTLINE OF THE REST OF THE BOOK

This book discusses both the principles and practice of computer networking. Most chapters start with a discussion of the relevant principles, followed by a number of examples that illustrate these principles. These examples are usually taken from the Internet and wireless networks such as the mobile phone network since these are both important and very different. Other examples will be given where relevant.

The book is structured according to the hybrid model of Fig. 1-23. Starting with Chap. 2, we begin working our way up the protocol hierarchy beginning at the bottom. We provide some background in the field of data communication that covers both wired and wireless transmission systems. This material is concerned with how to deliver information over physical channels, although we cover only the architectural rather than the hardware aspects. Several examples of the physical layer, such as the public switched telephone network, the mobile telephone network, and the cable television network are also discussed.

Chapters 3 and 4 discuss the data link layer in two parts. Chap. 3 looks at the problem of how to send packets across a link, including error detection and correction. We look at DSL (used for broadband Internet access over phone lines) as a real-world example of a data link protocol.

In Chap. 4, we examine the medium access sublayer. This is the part of the data link layer that deals with how to share a channel between multiple computers. The examples we look at include wireless, such as 802.11 and RFID, and wired LANs such as classic Ethernet. Link layer switches that connect LANs, such as switched Ethernet, are also discussed here.

Chapter 5 deals with the network layer, especially routing. Many routing algorithms, both static and dynamic, are covered. Even with good routing algorithms, though, if more traffic is offered than the network can handle, some packets will be delayed or discarded. We discuss this issue from how to prevent congestion to how to guarantee a certain quality of service. Connecting heterogeneous networks to form internetworks also leads to numerous problems that are discussed here. The network layer in the Internet is given extensive coverage.

Patent Owner's Ex. 2010
Page 099

Chapter 6 deals with the transport layer. Much of the emphasis is on connection-oriented protocols and reliability, since many applications need these. Both Internet transport protocols, UDP and TCP, are covered in detail, as are their performance issues.

Chapter 7 deals with the application layer, its protocols, and its applications. The first topic is DNS, which is the Internet's telephone book. Next comes email, including a discussion of its protocols. Then we move on to the Web, with detailed discussions of static and dynamic content, and what happens on the client and server sides. We follow this with a look at networked multimedia, including streaming audio and video. Finally, we discuss content-delivery networks, including peer-to-peer technology.

Chapter 8 is about network security. This topic has aspects that relate to all layers, so it is easiest to treat it after all the layers have been thoroughly explained. The chapter starts with an introduction to cryptography. Later, it shows how cryptography can be used to secure communication, email, and the Web. The chapter ends with a discussion of some areas in which security collides with privacy, freedom of speech, censorship, and other social issues.

Chapter 9 contains an annotated list of suggested readings arranged by chapter. It is intended to help those readers who would like to pursue their study of networking further. The chapter also has an alphabetical bibliography of all the references cited in this book.

The authors' Web site at Pearson:

http://www.pearsonhighered.com/tanenbaum

has a page with links to many tutorials, FAQs, companies, industry consortia, professional organizations, standards organizations, technologies, papers, and more.

1.9 SUMMARY

Computer networks have many uses, both for companies and for individuals, in the home and while on the move. Companies use networks of computers to share corporate information, typically using the client-server model with employee desktops acting as clients accessing powerful servers in the machine room. For individuals, networks offer access to a variety of information and entertainment resources, as well as a way to buy and sell products and services. Individuals often access the Internet via their phone or cable providers at home, though increasingly wireless access is used for laptops and phones. Technology advances are enabling new kinds of mobile applications and networks with computers embedded in appliances and other consumer devices. The same advances raise social issues such as privacy concerns.

Roughly speaking, networks can be divided into LANs, MANs, WANs, and internetworks. LANs typical cover a building and operate at high speeds. MANs

usually cover a city. An example is the cable television system, which is now used by many people to access the Internet. WANs may cover a country or a continent. Some of the technologies used to build these networks are point-to-point (e.g., a cable) while others are broadcast (e.g.,wireless). Networks can be interconnected with routers to form internetworks, of which the Internet is the largest and best known example. Wireless networks, for example 802.11 LANs and 3G mobile telephony, are also becoming extremely popular.

Network software is built around protocols, which are rules by which processes communicate. Most networks support protocol hierarchies, with each layer providing services to the layer above it and insulating them from the details of the protocols used in the lower layers. Protocol stacks are typically based either on the OSI model or on the TCP/IP model. Both have link, network, transport, and application layers, but they differ on the other layers. Design issues include reliability, resource allocation, growth, security, and more. Much of this book deals with protocols and their design.

Networks provide various services to their users. These services can range from connectionless best-efforts packet delivery to connection-oriented guaranteed delivery. In some networks, connectionless service is provided in one layer and connection-oriented service is provided in the layer above it.

Well-known networks include the Internet, the 3G mobile telephone network, and 802.11 LANs. The Internet evolved from the ARPANET, to which other networks were added to form an internetwork. The present-day Internet is actually a collection of many thousands of networks that use the TCP/IP protocol stack. The 3G mobile telephone network provides wireless and mobile access to the Internet at speeds of multiple Mbps, and, of course, carries voice calls as well. Wireless LANs based on the IEEE 802.11 standard are deployed in many homes and cafes and can provide connectivity at rates in excess of 100 Mbps. New kinds of networks are emerging too, such as embedded sensor networks and networks based on RFID technology.

Enabling multiple computers to talk to each other requires a large amount of standardization, both in the hardware and software. Organizations such as ITU-T, ISO, IEEE, and IAB manage different parts of the standardization process.

PROBLEMS

1. Imagine that you have trained your St. Bernard, Bernie, to carry a box of three 8-mm tapes instead of a flask of brandy. (When your disk fills up, you consider that an emergency.) These tapes each contain 7 gigabytes. The dog can travel to your side, wherever you may be, at 18 km/hour. For what range of distances does Bernie have a higher data rate than a transmission line whose data rate (excluding overhead) is 150 Mbps? How does your answer change if (i) Bernie's speed is doubled; (ii) each tape capacity is doubled; (iii) the data rate of the transmission line is doubled.

2. An alternative to a LAN is simply a big timesharing system with terminals for all users. Give two advantages of a client-server system using a LAN.

3. The performance of a client-server system is strongly influenced by two major network characteristics: the bandwidth of the network (that is, how many bits/sec it can transport) and the latency (that is, how many seconds it takes for the first bit to get from the client to the server). Give an example of a network that exhibits high bandwidth but also high latency. Then give an example of one that has both low bandwidth and low latency.

4. Besides bandwidth and latency, what other parameter is needed to give a good characterization of the quality of service offered by a network used for (i) digitized voice traffic? (ii) video traffic? (iii) financial transaction traffic?

5. A factor in the delay of a store-and-forward packet-switching system is how long it takes to store and forward a packet through a switch. If switching time is 10 μsec, is this likely to be a major factor in the response of a client-server system where the client is in New York and the server is in California? Assume the propagation speed in copper and fiber to be 2/3 the speed of light in vacuum.

6. A client-server system uses a satellite network, with the satellite at a height of 40,000 km. What is the best-case delay in response to a request?

7. In the future, when everyone has a home terminal connected to a computer network, instant public referendums on important pending legislation will become possible. Ultimately, existing legislatures could be eliminated, to let the will of the people be expressed directly. The positive aspects of such a direct democracy are fairly obvious; discuss some of the negative aspects.

8. Five routers are to be connected in a point-to-point subnet. Between each pair of routers, the designers may put a high-speed line, a medium-speed line, a low-speed line, or no line. If it takes 100 ms of computer time to generate and inspect each topology, how long will it take to inspect all of them?

9. A disadvantage of a broadcast subnet is the capacity wasted when multiple hosts attempt to access the channel at the same time. As a simplistic example, suppose that time is divided into discrete slots, with each of the n hosts attempting to use the channel with probability p during each slot. What fraction of the slots will be wasted due to collisions?

10. What are two reasons for using layered protocols? What is one possible disadvantage of using layered protocols?

11. The president of the Specialty Paint Corp. gets the idea to work with a local beer brewer to produce an invisible beer can (as an anti-litter measure). The president tells her legal department to look into it, and they in turn ask engineering for help. As a result, the chief engineer calls his counterpart at the brewery to discuss the technical aspects of the project. The engineers then report back to their respective legal departments, which then confer by telephone to arrange the legal aspects. Finally, the two corporate presidents discuss the financial side of the deal. What principle of a multilayer protocol in the sense of the OSI model does this communication mechanism violate?

12. Two networks each provide reliable connection-oriented service. One of them offers a reliable byte stream and the other offers a reliable message stream. Are these identical? If so, why is the distinction made? If not, give an example of how they differ.

13. What does "negotiation" mean when discussing network protocols? Give an example.

14. In Fig. 1-19, a service is shown. Are any other services implicit in this figure? If so, where? If not, why not?

15. In some networks, the data link layer handles transmission errors by requesting that damaged frames be retransmitted. If the probability of a frame's being damaged is p, what is the mean number of transmissions required to send a frame? Assume that acknowledgements are never lost.

16. A system has an n-layer protocol hierarchy. Applications generate messages of length M bytes. At each of the layers, an h-byte header is added. What fraction of the network bandwidth is filled with headers?

17. What is the main difference between TCP and UDP?

18. The subnet of Fig. 1-25(b) was designed to withstand a nuclear war. How many bombs would it take to partition the nodes into two disconnected sets? Assume that any bomb wipes out a node and all of the links connected to it.

19. The Internet is roughly doubling in size every 18 months. Although no one really knows for sure, one estimate put the number of hosts on it at 600 million in 2009. Use these data to compute the expected number of Internet hosts in the year 2018. Do you believe this? Explain why or why not.

20. When a file is transferred between two computers, two acknowledgement strategies are possible. In the first one, the file is chopped up into packets, which are individually acknowledged by the receiver, but the file transfer as a whole is not acknowledged. In the second one, the packets are not acknowledged individually, but the entire file is acknowledged when it arrives. Discuss these two approaches.

21. Mobile phone network operators need to know where their subscribers' mobile phones (hence their users) are located. Explain why this is bad for users. Now give reasons why this is good for users.

22. How long was a bit in the original 802.3 standard in meters? Use a transmission speed of 10 Mbps and assume the propagation speed in coax is 2/3 the speed of light in vacuum.

23. An image is 1600 × 1200 pixels with 3 bytes/pixel. Assume the image is uncompressed. How long does it take to transmit it over a 56-kbps modem channel? Over a 1-Mbps cable modem? Over a 10-Mbps Ethernet? Over 100-Mbps Ethernet? Over gigabit Ethernet?

24. Ethernet and wireless networks have some similarities and some differences. One property of Ethernet is that only one frame at a time can be transmitted on an Ethernet. Does 802.11 share this property with Ethernet? Discuss your answer.

25. List two advantages and two disadvantages of having international standards for network protocols.

26. When a system has a permanent part and a removable part (such as a CD-ROM drive and the CD-ROM), it is important that the system be standardized, so that different companies can make both the permanent and removable parts and everything still works together. Give three examples outside the computer industry where such international standards exist. Now give three areas outside the computer industry where they do not exist.

27. Suppose the algorithms used to implement the operations at layer k is changed. How does this impact operations at layers $k - 1$ and $k + 1$?

28. Suppose there is a change in the service (set of operations) provided by layer k. How does this impact services at layers k-1 and k+1?

29. Provide a list of reasons for why the response time of a client may be larger than the best-case delay.

30. What are the disadvantages of using small, fixed-length cells in ATM?

31. Make a list of activities that you do every day in which computer networks are used. How would your life be altered if these networks were suddenly switched off?

32. Find out what networks are used at your school or place of work. Describe the network types, topologies, and switching methods used there.

33. The *ping* program allows you to send a test packet to a given location and see how long it takes to get there and back. Try using *ping* to see how long it takes to get from your location to several known locations. From these data, plot the one-way transit time over the Internet as a function of distance. It is best to use universities since the location of their servers is known very accurately. For example, *berkeley.edu* is in Berkeley, California; *mit.edu* is in Cambridge, Massachusetts; *vu.nl* is in Amsterdam; The Netherlands; *www.usyd.edu.au* is in Sydney, Australia; and *www.uct.ac.za* is in Cape Town, South Africa.

34. Go to IETF's Web site, *www.ietf.org*, to see what they are doing. Pick a project you like and write a half-page report on the problem and the proposed solution.

35. The Internet is made up of a large number of networks. Their arrangement determines the topology of the Internet. A considerable amount of information about the Internet topology is available on line. Use a search engine to find out more about the Internet topology and write a short report summarizing your findings.

36. Search the Internet to find out some of the important peering points used for routing packets in the Internet at present.

37. Write a program that implements message flow from the top layer to the bottom layer of the 7-layer protocol model. Your program should include a separate protocol function for each layer. Protocol headers are sequence up to 64 characters. Each protocol function has two parameters: a message passed from the higher layer protocol (a char buffer) and the size of the message. This function attaches its header in front of the message, prints the new message on the standard output, and then invokes the protocol function of the lower-layer protocol. Program input is an application message (a sequence of 80 characters or less).

4

THE MEDIUM ACCESS CONTROL SUBLAYER

Network links can be divided into two categories: those using point-to-point connections and those using broadcast channels. We studied point-to-point links in Chap. 2; this chapter deals with broadcast links and their protocols.

In any broadcast network, the key issue is how to determine who gets to use the channel when there is competition for it. To make this point, consider a conference call in which six people, on six different telephones, are all connected so that each one can hear and talk to all the others. It is very likely that when one of them stops speaking, two or more will start talking at once, leading to chaos. In a face-to-face meeting, chaos is avoided by external means. For example, at a meeting, people raise their hands to request permission to speak. When only a single channel is available, it is much harder to determine who should go next. Many protocols for solving the problem are known. They form the contents of this chapter. In the literature, broadcast channels are sometimes referred to as **multiaccess channels** or **random access channels**.

The protocols used to determine who goes next on a multiaccess channel belong to a sublayer of the data link layer called the **MAC** (**Medium Access Control**) sublayer. The MAC sublayer is especially important in LANs, particularly wireless ones because wireless is naturally a broadcast channel. WANs, in contrast, use point-to-point links, except for satellite networks. Because multiaccess channels and LANs are so closely related, in this chapter we will discuss LANs in

Patent Owner's Ex. 2010
Page 105

general, including a few issues that are not strictly part of the MAC sublayer, but the main subject here will be control of the channel.

Technically, the MAC sublayer is the bottom part of the data link layer, so logically we should have studied it before examining all the point-to-point protocols in Chap. 3. Nevertheless, for most people, it is easier to understand protocols involving multiple parties after two-party protocols are well understood. For that reason we have deviated slightly from a strict bottom-up order of presentation.

4.1 THE CHANNEL ALLOCATION PROBLEM

The central theme of this chapter is how to allocate a single broadcast channel among competing users. The channel might be a portion of the wireless spectrum in a geographic region, or a single wire or optical fiber to which multiple nodes are connected. It does not matter. In both cases, the channel connects each user to all other users and any user who makes full use of the channel interferes with other users who also wish to use the channel.

We will first look at the shortcomings of static allocation schemes for bursty traffic. Then, we will lay out the key assumptions used to model the dynamic schemes that we examine in the following sections.

4.1.1 Static Channel Allocation

The traditional way of allocating a single channel, such as a telephone trunk, among multiple competing users is to chop up its capacity by using one of the multiplexing schemes we described in Sec. 2.5, such as FDM (Frequency Division Multiplexing). If there are N users, the bandwidth is divided into N equal-sized portions, with each user being assigned one portion. Since each user has a private frequency band, there is now no interference among users. When there is only a small and constant number of users, each of which has a steady stream or a heavy load of traffic, this division is a simple and efficient allocation mechanism. A wireless example is FM radio stations. Each station gets a portion of the FM band and uses it most of the time to broadcast its signal.

However, when the number of senders is large and varying or the traffic is bursty, FDM presents some problems. If the spectrum is cut up into N regions and fewer than N users are currently interested in communicating, a large piece of valuable spectrum will be wasted. And if more than N users want to communicate, some of them will be denied permission for lack of bandwidth, even if some of the users who have been assigned a frequency band hardly ever transmit or receive anything.

Even assuming that the number of users could somehow be held constant at N, dividing the single available channel into some number of static subchannels is

inherently inefficient. The basic problem is that when some users are quiescent, their bandwidth is simply lost. They are not using it, and no one else is allowed to use it either. A static allocation is a poor fit to most computer systems, in which data traffic is extremely bursty, often with peak traffic to mean traffic ratios of 1000:1. Consequently, most of the channels will be idle most of the time.

The poor performance of static FDM can easily be seen with a simple queueing theory calculation. Let us start by finding the mean time delay, T, to send a frame onto a channel of capacity C bps. We assume that the frames arrive randomly with an average arrival rate of λ frames/sec, and that the frames vary in length with an average length of $1/\mu$ bits. With these parameters, the service rate of the channel is μC frames/sec. A standard queueing theory result is

$$T = \frac{1}{\mu C - \lambda}$$

(For the curious, this result is for an "M/M/1" queue. It requires that the randomness of the times between frame arrivals and the frame lengths follow an exponential distribution, or equivalently be the result of a Poisson process.)

In our example, if C is 100 Mbps, the mean frame length, $1/\mu$, is 10,000 bits, and the frame arrival rate, λ, is 5000 frames/sec, then $T = 200$ μsec. Note that if we ignored the queueing delay and just asked how long it takes to send a 10,000-bit frame on a 100-Mbps network, we would get the (incorrect) answer of 100 μsec. That result only holds when there is no contention for the channel.

Now let us divide the single channel into N independent subchannels, each with capacity C/N bps. The mean input rate on each of the subchannels will now be λ/N. Recomputing T, we get

$$T_N = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT \qquad (4\text{-}1)$$

The mean delay for the divided channel is N times worse than if all the frames were somehow magically arranged orderly in a big central queue. This same result says that a bank lobby full of ATM machines is better off having a single queue feeding all the machines than a separate queue in front of each machine.

Precisely the same arguments that apply to FDM also apply to other ways of statically dividing the channel. If we were to use time division multiplexing (TDM) and allocate each user every Nth time slot, if a user does not use the allocated slot, it would just lie fallow. The same would hold if we split up the networks physically. Using our previous example again, if we were to replace the 100-Mbps network with 10 networks of 10 Mbps each and statically allocate each user to one of them, the mean delay would jump from 200 μsec to 2 msec.

Since none of the traditional static channel allocation methods work well at all with bursty traffic, we will now explore dynamic methods.

4.1.2 Assumptions for Dynamic Channel Allocation

Before we get to the first of the many channel allocation methods in this chapter, it is worthwhile to carefully formulate the allocation problem. Underlying all the work done in this area are the following five key assumptions:

1. **Independent Traffic**. The model consists of N independent **stations** (e.g., computers, telephones), each with a program or user that generates frames for transmission. The expected number of frames generated in an interval of length Δt is $\lambda \Delta t$, where λ is a constant (the arrival rate of new frames). Once a frame has been generated, the station is blocked and does nothing until the frame has been successfully transmitted.
2. **Single Channel**. A single channel is available for all communication. All stations can transmit on it and all can receive from it. The stations are assumed to be equally capable, though protocols may assign them different roles (e.g., priorities).
3. **Observable Collisions**. If two frames are transmitted simultaneously, they overlap in time and the resulting signal is garbled. This event is called a **collision**. All stations can detect that a collision has occurred. A collided frame must be transmitted again later. No errors other than those generated by collisions occur.
4. **Continuous or Slotted Time**. Time may be assumed continuous, in which case frame transmission can begin at any instant. Alternatively, time may be slotted or divided into discrete intervals (called slots). Frame transmissions must then begin at the start of a slot. A slot may contain 0, 1, or more frames, corresponding to an idle slot, a successful transmission, or a collision, respectively.
5. **Carrier Sense or No Carrier Sense**. With the carrier sense assumption, stations can tell if the channel is in use before trying to use it. No station will attempt to use the channel while it is sensed as busy. If there is no carrier sense, stations cannot sense the channel before trying to use it. They just go ahead and transmit. Only later can they determine whether the transmission was successful.

Some discussion of these assumptions is in order. The first one says that frame arrivals are independent, both across stations and at a particular station, and that frames are generated unpredictably but at a constant rate. Actually, this assumption is not a particularly good model of network traffic, as it is well known that packets come in bursts over a range of time scales (Paxson and Floyd, 1995; and Leland et al., 1994). Nonetheless, **Poisson models**, as they are frequently called, are useful because they are mathematically tractable. They help us analyze

protocols to understand roughly how performance changes over an operating range and how it compares with other designs.

The single-channel assumption is the heart of the model. No external ways to communicate exist. Stations cannot raise their hands to request that the teacher call on them, so we will have to come up with better solutions.

The remaining three assumptions depend on the engineering of the system, and we will say which assumptions hold when we examine a particular protocol.

The collision assumption is basic. Stations need some way to detect collisions if they are to retransmit frames rather than let them be lost. For wired channels, node hardware can be designed to detect collisions when they occur. The stations can then terminate their transmissions prematurely to avoid wasting capacity. This detection is much harder for wireless channels, so collisions are usually inferred after the fact by the lack of an expected acknowledgement frame. It is also possible for some frames involved in a collision to be successfully received, depending on the details of the signals and the receiving hardware. However, this situation is not the common case, so we will assume that all frames involved in a collision are lost. We will also see protocols that are designed to prevent collisions from occurring in the first place.

The reason for the two alternative assumptions about time is that slotted time can be used to improve performance. However, it requires the stations to follow a master clock or synchronize their actions with each other to divide time into discrete intervals. Hence, it is not always available. We will discuss and analyze systems with both kinds of time. For a given system, only one of them holds.

Similarly, a network may have carrier sensing or not have it. Wired networks will generally have carrier sense. Wireless networks cannot always use it effectively because not every station may be within radio range of every other station. Similarly, carrier sense will not be available in other settings in which a station cannot communicate directly with other stations, for example a cable modem in which stations must communicate via the cable headend. Note that the word ''carrier'' in this sense refers to a signal on the channel and has nothing to do with the common carriers (e.g., telephone companies) that date back to the days of the Pony Express.

To avoid any misunderstanding, it is worth noting that no multiaccess protocol guarantees reliable delivery. Even in the absence of collisions, the receiver may have copied some of the frame incorrectly for various reasons. Other parts of the link layer or higher layers provide reliability.

4.2 MULTIPLE ACCESS PROTOCOLS

Many algorithms for allocating a multiple access channel are known. In the following sections, we will study a small sample of the more interesting ones and give some examples of how they are commonly used in practice.

4.2.1 ALOHA

The story of our first MAC starts out in pristine Hawaii in the early 1970s. In this case, ''pristine'' can be interpreted as ''not having a working telephone system.'' This did not make life more pleasant for researcher Norman Abramson and his colleagues at the University of Hawaii who were trying to connect users on remote islands to the main computer in Honolulu. Stringing their own cables under the Pacific Ocean was not in the cards, so they looked for a different solution.

The one they found used short-range radios, with each user terminal sharing the same upstream frequency to send frames to the central computer. It included a simple and elegant method to solve the channel allocation problem. Their work has been extended by many researchers since then (Schwartz and Abramson, 2009). Although Abramson's work, called the ALOHA system, used ground-based radio broadcasting, the basic idea is applicable to any system in which uncoordinated users are competing for the use of a single shared channel.

We will discuss two versions of ALOHA here: pure and slotted. They differ with respect to whether time is continuous, as in the pure version, or divided into discrete slots into which all frames must fit.

Pure ALOHA

The basic idea of an ALOHA system is simple: let users transmit whenever they have data to be sent. There will be collisions, of course, and the colliding frames will be damaged. Senders need some way to find out if this is the case. In the ALOHA system, after each station has sent its frame to the central computer, this computer rebroadcasts the frame to all of the stations. A sending station can thus listen for the broadcast from the hub to see if its frame has gotten through. In other systems, such as wired LANs, the sender might be able to listen for collisions while transmitting.

If the frame was destroyed, the sender just waits a random amount of time and sends it again. The waiting time must be random or the same frames will collide over and over, in lockstep. Systems in which multiple users share a common channel in a way that can lead to conflicts are known as **contention** systems.

A sketch of frame generation in an ALOHA system is given in Fig. 4-1. We have made the frames all the same length because the throughput of ALOHA systems is maximized by having a uniform frame size rather than by allowing variable-length frames.

Whenever two frames try to occupy the channel at the same time, there will be a collision (as seen in Fig. 4-1) and both will be garbled. If the first bit of a new frame overlaps with just the last bit of a frame that has almost finished, both frames will be totally destroyed (i.e., have incorrect checksums) and both will have to be retransmitted later. The checksum does not (and should not) distinguish between a total loss and a near miss. Bad is bad.

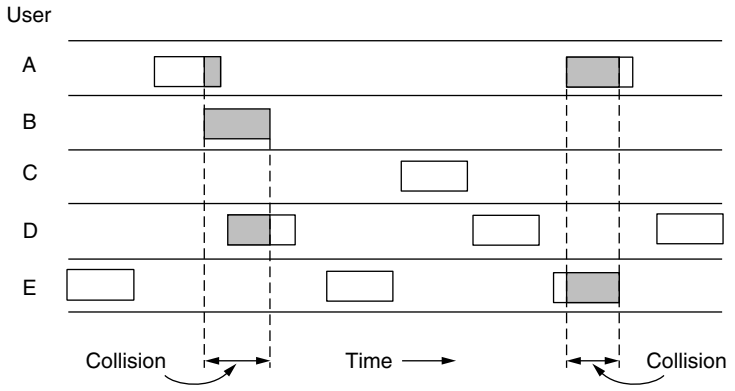


Figure 4-1. In pure ALOHA, frames are transmitted at completely arbitrary times.

An interesting question is: what is the efficiency of an ALOHA channel? In other words, what fraction of all transmitted frames escape collisions under these chaotic circumstances? Let us first consider an infinite collection of users typing at their terminals (stations). A user is always in one of two states: typing or waiting. Initially, all users are in the typing state. When a line is finished, the user stops typing, waiting for a response. The station then transmits a frame containing the line over the shared channel to the central computer and checks the channel to see if it was successful. If so, the user sees the reply and goes back to typing. If not, the user continues to wait while the station retransmits the frame over and over until it has been successfully sent.

Let the "frame time" denote the amount of time needed to transmit the standard, fixed-length frame (i.e., the frame length divided by the bit rate). At this point, we assume that the new frames generated by the stations are well modeled by a Poisson distribution with a mean of N frames per frame time. (The infinite-population assumption is needed to ensure that N does not decrease as users become blocked.) If $N > 1$, the user community is generating frames at a higher rate than the channel can handle, and nearly every frame will suffer a collision. For reasonable throughput, we would expect $0 < N < 1$.

In addition to the new frames, the stations also generate retransmissions of frames that previously suffered collisions. Let us further assume that the old and new frames combined are well modeled by a Poisson distribution, with mean of G frames per frame time. Clearly, $G \geq N$. At low load (i.e., $N \approx 0$), there will be few collisions, hence few retransmissions, so $G \approx N$. At high load, there will be many collisions, so $G > N$. Under all loads, the throughput, S, is just the offered load, G, times the probability, P_0, of a transmission succeeding—that is, $S = GP_0$, where P_0 is the probability that a frame does not suffer a collision.

A frame will not suffer a collision if no other frames are sent within one frame time of its start, as shown in Fig. 4-2. Under what conditions will the

shaded frame arrive undamaged? Let t be the time required to send one frame. If any other user has generated a frame between time t_0 and $t_0 + t$, the end of that frame will collide with the beginning of the shaded one. In fact, the shaded frame's fate was already sealed even before the first bit was sent, but since in pure ALOHA a station does not listen to the channel before transmitting, it has no way of knowing that another frame was already underway. Similarly, any other frame started between $t_0 + t$ and $t_0 + 2t$ will bump into the end of the shaded frame.

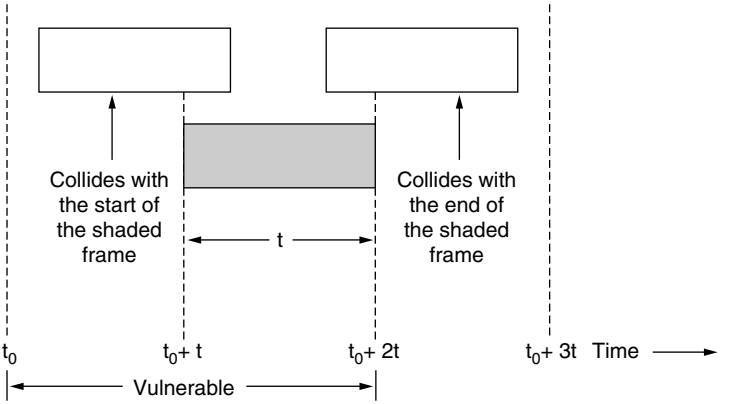


Figure 4-2. Vulnerable period for the shaded frame.

The probability that k frames are generated during a given frame time, in which G frames are expected, is given by the Poisson distribution

$$\Pr[k] = \frac{G^k \, e^{-G}}{k!} \tag{4-2}$$

so the probability of zero frames is just e^{-G}. In an interval two frame times long, the mean number of frames generated is $2G$. The probability of no frames being initiated during the entire vulnerable period is thus given by $P_0 = e^{-2G}$. Using $S = GP_0$, we get

$$S = Ge^{-2G}$$

The relation between the offered traffic and the throughput is shown in Fig. 4-3. The maximum throughput occurs at $G = 0.5$, with $S = 1/2e$, which is about 0.184. In other words, the best we can hope for is a channel utilization of 18%. This result is not very encouraging, but with everyone transmitting at will, we could hardly have expected a 100% success rate.

Slotted ALOHA

Soon after ALOHA came onto the scene, Roberts (1972) published a method for doubling the capacity of an ALOHA system. His proposal was to divide time into discrete intervals called **slots**, each interval corresponding to one frame. This

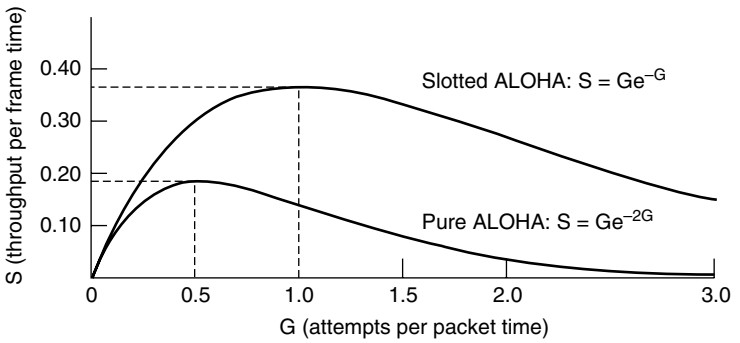


Figure 4-3. Throughput versus offered traffic for ALOHA systems.

approach requires the users to agree on slot boundaries. One way to achieve synchronization would be to have one special station emit a pip at the start of each interval, like a clock.

In Roberts' method, which has come to be known as **slotted ALOHA**—in contrast to Abramson's **pure ALOHA**—a station is not permitted to send whenever the user types a line. Instead, it is required to wait for the beginning of the next slot. Thus, the continuous time ALOHA is turned into a discrete time one. This halves the vulnerable period. To see this, look at Fig. 4-3 and imagine the collisions that are now possible. The probability of no other traffic during the same slot as our test frame is then e^{-G}, which leads to

$$S = Ge^{-G} \tag{4-3}$$

As you can see from Fig. 4-3, slotted ALOHA peaks at $G = 1$, with a throughput of $S = 1/e$ or about 0.368, twice that of pure ALOHA. If the system is operating at $G = 1$, the probability of an empty slot is 0.368 (from Eq. 4-2). The best we can hope for using slotted ALOHA is 37% of the slots empty, 37% successes, and 26% collisions. Operating at higher values of G reduces the number of empties but increases the number of collisions exponentially. To see how this rapid growth of collisions with G comes about, consider the transmission of a test frame. The probability that it will avoid a collision is e^{-G}, which is the probability that all the other stations are silent in that slot. The probability of a collision is then just $1 - e^{-G}$. The probability of a transmission requiring exactly k attempts (i.e., $k - 1$ collisions followed by one success) is

$$P_k = e^{-G}(1 - e^{-G})^{k-1}$$

The expected number of transmissions, E, per line typed at a terminal is then

$$E = \sum_{k=1}^{\infty} kP_k = \sum_{k=1}^{\infty} ke^{-G}(1 - e^{-G})^{k-1} = e^{G}$$

Patent Owner's Ex. 2010
Page 113

As a result of the exponential dependence of E upon G, small increases in the channel load can drastically reduce its performance.

Slotted ALOHA is notable for a reason that may not be initially obvious. It was devised in the 1970s, used in a few early experimental systems, then almost forgotten. When Internet access over the cable was invented, all of a sudden there was a problem of how to allocate a shared channel among multiple competing users. Slotted ALOHA was pulled out of the garbage can to save the day. Later, having multiple RFID tags talk to the same RFID reader presented another variation on the same problem. Slotted ALOHA, with a dash of other ideas mixed in, again came to the rescue. It has often happened that protocols that are perfectly valid fall into disuse for political reasons (e.g., some big company wants everyone to do things its way) or due to ever-changing technology trends. Then, years later some clever person realizes that a long-discarded protocol solves his current problem. For this reason, in this chapter we will study a number of elegant protocols that are not currently in widespread use but might easily be used in future applications, provided that enough network designers are aware of them. Of course, we will also study many protocols that are in current use as well.

4.2.2 Carrier Sense Multiple Access Protocols

With slotted ALOHA, the best channel utilization that can be achieved is $1/e$. This low result is hardly surprising, since with stations transmitting at will, without knowing what the other stations are doing there are bound to be many collisions. In LANs, however, it is often possible for stations to detect what other stations are doing, and thus adapt their behavior accordingly. These networks can achieve a much better utilization than $1/e$. In this section, we will discuss some protocols for improving performance.

Protocols in which stations listen for a carrier (i.e., a transmission) and act accordingly are called **carrier sense protocols**. A number of them have been proposed, and they were long ago analyzed in detail. For example, see Kleinrock and Tobagi (1975). Below we will look at several versions of carrier sense protocols.

Persistent and Nonpersistent CSMA

The first carrier sense protocol that we will study here is called **1-persistent CSMA** (**Carrier Sense Multiple Access**). That is a bit of a mouthful for the simplest CSMA scheme. When a station has data to send, it first listens to the channel to see if anyone else is transmitting at that moment. If the channel is idle, the stations sends its data. Otherwise, if the channel is busy, the station just waits until it becomes idle. Then the station transmits a frame. If a collision occurs, the

station waits a random amount of time and starts all over again. The protocol is called 1-persistent because the station transmits with a probability of 1 when it finds the channel idle.

You might expect that this scheme avoids collisions except for the rare case of simultaneous sends, but it in fact it does not. If two stations become ready in the middle of a third station's transmission, both will wait politely until the transmission ends, and then both will begin transmitting exactly simultaneously, resulting in a collision. If they were not so impatient, there would be fewer collisions.

More subtly, the propagation delay has an important effect on collisions. There is a chance that just after a station begins sending, another station will become ready to send and sense the channel. If the first station's signal has not yet reached the second one, the latter will sense an idle channel and will also begin sending, resulting in a collision. This chance depends on the number of frames that fit on the channel, or the **bandwidth-delay product** of the channel. If only a tiny fraction of a frame fits on the channel, which is the case in most LANs since the propagation delay is small, the chance of a collision happening is small. The larger the bandwidth-delay product, the more important this effect becomes, and the worse the performance of the protocol.

Even so, this protocol has better performance than pure ALOHA because both stations have the decency to desist from interfering with the third station's frame. Exactly the same holds for slotted ALOHA.

A second carrier sense protocol is **nonpersistent CSMA**. In this protocol, a conscious attempt is made to be less greedy than in the previous one. As before, a station senses the channel when it wants to send a frame, and if no one else is sending, the station begins doing so itself. However, if the channel is already in use, the station does not continually sense it for the purpose of seizing it immediately upon detecting the end of the previous transmission. Instead, it waits a random period of time and then repeats the algorithm. Consequently, this algorithm leads to better channel utilization but longer delays than 1-persistent CSMA.

The last protocol is **p-persistent CSMA**. It applies to slotted channels and works as follows. When a station becomes ready to send, it senses the channel. If it is idle, it transmits with a probability p. With a probability $q = 1 - p$, it defers until the next slot. If that slot is also idle, it either transmits or defers again, with probabilities p and q. This process is repeated until either the frame has been transmitted or another station has begun transmitting. In the latter case, the unlucky station acts as if there had been a collision (i.e., it waits a random time and starts again). If the station initially senses that the channel is busy, it waits until the next slot and applies the above algorithm. IEEE 802.11 uses a refinement of p-persistent CSMA that we will discuss in Sec. 4.4.

Figure 4-4 shows the computed throughput versus offered traffic for all three protocols, as well as for pure and slotted ALOHA.

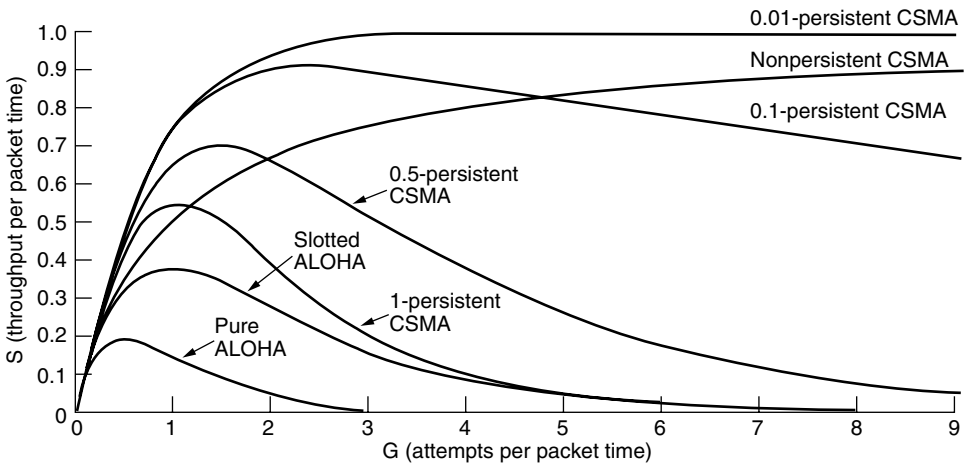


Figure 4-4. Comparison of the channel utilization versus load for various random access protocols.

CSMA with Collision Detection

Persistent and nonpersistent CSMA protocols are definitely an improvement over ALOHA because they ensure that no station begins to transmit while the channel is busy. However, if two stations sense the channel to be idle and begin transmitting simultaneously, their signals will still collide. Another improvement is for the stations to quickly detect the collision and abruptly stop transmitting, (rather than finishing them) since they are irretrievably garbled anyway. This strategy saves time and bandwidth.

This protocol, known as **CSMA/CD** (**CSMA with Collision Detection**), is the basis of the classic Ethernet LAN, so it is worth devoting some time to looking at it in detail. It is important to realize that collision detection is an analog process. The station's hardware must listen to the channel while it is transmitting. If the signal it reads back is different from the signal it is putting out, it knows that a collision is occurring. The implications are that a received signal must not be tiny compared to the transmitted signal (which is difficult for wireless, as received signals may be 1,000,000 times weaker than transmitted signals) and that the modulation must be chosen to allow collisions to be detected (e.g., a collision of two 0-volt signals may well be impossible to detect).

CSMA/CD, as well as many other LAN protocols, uses the conceptual model of Fig. 4-5. At the point marked t_0, a station has finished transmitting its frame. Any other station having a frame to send may now attempt to do so. If two or more stations decide to transmit simultaneously, there will be a collision. If a station detects a collision, it aborts its transmission, waits a random period of time, and then tries again (assuming that no other station has started transmitting in the

meantime). Therefore, our model for CSMA/CD will consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet (e.g., for lack of work).

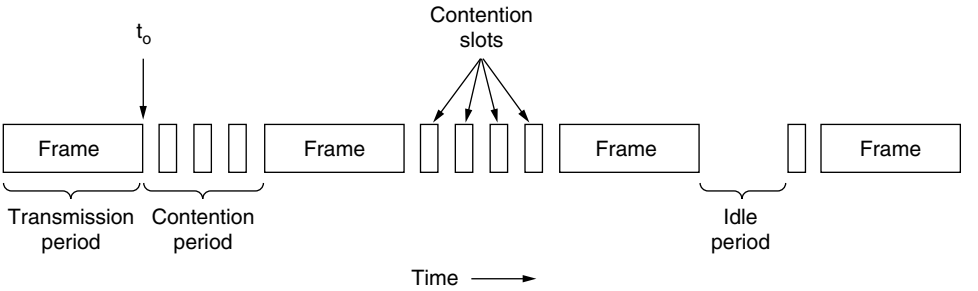


Figure 4-5. CSMA/CD can be in contention, transmission, or idle state.

Now let us look at the details of the contention algorithm. Suppose that two stations both begin transmitting at exactly time t_0. How long will it take them to realize that they have collided? The answer is vital to determining the length of the contention period and hence what the delay and throughput will be.

The minimum time to detect the collision is just the time it takes the signal to propagate from one station to the other. Based on this information, you might think that a station that has not heard a collision for a time equal to the full cable propagation time after starting its transmission can be sure it has seized the cable. By "seized," we mean that all other stations know it is transmitting and will not interfere. This conclusion is wrong.

Consider the following worst-case scenario. Let the time for a signal to propagate between the two farthest stations be τ. At t_0, one station begins transmitting. At $t_0 + \tau - \varepsilon$, an instant before the signal arrives at the most distant station, that station also begins transmitting. Of course, it detects the collision almost instantly and stops, but the little noise burst caused by the collision does not get back to the original station until time $2\tau - \varepsilon$. In other words, in the worst case a station cannot be sure that it has seized the channel until it has transmitted for 2τ without hearing a collision.

With this understanding, we can think of CSMA/CD contention as a slotted ALOHA system with a slot width of 2τ. On a 1-km long coaxial cable, $\tau \approx 5$ μsec. The difference for CSMA/CD compared to slotted ALOHA is that slots in which only one station transmits (i.e., in which the channel is seized) are followed by the rest of a frame. This difference will greatly improve performance if the frame time is much longer than the propagation time.

4.2.3 Collision-Free Protocols

Although collisions do not occur with CSMA/CD once a station has unambiguously captured the channel, they can still occur during the contention period. These collisions adversely affect the system performance, especially when the

bandwidth-delay product is large, such as when the cable is long (i.e., large τ) and the frames are short. Not only do collisions reduce bandwidth, but they make the time to send a frame variable, which is not a good fit for real-time traffic such as voice over IP. CSMA/CD is also not universally applicable.

In this section, we will examine some protocols that resolve the contention for the channel without any collisions at all, not even during the contention period. Most of these protocols are not currently used in major systems, but in a rapidly changing field, having some protocols with excellent properties available for future systems is often a good thing.

In the protocols to be described, we assume that there are exactly N stations, each programmed with a unique address from 0 to $N - 1$. It does not matter that some stations may be inactive part of the time. We also assume that propagation delay is negligible. The basic question remains: which station gets the channel after a successful transmission? We continue using the model of Fig. 4-5 with its discrete contention slots.

A Bit-Map Protocol

In our first collision-free protocol, the **basic bit-map method**, each contention period consists of exactly N slots. If station 0 has a frame to send, it transmits a 1 bit during the slot 0. No other station is allowed to transmit during this slot. Regardless of what station 0 does, station 1 gets the opportunity to transmit a 1 bit during slot 1, but only if it has a frame queued. In general, station j may announce that it has a frame to send by inserting a 1 bit into slot j. After all N slots have passed by, each station has complete knowledge of which stations wish to transmit. At that point, they begin transmitting frames in numerical order (see Fig. 4-6).

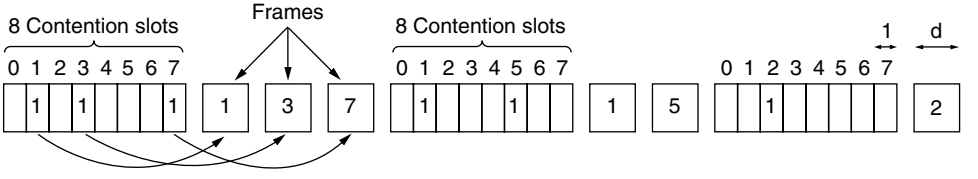


Figure 4-6. The basic bit-map protocol.

Since everyone agrees on who goes next, there will never be any collisions. After the last ready station has transmitted its frame, an event all stations can easily monitor, another N-bit contention period is begun. If a station becomes ready just after its bit slot has passed by, it is out of luck and must remain silent until every station has had a chance and the bit map has come around again.

Protocols like this in which the desire to transmit is broadcast before the actual transmission are called **reservation protocols** because they reserve channel ownership in advance and prevent collisions. Let us briefly analyze the performance of this protocol. For convenience, we will measure time in units of the contention bit slot, with data frames consisting of d time units.

Under conditions of low load, the bit map will simply be repeated over and over, for lack of data frames. Consider the situation from the point of view of a low-numbered station, such as 0 or 1. Typically, when it becomes ready to send, the ''current'' slot will be somewhere in the middle of the bit map. On average, the station will have to wait $N/2$ slots for the current scan to finish and another full N slots for the following scan to run to completion before it may begin transmitting.

The prospects for high-numbered stations are brighter. Generally, these will only have to wait half a scan ($N/2$ bit slots) before starting to transmit. High-numbered stations rarely have to wait for the next scan. Since low-numbered stations must wait on average $1.5N$ slots and high-numbered stations must wait on average $0.5N$ slots, the mean for all stations is N slots.

The channel efficiency at low load is easy to compute. The overhead per frame is N bits and the amount of data is d bits, for an efficiency of $d/(d + N)$.

At high load, when all the stations have something to send all the time, the N-bit contention period is prorated over N frames, yielding an overhead of only 1 bit per frame, or an efficiency of $d/(d + 1)$. The mean delay for a frame is equal to the sum of the time it queues inside its station, plus an additional $(N - 1)d + N$ once it gets to the head of its internal queue. This interval is how long it takes to wait for all other stations to have their turn sending a frame and another bitmap.

Token Passing

The essence of the bit-map protocol is that it lets every station transmit a frame in turn in a predefined order. Another way to accomplish the same thing is to pass a small message called a **token** from one station to the next in the same predefined order. The token represents permission to send. If a station has a frame queued for transmission when it receives the token, it can send that frame before it passes the token to the next station. If it has no queued frame, it simply passes the token.

In a **token ring** protocol, the topology of the network is used to define the order in which stations send. The stations are connected one to the next in a single ring. Passing the token to the next station then simply consists of receiving the token in from one direction and transmitting it out in the other direction, as seen in Fig. 4-7. Frames are also transmitted in the direction of the token. This way they will circulate around the ring and reach whichever station is the destination. However, to stop the frame circulating indefinitely (like the token), some station needs

to remove it from the ring. This station may be either the one that originally sent the frame, after it has gone through a complete cycle, or the station that was the intended recipient of the frame.

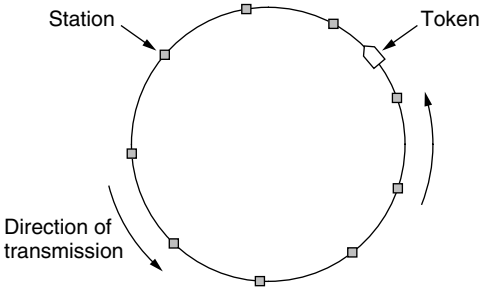


Figure 4-7. Token ring.

Note that we do not need a physical ring to implement token passing. The channel connecting the stations might instead be a single long bus. Each station then uses the bus to send the token to the next station in the predefined sequence. Possession of the token allows a station to use the bus to send one frame, as before. This protocol is called **token bus**.

The performance of token passing is similar to that of the bit-map protocol, though the contention slots and frames of one cycle are now intermingled. After sending a frame, each station must wait for all N stations (including itself) to send the token to their neighbors and the other $N - 1$ stations to send a frame, if they have one. A subtle difference is that, since all positions in the cycle are equivalent, there is no bias for low- or high-numbered stations. For token ring, each station is also sending the token only as far as its neighboring station before the protocol takes the next step. Each token does not need to propagate to all stations before the protocol advances to the next step.

Token rings have cropped up as MAC protocols with some consistency. An early token ring protocol (called "Token Ring" and standardized as IEEE 802.5) was popular in the 1980s as an alternative to classic Ethernet. In the 1990s, a much faster token ring called **FDDI** (**Fiber Distributed Data Interface**) was beaten out by switched Ethernet. In the 2000s, a token ring called **RPR** (**Resilient Packet Ring**) was defined as IEEE 802.17 to standardize the mix of metropolitan area rings in use by ISPs. We wonder what the 2010s will have to offer.

Binary Countdown

A problem with the basic bit-map protocol, and by extension token passing, is that the overhead is 1 bit per station, so it does not scale well to networks with thousands of stations. We can do better than that by using binary station addresses with a channel that combines transmissions. A station wanting to use the

channel now broadcasts its address as a binary bit string, starting with the high-order bit. All addresses are assumed to be the same length. The bits in each address position from different stations are BOOLEAN ORed together by the channel when they are sent at the same time. We will call this protocol **binary countdown**. It was used in Datakit (Fraser, 1987). It implicitly assumes that the transmission delays are negligible so that all stations see asserted bits essentially instantaneously.

To avoid conflicts, an arbitration rule must be applied: as soon as a station sees that a high-order bit position that is 0 in its address has been overwritten with a 1, it gives up. For example, if stations 0010, 0100, 1001, and 1010 are all trying to get the channel, in the first bit time the stations transmit 0, 0, 1, and 1, respectively. These are ORed together to form a 1. Stations 0010 and 0100 see the 1 and know that a higher-numbered station is competing for the channel, so they give up for the current round. Stations 1001 and 1010 continue.

The next bit is 0, and both stations continue. The next bit is 1, so station 1001 gives up. The winner is station 1010 because it has the highest address. After winning the bidding, it may now transmit a frame, after which another bidding cycle starts. The protocol is illustrated in Fig. 4-8. It has the property that higher-numbered stations have a higher priority than lower-numbered stations, which may be either good or bad, depending on the context.

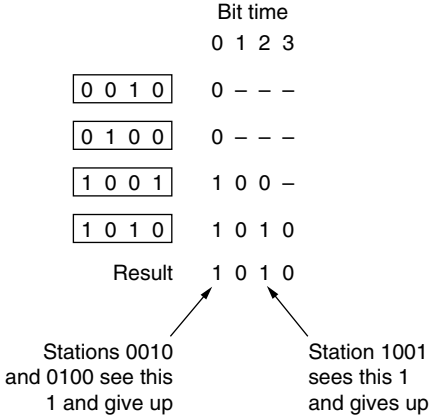


Figure 4-8. The binary countdown protocol. A dash indicates silence.

The channel efficiency of this method is $d/(d + \log_2 N)$. If, however, the frame format has been cleverly chosen so that the sender's address is the first field in the frame, even these $\log_2 N$ bits are not wasted, and the efficiency is 100%.

Binary countdown is an example of a simple, elegant, and efficient protocol that is waiting to be rediscovered. Hopefully, it will find a new home some day.

4.2.4 Limited-Contention Protocols

We have now considered two basic strategies for channel acquisition in a broadcast network: contention, as in CSMA, and collision-free protocols. Each strategy can be rated as to how well it does with respect to the two important performance measures, delay at low load and channel efficiency at high load. Under conditions of light load, contention (i.e., pure or slotted ALOHA) is preferable due to its low delay (since collisions are rare). As the load increases, contention becomes increasingly less attractive because the overhead associated with channel arbitration becomes greater. Just the reverse is true for the collision-free protocols. At low load, they have relatively high delay but as the load increases, the channel efficiency improves (since the overheads are fixed).

Obviously, it would be nice if we could combine the best properties of the contention and collision-free protocols, arriving at a new protocol that used contention at low load to provide low delay, but used a collision-free technique at high load to provide good channel efficiency. Such protocols, which we will call **limited-contention protocols**, do in fact exist, and will conclude our study of carrier sense networks.

Up to now, the only contention protocols we have studied have been symmetric. That is, each station attempts to acquire the channel with some probability, p, with all stations using the same p. Interestingly enough, the overall system performance can sometimes be improved by using a protocol that assigns different probabilities to different stations.

Before looking at the asymmetric protocols, let us quickly review the performance of the symmetric case. Suppose that k stations are contending for channel access. Each has a probability p of transmitting during each slot. The probability that some station successfully acquires the channel during a given slot is the probability that any one station transmits, with probability p, and all other $k-1$ stations defer, each with probability $1-p$. This value is $kp(1-p)^{k-1}$. To find the optimal value of p, we differentiate with respect to p, set the result to zero, and solve for p. Doing so, we find that the best value of p is $1/k$. Substituting $p = 1/k$, we get

$$\Pr[\text{success with optimal } p] = \left[\frac{k-1}{k}\right]^{k-1} \tag{4-4}$$

This probability is plotted in Fig. 4-9. For small numbers of stations, the chances of success are good, but as soon as the number of stations reaches even five, the probability has dropped close to its asymptotic value of $1/e$.

From Fig. 4-9, it is fairly obvious that the probability of some station acquiring the channel can be increased only by decreasing the amount of competition. The limited-contention protocols do precisely that. They first divide the stations into (not necessarily disjoint) groups. Only the members of group 0 are permitted

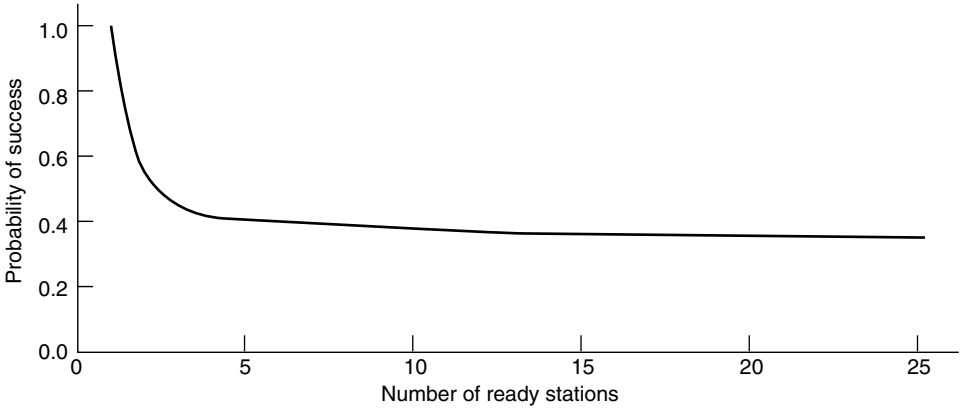


Figure 4-9. Acquisition probability for a symmetric contention channel.

to compete for slot 0. If one of them succeeds, it acquires the channel and transmits its frame. If the slot lies fallow or if there is a collision, the members of group 1 contend for slot 1, etc. By making an appropriate division of stations into groups, the amount of contention for each slot can be reduced, thus operating each slot near the left end of Fig. 4-9.

The trick is how to assign stations to slots. Before looking at the general case, let us consider some special cases. At one extreme, each group has but one member. Such an assignment guarantees that there will never be collisions because at most one station is contending for any given slot. We have seen such protocols before (e.g., binary countdown). The next special case is to assign two stations per group. The probability that both will try to transmit during a slot is p^2, which for a small p is negligible. As more and more stations are assigned to the same slot, the probability of a collision grows, but the length of the bit-map scan needed to give everyone a chance shrinks. The limiting case is a single group containing all stations (i.e., slotted ALOHA). What we need is a way to assign stations to slots dynamically, with many stations per slot when the load is low and few (or even just one) station per slot when the load is high.

The Adaptive Tree Walk Protocol

One particularly simple way of performing the necessary assignment is to use the algorithm devised by the U.S. Army for testing soldiers for syphilis during World War II (Dorfman, 1943). In short, the Army took a blood sample from N soldiers. A portion of each sample was poured into a single test tube. This mixed sample was then tested for antibodies. If none were found, all the soldiers in the group were declared healthy. If antibodies were present, two new mixed samples

were prepared, one from soldiers 1 through $N/2$ and one from the rest. The process was repeated recursively until the infected soldiers were determined.

For the computerized version of this algorithm (Capetanakis, 1979), it is convenient to think of the stations as the leaves of a binary tree, as illustrated in Fig. 4-10. In the first contention slot following a successful frame transmission, slot 0, all stations are permitted to try to acquire the channel. If one of them does so, fine. If there is a collision, then during slot 1 only those stations falling under node 2 in the tree may compete. If one of them acquires the channel, the slot following the frame is reserved for those stations under node 3. If, on the other hand, two or more stations under node 2 want to transmit, there will be a collision during slot 1, in which case it is node 4's turn during slot 2.

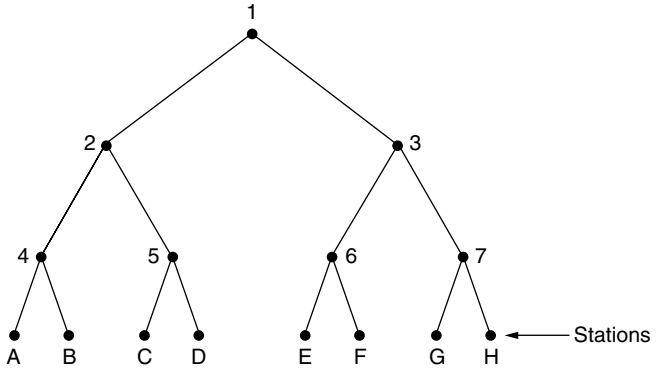


Figure 4-10. The tree for eight stations.

In essence, if a collision occurs during slot 0, the entire tree is searched, depth first, to locate all ready stations. Each bit slot is associated with some particular node in the tree. If a collision occurs, the search continues recursively with the node's left and right children. If a bit slot is idle or if only one station transmits in it, the searching of its node can stop because all ready stations have been located. (Were there more than one, there would have been a collision.)

When the load on the system is heavy, it is hardly worth the effort to dedicate slot 0 to node 1 because that makes sense only in the unlikely event that precisely one station has a frame to send. Similarly, one could argue that nodes 2 and 3 should be skipped as well for the same reason. Put in more general terms, at what level in the tree should the search begin? Clearly, the heavier the load, the farther down the tree the search should begin. We will assume that each station has a good estimate of the number of ready stations, q, for example, from monitoring recent traffic.

To proceed, let us number the levels of the tree from the top, with node 1 in Fig. 4-10 at level 0, nodes 2 and 3 at level 1, etc. Notice that each node at level i

has a fraction 2^{-i} of the stations below it. If the q ready stations are uniformly distributed, the expected number of them below a specific node at level i is just $2^{-i}q$. Intuitively, we would expect the optimal level to begin searching the tree to be the one at which the mean number of contending stations per slot is 1, that is, the level at which $2^{-i}q = 1$. Solving this equation, we find that $i = \log_2 q$.

Numerous improvements to the basic algorithm have been discovered and are discussed in some detail by Bertsekas and Gallager (1992). For example, consider the case of stations G and H being the only ones wanting to transmit. At node 1 a collision will occur, so 2 will be tried and discovered idle. It is pointless to probe node 3 since it is guaranteed to have a collision (we know that two or more stations under 1 are ready and none of them are under 2, so they must all be under 3). The probe of 3 can be skipped and 6 tried next. When this probe also turns up nothing, 7 can be skipped and node G tried next.

4.2.5 Wireless LAN Protocols

A system of laptop computers that communicate by radio can be regarded as a wireless LAN, as we discussed in Sec. 1.5.3. Such a LAN is an example of a broadcast channel. It also has somewhat different properties than a wired LAN, which leads to different MAC protocols. In this section, we will examine some of these protocols. In Sec. 4.4, we will look at 802.11 (WiFi) in detail.

A common configuration for a wireless LAN is an office building with access points (APs) strategically placed around the building. The APs are wired together using copper or fiber and provide connectivity to the stations that talk to them. If the transmission power of the APs and laptops is adjusted to have a range of tens of meters, nearby rooms become like a single cell and the entire building becomes like the cellular telephony systems we studied in Chap. 2, except that each cell only has one channel. This channel is shared by all the stations in the cell, including the AP. It typically provides megabit/sec bandwidths, up to 600 Mbps.

We have already remarked that wireless systems cannot normally detect a collision while it is occurring. The received signal at a station may be tiny, perhaps a million times fainter than the signal that is being transmitted. Finding it is like looking for a ripple on the ocean. Instead, acknowledgements are used to discover collisions and other errors after the fact.

There is an even more important difference between wireless LANs and wired LANs. A station on a wireless LAN may not be able to transmit frames to or receive frames from all other stations because of the limited radio range of the stations. In wired LANs, when one station sends a frame, all other stations receive it. The absence of this property in wireless LANs causes a variety of complications.

We will make the simplifying assumption that each radio transmitter has some fixed range, represented by a circular coverage region within which another station can sense and receive the station's transmission. It is important to realize that

in practice coverage regions are not nearly so regular because the propagation of radio signals depends on the environment. Walls and other obstacles that attenuate and reflect signals may cause the range to differ markedly in different directions. But a simple circular model will do for our purposes.

A naive approach to using a wireless LAN might be to try CSMA: just listen for other transmissions and only transmit if no one else is doing so. The trouble is, this protocol is not really a good way to think about wireless because what matters for reception is interference at the receiver, not at the sender. To see the nature of the problem, consider Fig. 4-11, where four wireless stations are illustrated. For our purposes, it does not matter which are APs and which are laptops. The radio range is such that *A* and *B* are within each other's range and can potentially interfere with one another. *C* can also potentially interfere with both *B* and *D*, but not with *A*.

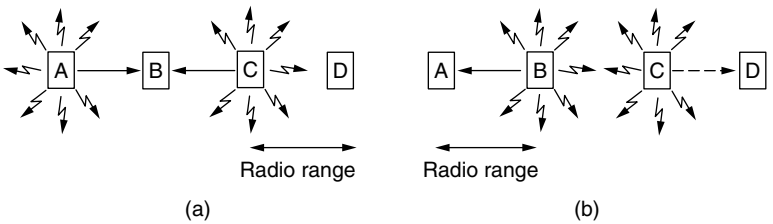


Figure 4-11. A wireless LAN. (a) *A* and *C* are hidden terminals when transmitting to *B*. (b) *B* and *C* are exposed terminals when transmitting to *A* and *D*.

First consider what happens when *A* and *C* transmit to *B*, as depicted in Fig. 4-11(a). If *A* sends and then *C* immediately senses the medium, it will not hear *A* because *A* is out of range. Thus *C* will falsely conclude that it can transmit to *B*. If *C* does start transmitting, it will interfere at *B*, wiping out the frame from *A*. (We assume here that no CDMA-type scheme is used to provide multiple channels, so collisions garble the signal and destroy both frames.) We want a MAC protocol that will prevent this kind of collision from happening because it wastes bandwidth. The problem of a station not being able to detect a potential competitor for the medium because the competitor is too far away is called the **hidden terminal problem**.

Now let us look at a different situation: *B* transmitting to *A* at the same time that *C* wants to transmit to *D,* as shown in Fig. 4-11(b). If *C* senses the medium, it will hear a transmission and falsely conclude that it may not send to *D* (shown as a dashed line). In fact, such a transmission would cause bad reception only in the zone between *B* and *C*, where neither of the intended receivers is located. We want a MAC protocol that prevents this kind of deferral from happening because it wastes bandwidth. The problem is called the **exposed terminal problem**.

The difficulty is that, before starting a transmission, a station really wants to know whether there is radio activity around the receiver. CSMA merely tells it

whether there is activity near the transmitter by sensing the carrier. With a wire, all signals propagate to all stations, so this distinction does not exist. However, only one transmission can then take place at once anywhere in the system. In a system based on short-range radio waves, multiple transmissions can occur simultaneously if they all have different destinations and these destinations are out of range of one another. We want this concurrency to happen as the cell gets larger and larger, in the same way that people at a party should not wait for everyone in the room to go silent before they talk; multiple conversations can take place at once in a large room as long as they are not directed to the same location.

An early and influential protocol that tackles these problems for wireless LANs is **MACA** (**Multiple Access with Collision Avoidance**) (Karn, 1990). The basic idea behind it is for the sender to stimulate the receiver into outputting a short frame, so stations nearby can detect this transmission and avoid transmitting for the duration of the upcoming (large) data frame. This technique is used instead of carrier sense.

MACA is illustrated in Fig. 4-12. Let us see how *A* sends a frame to *B*. *A* starts by sending an **RTS** (**Request To Send**) frame to *B*, as shown in Fig. 4-12(a). This short frame (30 bytes) contains the length of the data frame that will eventually follow. Then *B* replies with a **CTS** (**Clear To Send**) frame, as shown in Fig. 4-12(b). The CTS frame contains the data length (copied from the RTS frame). Upon receipt of the CTS frame, *A* begins transmission.

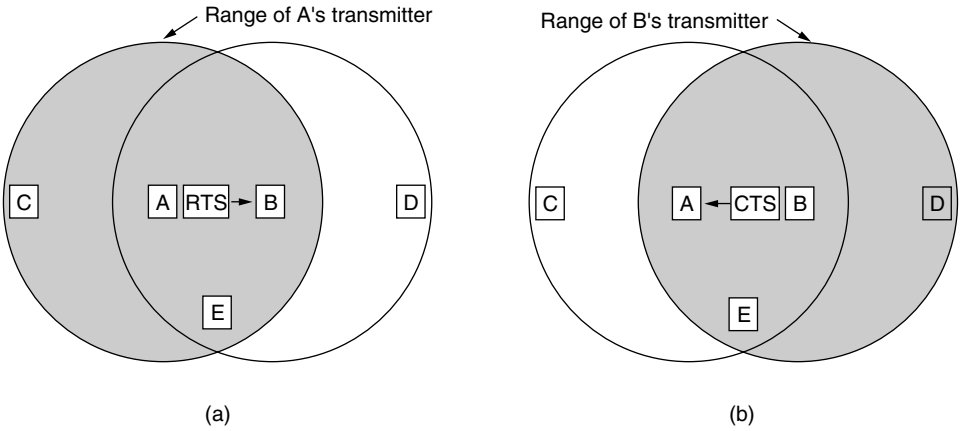


Figure 4-12. The MACA protocol. (a) *A* sending an RTS to *B*. (b) *B* responding with a CTS to *A*.

Now let us see how stations overhearing either of these frames react. Any station hearing the RTS is clearly close to *A* and must remain silent long enough for the CTS to be transmitted back to *A* without conflict. Any station hearing the CTS is clearly close to *B* and must remain silent during the upcoming data transmission, whose length it can tell by examining the CTS frame.

In Fig. 4-12, *C* is within range of *A* but not within range of *B*. Therefore, it hears the RTS from *A* but not the CTS from *B*. As long as it does not interfere with the CTS, it is free to transmit while the data frame is being sent. In contrast, *D* is within range of *B* but not *A*. It does not hear the RTS but does hear the CTS. Hearing the CTS tips it off that it is close to a station that is about to receive a frame, so it defers sending anything until that frame is expected to be finished. Station *E* hears both control messages and, like *D*, must be silent until the data frame is complete.

Despite these precautions, collisions can still occur. For example, *B* and *C* could both send RTS frames to *A* at the same time. These will collide and be lost. In the event of a collision, an unsuccessful transmitter (i.e., one that does not hear a CTS within the expected time interval) waits a random amount of time and tries again later.

4.3 ETHERNET

We have now finished our discussion of channel allocation protocols in the abstract, so it is time to see how these principles apply to real systems. Many of the designs for personal, local, and metropolitan area networks have been standardized under the name of IEEE 802. A few have survived but many have not, as we saw in Fig. 1-38. Some people who believe in reincarnation think that Charles Darwin came back as a member of the IEEE Standards Association to weed out the unfit. The most important of the survivors are 802.3 (Ethernet) and 802.11 (wireless LAN). Bluetooth (wireless PAN) is widely deployed but has now been standardized outside of 802.15. With 802.16 (wireless MAN), it is too early to tell. Please consult the 6th edition of this book to find out.

We will begin our study of real systems with Ethernet, probably the most ubiquitous kind of computer network in the world. Two kinds of Ethernet exist: **classic Ethernet**, which solves the multiple access problem using the techniques we have studied in this chapter; and **switched Ethernet**, in which devices called switches are used to connect different computers. It is important to note that, while they are both referred to as Ethernet, they are quite different. Classic Ethernet is the original form and ran at rates from 3 to 10 Mbps. Switched Ethernet is what Ethernet has become and runs at 100, 1000, and 10,000 Mbps, in forms called fast Ethernet, gigabit Ethernet, and 10 gigabit Ethernet. In practice, only switched Ethernet is used nowadays.

We will discuss these historical forms of Ethernet in chronological order showing how they developed. Since Ethernet and IEEE 802.3 are identical except for a minor difference (which we will discuss shortly), many people use the terms "Ethernet" and "IEEE 802.3" interchangeably. We will do so, too. For more information about Ethernet, see Spurgeon (2000).

4.3.1 Classic Ethernet Physical Layer

The story of Ethernet starts about the same time as that of ALOHA, when a student named Bob Metcalfe got his bachelor's degree at M.I.T. and then moved up the river to get his Ph.D. at Harvard. During his studies, he was exposed to Abramson's work. He became so interested in it that after graduating from Harvard, he decided to spend the summer in Hawaii working with Abramson before starting work at Xerox PARC (Palo Alto Research Center). When he got to PARC, he saw that the researchers there had designed and built what would later be called personal computers. But the machines were isolated. Using his knowledge of Abramson's work, he, together with his colleague David Boggs, designed and implemented the first local area network (Metcalfe and Boggs, 1976). It used a single long, thick coaxial cable and ran at 3 Mbps.

They called the system **Ethernet** after the *luminiferous ether*, through which electromagnetic radiation was once thought to propagate. (When the 19th-century British physicist James Clerk Maxwell discovered that electromagnetic radiation could be described by a wave equation, scientists assumed that space must be filled with some ethereal medium in which the radiation was propagating. Only after the famous Michelson-Morley experiment in 1887 did physicists discover that electromagnetic radiation could propagate in a vacuum.)

The Xerox Ethernet was so successful that DEC, Intel, and Xerox drew up a standard in 1978 for a 10-Mbps Ethernet, called the **DIX standard**. With a minor change, the DIX standard became the IEEE 802.3 standard in 1983. Unfortunately for Xerox, it already had a history of making seminal inventions (such as the personal computer) and then failing to commercialize on them, a story told in *Fumbling the Future* (Smith and Alexander, 1988). When Xerox showed little interest in doing anything with Ethernet other than helping standardize it, Metcalfe formed his own company, 3Com, to sell Ethernet adapters for PCs. It sold many millions of them.

Classic Ethernet snaked around the building as a single long cable to which all the computers were attached. This architecture is shown in Fig. 4-13. The first variety, popularly called **thick Ethernet,** resembled a yellow garden hose, with markings every 2.5 meters to show where to attach computers. (The 802.3 standard did not actually *require* the cable to be yellow, but it did *suggest* it.) It was succeeded by **thin Ethernet,** which bent more easily and made connections using industry-standard BNC connectors. Thin Ethernet was much cheaper and easier to install, but it could run for only 185 meters per segment (instead of 500 m with thick Ethernet), each of which could handle only 30 machines (instead of 100).

Each version of Ethernet has a maximum cable length per segment (i.e., unamplified length) over which the signal will propagate. To allow larger networks, multiple cables can be connected by **repeaters**. A repeater is a physical layer device that receives, amplifies (i.e., regenerates), and retransmits signals in both directions. As far as the software is concerned, a series of cable segments

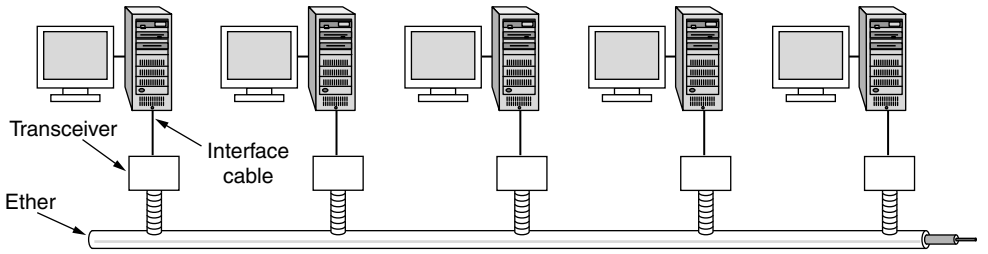


Figure 4-13. Architecture of classic Ethernet.

connected by repeaters is no different from a single cable (except for a small amount of delay introduced by the repeaters).

Over each of these cables, information was sent using the Manchester encoding we studied in Sec. 2.5. An Ethernet could contain multiple cable segments and multiple repeaters, but no two transceivers could be more than 2.5 km apart and no path between any two transceivers could traverse more than four repeaters. The reason for this restriction was so that the MAC protocol, which we will look at next, would work correctly.

4.3.2 Classic Ethernet MAC Sublayer Protocol

The format used to send frames is shown in Fig. 4-14. First comes a *Preamble* of 8 bytes, each containing the bit pattern 10101010 (with the exception of the last byte, in which the last 2 bits are set to 11). This last byte is called the *Start of Frame* delimiter for 802.3. The Manchester encoding of this pattern produces a 10-MHz square wave for 6.4 μsec to allow the receiver's clock to synchronize with the sender's. The last two 1 bits tell the receiver that the rest of the frame is about to start.

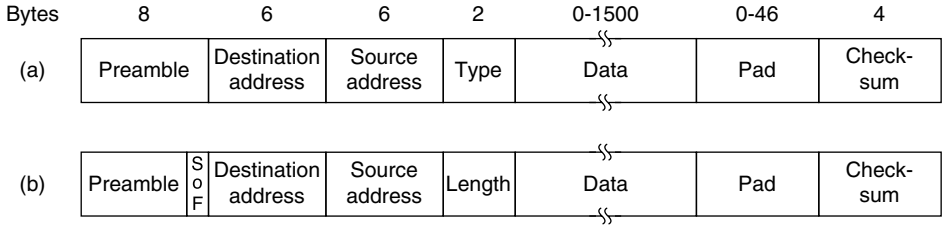


Figure 4-14. Frame formats. (a) Ethernet (DIX). (b) IEEE 802.3.

Next come two addresses, one for the destination and one for the source. They are each 6 bytes long. The first transmitted bit of the destination address is a 0 for

ordinary addresses and a 1 for group addresses. Group addresses allow multiple stations to listen to a single address. When a frame is sent to a group address, all the stations in the group receive it. Sending to a group of stations is called **multicasting**. The special address consisting of all 1 bits is reserved for **broadcasting**. A frame containing all 1s in the destination field is accepted by all stations on the network. Multicasting is more selective, but it involves group management to define which stations are in the group. Conversely, broadcasting does not differentiate between stations at all, so it does not require any group management.

An interesting feature of station source addresses is that they are globally unique, assigned centrally by IEEE to ensure that no two stations anywhere in the world have the same address. The idea is that any station can uniquely address any other station by just giving the right 48-bit number. To do this, the first 3 bytes of the address field are used for an **OUI** (**Organizationally Unique Identifier**). Values for this field are assigned by IEEE and indicate a manufacturer. Manufacturers are assigned blocks of 2^{24} addresses. The manufacturer assigns the last 3 bytes of the address and programs the complete address into the NIC before it is sold.

Next comes the *Type* or *Length* field, depending on whether the frame is Ethernet or IEEE 802.3. Ethernet uses a *Type* field to tell the receiver what to do with the frame. Multiple network-layer protocols may be in use at the same time on the same machine, so when an Ethernet frame arrives, the operating system has to know which one to hand the frame to. The *Type* field specifies which process to give the frame to. For example, a type code of 0x0800 means that the data contains an IPv4 packet.

IEEE 802.3, in its wisdom, decided that this field would carry the length of the frame, since the Ethernet length was determined by looking inside the data—a layering violation if ever there was one. Of course, this meant there was no way for the receiver to figure out what to do with an incoming frame. That problem was handled by the addition of another header for the **LLC** (**Logical Link Control**) protocol within the data. It uses 8 bytes to convey the 2 bytes of protocol type information.

Unfortunately, by the time 802.3 was published, so much hardware and software for DIX Ethernet was already in use that few manufacturers and users were enthusiastic about repackaging the *Type* and *Length* fields. In 1997, IEEE threw in the towel and said that both ways were fine with it. Fortunately, all the *Type* fields in use before 1997 had values greater than 1500, then well established as the maximum data size. Now the rule is that any number there less than or equal to 0x600 (1536) can be interpreted as *Length*, and any number greater than 0x600 can be interpreted as *Type*. Now IEEE can maintain that everyone is using its standard and everybody else can keep on doing what they were already doing (not bothering with LLC) without feeling guilty about it.

Next come the data, up to 1500 bytes. This limit was chosen somewhat arbitrarily at the time the Ethernet standard was cast in stone, mostly based on the fact

that a transceiver needs enough RAM to hold an entire frame and RAM was expensive in 1978. A larger upper limit would have meant more RAM, and hence a more expensive transceiver.

In addition to there being a maximum frame length, there is also a minimum frame length. While a data field of 0 bytes is sometimes useful, it causes a problem. When a transceiver detects a collision, it truncates the current frame, which means that stray bits and pieces of frames appear on the cable all the time. To make it easier to distinguish valid frames from garbage, Ethernet requires that valid frames must be at least 64 bytes long, from destination address to checksum, including both. If the data portion of a frame is less than 46 bytes, the *Pad* field is used to fill out the frame to the minimum size.

Another (and more important) reason for having a minimum length frame is to prevent a station from completing the transmission of a short frame before the first bit has even reached the far end of the cable, where it may collide with another frame. This problem is illustrated in Fig. 4-15. At time 0, station *A*, at one end of the network, sends off a frame. Let us call the propagation time for this frame to reach the other end τ. Just before the frame gets to the other end (i.e., at time $\tau - \varepsilon$), the most distant station, *B*, starts transmitting. When *B* detects that it is receiving more power than it is putting out, it knows that a collision has occurred, so it aborts its transmission and generates a 48-bit noise burst to warn all other stations. In other words, it jams the ether to make sure the sender does not miss the collision. At about time 2τ, the sender sees the noise burst and aborts its transmission, too. It then waits a random time before trying again.

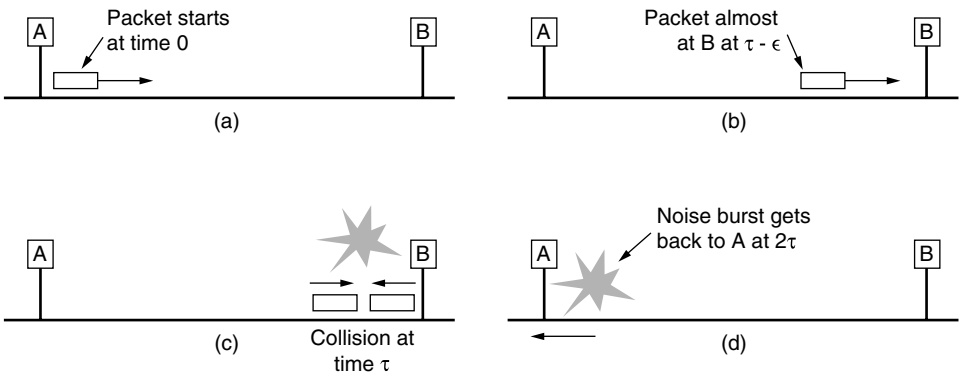


Figure 4-15. Collision detection can take as long as 2τ.

If a station tries to transmit a very short frame, it is conceivable that a collision will occur, but the transmission will have completed before the noise burst gets back to the station at 2τ. The sender will then incorrectly conclude that the frame was successfully sent. To prevent this situation from occurring, all frames must take more than 2τ to send so that the transmission is still taking place when

the noise burst gets back to the sender. For a 10-Mbps LAN with a maximum length of 2500 meters and four repeaters (from the 802.3 specification), the round-trip time (including time to propagate through the four repeaters) has been determined to be nearly 50 μsec in the worst case. Therefore, the shortest allowed frame must take at least this long to transmit. At 10 Mbps, a bit takes 100 nsec, so 500 bits is the smallest frame that is guaranteed to work. To add some margin of safety, this number was rounded up to 512 bits or 64 bytes.

The final field is the *Checksum*. It is a 32-bit CRC of the kind we studied in Sec. 3.2. In fact, it is defined exactly by the generator polynomial we gave there, which popped up for PPP, ADSL, and other links too. This CRC is an error-detecting code that is used to determine if the bits of the frame have been received correctly. It just does error detection, with the frame dropped if an error is detected.

CSMA/CD with Binary Exponential Backoff

Classic Ethernet uses the 1-persistent CSMA/CD algorithm that we studied in Sec. 4.2. This descriptor just means that stations sense the medium when they have a frame to send and send the frame as soon as the medium becomes idle. They monitor the channel for collisions as they send. If there is a collision, they abort the transmission with a short jam signal and retransmit after a random interval.

Let us now see how the random interval is determined when a collision occurs, as it is a new method. The model is still that of Fig. 4-5. After a collision, time is divided into discrete slots whose length is equal to the worst-case round-trip propagation time on the ether (2τ). To accommodate the longest path allowed by Ethernet, the slot time has been set to 512 bit times, or 51.2 μsec.

After the first collision, each station waits either 0 or 1 slot times at random before trying again. If two stations collide and each one picks the same random number, they will collide again. After the second collision, each one picks either 0, 1, 2, or 3 at random and waits that number of slot times. If a third collision occurs (the probability of this happening is 0.25), the next time the number of slots to wait is chosen at random from the interval 0 to $2^3 - 1$.

In general, after i collisions, a random number between 0 and $2^i - 1$ is chosen, and that number of slots is skipped. However, after 10 collisions have been reached, the randomization interval is frozen at a maximum of 1023 slots. After 16 collisions, the controller throws in the towel and reports failure back to the computer. Further recovery is up to higher layers.

This algorithm, called **binary exponential backoff**, was chosen to dynamically adapt to the number of stations trying to send. If the randomization interval for all collisions were 1023, the chance of two stations colliding for a second time would be negligible, but the average wait after a collision would be hundreds of slot times, introducing significant delay. On the other hand, if each station always

delayed for either 0 or 1 slots, then if 100 stations ever tried to send at once they would collide over and over until 99 of them picked 1 and the remaining station picked 0. This might take years. By having the randomization interval grow exponentially as more and more consecutive collisions occur, the algorithm ensures a low delay when only a few stations collide but also ensures that the collisions are resolved in a reasonable interval when many stations collide. Truncating the backoff at 1023 keeps the bound from growing too large.

If there is no collision, the sender assumes that the frame was probably successfully delivered. That is, neither CSMA/CD nor Ethernet provides acknowledgements. This choice is appropriate for wired and optical fiber channels that have low error rates. Any errors that do occur must then be detected by the CRC and recovered by higher layers. For wireless channels that have more errors, we will see that acknowledgements are used.

4.3.3 Ethernet Performance

Now let us briefly examine the performance of classic Ethernet under conditions of heavy and constant load, that is, with k stations always ready to transmit. A rigorous analysis of the binary exponential backoff algorithm is complicated. Instead, we will follow Metcalfe and Boggs (1976) and assume a constant retransmission probability in each slot. If each station transmits during a contention slot with probability p, the probability A that some station acquires the channel in that slot is

$$A = kp(1-p)^{k-1} \tag{4-5}$$

A is maximized when $p = 1/k$, with $A \to 1/e$ as $k \to \infty$. The probability that the contention interval has exactly j slots in it is $A(1-A)^{j-1}$, so the mean number of slots per contention is given by

$$\sum_{j=0}^{\infty} jA(1-A)^{j-1} = \frac{1}{A}$$

Since each slot has a duration 2τ, the mean contention interval, w, is $2\tau/A$. Assuming optimal p, the mean number of contention slots is never more than e, so w is at most $2\tau e \approx 5.4\tau$.

If the mean frame takes P sec to transmit, when many stations have frames to send,

$$\text{Channel efficiency} = \frac{P}{P + 2\tau/A} \tag{4-6}$$

Here we see where the maximum cable distance between any two stations enters into the performance figures. The longer the cable, the longer the contention interval, which is why the Ethernet standard specifies a maximum cable length.

It is instructive to formulate Eq. (4-6) in terms of the frame length, F, the network bandwidth, B, the cable length, L, and the speed of signal propagation, c, for the optimal case of e contention slots per frame. With $P = F/B$, Eq. (4-6) becomes

$$\text{Channel efficiency} = \frac{1}{1 + 2BLe/cF} \tag{4-7}$$

When the second term in the denominator is large, network efficiency will be low. More specifically, increasing network bandwidth or distance (the BL product) reduces efficiency for a given frame size. Unfortunately, much research on network hardware is aimed precisely at increasing this product. People want high bandwidth over long distances (fiber optic MANs, for example), yet classic Ethernet implemented in this manner is not the best system for these applications. We will see other ways of implementing Ethernet in the next section.

In Fig. 4-16, the channel efficiency is plotted versus the number of ready stations for $2\tau = 51.2$ μsec and a data rate of 10 Mbps, using Eq. (4-7). With a 64-byte slot time, it is not surprising that 64-byte frames are not efficient. On the other hand, with 1024-byte frames and an asymptotic value of e 64-byte slots per contention interval, the contention period is 174 bytes long and the efficiency is 85%. This result is much better than the 37% efficiency of slotted ALOHA.

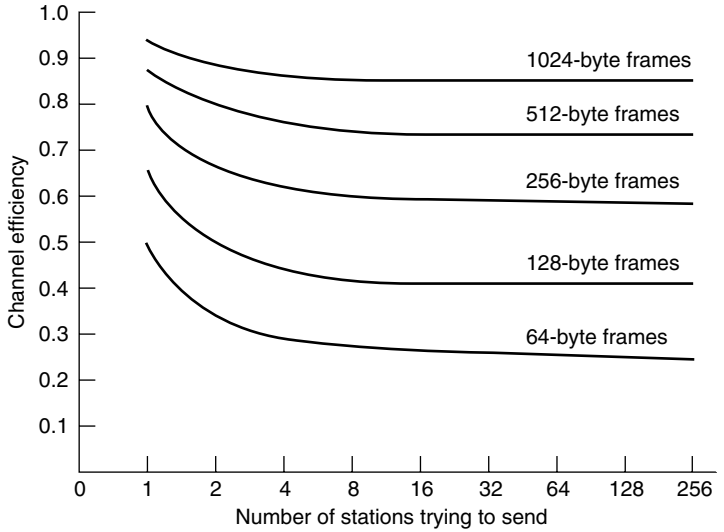


Figure 4-16. Efficiency of Ethernet at 10 Mbps with 512-bit slot times.

It is probably worth mentioning that there has been a large amount of theoretical performance analysis of Ethernet (and other networks). Most of the results should be taken with a grain (or better yet, a metric ton) of salt, for two reasons.

First, virtually all of the theoretical work assumes Poisson traffic. As researchers have begun looking at real data, it now appears that network traffic is rarely Poisson. Instead, it is self-similar or bursty over a range of time scales (Paxson and Floyd, 1995; and Leland et al., 1994). What this means is that averaging over long periods of time does not smooth out the traffic. As well as using questionable models, many of the analyses focus on the ''interesting'' performance cases of abnormally high load. Boggs et al. (1988) showed by experimentation that Ethernet works well in reality, even at moderately high load.

4.3.4 Switched Ethernet

Ethernet soon began to evolve away from the single long cable architecture of classic Ethernet. The problems associated with finding breaks or loose connections drove it toward a different kind of wiring pattern, in which each station has a dedicated cable running to a central **hub**. A hub simply connects all the attached wires electrically, as if they were soldered together. This configuration is shown in Fig. 4-17(a).

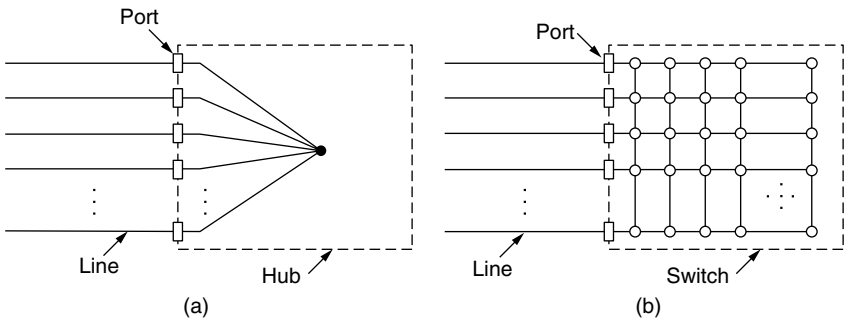


Figure 4-17. (a) Hub. (b) Switch.

The wires were telephone company twisted pairs, since most office buildings were already wired this way and normally plenty of spares were available. This reuse was a win, but it did reduce the maximum cable run from the hub to 100 meters (200 meters if high quality Category 5 twisted pairs were used). Adding or removing a station is simpler in this configuration, and cable breaks can be detected easily. With the advantages of being able to use existing wiring and ease of maintenance, twisted-pair hubs quickly became the dominant form of Ethernet.

However, hubs do not increase capacity because they are logically equivalent to the single long cable of classic Ethernet. As more and more stations are added, each station gets a decreasing share of the fixed capacity. Eventually, the LAN will saturate. One way out is to go to a higher speed, say, from 10 Mbps to 100 Mbps, 1 Gbps, or even higher speeds. But with the growth of multimedia and powerful servers, even a 1-Gbps Ethernet can become saturated.

Patent Owner's Ex. 2010
Page 136

Fortunately, there is an another way to deal with increased load: switched Ethernet. The heart of this system is a **switch** containing a high-speed backplane that connects all of the ports, as shown in Fig. 4-17(b). From the outside, a switch looks just like a hub. They are both boxes, typically with 4 to 48 ports, each with a standard RJ-45 connector for a twisted-pair cable. Each cable connects the switch or hub to a single computer, as shown in Fig. 4-18. A switch has the same advantages as a hub, too. It is easy to add or remove a new station by plugging or unplugging a wire, and it is easy to find most faults since a flaky cable or port will usually affect just one station. There is still a shared component that can fail—the switch itself—but if all stations lose connectivity the IT folks know what to do to fix the problem: replace the whole switch.

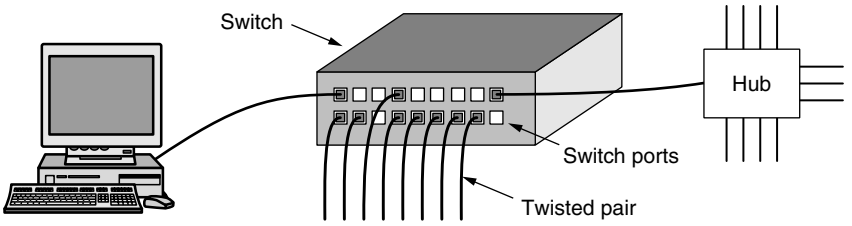


Figure 4-18. An Ethernet switch.

Inside the switch, however, something very different is happening. Switches only output frames to the ports for which those frames are destined. When a switch port receives an Ethernet frame from a station, the switch checks the Ethernet addresses to see which port the frame is destined for. This step requires the switch to be able to work out which ports correspond to which addresses, a process that we will describe in Sec. 4.8 when we get to the general case of switches connected to other switches. For now, just assume that the switch knows the frame's destination port. The switch then forwards the frame over its high-speed backplane to the destination port. The backplane typically runs at many Gbps, using a proprietary protocol that does not need to be standardized because it is entirely hidden inside the switch. The destination port then transmits the frame on the wire so that it reaches the intended station. None of the other ports even knows the frame exists.

What happens if more than one of the stations or ports wants to send a frame at the same time? Again, switches differ from hubs. In a hub, all stations are in the same **collision domain**. They must use the CSMA/CD algorithm to schedule their transmissions. In a switch, each port is its own independent collision domain. In the common case that the cable is full duplex, both the station and the port can send a frame on the cable at the same time, without worrying about other ports and stations. Collisions are now impossible and CSMA/CD is not needed. However, if the cable is half duplex, the station and the port must contend for transmission with CSMA/CD in the usual way.

A switch improves performance over a hub in two ways. First, since there are no collisions, the capacity is used more efficiently. Second, and more importantly, with a switch multiple frames can be sent simultaneously (by different stations). These frames will reach the switch ports and travel over the switch's backplane to be output on the proper ports. However, since two frames might be sent to the same output port at the same time, the switch must have buffering so that it can temporarily queue an input frame until it can be transmitted to the output port. Overall, these improvements give a large performance win that is not possible with a hub. The total system throughput can often be increased by an order of magnitude, depending on the number of ports and traffic patterns.

The change in the ports on which frames are output also has security benefits. Most LAN interfaces have a **promiscuous mode**, in which *all* frames are given to each computer, not just those addressed to it. With a hub, every computer that is attached can see the traffic sent between all of the other computers. Spies and busybodies love this feature. With a switch, traffic is forwarded only to the ports where it is destined. This restriction provides better isolation so that traffic will not easily escape and fall into the wrong hands. However, it is better to encrypt traffic if security is really needed.

Because the switch just expects standard Ethernet frames on each input port, it is possible to use some of the ports as concentrators. In Fig. 4-18, the port in the upper-right corner is connected not to a single station, but to a 12-port hub instead. As frames arrive at the hub, they contend for the ether in the usual way, including collisions and binary backoff. Successful frames make it through the hub to the switch and are treated there like any other incoming frames. The switch does not know they had to fight their way in. Once in the switch, they are sent to the correct output line over the high-speed backplane. It is also possible that the correct destination was one on the lines attached to the hub, in which case the frame has already been delivered so the switch just drops it. Hubs are simpler and cheaper than switches, but due to falling switch prices they have become an endangered species. Modern networks largely use switched Ethernet. Nevertheless, legacy hubs still exist.

4.3.5 Fast Ethernet

At the same time that switches were becoming popular, the speed of 10-Mbps Ethernet was coming under pressure. At first, 10 Mbps seemed like heaven, just as cable modems seemed like heaven to the users of telephone modems. But the novelty wore off quickly. As a kind of corollary to Parkinson's Law (''Work expands to fill the time available for its completion''), it seemed that data expanded to fill the bandwidth available for their transmission.

Many installations needed more bandwidth and thus had numerous 10-Mbps LANs connected by a maze of repeaters, hubs, and switches, although to the network managers it sometimes felt that they were being held together by bubble

gum and chicken wire. But even with Ethernet switches, the maximum bandwidth of a single computer was limited by the cable that connected it to the switch port.

It was in this environment that IEEE reconvened the 802.3 committee in 1992 with instructions to come up with a faster LAN. One proposal was to keep 802.3 exactly as it was, but just make it go faster. Another proposal was to redo it totally and give it lots of new features, such as real-time traffic and digitized voice, but just keep the old name (for marketing reasons). After some wrangling, the committee decided to keep 802.3 the way it was, and just make it go faster. This strategy would get the job done before the technology changed and avoid unforeseen problems with a brand new design. The new design would also be backward-compatible with existing Ethernet LANs. The people behind the losing proposal did what any self-respecting computer-industry people would have done under these circumstances: they stomped off and formed their own committee and standardized their LAN anyway (eventually as 802.12). It flopped miserably.

The work was done quickly (by standards committees' norms), and the result, 802.3u, was approved by IEEE in June 1995. Technically, 802.3u is not a new standard, but an addendum to the existing 802.3 standard (to emphasize its backward compatibility). This strategy is used a lot. Since practically everyone calls it **fast Ethernet**, rather than 802.3u, we will do that, too.

The basic idea behind fast Ethernet was simple: keep all the old frame formats, interfaces, and procedural rules, but reduce the bit time from 100 nsec to 10 nsec. Technically, it would have been possible to copy 10-Mbps classic Ethernet and still detect collisions on time by just reducing the maximum cable length by a factor of 10. However, the advantages of twisted-pair wiring were so overwhelming that fast Ethernet is based entirely on this design. Thus, all fast Ethernet systems use hubs and switches; multidrop cables with vampire taps or BNC connectors are not permitted.

Nevertheless, some choices still had to be made, the most important being which wire types to support. One contender was Category 3 twisted pair. The argument for it was that practically every office in the Western world had at least four Category 3 (or better) twisted pairs running from it to a telephone wiring closet within 100 meters. Sometimes two such cables existed. Thus, using Category 3 twisted pair would make it possible to wire up desktop computers using fast Ethernet without having to rewire the building, an enormous advantage for many organizations.

The main disadvantage of a Category 3 twisted pair is its inability to carry 100 Mbps over 100 meters, the maximum computer-to-hub distance specified for 10-Mbps hubs. In contrast, Category 5 twisted pair wiring can handle 100 m easily, and fiber can go much farther. The compromise chosen was to allow all three possibilities, as shown in Fig. 4-19, but to pep up the Category 3 solution to give it the additional carrying capacity needed.

The Category 3 UTP scheme, called **100Base-T4**, used a signaling speed of 25 MHz, only 25% faster than standard Ethernet's 20 MHz. (Remember that

Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps (Cat 5 UTP)
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

Figure 4-19. The original fast Ethernet cabling.

Manchester encoding, discussed in Sec. 2.5, requires two clock periods for each of the 10 million bits sent each second.) However, to achieve the necessary bit rate, 100Base-T4 requires four twisted pairs. Of the four pairs, one is always to the hub, one is always from the hub, and the other two are switchable to the current transmission direction. To get 100 Mbps out of the three twisted pairs in the transmission direction, a fairly involved scheme is used on each twisted pair. It involves sending ternary digits with three different voltage levels. This scheme is not likely to win any prizes for elegance, and we will skip the details. However, since standard telephone wiring for decades has had four twisted pairs per cable, most offices are able to use the existing wiring plant. Of course, it means giving up your office telephone, but that is surely a small price to pay for faster email.

100Base-T4 fell by the wayside as many office buildings were rewired with Category 5 UTP for **100Base-TX** Ethernet, which came to dominate the market. This design is simpler because the wires can handle clock rates of 125 MHz. Only two twisted pairs per station are used, one to the hub and one from it. Neither straight binary coding (i.e., NRZ) nor Manchester coding is used. Instead, the **4B/5B** encoding we described in Sec 2.5 is used. 4 data bits are encoded as 5 signal bits and sent at 125 MHz to provide 100 Mbps. This scheme is simple but has sufficient transitions for synchronization and uses the bandwidth of the wire relatively well. The 100Base-TX system is full duplex; stations can transmit at 100 Mbps on one twisted pair and receive at 100 Mbps on another twisted pair at the same time.

The last option, **100Base-FX**, uses two strands of multimode fiber, one for each direction, so it, too, can run full duplex with 100 Mbps in each direction. In this setup, the distance between a station and the switch can be up to 2 km.

Fast Ethernet allows interconnection by either hubs or switches. To ensure that the CSMA/CD algorithm continues to work, the relationship between the minimum frame size and maximum cable length must be maintained as the network speed goes up from 10 Mbps to 100 Mbps. So, either the minimum frame size of 64 bytes must go up or the maximum cable length of 2500 m must come down, proportionally. The easy choice was for the maximum distance between any two stations to come down by a factor of 10, since a hub with 100-m cables falls within this new maximum already. However, 2-km 100Base-FX cables are

too long to permit a 100-Mbps hub with the normal Ethernet collision algorithm. These cables must instead be connected to a switch and operate in a full-duplex mode so that there are no collisions.

Users quickly started to deploy fast Ethernet, but they were not about to throw away 10-Mbps Ethernet cards on older computers. As a consequence, virtually all fast Ethernet switches can handle a mix of 10-Mbps and 100-Mbps stations. To make upgrading easy, the standard itself provides a mechanism called **auto-negotiation** that lets two stations automatically negotiate the optimum speed (10 or 100 Mbps) and duplexity (half or full). It works well most of the time but is known to lead to duplex mismatch problems when one end of the link autonegotiates but the other end does not and is set to full-duplex mode (Shalunov and Carlson, 2005). Most Ethernet products use this feature to configure themselves.

4.3.6 Gigabit Ethernet

The ink was barely dry on the fast Ethernet standard when the 802 committee began working on a yet faster Ethernet, quickly dubbed **gigabit Ethernet**. IEEE ratified the most popular form as 802.3ab in 1999. Below we will discuss some of the key features of gigabit Ethernet. More information is given by Spurgeon (2000).

The committee's goals for gigabit Ethernet were essentially the same as the committee's goals for fast Ethernet: increase performance tenfold while maintaining compatibility with all existing Ethernet standards. In particular, gigabit Ethernet had to offer unacknowledged datagram service with both unicast and broadcast, use the same 48-bit addressing scheme already in use, and maintain the same frame format, including the minimum and maximum frame sizes. The final standard met all these goals.

Like fast Ethernet, all configurations of gigabit Ethernet use point-to-point links. In the simplest configuration, illustrated in Fig. 4-20(a), two computers are directly connected to each other. The more common case, however, uses a switch or a hub connected to multiple computers and possibly additional switches or hubs, as shown in Fig. 4-20(b). In both configurations, each individual Ethernet cable has exactly two devices on it, no more and no fewer.

Also like fast Ethernet, gigabit Ethernet supports two different modes of operation: full-duplex mode and half-duplex mode. The "normal" mode is full-duplex mode, which allows traffic in both directions at the same time. This mode is used when there is a central switch connected to computers (or other switches) on the periphery. In this configuration, all lines are buffered so each computer and switch is free to send frames whenever it wants to. The sender does not have to sense the channel to see if anybody else is using it because contention is impossible. On the line between a computer and a switch, the computer is the only possible sender to the switch, and the transmission will succeed even if the switch is currently sending a frame to the computer (because the line is full duplex). Since

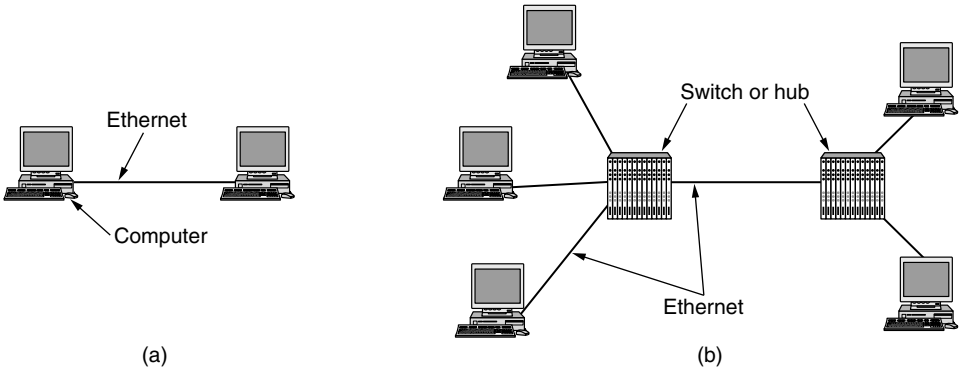


Figure 4-20. (a) A two-station Ethernet. (b) A multistation Ethernet.

no contention is possible, the CSMA/CD protocol is not used, so the maximum length of the cable is determined by signal strength issues rather than by how long it takes for a noise burst to propagate back to the sender in the worst case. Switch\%es are free to mix and match speeds. Autonegotiation is supported just as in fast Ethernet, only now the choice is among 10, 100, and 1000 Mbps.

The other mode of operation, half-duplex, is used when the computers are connected to a hub rather than a switch. A hub does not buffer incoming frames. Instead, it electrically connects all the lines internally, simulating the multidrop cable used in classic Ethernet. In this mode, collisions are possible, so the standard CSMA/CD protocol is required. Because a 64-byte frame (the shortest allowed) can now be transmitted 100 times faster than in classic Ethernet, the maximum cable length must be 100 times less, or 25 meters, to maintain the essential property that the sender is still transmitting when the noise burst gets back to it, even in the worst case. With a 2500-meter-long cable, the sender of a 64-byte frame at 1 Gbps would be long finished before the frame got even a tenth of the way to the other end, let alone to the end and back.

This length restriction was painful enough that two features were added to the standard to increase the maximum cable length to 200 meters, which is probably enough for most offices. The first feature, called **carrier extension**, essentially tells the hardware to add its own padding after the normal frame to extend the frame to 512 bytes. Since this padding is added by the sending hardware and removed by the receiving hardware, the software is unaware of it, meaning that no changes are needed to existing software. The downside is that using 512 bytes worth of bandwidth to transmit 46 bytes of user data (the payload of a 64-byte frame) has a line efficiency of only 9%.

The second feature, called **frame bursting**, allows a sender to transmit a concatenated sequence of multiple frames in a single transmission. If the total burst is less than 512 bytes, the hardware pads it again. If enough frames are waiting for transmission, this scheme is very efficient and preferred over carrier extension.

In all fairness, it is hard to imagine an organization buying modern computers with gigabit Ethernet cards and then connecting them with an old-fashioned hub to simulate classic Ethernet with all its collisions. Gigabit Ethernet interfaces and switches used to be expensive, but their prices fell rapidly as sales volumes picked up. Still, backward compatibility is sacred in the computer industry, so the committee was required to put it in. Today, most computers ship with an Ethernet interface that is capable of 10-, 100-, and 1000-Mbps operation and compatible with all of them.

Gigabit Ethernet supports both copper and fiber cabling, as listed in Fig. 4-21. Signaling at or near 1 Gbps requires encoding and sending a bit every nanosecond. This trick was initially accomplished with short, shielded copper cables (the 1000Base-CX version) and optical fibers. For the optical fibers, two wavelengths are permitted and result in two different versions: 0.85 microns (short, for 1000Base-SX) and 1.3 microns (long, for 1000Base-LX).

Name	Cable	Max. segment	Advantages
1000Base-SX	Fiber optics	550 m	Multimode fiber (50, 62.5 microns)
1000Base-LX	Fiber optics	5000 m	Single (10 μ) or multimode (50, 62.5 μ)
1000Base-CX	2 Pairs of STP	25 m	Shielded twisted pair
1000Base-T	4 Pairs of UTP	100 m	Standard category 5 UTP

Figure 4-21. Gigabit Ethernet cabling.

Signaling at the short wavelength can be achieved with cheaper LEDs. It is used with multimode fiber and is useful for connections within a building, as it can run up to 500 m for 50-micron fiber. Signaling at the long wavelength requires more expensive lasers. On the other hand, when combined with single-mode (10-micron) fiber, the cable length can be up to 5 km. This limit allows long distance connections between buildings, such as for a campus backbone, as a dedicated point-to-point link. Later variations of the standard allowed even longer links over single-mode fiber.

To send bits over these versions of gigabit Ethernet, the **8B/10B** encoding we described in Sec. 2.5 was borrowed from another networking technology called Fibre Channel. That scheme encodes 8 bits of data into 10-bit codewords that are sent over the wire or fiber, hence the name 8B/10B. The codewords were chosen so that they could be balanced (i.e., have the same number of 0s and 1s) with sufficient transitions for clock recovery. Sending the coded bits with NRZ requires a signaling bandwidth of 25% more than that required for the uncoded bits, a big improvement over the 100% expansion of Manchester coding.

However, all of these options required new copper or fiber cables to support the faster signaling. None of them made use of the large amount of Category 5 UTP that had been installed along with fast Ethernet. Within a year, 1000Base-T

came along to fill this gap, and it has been the most popular form of gigabit Ethernet ever since. People apparently dislike rewiring their buildings.

More complicated signaling is needed to make Ethernet run at 1000 Mbps over Category 5 wires. To start, all four twisted pairs in the cable are used, and each pair is used in both directions at the same time by using digital signal processing to separate signals. Over each wire, five voltage levels that carry 2 bits are used for signaling at 125 Msymbols/sec. The mapping to produce the symbols from the bits is not straightforward. It involves scrambling, for transitions, followed by an error correcting code in which four values are embedded into five signal levels.

A speed of 1 Gbps is quite fast. For example, if a receiver is busy with some other task for even 1 msec and does not empty the input buffer on some line, up to 1953 frames may have accumulated in that gap. Also, when a computer on a gigabit Ethernet is shipping data down the line to a computer on a classic Ethernet, buffer overruns are very likely. As a consequence of these two observations, gigabit Ethernet supports flow control. The mechanism consists of one end sending a special control frame to the other end telling it to pause for some period of time. These PAUSE control frames are normal Ethernet frames containing a type of 0x8808. Pauses are given in units of the minimum frame time. For gigabit Ethernet, the time unit is 512 nsec, allowing for pauses as long as 33.6 msec.

There is one more extension that was introduced along with gigabit Ethernet. **Jumbo frames** allow for frames to be longer than 1500 bytes, usually up to 9 KB. This extension is proprietary. It is not recognized by the standard because if it is used then Ethernet is no longer compatible with earlier versions, but most vendors support it anyway. The rationale is that 1500 bytes is a short unit at gigabit speeds. By manipulating larger blocks of information, the frame rate can be decreased, along with the processing associated with it, such as interrupting the processor to say that a frame has arrived, or splitting up and recombining messages that were too long to fit in one Ethernet frame.

4.3.7 10-Gigabit Ethernet

As soon as gigabit Ethernet was standardized, the 802 committee got bored and wanted to get back to work. IEEE told them to start on 10-gigabit Ethernet. This work followed much the same pattern as the previous Ethernet standards, with standards for fiber and shielded copper cable appearing first in 2002 and 2004, followed by the standard for copper twisted pair in 2006.

10 Gbps is a truly prodigious speed, 1000x faster than the original Ethernet. Where could it be needed? The answer is inside data centers and exchanges to connect high-end routers, switches, and servers, as well as in long-distance, high bandwidth trunks between offices that are enabling entire metropolitan area networks based on Ethernet and fiber. The long distance connections use optical fiber, while the short connections may use copper or fiber.

All versions of 10-gigabit Ethernet support only full-duplex operation. CSMA/CD is no longer part of the design, and the standards concentrate on the details of physical layers that can run at very high speed. Compatibility still matters, though, so 10-gigabit Ethernet interfaces autonegotiate and fall back to the highest speed supported by both ends of the line.

The main kinds of 10-gigabit Ethernet are listed in Fig. 4-22. Multimode fiber with the 0.85μ (short) wavelength is used for medium distances, and single-mode fiber at 1.3μ (long) and 1.5μ (extended) is used for long distances. 10GBase-ER can run for distances of 40 km, making it suitable for wide area applications. All of these versions send a serial stream of information that is produced by scrambling the data bits, then encoding them with a **64B/66B** code. This encoding has less overhead than an 8B/10B code.

Name	Cable	Max. segment	Advantages
10GBase-SR	Fiber optics	Up to 300 m	Multimode fiber (0.85μ)
10GBase-LR	Fiber optics	10 km	Single-mode fiber (1.3μ)
10GBase-ER	Fiber optics	40 km	Single-mode fiber (1.5μ)
10GBase-CX4	4 Pairs of twinax	15 m	Twinaxial copper
10GBase-T	4 Pairs of UTP	100 m	Category 6a UTP

Figure 4-22. 10-Gigabit Ethernet cabling.

The first copper version defined, 10GBase-CX4, uses a cable with four pairs of twinaxial copper wiring. Each pair uses 8B/10B coding and runs at 3.125 Gsymbols/second to reach 10 Gbps. This version is cheaper than fiber and was early to market, but it remains to be seen whether it will be beat out in the long run by 10-gigabit Ethernet over more garden variety twisted pair wiring.

10GBase-T is the version that uses UTP cables. While it calls for Category 6a wiring, for shorter runs, it can use lower categories (including Category 5) to allow some reuse of installed cabling. Not surprisingly, the physical layer is quite involved to reach 10 Gbps over twisted pair. We will only sketch some of the high-level details. Each of the four twisted pairs is used to send 2500 Mbps in both directions. This speed is reached using a signaling rate of 800 Msymbols/sec with symbols that use 16 voltage levels. The symbols are produced by scrambling the data, protecting it with a LDPC (Low Density Parity Check) code, and further coding for error correction.

10-gigabit Ethernet is still shaking out in the market, but the 802.3 committee has already moved on. At the end of 2007, IEEE created a group to standardize Ethernet operating at 40 Gbps and 100 Gbps. This upgrade will let Ethernet compete in very high-performance settings, including long-distance connections in backbone networks and short connections over the equipment backplanes. The standard is not yet complete, but proprietary products are already available.

Patent Owner's Ex. 2010
Page 145

4.3.8 Retrospective on Ethernet

Ethernet has been around for over 30 years and has no serious competitors in sight, so it is likely to be around for many years to come. Few CPU architectures, operating systems, or programming languages have been king of the mountain for three decades going on strong. Clearly, Ethernet did something right. What?

Probably the main reason for its longevity is that Ethernet is simple and flexible. In practice, simple translates into reliable, cheap, and easy to maintain. Once the hub and switch architecture was adopted, failures became extremely rare. People hesitate to replace something that works perfectly all the time, especially when they know that an awful lot of things in the computer industry work very poorly, so that many so-called "upgrades" are worse than what they replaced.

Simple also translates into cheap. Twisted-pair wiring is relatively inexpensive as are the hardware components. They may start out expensive when there is a transition, for example, new gigabit Ethernet NICs or switches, but they are merely additions to a well established network (not a replacement of it) and the prices fall quickly as the sales volume picks up.

Ethernet is easy to maintain. There is no software to install (other than the drivers) and not much in the way of configuration tables to manage (and get wrong). Also, adding new hosts is as simple as just plugging them in.

Another point is that Ethernet interworks easily with TCP/IP, which has become dominant. IP is a connectionless protocol, so it fits perfectly with Ethernet, which is also connectionless. IP fits much less well with connection-oriented alternatives such as ATM. This mismatch definitely hurt ATM's chances.

Lastly, and perhaps most importantly, Ethernet has been able to evolve in certain crucial ways. Speeds have gone up by several orders of magnitude and hubs and switches have been introduced, but these changes have not required changing the software and have often allowed the existing cabling to be reused for a time. When a network salesman shows up at a large installation and says "I have this fantastic new network for you. All you have to do is throw out all your hardware and rewrite all your software," he has a problem.

Many alternative technologies that you have probably not even heard of were faster than Ethernet when they were introduced. As well as ATM, this list includes FDDI (Fiber Distributed Data Interface) and Fibre Channel,† two ring-based optical LANs. Both were incompatible with Ethernet. Neither one made it. They were too complicated, which led to complex chips and high prices. The lesson that should have been learned here was KISS (Keep It Simple, Stupid). Eventually, Ethernet caught up with them in terms of speed, often by borrowing some of their technology, for example, the 4B/5B coding from FDDI and the 8B/10B coding from Fibre Channel. Then they had no advantages left and quietly died off or fell into specialized roles.

† It is called "Fibre Channel" and not "Fiber Channel" because the document editor was British.

It looks like Ethernet will continue to expand in its applications for some time. 10-gigabit Ethernet has freed it from the distance constraints of CSMA/CD. Much effort is being put into **carrier-grade Ethernet** to let network providers offer Ethernet-based services to their customers for metropolitan and wide area networks (Fouli and Maler, 2009). This application carries Ethernet frames long distances over fiber and calls for better management features to help operators offer reliable, high-quality services. Very high speed networks are also finding uses in backplanes connecting components in large routers or servers. Both of these uses are in addition to that of sending frames between computers in offices.

4.4 WIRELESS LANS

Wireless LANs are increasingly popular, and homes, offices, cafes, libraries, airports, zoos, and other public places are being outfitted with them to connect computers, PDAs, and smart phones to the Internet. Wireless LANs can also be used to let two or more nearby computers communicate without using the Internet.

The main wireless LAN standard is 802.11. We gave some background information on it in Sec. 1.5.3. Now it is time to take a closer look at the technology. In the following sections, we will look at the protocol stack, physical-layer radio transmission techniques, the MAC sublayer protocol, the frame structure, and the services provided. For more information about 802.11, see Gast (2005). To get the truth from the mouth of the horse, consult the published standard, IEEE 802.11-2007 itself.

4.4.1 The 802.11 Architecture and Protocol Stack

802.11 networks can be used in two modes. The most popular mode is to connect clients, such as laptops and smart phones, to another network, such as a company intranet or the Internet. This mode is shown in Fig. 4-23(a). In infrastructure mode, each client is associated with an **AP** (**Access Point**) that is in turn connected to the other network. The client sends and receives its packets via the AP. Several access points may be connected together, typically by a wired network called a **distribution system**, to form an extended 802.11 network. In this case, clients can send frames to other clients via their APs.

The other mode, shown in Fig. 4-23(b), is an **ad hoc network**. This mode is a collection of computers that are associated so that they can directly send frames to each other. There is no access point. Since Internet access is the killer application for wireless, ad hoc networks are not very popular.

Now we will look at the protocols. All the 802 protocols, including 802.11 and Ethernet, have a certain commonality of structure. A partial view of the 802.11 protocol stack is given in Fig. 4-24. The stack is the same for clients and

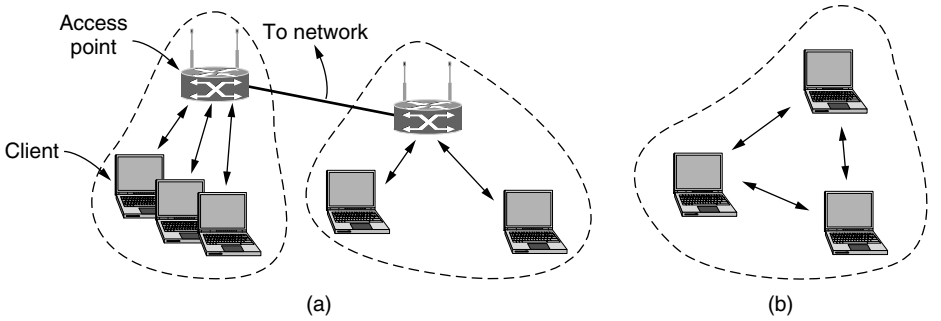


Figure 4-23. 802.11 architecture. (a) Infrastructure mode. (b) Ad-hoc mode.

APs. The physical layer corresponds fairly well to the OSI physical layer, but the data link layer in all the 802 protocols is split into two or more sublayers. In 802.11, the MAC (Medium Access Control) sublayer determines how the channel is allocated, that is, who gets to transmit next. Above it is the LLC (Logical Link Control) sublayer, whose job it is to hide the differences between the different 802 variants and make them indistinguishable as far as the network layer is concerned. This could have been a significant responsibility, but these days the LLC is a glue layer that identifies the protocol (e.g., IP) that is carried within an 802.11 frame.

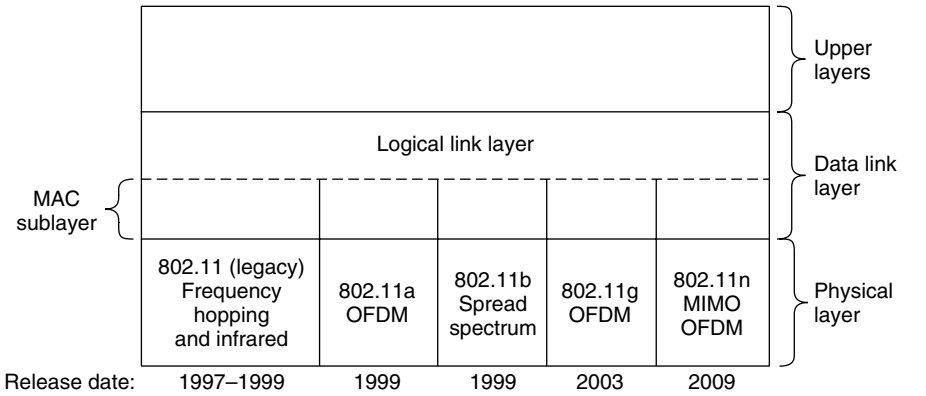


Figure 4-24. Part of the 802.11 protocol stack.

Several transmission techniques have been added to the physical layer as 802.11 has evolved since it first appeared in 1997. Two of the initial techniques, infrared in the manner of television remote controls and frequency hopping in the 2.4-GHz band, are now defunct. The third initial technique, direct sequence spread spectrum at 1 or 2 Mbps in the 2.4-GHz band, was extended to run at rates up to 11 Mbps and quickly became a hit. It is now known as 802.11b.

To give wireless junkies a much-wanted speed boost, new transmission techniques based on the OFDM (Orthogonal Frequency Division Multiplexing) scheme we described in Sec. 2.5.3 were introduced in 1999 and 2003. The first is called 802.11a and uses a different frequency band, 5 GHz. The second stuck with 2.4 GHz and compatibility. It is called 802.11g. Both give rates up to 54 Mbps.

Most recently, transmission techniques that simultaneously use multiple antennas at the transmitter and receiver for a speed boost were finalized as 802.11n in Oct. 2009. With four antennas and wider channels, the 802.11 standard now defines rates up to a startling 600 Mbps.

We will now examine each of these transmission techniques briefly. We will only cover those that are in use, however, skipping the legacy 802.11 transmission methods. Technically, these belong to the physical layer and should have been examined in Chap. 2, but since they are so closely tied to LANs in general and the 802.11 LAN in particular, we treat them here instead.

4.4.2 The 802.11 Physical Layer

Each of the transmission techniques makes it possible to send a MAC frame over the air from one station to another. They differ, however, in the technology used and speeds achievable. A detailed discussion of these technologies is far beyond the scope of this book, but a few words on each one will relate the techniques to the material we covered in Sec. 2.5 and will provide interested readers with the key terms to search for elsewhere for more information.

All of the 802.11 techniques use short-range radios to transmit signals in either the 2.4-GHz or the 5-GHz ISM frequency bands, both described in Sec. 2.3.3. These bands have the advantage of being unlicensed and hence freely available to any transmitter willing to meet some restrictions, such as radiated power of at most 1 W (though 50 mW is more typical for wireless LAN radios). Unfortunately, this fact is also known to the manufacturers of garage door openers, cordless phones, microwave ovens, and countless other devices, all of which compete with laptops for the same spectrum. The 2.4-GHz band tends to be more crowded than the 5-GHz band, so 5 GHz can be better for some applications even though it has shorter range due to the higher frequency.

All of the transmission methods also define multiple rates. The idea is that different rates can be used depending on the current conditions. If the wireless signal is weak, a low rate can be used. If the signal is clear, the highest rate can be used. This adjustment is called **rate adaptation**. Since the rates vary by a factor of 10 or more, good rate adaptation is important for good performance. Of course, since it is not needed for interoperability, the standards do not say how rate adaptation should be done.

The first transmission method we shall look at is **802.11b**. It is a spread-spectrum method that supports rates of 1, 2, 5.5, and 11 Mbps, though in practice the operating rate is nearly always 11 Mbps. It is similar to the CDMA system we

examined in Sec. 2.5, except that there is only one spreading code that is shared by all users. Spreading is used to satisfy the FCC requirement that power be spread over the ISM band. The spreading sequence used by 201.11b is a **Barker sequence**. It has the property that its autocorrelation is low except when the sequences are aligned. This property allows a receiver to lock onto the start of a transmission. To send at a rate of 1 Mbps, the Barker sequence is used with BPSK modulation to send 1 bit per 11 chips. The chips are transmitted at a rate of 11 Mchips/sec. To send at 2 Mbps, it is used with QPSK modulation to send 2 bits per 11 chips. The higher rates are different. These rates use a technique called **CCK** (**Complementary Code Keying**) to construct codes instead of the Barker sequence. The 5.5-Mbps rate sends 4 bits in every 8-chip code, and the 11-Mbps rate sends 8 bits in every 8-chip code.

Next we come to **802.11a**, which supports rates up to 54 Mbps in the 5-GHz ISM band. You might have expected that 802.11a to come before 802.11b, but that was not the case. Although the 802.11a group was set up first, the 802.11b standard was approved first and its product got to market well ahead of the 802.11a products, partly because of the difficulty of operating in the higher 5-GHz band.

The 802.11a method is based on **OFDM** (**Orthogonal Frequency Division Multiplexing**) because OFDM uses the spectrum efficiently and resists wireless signal degradations such as multipath. Bits are sent over 52 subcarriers in parallel, 48 carrying data and 4 used for synchronization. Each symbol lasts 4μs and sends 1, 2, 4, or 6 bits. The bits are coded for error correction with a binary convolutional code first, so only 1/2, 2/3, or 3/4 of the bits are not redundant. With different combinations, 802.11a can run at eight different rates, ranging from 6 to 54 Mbps. These rates are significantly faster than 802.11b rates, and there is less interference in the 5-GHz band. However, 802.11b has a range that is about seven times greater than that of 802.11a, which is more important in many situations.

Even with the greater range, the 802.11b people had no intention of letting this upstart win the speed championship. Fortunately, in May 2002, the FCC dropped its long-standing rule requiring all wireless communications equipment operating in the ISM bands in the U.S. to use spread spectrum, so it got to work on **802.11g**, which was approved by IEEE in 2003. It copies the OFDM modulation methods of 802.11a but operates in the narrow 2.4-GHz ISM band along with 802.11b. It offers the same rates as 802.11a (6 to 54 Mbps) plus of course compatibility with any 802.11b devices that happen to be nearby. All of these different choices can be confusing for customers, so it is common for products to support 802.11a/b/g in a single NIC.

Not content to stop there, the IEEE committee began work on a high-throughput physical layer called **802.11n**. It was ratified in 2009. The goal for 802.11n was throughput of at least 100 Mbps after all the wireless overheads were removed. This goal called for a raw speed increase of at least a factor of four. To make it happen, the committee doubled the channels from 20 MHz to 40 MHz and

reduced framing overheads by allowing a group of frames to be sent together. More significantly, however, 802.11n uses up to four antennas to transmit up to four streams of information at the same time. The signals of the streams interfere at the receiver, but they can be separated using **MIMO** (**Multiple Input Multiple Output**) communications techniques. The use of multiple antennas gives a large speed boost, or better range and reliability instead. MIMO, like OFDM, is one of those clever communications ideas that is changing wireless designs and which we are all likely to hear a lot about in the future. For a brief introduction to multiple antennas in 802.11 see Halperin et al. (2010).

4.4.3 The 802.11 MAC Sublayer Protocol

Let us now return from the land of electrical engineering to the land of computer science. The 802.11 MAC sublayer protocol is quite different from that of Ethernet, due to two factors that are fundamental to wireless communication.

First, radios are nearly always half duplex, meaning that they cannot transmit and listen for noise bursts at the same time on a single frequency. The received signal can easily be a million times weaker than the transmitted signal, so it cannot be heard at the same time. With Ethernet, a station just waits until the ether goes silent and then starts transmitting. If it does not receive a noise burst back while transmitting the first 64 bytes, the frame has almost assuredly been delivered correctly. With wireless, this collision detection mechanism does not work.

Instead, 802.11 tries to avoid collisions with a protocol called **CSMA/CA** (**CSMA with Collision Avoidance**). This protocol is conceptually similar to Ethernet's CSMA/CD, with channel sensing before sending and exponential back off after collisions. However, a station that has a frame to send starts with a random backoff (except in the case that it has not used the channel recently and the channel is idle). It does not wait for a collision. The number of slots to backoff is chosen in the range 0 to, say, 15 in the case of the OFDM physical layer. The station waits until the channel is idle, by sensing that there is no signal for a short period of time (called the DIFS, as we explain below), and counts down idle slots, pausing when frames are sent. It sends its frame when the counter reaches 0. If the frame gets through, the destination immediately sends a short acknowledgement. Lack of an acknowledgement is inferred to indicate an error, whether a collision or otherwise. In this case, the sender doubles the backoff period and tries again, continuing with exponential backoff as in Ethernet until the frame has been successfully transmitted or the maximum number of retransmissions has been reached.

An example timeline is shown in Fig. 4-25. Station *A* is the first to send a frame. While *A* is sending, stations *B* and *C* become ready to send. They see that the channel is busy and wait for it to become idle. Shortly after *A* receives an acknowledgement, the channel goes idle. However, rather than sending a frame right away and colliding, *B* and *C* both perform a backoff. *C* picks a short backoff,

and thus sends first. *B* pauses its countdown while it senses that *C* is using the channel, and resumes after *C* has received an acknowledgement. *B* soon completes its backoff and sends its frame.

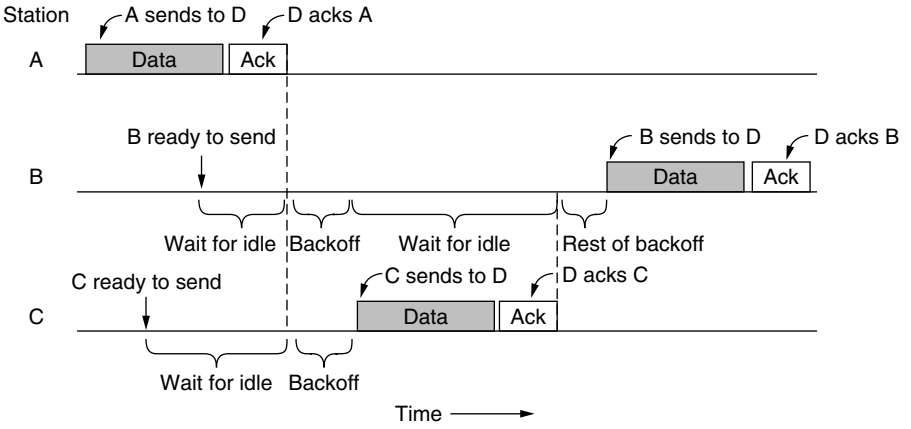


Figure 4-25. Sending a frame with CSMA/CA.

Compared to Ethernet, there are two main differences. First, starting backoffs early helps to avoid collisions. This avoidance is worthwhile because collisions are expensive, as the entire frame is transmitted even if one occurs. Second, acknowledgements are used to infer collisions because collisions cannot be detected.

This mode of operation is called **DCF** (**Distributed Coordination Function**) because each station acts independently, without any kind of central control. The standard also includes an optional mode of operation called **PCF** (**Point Coordination Function**) in which the access point controls all activity in its cell, just like a cellular base station. However, PCF is not used in practice because there is normally no way to prevent stations in another nearby network from transmitting competing traffic.

The second problem is that the transmission ranges of different stations may be different. With a wire, the system is engineered so that all stations can hear each other. With the complexities of RF propagation this situation does not hold for wireless stations. Consequently, situations such as the hidden terminal problem mentioned earlier and illustrated again in Fig. 4-26(a) can arise. Since not all stations are within radio range of each other, transmissions going on in one part of a cell may not be received elsewhere in the same cell. In this example, station *C* is transmitting to station *B*. If *A* senses the channel, it will not hear anything and will falsely conclude that it may now start transmitting to *B*. This decision leads to a collision.

The inverse situation is the exposed terminal problem, illustrated in Fig. 4-26(b). Here, *B* wants to send to *C*, so it listens to the channel. When it hears a

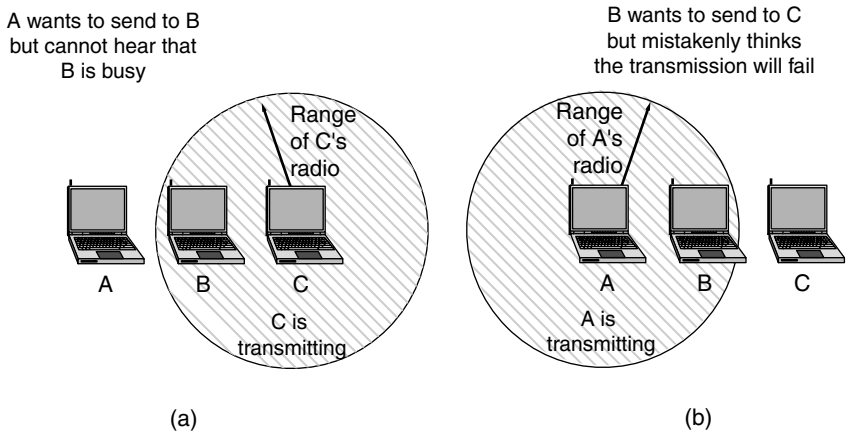


Figure 4-26. (a) The hidden terminal problem. (b) The exposed terminal problem.

transmission, it falsely concludes that it may not send to *C*, even though *A* may in fact be transmitting to *D* (not shown). This decision wastes a transmission opportunity.

To reduce ambiguities about which station is sending, 802.11 defines channel sensing to consist of both physical sensing and virtual sensing. Physical sensing simply checks the medium to see if there is a valid signal. With virtual sensing, each station keeps a logical record of when the channel is in use by tracking the **NAV** (**Network Allocation Vector**). Each frame carries a NAV field that says how long the sequence of which this frame is part will take to complete. Stations that overhear this frame know that the channel will be busy for the period indicated by the NAV, regardless of whether they can sense a physical signal. For example, the NAV of a data frame includes the time needed to send an acknowledgement. All stations that hear the data frame will defer during the acknowledgement period, whether or not they can hear the acknowledgement.

An optional RTS/CTS mechanism uses the NAV to prevent terminals from sending frames at the same time as hidden terminals. It is shown in Fig. 4-27. In this example, *A* wants to send to *B*. *C* is a station within range of *A* (and possibly within range of *B*, but that does not matter). *D* is a station within range of *B* but not within range of *A*.

The protocol starts when *A* decides it wants to send data to *B*. *A* begins by sending an RTS frame to *B* to request permission to send it a frame. If *B* receives this request, it answers with a CTS frame to indicate that the channel is clear to send. Upon receipt of the CTS, *A* sends its frame and starts an ACK timer. Upon correct receipt of the data frame, *B* responds with an ACK frame, completing the exchange. If *A*'s ACK timer expires before the ACK gets back to it, it is treated as a collision and the whole protocol is run again after a backoff.

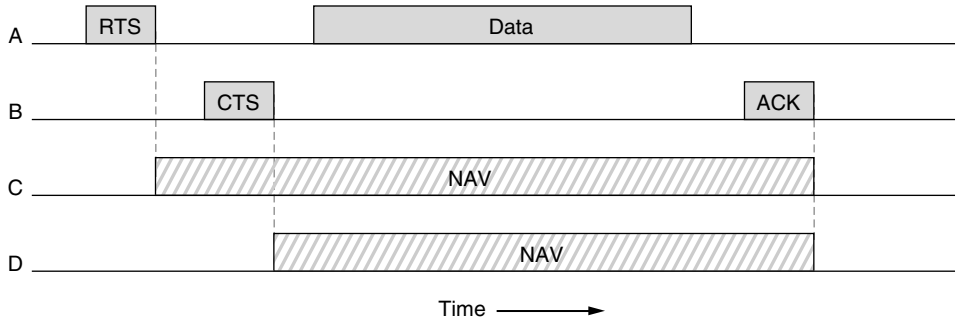


Figure 4-27. Virtual channel sensing using CSMA/CA.

Now let us consider this exchange from the viewpoints of *C* and *D*. *C* is within range of *A*, so it may receive the RTS frame. If it does, it realizes that someone is going to send data soon. From the information provided in the RTS request, it can estimate how long the sequence will take, including the final ACK. So, for the good of all, it desists from transmitting anything until the exchange is completed. It does so by updating its record of the NAV to indicate that the channel is busy, as shown in Fig. 4-27. *D* does not hear the RTS, but it does hear the CTS, so it also updates its NAV. Note that the NAV signals are not transmitted; they are just internal reminders to keep quiet for a certain period of time.

However, while RTS/CTS sounds good in theory, it is one of those designs that has proved to be of little value in practice. Several reasons why it is seldom used are known. It does not help for short frames (which are sent in place of the RTS) or for the AP (which everyone can hear, by definition). For other situations, it only slows down operation. RTS/CTS in 802.11 is a little different than in the MACA protocol we saw in Sec 4.2 because everyone hearing the RTS or CTS remains quiet for the duration to allow the ACK to get through without collision. Because of this, it does not help with exposed terminals as MACA did, only with hidden terminals. Most often there are few hidden terminals, and CSMA/CA already helps them by slowing down stations that transmit unsuccessfully, whatever the cause, to make it more likely that transmissions will succeed.

CSMA/CA with physical and virtual sensing is the core of the 802.11 protocol. However, there are several other mechanisms that have been developed to go with it. Each of these mechanisms was driven by the needs of real operation, so we will look at them briefly.

The first need we will look at is reliability. In contrast to wired networks, wireless networks are noisy and unreliable, in no small part due to interference from other kinds of devices, such as microwave ovens, which also use the unlicensed ISM bands. The use of acknowledgements and retransmissions is of little help if the probability of getting a frame through is small in the first place.

The main strategy that is used to increase successful transmissions is to lower the transmission rate. Slower rates use more robust modulations that are more likely to be received correctly for a given signal-to-noise ratio. If too many frames are lost, a station can lower the rate. If frames are delivered with little loss, a station can occasionally test a higher rate to see if it should be used.

Another strategy to improve the chance of the frame getting through undamaged is to send shorter frames. If the probability of any bit being in error is p, the probability of an n-bit frame being received entirely correctly is $(1 - p)^n$. For example, for $p = 10^{-4}$, the probability of receiving a full Ethernet frame (12,144 bits) correctly is less than 30%. Most frames will be lost. But if the frames are only a third as long (4048 bits) two thirds of them will be received correctly. Now most frames will get through and fewer retransmissions will be needed.

Shorter frames can be implemented by reducing the maximum size of the message that is accepted from the network layer. Alternatively, 802.11 allows frames to be split into smaller pieces, called **fragments**, each with its own checksum. The fragment size is not fixed by the standard, but is a parameter that can be adjusted by the AP. The fragments are individually numbered and acknowledged using a stop-and-wait protocol (i.e., the sender may not transmit fragment $k + 1$ until it has received the acknowledgement for fragment k). Once the channel has been acquired, multiple fragments are sent as a burst. They go one after the other with an acknowledgement (and possibly retransmissions) in between, until either the whole frame has been successfully sent or the transmission time reaches the maximum allowed. The NAV mechanism keeps other stations quiet only until the next acknowledgement, but another mechanism (see below) is used to allow a burst of fragments to be sent without other stations sending a frame in the middle.

The second need we will discuss is saving power. Battery life is always an issue with mobile wireless devices. The 802.11 standard pays attention to the issue of power management so that clients need not waste power when they have neither information to send nor to receive.

The basic mechanism for saving power builds on **beacon frames**. Beacons are periodic broadcasts by the AP (e.g., every 100 msec). The frames advertise the presence of the AP to clients and carry system parameters, such as the identifier of the AP, the time, how long until the next beacon, and security settings.

Clients can set a power-management bit in frames that they send to the AP to tell it that they are entering **power-save mode**. In this mode, the client can doze and the AP will buffer traffic intended for it. To check for incoming traffic, the client wakes up for every beacon, and checks a traffic map that is sent as part of the beacon. This map tells the client if there is buffered traffic. If so, the client sends a poll message to the AP, which then sends the buffered traffic. The client can then go back to sleep until the next beacon is sent.

Another power-saving mechanism, called **APSD** (**Automatic Power Save Delivery**), was also added to 802.11 in 2005. With this new mechanism, the AP buffers frames and sends them to a client just after the client sends frames to the

AP. The client can then go to sleep until it has more traffic to send (and receive). This mechanism works well for applications such as VoIP that have frequent traffic in both directions. For example, a VoIP wireless phone might use it to send and receive frames every 20 msec, much more frequently than the beacon interval of 100 msec, while dozing in between.

The third and last need we will examine is quality of service. When the VoIP traffic in the preceding example competes with peer-to-peer traffic, the VoIP traffic will suffer. It will be delayed due to contention with the high-bandwidth peer-to-peer traffic, even though the VoIP bandwidth is low. These delays are likely to degrade the voice calls. To prevent this degradation, we would like to let the VoIP traffic go ahead of the peer-to-peer traffic, as it is of higher priority.

IEEE 802.11 has a clever mechanism to provide this kind of quality of service that was introduced as set of extensions under the name 802.11e in 2005. It works by extending CSMA/CA with carefully defined intervals between frames. After a frame has been sent, a certain amount of idle time is required before any station may send a frame to check that the channel is no longer in use. The trick is to define different time intervals for different kinds of frames.

Five intervals are depicted in Fig. 4-28. The interval between regular data frames is called the **DIFS** (**DCF InterFrame Spacing**). Any station may attempt to acquire the channel to send a new frame after the medium has been idle for DIFS. The usual contention rules apply, and binary exponential backoff may be needed if a collision occurs. The shortest interval is **SIFS** (**Short InterFrame Spacing**). It is used to allow the parties in a single dialog the chance to go first. Examples include letting the receiver send an ACK, other control frame sequences like RTS and CTS, or letting a sender transmit a burst of fragments. Sending the next fragment after waiting only SIFS is what prevents another station from jumping in with a frame in the middle of the exchange.

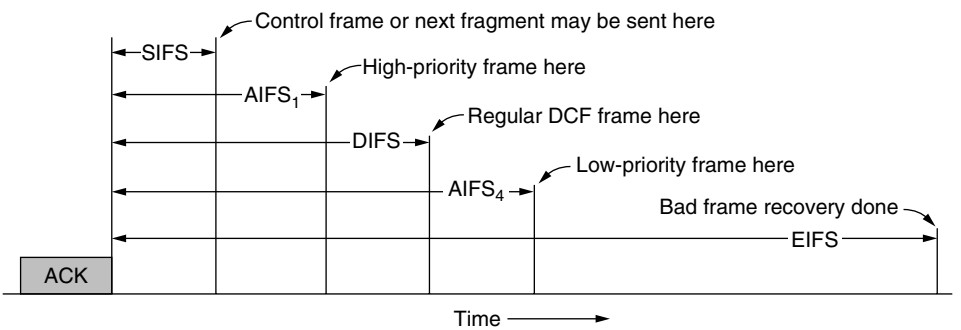


Figure 4-28. Interframe spacing in 802.11.

The two **AIFS** (**Arbitration InterFrame Space**) intervals show examples of two different priority levels. The short interval, $AIFS_1$, is smaller than DIFS but longer than SIFS. It can be used by the AP to move voice or other high-priority

traffic to the head of the line. The AP will wait for a shorter interval before it sends the voice traffic, and thus send it before regular traffic. The long interval, $AIFS_4$, is larger than DIFS. It is used for background traffic that can be deferred until after regular traffic. The AP will wait for a longer interval before it sends this traffic, giving regular traffic the opportunity to transmit first. The complete quality of service mechanism defines four different priority levels that have different backoff parameters as well as different idle parameters.

The last time interval, **EIFS** (**Extended InterFrame Spacing**), is used only by a station that has just received a bad or unknown frame, to report the problem. The idea is that since the receiver may have no idea of what is going on, it should wait a while to avoid interfering with an ongoing dialog between two stations.

A further part of the quality of service extensions is the notion of a **TXOP** or **transmission opportunity**. The original CSMA/CA mechanism let stations send one frame at a time. This design was fine until the range of rates increased. With 802.11a/g, one station might be sending at 6 Mbps and another station be sending at 54 Mbps. They each get to send one frame, but the 6-Mbps station takes nine times as long (ignoring fixed overheads) as the 54-Mbps station to send its frame. This disparity has the unfortunate side effect of slowing down a fast sender who is competing with a slow sender to roughly the rate of the slow sender. For example, again ignoring fixed overheads, when sending alone the 6-Mbps and 54-Mbps senders will get their own rates, but when sending together they will both get 5.4 Mbps on average. It is a stiff penalty for the fast sender. This issue is known as the **rate anomaly** (Heusse et al., 2003).

With transmission opportunities, each station gets an equal amount of airtime, not an equal number of frames. Stations that send at a higher rate for their airtime will get higher throughput. In our example, when sending together the 6-Mbps and 54-Mbps senders will now get 3 Mbps and 27 Mbps, respectively.

4.4.4 The 802.11 Frame Structure

The 802.11 standard defines three different classes of frames in the air: data, control, and management. Each of these has a header with a variety of fields used within the MAC sublayer. In addition, there are some headers used by the physical layer, but these mostly deal with the modulation techniques used, so we will not discuss them here.

We will look at the format of the data frame as an example. It is shown in Fig. 4-29. First comes the *Frame control* field, which is made up of 11 subfields. The first of these is the *Protocol version*, set to 00. It is there to allow future versions of 802.11 to operate at the same time in the same cell. Then come the *Type* (data, control, or management) and *Subtype* fields (e.g., RTS or CTS). For a regular data frame (without quality of service), they are set to 10 and 0000 in binary. The *To DS* and *From DS* bits are set to indicate whether the frame is going to or coming from the network connected to the APs, which is called the distribution

system. The *More fragments* bit means that more fragments will follow. The *Retry* bit marks a retransmission of a frame sent earlier. The *Power management* bit indicates that the sender is going into power-save mode. The *More data* bit indicates that the sender has additional frames for the receiver. The *Protected Frame* bit indicates that the frame body has been encrypted for security. We will discuss security briefly in the next section. Finally, the *Order* bit tells the receiver that the higher layer expects the sequence of frames to arrive strictly in order.

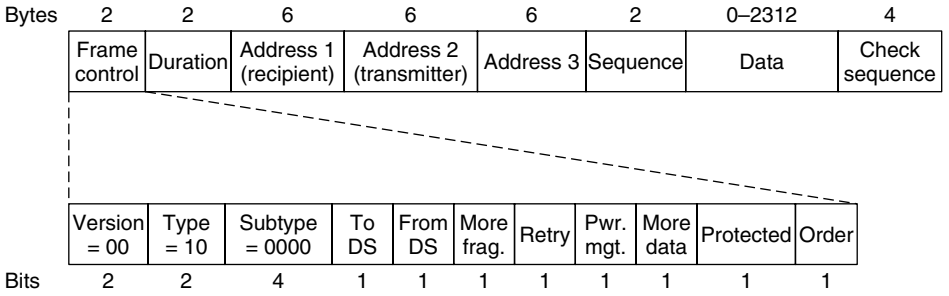


Figure 4-29. Format of the 802.11 data frame.

The second field of the data frame, the *Duration* field, tells how long the frame and its acknowledgement will occupy the channel, measured in microseconds. It is present in all types of frames, including control frames, and is what stations use to manage the NAV mechanism.

Next come addresses. Data frames sent to or from an AP have three addresses, all in standard IEEE 802 format. The first address is the receiver, and the second address is the transmitter. They are obviously needed, but what is the third address for? Remember that the AP is simply a relay point for frames as they travel between a client and another point on the network, perhaps a distant client or a portal to the Internet. The third address gives this distant endpoint.

The *Sequence* field numbers frames so that duplicates can be detected. Of the 16 bits available, 4 identify the fragment and 12 carry a number that is advanced with each new transmission. The *Data* field contains the payload, up to 2312 bytes. The first bytes of this payload are in a format known as **LLC** (**Logical Link Control**). This layer is the glue that identifies the higher-layer protocol (e.g., IP) to which the payloads should be passed. Last comes the *Frame check sequence*, which is the same 32-bit CRC we saw in Sec. 3.2.2 and elsewhere.

Management frames have the same format as data frames, plus a format for the data portion that varies with the subtype (e.g., parameters in beacon frames). Control frames are short. Like all frames, they have the *Frame control*, *Duration*, and *Frame check sequence* fields. However, they may have only one address and no data portion. Most of the key information is conveyed with the *Subtype* field (e.g., ACK, RTS and CTS).

4.4.5 Services

The 802.11 standard defines the services that the clients, the access points, and the network connecting them must be a conformant wireless LAN. These services cluster into several groups.

The **association** service is used by mobile stations to connect themselves to APs. Typically, it is used just after a station moves within radio range of the AP. Upon arrival, the station learns the identity and capabilities of the AP, either from beacon frames or by directly asking the AP. The capabilities include the data rates supported, security arrangements, power-saving capabilities, quality of service support, and more. The station sends a request to associate with the AP. The AP may accept or reject the request.

Reassociation lets a station change its preferred AP. This facility is useful for mobile stations moving from one AP to another AP in the same extended 802.11 LAN, like a handover in the cellular network. If it is used correctly, no data will be lost as a consequence of the handover. (But 802.11, like Ethernet, is just a best-effort service.) Either the station or the AP may also **disassociate**, breaking their relationship. A station should use this service before shutting down or leaving the network. The AP may use it before going down for maintenance.

Stations must also **authenticate** before they can send frames via the AP, but authentication is handled in different ways depending on the choice of security scheme. If the 802.11 network is "open," anyone is allowed to use it. Otherwise, credentials are needed to authenticate. The recommended scheme, called **WPA2** (**WiFi Protected Access 2**), implements security as defined in the 802.11i standard. (Plain WPA is an interim scheme that implements a subset of 802.11i. We will skip it and go straight to the complete scheme.) With WPA2, the AP can talk to an authentication server that has a username and password database to determine if the station is allowed to access the network. Alternatively a pre-shared key, which is a fancy name for a network password, may be configured. Several frames are exchanged between the station and the AP with a challenge and response that lets the station prove it has the right credentials. This exchange happens after association.

The scheme that was used before WPA is called **WEP** (**Wired Equivalent Privacy**). For this scheme, authentication with a preshared key happens before association. However, its use is discouraged because of design flaws that make WEP easy to compromise. The first practical demonstration that WEP was broken came when Adam Stubblefield was a summer intern at AT&T (Stubblefield et al., 2002). He was able to code up and test an attack in one week, much of which was spent getting permission from management to buy the WiFi cards needed for experiments. Software to crack WEP passwords is now freely available.

Once frames reach the AP, the **distribution** service determines how to route them. If the destination is local to the AP, the frames can be sent out directly over the air. Otherwise, they will have to be forwarded over the wired network. The

integration service handles any translation that is needed for a frame to be sent outside the 802.11 LAN, or to arrive from outside the 802.11 LAN. The common case here is connecting the wireless LAN to the Internet.

Data transmission is what it is all about, so 802.11 naturally provides a **data delivery** service. This service lets stations transmit and receive data using the protocols we described earlier in this chapter. Since 802.11 is modeled on Ethernet and transmission over Ethernet is not guaranteed to be 100% reliable, transmission over 802.11 is not guaranteed to be reliable either. Higher layers must deal with detecting and correcting errors.

Wireless is a broadcast signal. For information sent over a wireless LAN to be kept confidential, it must be encrypted. This goal is accomplished with a **privacy** service that manages the details of encryption and decryption. The encryption algorithm for WPA2 is based on **AES** (**Advanced Encryption Standard**), a U.S. government standard approved in 2002. The keys that are used for encryption are determined during the authentication procedure.

To handle traffic with different priorities, there is a **QOS traffic scheduling** service. It uses the protocols we described to give voice and video traffic preferential treatment compared to best-effort and background traffic. A companion service also provides higher-layer timer synchronization. This lets stations coordinate their actions, which may be useful for media processing.

Finally, there are two services that help stations manage their use of the spectrum. The **transmit power control** service gives stations the information they need to meet regulatory limits on transmit power that vary from region to region. The **dynamic frequency selection** service give stations the information they need to avoid transmitting on frequencies in the 5-GHz band that are being used for radar in the proximity.

With these services, 802.11 provides a rich set of functionality for connecting nearby mobile clients to the Internet. It has been a huge success, and the standard has repeatedly been amended to add more functionality. For a perspective on where the standard has been and where it is heading, see Hiertz et al. (2010).

4.5 BROADBAND WIRELESS

We have been indoors too long. Let us go outdoors, where there is quite a bit of interesting networking over the so-called "last mile." With the deregulation of the telephone systems in many countries, competitors to the entrenched telephone companies are now often allowed to offer local voice and high-speed Internet service. There is certainly plenty of demand. The problem is that running fiber or coax to millions of homes and businesses is prohibitively expensive. What is a competitor to do?

The answer is broadband wireless. Erecting a big antenna on a hill just outside of town is much easier and cheaper than digging many trenches and stringing

cables. Thus, companies have begun to experiment with providing multimegabit wireless communication services for voice, Internet, movies on demand, etc.

To stimulate the market, IEEE formed a group to standardize a broadband wireless metropolitan area network. The next number available in the 802 numbering space was **802.16**, so the standard got this number. Informally the technology is called **WiMAX** (**Worldwide Interoperability for Microwave Access**). We will use the terms 802.16 and WiMAX interchangeably.

The first 802.16 standard was approved in December 2001. Early versions provided a wireless local loop between fixed points with a line of sight to each other. This design soon changed to make WiMAX a more competitive alternative to cable and DSL for Internet access. By January 2003, 802.16 had been revised to support non-line-of-sight links by using OFDM technology at frequencies between 2 GHz and 10 GHz. This change made deployment much easier, though stations were still fixed locations. The rise of 3G cellular networks posed a threat by promising high data rates *and* mobility. In response, 802.16 was enhanced again to allow mobility at vehicular speeds by December 2005. Mobile broadband Internet access is the target of the current standard, IEEE 802.16-2009.

Like the other 802 standards, 802.16 was heavily influenced by the OSI model, including the (sub)layers, terminology, service primitives, and more. Unfortunately, also like OSI, it is fairly complicated. In fact, the **WiMAX Forum** was created to define interoperable subsets of the standard for commercial offerings. In the following sections, we will give a brief description of some of the highlights of the common forms of 802.16 air interface, but this treatment is far from complete and leaves out many details. For additional information about WiMAX and broadband wireless in general, see Andrews et al. (2007).

4.5.1 Comparison of 802.16 with 802.11 and 3G

At this point you may be thinking: why devise a new standard? Why not just use 802.11 or 3G? In fact, WiMAX combines aspects of both 802.11 and 3G, making it more like a 4G technology.

Like 802.11, WiMAX is all about wirelessly connecting devices to the Internet at megabit/sec speeds, instead of using cable or DSL. The devices may be mobile, or at least portable. WiMAX did not start by adding low-rate data on the side of voice-like cellular networks; 802.16 was designed to carry IP packets over the air and to connect to an IP-based wired network with a minimum of fuss. The packets may carry peer-to-peer traffic, VoIP calls, or streaming media to support a range of applications. Also like 802.11, it is based on OFDM technology to ensure good performance in spite of wireless signal degradations such as multipath fading, and on MIMO technology to achieve high levels of throughput.

However, WiMAX is more like 3G (and thus unlike 802.11) in several key respects. The key technical problem is to achieve high capacity by the efficient use of spectrum, so that a large number of subscribers in a coverage area can all get

high throughput. The typical distances are at least 10 times larger than for an 802.11 network. Consequently, WiMAX base stations are more powerful than 802.11 Access Points (APs). To handle weaker signals over larger distances, the base station uses more power and better antennas, and it performs more processing to handle errors. To maximize throughput, transmissions are carefully scheduled by the base station for each particular subscriber; spectrum use is not left to chance with CSMA/CA, which may waste capacity with collisions.

Licensed spectrum is the expected case for WiMAX, typically around 2.5 GHz in the U.S. The whole system is substantially more optimized than 802.11. This complexity is worth it, considering the large amount of money involved for licensed spectrum. Unlike 802.11, the result is a managed and reliable service with good support for quality of service.

With all of these features, 802.16 most closely resembles the 4G cellular networks that are now being standardized under the name **LTE** (**Long Term Evolution**). While 3G cellular networks are based on CDMA and support voice and data, 4G cellular networks will be based on OFDM with MIMO, and they will target data, with voice as just one application. It looks like WiMAX and 4G are on a collision course in terms of technology and applications. Perhaps this convergence is unsurprising, given that the Internet is the killer application and OFDM and MIMO are the best-known technologies for efficiently using the spectrum.

4.5.2 The 802.16 Architecture and Protocol Stack

The 802.16 architecture is shown in Fig. 4-30. Base stations connect directly to the provider's backbone network, which is in turn connected to the Internet. The base stations communicate with stations over the wireless air interface. Two kinds of stations exist. Subscriber stations remain in a fixed location, for example, broadband Internet access for homes. Mobile stations can receive service while they are moving, for example, a car equipped with WiMAX.

The 802.16 protocol stack that is used across the air interface is shown in Fig. 4-31. The general structure is similar to that of the other 802 networks, but with more sublayers. The bottom layer deals with transmission, and here we have shown only the popular offerings of 802.16, fixed and mobile WiMAX. There is a different physical layer for each offering. Both layers operate in licensed spectrum below 11 GHz and use OFDM, but in different ways.

Above the physical layer, the data link layer consists of three sublayers. The bottom one deals with privacy and security, which is far more crucial for public outdoor networks than for private indoor networks. It manages encryption, decryption, and key management.

Next comes the MAC common sublayer part. This part is where the main protocols, such as channel management, are located. The model here is that the base station completely controls the system. It can schedule the downlink (i.e., base to subscriber) channels very efficiently and plays a major role in managing

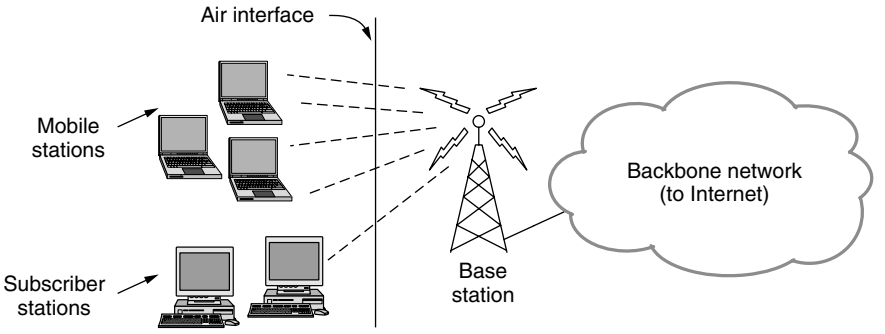


Figure 4-30. The 802.16 architecture.

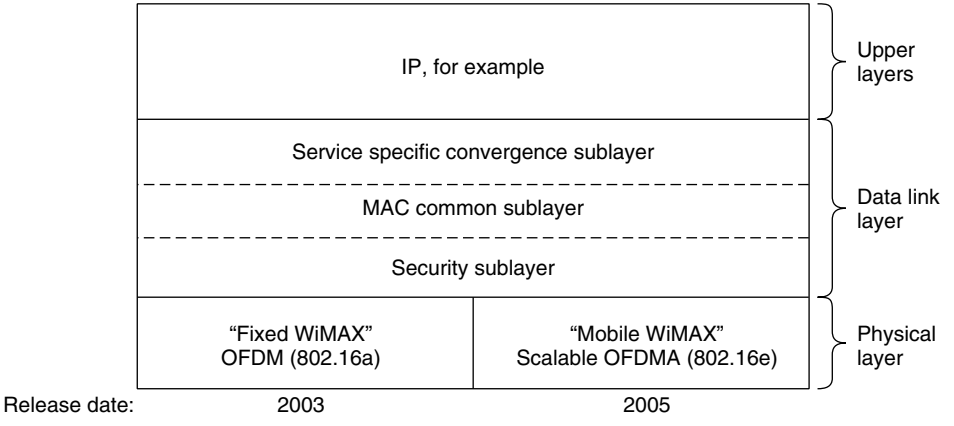


Figure 4-31. The 802.16 protocol stack.

the uplink (i.e., subscriber to base) channels as well. An unusual feature of this MAC sublayer is that, unlike those of the other 802 protocols, it is completely connection oriented, in order to provide quality of service guarantees for telephony and multimedia communication.

The service-specific convergence sublayer takes the place of the logical link sublayer in the other 802 protocols. Its function is to provide an interface to the network layer. Different convergence layers are defined to integrate seamlessly with different upper layers. The important choice is IP, though the standard defines mappings for protocols such as Ethernet and ATM too. Since IP is connectionless and the 802.16 MAC sublayer is connection-oriented, this layer must map between addresses and connections.

Patent Owner's Ex. 2010
Page 163

4.5.3 The 802.16 Physical Layer

Most WiMAX deployments use licensed spectrum around either 3.5 GHz or 2.5 GHz. As with 3G, finding available spectrum is a key problem. To help, the 802.16 standard is designed for flexibility. It allows operation from 2 GHz to 11 GHz. Channels of different sizes are supported, for example, 3.5 MHz for fixed WiMAX and from 1.25 MHz to 20 MHz for mobile WiMAX.

Transmissions are sent over these channels with OFDM, the technique we described in Sec. 2.5.3. Compared to 802.11, the 802.16 OFDM design is optimized to make the most out of licensed spectrum and wide area transmissions. The channel is divided into more subcarriers with a longer symbol duration to tolerate larger wireless signal degradations; WiMAX parameters are around 20 times larger than comparable 802.11 parameters. For example, in mobile WiMAX there are 512 subcarriers for a 5-MHz channel and the time to send a symbol on each subcarrier is roughly 100 μsec.

Symbols on each subcarrier are sent with QPSK, QAM-16, or QAM-64, modulation schemes we described in Sec. 2.5.3. When the mobile or subscriber station is near the base station and the received signal has a high signal-to-noise ratio (SNR), QAM-64 can be used to send 6 bits per symbol. To reach distant stations with a low SNR, QPSK can be used to deliver 2 bits per symbol. The data is first coded for error correction with the convolutional coding (or better schemes) that we described in Sec. 3.2.1. This coding is common on noisy channels to tolerate some bit errors without needing to send retransmissions. In fact, the modulation and coding methods should sound familiar by now as they are used for many networks we have studied, including 802.11 cable, and DSL. The net result is that a base station can support up to 12.6 Mbps of downlink traffic and 6.2 Mbps of uplink traffic per 5-MHz channel and pair of antennas.

One thing the designers of 802.16 did not like was a certain aspect of the way GSM and DAMPS work. Both of those systems use equal frequency bands for upstream and downstream traffic. That is, they implicitly assume there is as much upstream traffic as downstream traffic. For voice, traffic is symmetric for the most part, but for Internet access (and certainly Web surfing) there is often more downstream traffic than upstream traffic. The ratio is often 2:1, 3:1, or more:1.

So, the designers chose a flexible scheme for dividing the channel between stations, called **OFDMA** (**Orthogonal Frequency Division Multiple Access**). With OFDMA, different sets of subcarriers can be assigned to different stations, so that more than one station can send or receive at once. If this were 802.11, all subcarriers would be used by one station to send at any given moment. The added flexibility in how bandwidth is assigned can increase performance because a given subcarrier might be faded at one receiver due to multipath effects but clear at another. Subcarriers can be assigned to the stations that can use them best.

As well as having asymmetric traffic, stations usually alternate between sending and receiving. This method is called **TDD** (**Time Division Duplex**). The

alternative method, in which a station sends and receives at the same time (on different subcarrier frequencies), is called **FDD** (**Frequency Division Duplex**). WiMAX allows both methods, but TDD is preferred because it is easier to implement and more flexible.

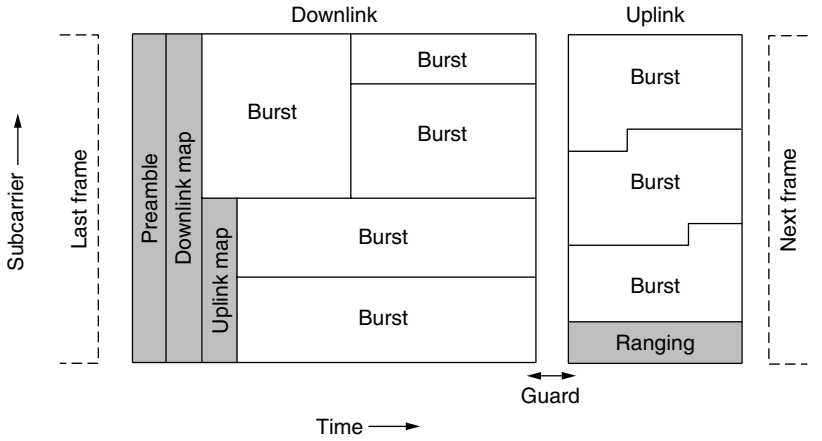


Figure 4-32. Frame structure for OFDMA with time division duplexing.

Fig. 4-32 shows an example of the frame structure that is repeated over time. It starts with a preamble to synchronize all stations, followed by downlink transmissions from the base station. First, the base station sends maps that tell all stations how the downlink and uplink subcarriers are assigned over the frame. The base station controls the maps, so it can allocate different amounts of bandwidth to stations from frame to frame depending on the needs of each station.

Next, the base station sends bursts of traffic to different subscriber and mobile stations on the subcarriers at the times given in the map. The downlink transmissions end with a guard time for stations to switch from receiving to transmitting. Finally, the subscriber and mobile stations send their bursts of traffic to the base station in the uplink positions that were reserved for them in the map. One of these uplink bursts is reserved for **ranging**, which is the process by which new stations adjust their timing and request initial bandwidth to connect to the base station. Since no connection is set up at this stage, new stations just transmit and hope there is no collision.

4.5.4 The 802.16 MAC Sublayer Protocol

The data link layer is divided into three sublayers, as we saw in Fig. 4-31. Since we will not study cryptography until Chap. 8, it is difficult to explain now how the security sublayer works. Suffice it to say that encryption is used to keep secret all data transmitted. Only the frame payloads are encrypted; the headers

are not. This property means that a snooper can see who is talking to whom but cannot tell what they are saying to each other.

If you already know something about cryptography, what follows is a one-paragraph explanation of the security sublayer. If you know nothing about cryptography, you are not likely to find the next paragraph terribly enlightening (but you might consider rereading it after finishing Chap. 8).

When a subscriber connects to a base station, they perform mutual authentication with RSA public-key cryptography using X.509 certificates. The payloads themselves are encrypted using a symmetric-key system, either AES (Rijndael) or DES with cipher block chaining. Integrity checking uses SHA-1. Now that was not so bad, was it?

Let us now look at the MAC common sublayer part. The MAC sublayer is connection-oriented and point-to-multipoint, which means that one base station communicates with multiple subscriber stations. Much of this design is borrowed from cable modems, in which one cable headend controls the transmissions of multiple cable modems at the customer premises.

The downlink direction is fairly straightforward. The base station controls the physical-layer bursts that are used to send information to the different subscriber stations. The MAC sublayer simply packs its frames into this structure. To reduce overhead, there are several different options. For example, MAC frames may be sent individually, or packed back-to-back into a group.

The uplink channel is more complicated since there are competing subscribers that need access to it. Its allocation is tied closely to the quality of service issue. Four classes of service are defined, as follows:

1. Constant bit rate service.
2. Real-time variable bit rate service.
3. Non-real-time variable bit rate service.
4. Best-effort service.

All service in 802.16 is connection-oriented. Each connection gets one of these service classes, determined when the connection is set up. This design is different from that of 802.11 or Ethernet, which are connectionless in the MAC sublayer.

Constant bit rate service is intended for transmitting uncompressed voice. This service needs to send a predetermined amount of data at predetermined time intervals. It is accommodated by dedicating certain bursts to each connection of this type. Once the bandwidth has been allocated, the bursts are available automatically, without the need to ask for each one.

Real-time variable bit rate service is for compressed multimedia and other soft real-time applications in which the amount of bandwidth needed at each instant may vary. It is accommodated by the base station polling the subscriber at a fixed interval to ask how much bandwidth is needed this time.

Non-real-time variable bit rate service is for heavy transmissions that are not real time, such as large file transfers. For this service, the base station polls the subscriber often, but not at rigidly prescribed time intervals. Connections with this service can also use best-effort service, described next, to request bandwidth.

Best-effort service is for everything else. No polling is done and the subscriber must contend for bandwidth with other best-effort subscribers. Requests for bandwidth are sent in bursts marked in the uplink map as available for contention. If a request is successful, its success will be noted in the next downlink map. If it is not successful, the unsuccessful subscriber have to try again later. To minimize collisions, the Ethernet binary exponential backoff algorithm is used.

4.5.5 The 802.16 Frame Structure

All MAC frames begin with a generic header. The header is followed by an optional payload and an optional checksum (CRC), as illustrated in Fig. 4-33. The payload is not needed in control frames, for example, those requesting channel slots. The checksum is (surprisingly) also optional, due to the error correction in the physical layer and the fact that no attempt is ever made to retransmit real-time frames. If no retransmissions will be attempted, why even bother with a checksum? But if there is a checksum, it is the standard IEEE 802 CRC, and acknowledgements and retransmissions are used for reliability.

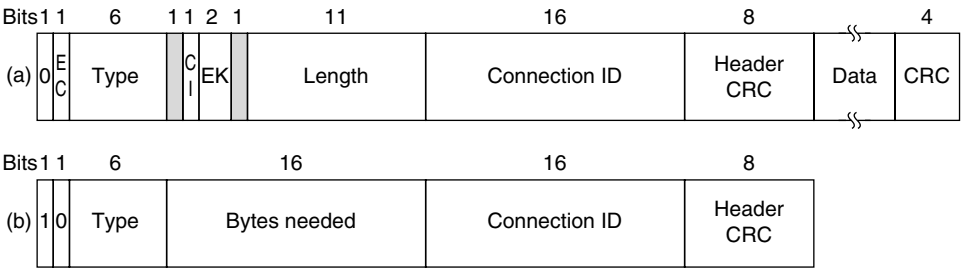


Figure 4-33. (a) A generic frame. (b) A bandwidth request frame.

A quick rundown of the header fields of Fig. 4-33(a) follows. The *EC* bit tells whether the payload is encrypted. The *Type* field identifies the frame type, mostly telling whether packing and fragmentation are present. The *CI* field indicates the presence or absence of the final checksum. The *EK* field tells which of the encryption keys is being used (if any). The *Length* field gives the complete length of the frame, including the header. The *Connection identifier* tells which connection this frame belongs to. Finally, the *Header CRC* field is a checksum over the header only, using the polynomial $x^8 + x^2 + x + 1$.

The 802.16 protocol has many kinds of frames. An example of a different type of frame, one that is used to request bandwidth, is shown in Fig. 4-33(b). It

starts with a 1 bit instead of a 0 bit and is otherwise similar to the generic header except that the second and third bytes form a 16-bit number telling how much bandwidth is needed to carry the specified number of bytes. Bandwidth request frames do not carry a payload or full-frame CRC.

A great deal more could be said about 802.16, but this is not the place to say it. For more information, please consult the IEEE 802.16-2009 standard itself.

4.6 BLUETOOTH

In 1994, the L. M. Ericsson company became interested in connecting its mobile phones to other devices (e.g., laptops) without cables. Together with four other companies (IBM, Intel, Nokia, and Toshiba), it formed a SIG (Special Interest Group, i.e., consortium) in 1998 to develop a wireless standard for interconnecting computing and communication devices and accessories using short-range, low-power, inexpensive wireless radios. The project was named **Bluetooth**, after Harald Blaatand (Bluetooth) II (940–981), a Viking king who unified (i.e., conquered) Denmark and Norway, also without cables.

Bluetooth 1.0 was released in July 1999, and since then the SIG has never looked back. All manner of consumer electronic devices now use Bluetooth, from mobile phones and laptops to headsets, printers, keyboards, mice, gameboxes, watches, music players, navigation units, and more. The Bluetooth protocols let these devices find and connect to each other, an act called **pairing**, and securely transfer data.

The protocols have evolved over the past decade, too. After the initial protocols stabilized, higher data rates were added to Bluetooth 2.0 in 2004. With the 3.0 release in 2009, Bluetooth can be used for device pairing in combination with 802.11 for high-throughput data transfer. The 4.0 release in December 2009 specified low-power operation. That will be handy for people who do not want to change the batteries regularly in all of those devices around the house. We will cover the main aspects of Bluetooth below.

4.6.1 Bluetooth Architecture

Let us start our study of the Bluetooth system with a quick overview of what it contains and what it is intended to do. The basic unit of a Bluetooth system is a **piconet**, which consists of a master node and up to seven active slave nodes within a distance of 10 meters. Multiple piconets can exist in the same (large) room and can even be connected via a bridge node that takes part in multiple piconets, as in Fig. 4-34. An interconnected collection of piconets is called a **scatternet**.

In addition to the seven active slave nodes in a piconet, there can be up to 255 parked nodes in the net. These are devices that the master has switched to a low-power state to reduce the drain on their batteries. In parked state, a device cannot

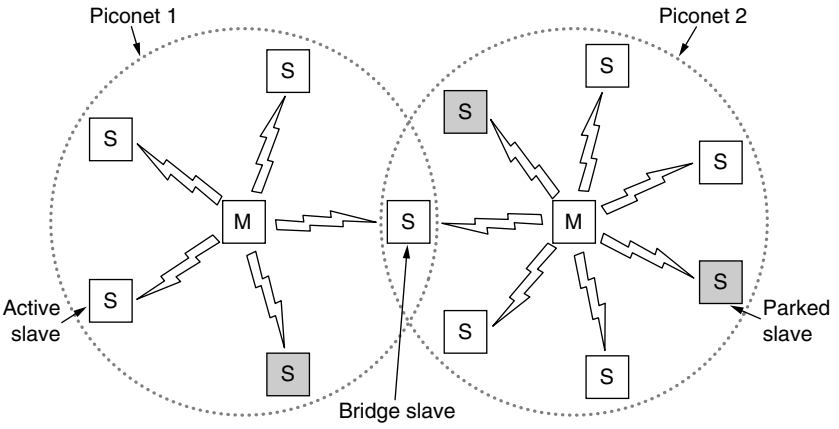


Figure 4-34. Two piconets can be connected to form a scatternet.

do anything except respond to an activation or beacon signal from the master. Two intermediate power states, hold and sniff, also exist, but these will not concern us here.

The reason for the master/slave design is that the designers intended to facilitate the implementation of complete Bluetooth chips for under $5. The consequence of this decision is that the slaves are fairly dumb, basically just doing whatever the master tells them to do. At its heart, a piconet is a centralized TDM system, with the master controlling the clock and determining which device gets to communicate in which time slot. All communication is between the master and a slave; direct slave-slave communication is not possible.

4.6.2 Bluetooth Applications

Most network protocols just provide channels between communicating entities and let application designers figure out what they want to use them for. For example, 802.11 does not specify whether users should use their notebook computers for reading email, surfing the Web, or something else. In contrast, the Bluetooth SIG specifies particular applications to be supported and provides different protocol stacks for each one. At the time of writing, there are 25 applications, which are called **profiles**. Unfortunately, this approach leads to a very large amount of complexity. We will omit the complexity here but will briefly look at the profiles to see more clearly what the Bluetooth SIG is trying to accomplish.

Six of the profiles are for different uses of audio and video. For example, the intercom profile allows two telephones to connect as walkie-talkies. The headset and hands-free profiles both provide voice communication between a headset and its base station, as might be used for hands-free telephony while driving a car.

Other profiles are for streaming stereo-quality audio and video, say, from a portable music player to headphones, or from a digital camera to a TV.

The human interface device profile is for connecting keyboards and mice to computers. Other profiles let a mobile phone or other computer receive images from a camera or send images to a printer. Perhaps of more interest is a profile to use a mobile phone as a remote control for a (Bluetooth-enabled) TV.

Still other profiles enable networking. The personal area network profile lets Bluetooth devices form an ad hoc network or remotely access another network, such as an 802.11 LAN, via an access point. The dial-up networking profile was actually the original motivation for the whole project. It allows a notebook computer to connect to a mobile phone containing a built-in modem without using wires.

Profiles for higher-layer information exchange have also been defined. The synchronization profile is intended for loading data into a mobile phone when it leaves home and collecting data from it when it returns.

We will skip the rest of the profiles, except to mention that some profiles serve as building blocks on which the above profiles are built. The generic access profile, on which all of the other profiles are built, provides a way to establish and maintain secure links (channels) between the master and the slaves. The other generic profiles define the basics of object exchange and audio and video transport. Utility profiles are used widely for functions such as emulating a serial line, which is especially useful for many legacy applications.

Was it really necessary to spell out all these applications in detail and provide different protocol stacks for each one? Probably not, but there were a number of different working groups that devised different parts of the standard, and each one just focused on its specific problem and generated its own profile. Think of this as Conway's Law in action. (In the April 1968 issue of *Datamation* magazine, Melvin Conway observed that if you assign *n* people to write a compiler, you will get an *n*-pass compiler, or more generally, the software structure mirrors the structure of the group that produced it.) It would probably have been possible to get away with two protocol stacks instead of 25, one for file transfer and one for streaming real-time communication.

4.6.3 The Bluetooth Protocol Stack

The Bluetooth standard has many protocols grouped loosely into the layers shown in Fig. 4-35. The first observation to make is that the structure does not follow the OSI model, the TCP/IP model, the 802 model, or any other model.

The bottom layer is the physical radio layer, which corresponds fairly well to the physical layer in the OSI and 802 models. It deals with radio transmission and modulation. Many of the concerns here have to do with the goal of making the system inexpensive so that it can become a mass-market item.

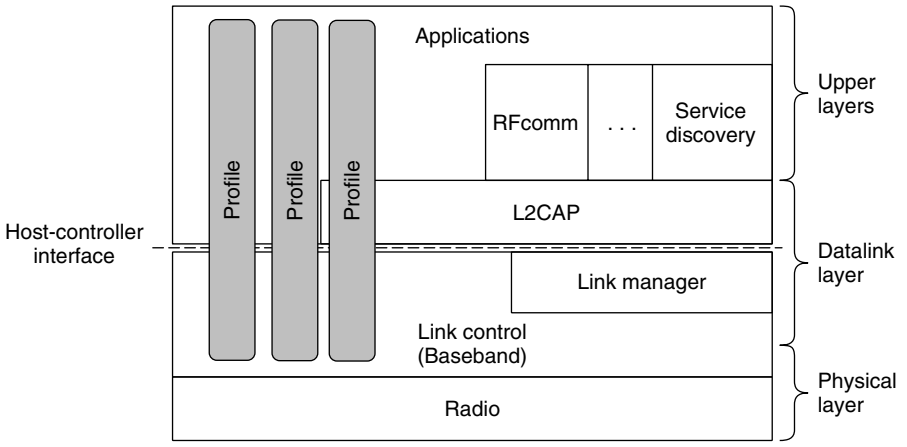


Figure 4-35. The Bluetooth protocol architecture.

The link control (or baseband) layer is somewhat analogous to the MAC sublayer but also includes elements of the physical layer. It deals with how the master controls time slots and how these slots are grouped into frames.

Next come two protocols that use the link control protocol. The link manager handles the establishment of logical channels between devices, including power management, pairing and encryption, and quality of service. It lies below the host controller interface line. This interface is a convenience for implementation: typically, the protocols below the line will be implemented on a Bluetooth chip, and the protocols above the line will be implemented on the Bluetooth device that hosts the chip.

The link protocol above the line is **L2CAP** (**Logical Link Control Adaptation Protocol**). It frames variable-length messages and provides reliability if needed. Many protocols use L2CAP, such as the two utility protocols that are shown. The service discovery protocol is used to locate services within the network. The RFcomm (Radio Frequency communication) protocol emulates the standard serial port found on PCs for connecting the keyboard, mouse, and modem, among other devices.

The top layer is where the applications are located. The profiles are represented by vertical boxes because they each define a slice of the protocol stack for a particular purpose. Specific profiles, such as the headset profile, usually contain only those protocols needed by that application and no others. For example, profiles may include L2CAP if they have packets to send but skip L2CAP if they have only a steady flow of audio samples.

In the following sections, we will examine the Bluetooth radio layer and various link protocols, since these roughly correspond to the physical and MAC sublayers in the other procotol stacks we have studied.

4.6.4 The Bluetooth Radio Layer

The radio layer moves the bits from master to slave, or vice versa. It is a low-power system with a range of 10 meters operating in the same 2.4-GHz ISM band as 802.11. The band is divided into 79 channels of 1 MHz each. To coexist with other networks using the ISM band, frequency hopping spread spectrum is used. There can be up to 1600 hops/sec over slots with a dwell time of 625 μsec. All the nodes in a piconet hop frequencies simultaneously, following the slot timing and pseudorandom hop sequence dictated by the master.

Unfortunately, it turned out that early versions of Bluetooth and 802.11 interfered enough to ruin each other's transmissions. Some companies responded by banning Bluetooth altogether, but eventually a technical solution was devised. The solution is for Bluetooth to adapt its hop sequence to exclude channels on which there are other RF signals. This process reduces the harmful interference. It is called **adaptive frequency hopping**.

Three forms of modulation are used to send bits on a channel. The basic scheme is to use frequency shift keying to send a 1-bit symbol every microsecond, giving a gross data rate of 1 Mbps. Enhanced rates were introduced with the 2.0 version of Bluetooth. These rates use phase shift keying to send either 2 or 3 bits per symbol, for gross data rates of 2 or 3 Mbps. The enhanced rates are only used in the data portion of frames.

4.6.5 The Bluetooth Link Layers

The link control (or baseband) layer is the closest thing Bluetooth has to a MAC sublayer. It turns the raw bit stream into frames and defines some key formats. In the simplest form, the master in each piconet defines a series of 625-μsec time slots, with the master's transmissions starting in the even slots and the slaves' transmissions starting in the odd ones. This scheme is traditional time division multiplexing, with the master getting half the slots and the slaves sharing the other half. Frames can be 1, 3, or 5 slots long. Each frame has an overhead of 126 bits for an access code and header, plus a settling time of 250–260 μsec per hop to allow the inexpensive radio circuits to become stable. The payload of the frame can be encrypted for confidentiality with a key that is chosen when the master and slave connect. Hops only happen between frames, not during a frame. The result is that a 5-slot frame is much more efficient than a 1-slot frame because the overhead is constant but more data is sent.

The link manager protocol sets up logical channels, called **links**, to carry frames between the master and a slave device that have discovered each other. A pairing procedure is followed to make sure that the two devices are allowed to communicate before the link is used. The old pairing method is that both devices must be configured with the same four-digit PIN (Personal Identification Number). The matching PIN is how each device would know that it was connecting to

the right remote device. However, unimaginative users and devices default to PINs such as "0000" and "1234" meant that this method provided very little security in practice.

The new **secure simple pairing** method enables users to confirm that both devices are displaying the same passkey, or to observe the passkey on one device and enter it into the second device. This method is more secure because users do not have to choose or set a PIN. They merely confirm a longer, device-generated passkey. Of course, it cannot be used on some devices with limited input/output, such as a hands-free headset.

Once pairing is complete, the link manager protocol sets up the links. Two main kinds of links exist to carry user data. The first is the **SCO** (**Synchronous Connection Oriented**) link. It is used for real-time data, such as telephone connections. This type of link is allocated a fixed slot in each direction. A slave may have up to three SCO links with its master. Each SCO link can transmit one 64,000-bps PCM audio channel. Due to the time-critical nature of SCO links, frames sent over them are never retransmitted. Instead, forward error correction can be used to increase reliability.

The other kind is the **ACL** (**Asynchronous ConnectionLess**) link. This type of link is used for packet-switched data that is available at irregular intervals. ACL traffic is delivered on a best-effort basis. No guarantees are given. Frames can be lost and may have to be retransmitted. A slave may have only one ACL link to its master.

The data sent over ACL links come from the L2CAP layer. This layer has four major functions. First, it accepts packets of up to 64 KB from the upper layers and breaks them into frames for transmission. At the far end, the frames are reassembled into packets. Second, it handles the multiplexing and demultiplexing of multiple packet sources. When a packet has been reassembled, the L2CAP layer determines which upper-layer protocol to hand it to, for example, RFcomm or service discovery. Third, L2CAP handles error control and retransmission. It detects errors and resends packets that were not acknowledged. Finally, L2CAP enforces quality of service requirements between multiple links.

4.6.6 The Bluetooth Frame Structure

Bluetooth defines several frame formats, the most important of which is shown in two forms in Fig. 4-36. It begins with an access code that usually identifies the master so that slaves within radio range of two masters can tell which traffic is for them. Next comes a 54-bit header containing typical MAC sublayer fields. If the frame is sent at the basic rate, the data field comes next. It has up to 2744 bits for a five-slot transmission. For a single time slot, the format is the same except that the data field is 240 bits.

If the frame is sent at the enhanced rate, the data portion may have up to two or three times as many bits because each symbol carries 2 or 3 bits instead of 1

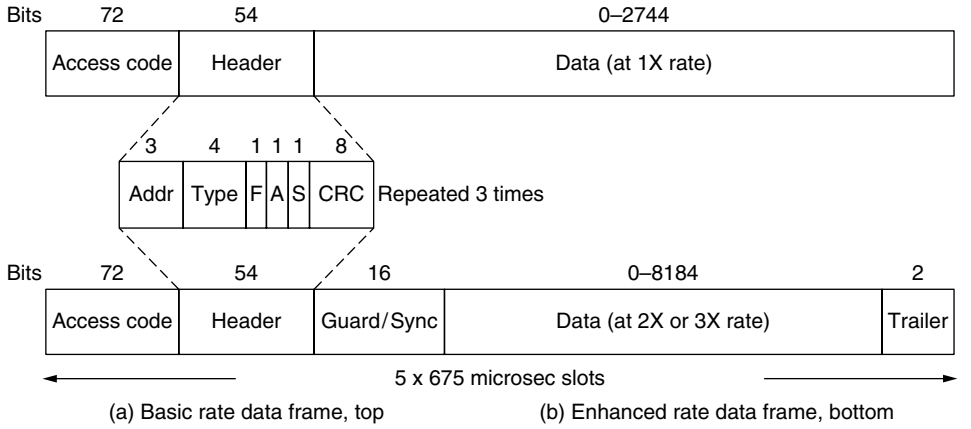


Figure 4-36. Typical Bluetooth data frame at (a) basic and (b) enhanced, data rates.

bit. These data are preceded by a guard field and a synchronization pattern that is used to switch to the faster data rate. That is, the access code and header are carried at the basic rate and only the data portion is carried at the faster rate. Enhanced-rate frames end with a short trailer.

Let us take a quick look at the common header. The *Address* field identifies which of the eight active devices the frame is intended for. The *Type* field identifies the frame type (ACL, SCO, poll, or null), the type of error correction used in the data field, and how many slots long the frame is. The *Flow* bit is asserted by a slave when its buffer is full and cannot receive any more data. This bit enables a primitive form of flow control. The *Acknowledgement* bit is used to piggyback an ACK onto a frame. The *Sequence* bit is used to number the frames to detect retransmissions. The protocol is stop-and-wait, so 1 bit is enough. Then comes the 8-bit header *Checksum*. The entire 18-bit header is repeated three times to form the 54-bit header shown in Fig. 4-36. On the receiving side, a simple circuit examines all three copies of each bit. If all three are the same, the bit is accepted. If not, the majority opinion wins. Thus, 54 bits of transmission capacity are used to send 10 bits of header. The reason is that to reliably send data in a noisy environment using cheap, low-powered (2.5 mW) devices with little computing capacity, a great deal of redundancy is needed.

Various formats are used for the data field for ACL and SCO frames. The basic-rate SCO frames are a simple example to study: the data field is always 240 bits. Three variants are defined, permitting 80, 160, or 240 bits of actual payload, with the rest being used for error correction. In the most reliable version (80-bit payload), the contents are just repeated three times, the same as the header.

We can work out the capacity with this frame as follows. Since the slave may use only the odd slots, it gets 800 slots/sec, just as the master does. With an 80-bit

payload, the channel capacity from the slave is 64,000 bps as is the channel capacity from the master. This capacity is exactly enough for a single full-duplex PCM voice channel (which is why a hop rate of 1600 hops/sec was chosen). That is, despite a raw bandwidth of 1 Mbps, a single full-duplex uncompressed voice channel can completely saturate the piconet. The efficiency of 13% is the result of spending 41% of the capacity on settling time, 20% on headers, and 26% on repetition coding. This shortcoming highlights the value of the enhanced rates and frames of more than a single slot.

There is much more to be said about Bluetooth, but no more space to say it here. For the curious, the Bluetooth 4.0 specification contains all the details.

4.7 RFID

We have looked at MAC designs from LANs up to MANs and down to PANs. As a last example, we will study a category of low-end wireless devices that people may not recognize as forming a computer network: the **RFID** (**Radio Frequency IDentification**) tags and readers that we described in Sec. 1.5.4.

RFID technology takes many forms, used in smartcards, implants for pets, passports, library books, and more. The form that we will look at was developed in the quest for an **EPC** (**Electronic Product Code**) that started with the Auto-ID Center at the Massachusetts Institute of Technology in 1999. An EPC is a replacement for a barcode that can carry a larger amount of information and is electronically readable over distances up to 10 m, even when it is not visible. It is different technology than, for example, the RFID used in passports,which must be placed quite close to a reader to perform a transaction. The ability to communicate over a distance makes EPCs more relevant to our studies.

EPCglobal was formed in 2003 to commercialize the RFID technology developed by the Auto-ID Center. The effort got a boost in 2005 when Walmart required its top 100 suppliers to label all shipments with RFID tags. Widespread deployment has been hampered by the difficulty of competing with cheap printed barcodes, but new uses, such as in drivers licenses, are now growing. We will describe the second generation of this technology, which is informally called **EPC Gen 2** (EPCglobal, 2008).

4.7.1 EPC Gen 2 Architecture

The architecture of an EPC Gen 2 RFID network is shown in Fig. 4-37. It has two key components: tags and readers. RFID tags are small, inexpensive devices that have a unique 96-bit EPC identifier and a small amount of memory that can be read and written by the RFID reader. The memory might be used to record the location history of an item, for example, as it moves through the supply chain.

Often, the tags look like stickers that can be placed on, for example, pairs of jeans on the shelves in a store. Most of the sticker is taken up by an antenna that is printed onto it. A tiny dot in the middle is the RFID integrated circuit. Alternatively, the RFID tags can be integrated into an object, such as a driver's license. In both cases, the tags have no battery and they must gather power from the radio transmissions of a nearby RFID reader to run. This kind of tag is called a "Class 1" tag to distinguish it from more capable tags that have batteries.

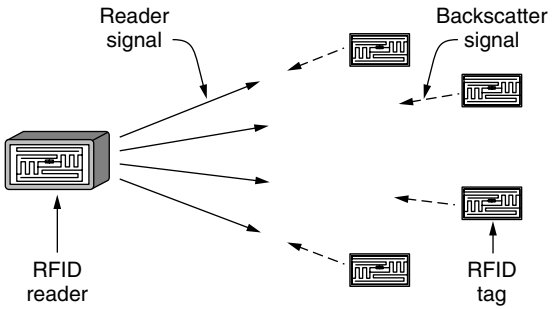


Figure 4-37. RFID architecture.

The readers are the intelligence in the system, analogous to base stations and access points in cellular and WiFi networks. Readers are much more powerful than tags. They have their own power sources, often have multiple antennas, and are in charge of when tags send and receive messages. As there will commonly be multiple tags within the reading range, the readers must solve the multiple access problem. There may be multiple readers that can contend with each other in the same area, too.

The main job of the reader is to inventory the tags in the neighborhood, that is, to discover the identifiers of the nearby tags. The inventory is accomplished with the physical layer protocol and the tag-identification protocol that are outlined in the following sections.

4.7.2 EPC Gen 2 Physical Layer

The physical layer defines how bits are sent between the RFID reader and tags. Much of it uses methods for sending wireless signals that we have seen previously. In the U.S., transmissions are sent in the unlicensed 902–928 MHz ISM band. This band falls in the UHF (Ultra High Frequency) range, so the tags are referred to as UHF RFID tags. The reader performs frequency hopping at least every 400 msec to spread its signal across the channel, to limit interference and satisfy regulatory requirements. The reader and tags use forms of ASK (Amplitude Shift Keying) modulation that we described in Sec. 2.5.2 to encode bits. They take turns to send bits, so the link is half duplex.

There are two main differences from other physical layers that we have studied. The first is that the reader is always transmitting a signal, regardless of whether it is the reader or tag that is communicating. Naturally, the reader transmits a signal to send bits to tags. For the tags to send bits to the reader, the reader transmits a fixed carrier signal that carries no bits. The tags harvest this signal to get the power they need to run; otherwise, a tag would not be able to transmit in the first place. To send data, a tag changes whether it is reflecting the signal from the reader, like a radar signal bouncing off a target, or absorbing it.

This method is called **backscatter**. It differs from all the other wireless situations we have seen so far, in which the sender and receiver never both transmit at the same time. Backscatter is a low-energy way for the tag to create a weak signal of its own that shows up at the reader. For the reader to decode the incoming signal, it must filter out the outgoing signal that it is transmitting. Because the tag signal is weak, tags can only send bits to the reader at a low rate, and tags cannot receive or even sense transmissions from other tags.

The second difference is that very simple forms of modulation are used so that they can be implemented on a tag that runs on very little power and costs only a few cents to make. To send data to the tags, the reader uses two amplitude levels. Bits are determined to be either a 0 or a 1, depending on how long the reader waits before a low-power period. The tag measures the time between low-power periods and compares this time to a reference measured during a preamble. As shown in Fig. 4-38, 1s are longer than 0s.

Tag responses consist of the tag alternating its backscatter state at fixed intervals to create a series of pulses in the signal. Anywhere from one to eight pulse periods can be used to encode each 0 or 1, depending on the need for reliability. 1s have fewer transitions than 0s, as is shown with an example of two-pulse period coding in Fig. 4-38.

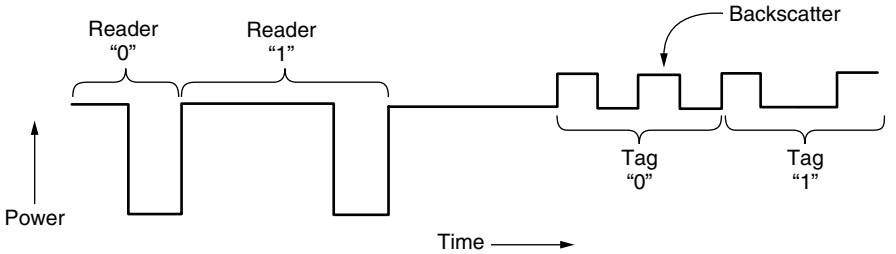


Figure 4-38. Reader and tag backscatter signals.

4.7.3 EPC Gen 2 Tag Identification Layer

To inventory the nearby tags, the reader needs to receive a message from each tag that gives the identifier for the tag. This situation is a multiple access problem for which the number of tags is unknown in the general case. The reader might

broadcast a query to ask all tags to send their identifiers. However, tags that replied right away would then collide in much the same way as stations on a classic Ethernet.

We have seen many ways of tackling the multiple access problem in this chapter. The closest protocol for the current situation, in which the tags cannot hear each others' transmissions, is slotted ALOHA, one of the earliest protocols we studied. This protocol is adapted for use in Gen 2 RFID.

The sequence of messages used to identify a tag is shown in Fig. 4-39. In the first slot (slot 0), the reader sends a *Query* message to start the process. Each *QRepeat* message advances to the next slot. The reader also tells the tags the range of slots over which to randomize transmissions. Using a range is necessary because the reader synchronizes tags when it starts the process; unlike stations on an Ethernet, tags do not wake up with a message at a time of their choosing.

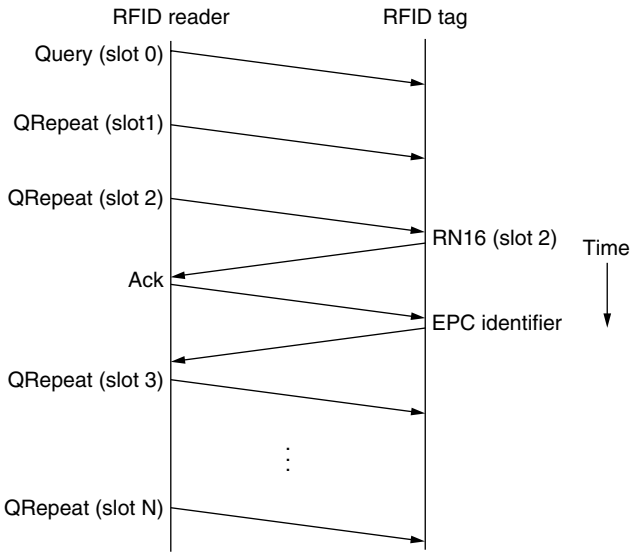


Figure 4-39. Example message exchange to identify a tag.

Tags pick a random slot in which to reply. In Fig. 4-39, the tag replies in slot 2. However, tags do not send their identifiers when they first reply. Instead, a tag sends a short 16-bit random number in an *RN16* message. If there is no collision, the reader receives this message and sends an *ACK* message of its own. At this stage, the tag has acquired the slot and sends its EPC identifier.

The reason for this exchange is that EPC identifiers are long, so collisions on these messages would be expensive. Instead, a short exchange is used to test whether the tag can safely use the slot to send its identifier. Once its identifier has been successfully transmitted, the tag temporarily stops responding to new *Query* messages so that all the remaining tags can be identified.

A key problem is for the reader to adjust the number of slots to avoid collisions, but without using so many slots that performance suffers. This adjustment is analogous to binary exponential backoff in Ethernet. If the reader sees too many slots with no responses or too many slots with collisions, it can send a *QAdjust* message to decrease or increase the range of slots over which the tags are responding.

The RFID reader can perform other operations on the tags. For example, it can select a subset of tags before running an inventory, allowing it to collect responses from, say, tagged jeans but not tagged shirts. The reader can also write data to tags as they are identified. This feature could be used to record the point of sale or other relevant information.

4.7.4 Tag Identification Message Formats

The format of the *Query* message is shown in Fig. 4-40 as an example of a reader-to-tag message. The message is compact because the downlink rates are limited, from 27 kbps up to 128 kbps. The *Command* field carries the code 1000 to identify the message as a *Query*.

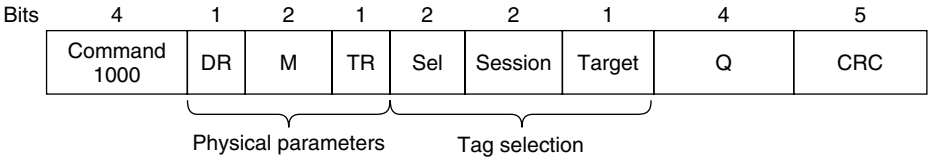


Figure 4-40. Format of the Query message.

The next flags, *DR*, *M*, and *TR*, determine the physical layer parameters for reader transmissions and tag responses. For example, the response rate may be set to between 5 kbps and 640 kbps. We will skip over the details of these flags.

Then come three fields, *Sel*, *Session*, and *Target*, that select the tags to respond. As well as the readers being able to select a subset of identifiers, the tags keep track of up to four concurrent sessions and whether they have been identified in those sessions. In this way, multiple readers can operate in overlapping coverage areas by using different sessions.

Next is the most important parameter for this command, Q. This field defines the range of slots over which tags will respond, from 0 to 2^Q-1. Finally, there is a CRC to protect the message fields. At 5 bits, it is shorter than most CRCs we have seen, but the *Query* message is much shorter than most packets too.

Tag-to-reader messages are simpler. Since the reader is in control, it knows what message to expect in response to each of its transmissions. The tag responses simply carry data, such as the EPC identifier.

Originally the tags were just for identification purposes. However, they have grown over time to resemble very small computers. Some research tags have sensors and are able to run small programs to gather and process data (Sample et al., 2008). One vision for this technology is the "Internet of things" that connects objects in the physical world to the Internet (Welbourne et al., 2009; and Gershenfeld et al., 2004).

4.8 DATA LINK LAYER SWITCHING

Many organizations have multiple LANs and wish to connect them. Would it not be convenient if we could just join the LANs together to make a larger LAN? In fact, we can do this when the connections are made with devices called **bridges**. The Ethernet switches we described in Sec. 4.3.4 are a modern name for bridges; they provide functionality that goes beyond classic Ethernet and Ethernet hubs to make it easy to join multiple LANs into a larger and faster network. We shall use the terms "bridge" and "switch" interchangeably.

Bridges operate in the data link layer, so they examine the data link layer addresses to forward frames. Since they are not supposed to examine the payload field of the frames they forward, they can handle IP packets as well as other kinds of packets, such as AppleTalk packets. In contrast, *routers* examine the addresses in packets and route based on them, so they only work with the protocols that they were designed to handle.

In this section, we will look at how bridges work and are used to join multiple physical LANs into a single logical LAN. We will also look at how to do the reverse and treat one physical LAN as multiple logical LANs, called **VLANs** (**Virtual LANs**). Both technologies provide useful flexibility for managing networks. For a comprehensive treatment of bridges, switches, and related topics, see Seifert and Edwards (2008) and Perlman (2000).

4.8.1 Uses of Bridges

Before getting into the technology of bridges, let us take a look at some common situations in which bridges are used. We will mention three reasons why a single organization may end up with multiple LANs.

First, many university and corporate departments have their own LANs to connect their own personal computers, servers, and devices such as printers. Since the goals of the various departments differ, different departments may set up different LANs, without regard to what other departments are doing. Sooner or later, though, there is a need for interaction, so bridges are needed. In this example, multiple LANs come into existence due to the autonomy of their owners.

Second, the organization may be geographically spread over several buildings separated by considerable distances. It may be cheaper to have separate LANs in each building and connect them with bridges and a few long-distance fiber optic links than to run all the cables to a single central switch. Even if laying the cables is easy to do, there are limits on their lengths (e.g., 200 m for twisted-pair gigabit Ethernet). The network would not work for longer cables due to the excessive signal attenuation or round-trip delay. The only solution is to partition the LAN and install bridges to join the pieces to increase the total physical distance that can be covered.

Third, it may be necessary to split what is logically a single LAN into separate LANs (connected by bridges) to accommodate the load. At many large universities, for example, thousands of workstations are available for student and faculty computing. Companies may also have thousands of employees. The scale of this system precludes putting all the workstations on a single LAN—there are more computers than ports on any Ethernet hub and more stations than allowed on a single classic Ethernet.

Even if it were possible to wire all the workstations together, putting more stations on an Ethernet hub or classic Ethernet would not add capacity. All of the stations share the same, fixed amount of bandwidth. The more stations there are, the less average bandwidth per station.

However, two separate LANs have twice the capacity of a single LAN. Bridges let the LANs be joined together while keeping this capacity. The key is not to send traffic onto ports where it is not needed, so that each LAN can run at full speed. This behavior also increases reliability, since on a single LAN a defective node that keeps outputting a continuous stream of garbage can clog up the entire LAN. By deciding what to forward and what not to forward, bridges act like fire doors in a building, preventing a single node that has gone berserk from bringing down the entire system.

To make these benefits easily available, ideally bridges should be completely transparent. It should be possible to go out and buy bridges, plug the LAN cables into the bridges, and have everything work perfectly, instantly. There should be no hardware changes required, no software changes required, no setting of address switches, no downloading of routing tables or parameters, nothing at all. Just plug in the cables and walk away. Furthermore, the operation of the existing LANs should not be affected by the bridges at all. As far as the stations are concerned, there should be no observable difference whether or not they are part of a bridged LAN. It should be as easy to move stations around the bridged LAN as it is to move them around a single LAN.

Surprisingly enough, it is actually possible to create bridges that are transparent. Two algorithms are used: a backward learning algorithm to stop traffic being sent where it is not needed; and a spanning tree algorithm to break loops that may be formed when switches are cabled together willy-nilly. Let us now take a look at these algorithms in turn to learn how this magic is accomplished.

4.8.2 Learning Bridges

The topology of two LANs bridged together is shown in Fig. 4-41 for two cases. On the left-hand side, two multidrop LANs, such as classic Ethernets, are joined by a special station—the bridge—that sits on both LANs. On the right-hand side, LANs with point-to-point cables, including one hub, are joined together. The bridges are the devices to which the stations and hub are attached. If the LAN technology is Ethernet, the bridges are better known as Ethernet switches.

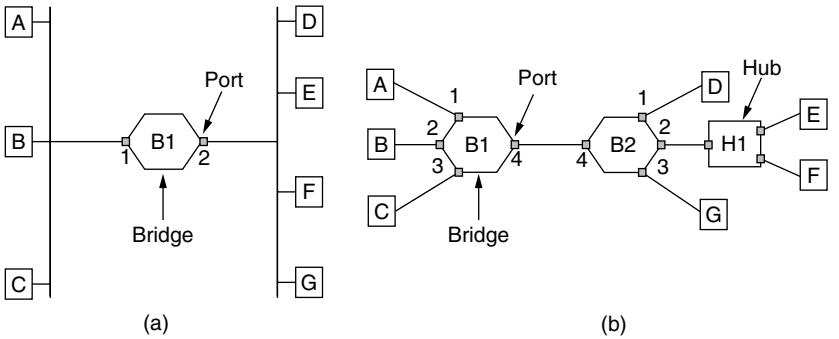


Figure 4-41. (a) Bridge connecting two multidrop LANs. (b) Bridges (and a hub) connecting seven point-to-point stations.

Bridges were developed when classic Ethernets were in use, so they are often shown in topologies with multidrop cables, as in Fig. 4-41(a). However, all the topologies that are encountered today are comprised of point-to-point cables and switches. The bridges work the same way in both settings. All of the stations attached to the same port on a bridge belong to the same collision domain, and this is different than the collision domain for other ports. If there is more than one station, as in a classic Ethernet, a hub, or a half-duplex link, the CSMA/CD protocol is used to send frames.

There is a difference, however, in how the bridged LANs are built. To bridge multidrop LANs, a bridge is added as a new station on each of the multidrop LANs, as in Fig. 4-41(a). To bridge point-to-point LANs, the hubs are either connected to a bridge or, preferably, replaced with a bridge to increase performance. In Fig. 4-41(b), bridges have replaced all but one hub.

Different kinds of cables can also be attached to one bridge. For example, the cable connecting bridge *B1* to bridge *B2* in Fig. 4-41(b) might be a long-distance fiber optic link, while the cable connecting the bridges to stations might be a short-haul twisted-pair line. This arrangement is useful for bridging LANs in different buildings.

Now let us consider what happens inside the bridges. Each bridge operates in promiscuous mode, that is, it accepts every frame transmitted by the stations

attached to each of its ports. The bridge must decide whether to forward or discard each frame, and, if the former, on which port to output the frame. This decision is made by using the destination address. As an example, consider the topology of Fig. 4-41(a). If station *A* sends a frame to station *B*, bridge *B1* will receive the frame on port 1. This frame can be immediately discarded without further ado because it is already on the correct port. However, in the topology of Fig. 4-41(b) suppose that *A* sends a frame to *D*. Bridge *B1* will receive the frame on port 1 and output it on port 4. Bridge *B2* will then receive the frame on its port 4 and output it on its port 1.

A simple way to implement this scheme is to have a big (hash) table inside the bridge. The table can list each possible destination and which output port it belongs on. For example, in Fig. 4-41(b), the table at *B1* would list *D* as belonging to port 4, since all *B1* has to know is which port to put frames on to reach *D*. That, in fact, more forwarding will happen later when the frame hits *B2* is not of interest to *B1*.

When the bridges are first plugged in, all the hash tables are empty. None of the bridges know where any of the destinations are, so they use a flooding algorithm: every incoming frame for an unknown destination is output on all the ports to which the bridge is connected except the one it arrived on. As time goes on, the bridges learn where destinations are. Once a destination is known, frames destined for it are put only on the proper port; they are not flooded.

The algorithm used by the bridges is **backward learning**. As mentioned above, the bridges operate in promiscuous mode, so they see every frame sent on any of their ports. By looking at the source addresses, they can tell which machines are accessible on which ports. For example, if bridge *B1* in Fig. 4-41(b) sees a frame on port 3 coming from *C*, it knows that *C* must be reachable via port 3, so it makes an entry in its hash table. Any subsequent frame addressed to *C* coming in to *B1* on any other port will be forwarded to port 3.

The topology can change as machines and bridges are powered up and down and moved around. To handle dynamic topologies, whenever a hash table entry is made, the arrival time of the frame is noted in the entry. Whenever a frame whose source is already in the table arrives, its entry is updated with the current time. Thus, the time associated with every entry tells the last time a frame from that machine was seen.

Periodically, a process in the bridge scans the hash table and purges all entries more than a few minutes old. In this way, if a computer is unplugged from its LAN, moved around the building, and plugged in again somewhere else, within a few minutes it will be back in normal operation, without any manual intervention. This algorithm also means that if a machine is quiet for a few minutes, any traffic sent to it will have to be flooded until it next sends a frame itself.

The routing procedure for an incoming frame depends on the port it arrives on (the source port) and the address to which it is destined (the destination address). The procedure is as follows.

1. If the port for the destination address is the same as the source port, discard the frame.
2. If the port for the destination address and the source port are different, forward the frame on to the destination port.
3. If the destination port is unknown, use flooding and send the frame on all ports except the source port.

You might wonder whether the first case can occur with point-to-point links. The answer is that it can occur if hubs are used to connect a group of computers to a bridge. An example is shown in Fig. 4-41(b) where stations *E* and *F* are connected to hub *H* 1, which is in turn connected to bridge *B2*. If *E* sends a frame to *F*, the hub will relay it to *B2* as well as to *F*. That is what hubs do—they wire all ports together so that a frame input on one port is simply output on all other ports. The frame will arrive at *B2* on port 4, which is already the right output port to reach the destination. Bridge *B2* need only discard the frame.

As each frame arrives, this algorithm must be applied, so it is usually implemented with special-purpose VLSI chips. The chips do the lookup and update the table entry, all in a few microseconds. Because bridges only look at the MAC addresses to decide how to forward frames, it is possible to start forwarding as soon as the destination header field has come in, before the rest of the frame has arrived (provided the output line is available, of course). This design reduces the latency of passing through the bridge, as well as the number of frames that the bridge must be able to buffer. It is referred to as **cut-through switching** or **wormhole routing** and is usually handled in hardware.

We can look at the operation of a bridge in terms of protocol stacks to understand what it means to be a link layer device. Consider a frame sent from station *A* to station *D* in the configuration of Fig. 4-41(a), in which the LANs are Ethernet. The frame will pass through one bridge. The protocol stack view of processing is shown in Fig. 4-42.

The packet comes from a higher layer and descends into the Ethernet MAC layer. It acquires an Ethernet header (and also a trailer, not shown in the figure). This unit is passed to the physical layer, goes out over the cable, and is picked up by the bridge.

In the bridge, the frame is passed up from the physical layer to the Ethernet MAC layer. This layer has extended processing compared to the Ethernet MAC layer at a station. It passes the frame to a relay, still within the MAC layer. The bridge relay function uses only the Ethernet MAC header to determine how to handle the frame. In this case, it passes the frame to the Ethernet MAC layer of the port used to reach station *D*, and the frame continues on its way.

In the general case, relays at a given layer can rewrite the headers for that layer. VLANs will provide an example shortly. In no case should the bridge look inside the frame and learn that it is carrying an IP packet; that is irrelevant to the

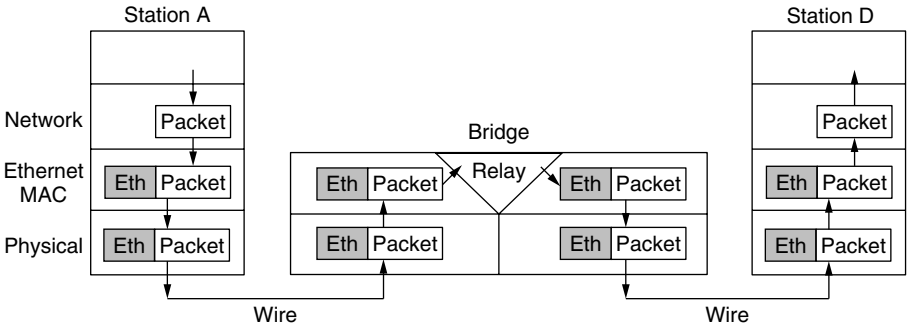


Figure 4-42. Protocol processing at a bridge.

bridge processing and would violate protocol layering. Also note that a bridge with k ports will have k instances of MAC and physical layers. The value of k is 2 for our simple example.

4.8.3 Spanning Tree Bridges

To increase reliability, redundant links can be used between bridges. In the example of Fig. 4-43, there are two links in parallel between a pair of bridges. This design ensures that if one link is cut, the network will not be partitioned into two sets of computers that cannot talk to each other.

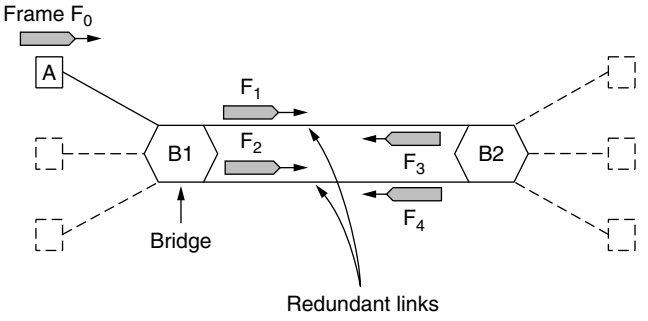


Figure 4-43. Bridges with two parallel links.

However, this redundancy introduces some additional problems because it creates loops in the topology. An example of these problems can be seen by looking at how a frame sent by A to a previously unobserved destination is handled in Fig. 4-43. Each bridge follows the normal rule for handling unknown destinations, which is to flood the frame. Call the frame from A that reaches bridge $B1$ frame F_0. The bridge sends copies of this frame out all of its other ports. We

will only consider the bridge ports that connect *B1* to *B2* (though the frame will be sent out the other ports, too). Since there are two links from *B1* to *B2*, two copies of the frame will reach *B2*. They are shown in Fig. 4-43 as F_1 and F_2.

Shortly thereafter, bridge *B2* receives these frames. However, it does not (and cannot) know that they are copies of the same frame, rather than two different frames sent one after the other. So bridge *B2* takes F_1 and sends copies of it out all the other ports, and it also takes F_2 and sends copies of it out all the other ports. This produces frames F_3 and F_4 that are sent along the two links back to *B1*. Bridge *B1* then sees two new frames with unknown destinations and copies them again. This cycle goes on forever.

The solution to this difficulty is for the bridges to communicate with each other and overlay the actual topology with a spanning tree that reaches every bridge. In effect, some potential connections between bridges are ignored in the interest of constructing a fictitious loop-free topology that is a subset of the actual topology.

For example, in Fig. 4-44 we see five bridges that are interconnected and also have stations connected to them. Each station connects to only one bridge. There are some redundant connections between the bridges so that frames will be forwarded in loops if all of the links are used. This topology can be thought of as a graph in which the bridges are the nodes and the point-to-point links are the edges. The graph can be reduced to a spanning tree, which has no cycles by definition, by dropping the links shown as dashed lines in Fig. 4-44. Using this spanning tree, there is exactly one path from every station to every other station. Once the bridges have agreed on the spanning tree, all forwarding between stations follows the spanning tree. Since there is a unique path from each source to each destination, loops are impossible.

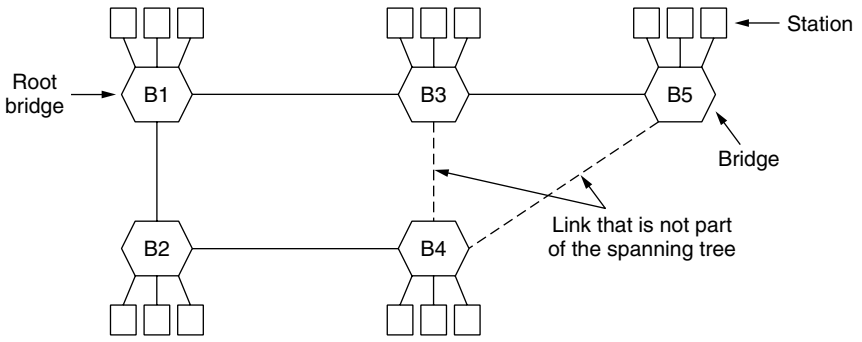


Figure 4-44. A spanning tree connecting five bridges. The dashed lines are links that are not part of the spanning tree.

To build the spanning tree, the bridges run a distributed algorithm. Each bridge periodically broadcasts a configuration message out all of its ports to its

neighbors and processes the messages it receives from other bridges, as described next. These messages are not forwarded, since their purpose is to build the tree, which can then be used for forwarding.

The bridges must first choose one bridge to be the root of the spanning tree. To make this choice, they each include an identifier based on their MAC address in the configuration message, as well as the identifier of the bridge they believe to be the root. MAC addresses are installed by the manufacturer and guaranteed to be unique worldwide, which makes these identifiers convenient and unique. The bridges choose the bridge with the lowest identifier to be the root. After enough messages have been exchanged to spread the news, all bridges will agree on which bridge is the root. In Fig. 4-44, bridge *B1* has the lowest identifier and becomes the root.

Next, a tree of shortest paths from the root to every bridge is constructed. In Fig. 4-44, bridges *B2* and *B3* can each be reached from bridge *B1* directly, in one hop that is a shortest path. Bridge *B4* can be reached in two hops, via either *B2* or *B3*. To break this tie, the path via the bridge with the lowest identifier is chosen, so *B4* is reached via *B2*. Bridge *B5* can be reached in two hops via *B3*.

To find these shortest paths, bridges include the distance from the root in their configuration messages. Each bridge remembers the shortest path it finds to the root. The bridges then turn off ports that are not part of the shortest path.

Although the tree spans all the bridges, not all the links (or even bridges) are necessarily present in the tree. This happens because turning off the ports prunes some links from the network to prevent loops. Even after the spanning tree has been established, the algorithm continues to run during normal operation to automatically detect topology changes and update the tree.

The algorithm for constructing the spanning tree was invented by Radia Perlman. Her job was to solve the problem of joining LANs without loops. She was given a week to do it, but she came up with the idea for the spanning tree algorithm in a day. Fortunately, this left her enough time to write it as a poem (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.

The spanning tree algorithm was then standardized as IEEE 802.1D and used for many years. In 2001, it was revised to more rapidly find a new spanning tree after a topology change. For a detailed treatment of bridges, see Perlman (2000).

4.8.4 Repeaters, Hubs, Bridges, Switches, Routers, and Gateways

So far in this book, we have looked at a variety of ways to get frames and packets from one computer to another. We have mentioned repeaters, hubs, bridges, switches, routers, and gateways. All of these devices are in common use, but they all differ in subtle and not-so-subtle ways. Since there are so many of them, it is probably worth taking a look at them together to see what the similarities and differences are.

The key to understanding these devices is to realize that they operate in different layers, as illustrated in Fig. 4-45(a). The layer matters because different devices use different pieces of information to decide how to switch. In a typical scenario, the user generates some data to be sent to a remote machine. Those data are passed to the transport layer, which then adds a header (for example, a TCP header) and passes the resulting unit down to the network layer. The network layer adds its own header to form a network layer packet (e.g., an IP packet). In Fig. 4-45(b), we see the IP packet shaded in gray. Then the packet goes to the data link layer, which adds its own header and checksum (CRC) and gives the resulting frame to the physical layer for transmission, for example, over a LAN.

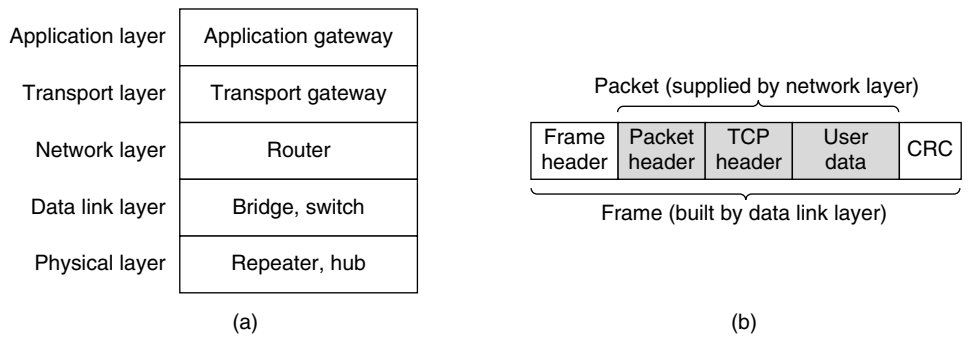


Figure 4-45. (a) Which device is in which layer. (b) Frames, packets, and headers.

Now let us look at the switching devices and see how they relate to the packets and frames. At the bottom, in the physical layer, we find the repeaters. These are analog devices that work with signals on the cables to which they are connected. A signal appearing on one cable is cleaned up, amplified, and put out on another cable. Repeaters do not understand frames, packets, or headers. They understand the symbols that encode bits as volts. Classic Ethernet, for example, was

designed to allow four repeaters that would boost the signal to extend the maximum cable length from 500 meters to 2500 meters.

Next we come to the hubs. A hub has a number of input lines that it joins electrically. Frames arriving on any of the lines are sent out on all the others. If two frames arrive at the same time, they will collide, just as on a coaxial cable. All the lines coming into a hub must operate at the same speed. Hubs differ from repeaters in that they do not (usually) amplify the incoming signals and are designed for multiple input lines, but the differences are slight. Like repeaters, hubs are physical layer devices that do not examine the link layer addresses or use them in any way.

Now let us move up to the data link layer, where we find bridges and switches. We just studied bridges at some length. A bridge connects two or more LANs. Like a hub, a modern bridge has multiple ports, usually enough for 4 to 48 input lines of a certain type. Unlike in a hub, each port is isolated to be its own collision domain; if the port has a full-duplex point-to-point line, the CSMA/CD algorithm is not needed. When a frame arrives, the bridge extracts the destination address from the frame header and looks it up in a table to see where to send the frame. For Ethernet, this address is the 48-bit destination address shown in Fig. 4-14. The bridge only outputs the frame on the port where it is needed and can forward multiple frames at the same time.

Bridges offer much better performance than hubs, and the isolation between bridge ports also means that the input lines may run at different speeds, possibly even with different network types. A common example is a bridge with ports that connect to 10-, 100-, and 1000-Mbps Ethernet. Buffering within the bridge is needed to accept a frame on one port and transmit the frame out on a different port. If frames come in faster than they can be retransmitted, the bridge may run out of buffer space and have to start discarding frames. For example, if a gigabit Ethernet is pouring bits into a 10-Mbps Ethernet at top speed, the bridge will have to buffer them, hoping not to run out of memory. This problem still exists even if all the ports run at the same speed because more than one port may be sending frames to a given destination port.

Bridges were originally intended to be able to join different kinds of LANs, for example, an Ethernet and a Token Ring LAN. However, this never worked well because of differences between the LANs. Different frame formats require copying and reformatting, which takes CPU time, requires a new checksum calculation, and introduces the possibility of undetected errors due to bad bits in the bridge's memory. Different maximum frame lengths are also a serious problem with no good solution. Basically, frames that are too large to be forwarded must be discarded. So much for transparency.

Two other areas where LANs can differ are security and quality of service. Some LANs have link-layer encryption, for example 802.11, and some do not, for example Ethernet. Some LANs have quality of service features such as priorities, for example 802.11, and some do not, for example Ethernet. Consequently, when

a frame must travel between these LANs, the security or quality of service expected by the sender may not be able to be provided. For all of these reasons, modern bridges usually work for one network type, and routers, which we will come to soon, are used instead to join networks of different types.

Switches are modern bridges by another name. The differences are more to do with marketing than technical issues, but there are a few points worth knowing. Bridges were developed when classic Ethernet was in use, so they tend to join relatively few LANs and thus have relatively few ports. The term ''switch'' is more popular nowadays. Also, modern installations all use point-to-point links, such as twisted-pair cables, so individual computers plug directly into a switch and thus the switch will tend to have many ports. Finally, ''switch'' is also used as a general term. With a bridge, the functionality is clear. On the other hand, a switch may refer to an Ethernet switch or a completely different kind of device that makes forwarding decisions, such as a telephone switch.

So far, we have seen repeaters and hubs, which are actually quite similar, as well as bridges and switches, which are even more similar to each other. Now we move up to routers, which are different from all of the above. When a packet comes into a router, the frame header and trailer are stripped off and the packet located in the frame's payload field (shaded in Fig. 4-45) is passed to the routing software. This software uses the packet header to choose an output line. For an IP packet, the packet header will contain a 32-bit (IPv4) or 128-bit (IPv6) address, but not a 48-bit IEEE 802 address. The routing software does not see the frame addresses and does not even know whether the packet came in on a LAN or a point-to-point line. We will study routers and routing in Chap. 5.

Up another layer, we find transport gateways. These connect two computers that use different connection-oriented transport protocols. For example, suppose a computer using the connection-oriented TCP/IP protocol needs to talk to a computer using a different connection-oriented transport protocol called SCTP. The transport gateway can copy the packets from one connection to the other, reformatting them as need be.

Finally, application gateways understand the format and contents of the data and can translate messages from one format to another. An email gateway could translate Internet messages into SMS messages for mobile phones, for example. Like ''switch,'' ''gateway'' is somewhat of a general term. It refers to a forwarding process that runs at a high layer.

4.8.5 Virtual LANs

In the early days of local area networking, thick yellow cables snaked through the cable ducts of many office buildings. Every computer they passed was plugged in. No thought was given to which computer belonged on which LAN. All the people in adjacent offices were put on the same LAN, whether they belonged together or not. Geography trumped corporate organization charts.

With the advent of twisted pair and hubs in the 1990s, all that changed. Buildings were rewired (at considerable expense) to rip out all the yellow garden hoses and install twisted pairs from every office to central wiring closets at the end of each corridor or in a central machine room, as illustrated in Fig. 4-46. If the Vice President in Charge of Wiring was a visionary, Category 5 twisted pairs were installed; if he was a bean counter, the existing (Category 3) telephone wiring was used (only to be replaced a few years later, when fast Ethernet emerged).

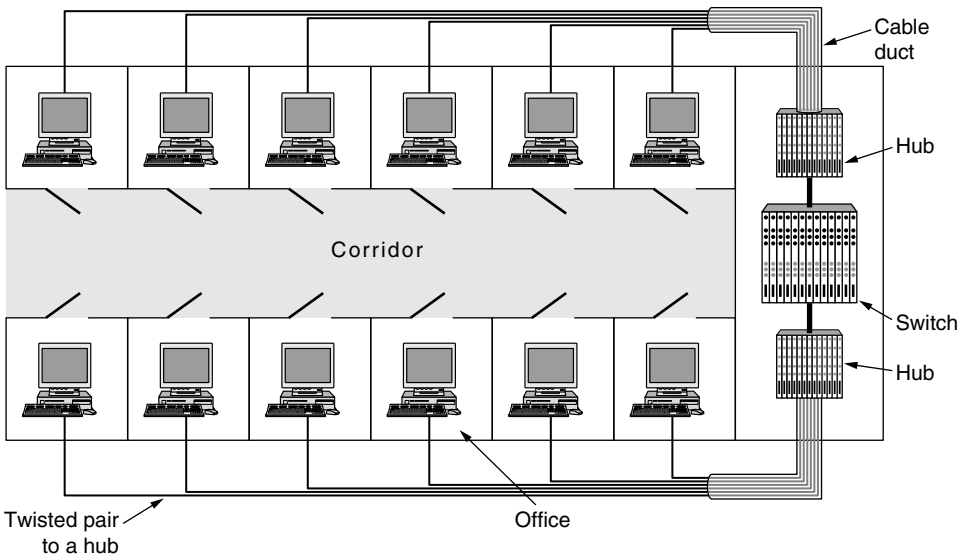


Figure 4-46. A building with centralized wiring using hubs and a switch.

Today, the cables have changed and hubs have become switches, but the wiring pattern is still the same. This pattern makes it possible to configure LANs logically rather than physically. For example, if a company wants *k* LANs, it could buy *k* switches. By carefully choosing which connectors to plug into which switches, the occupants of a LAN can be chosen in a way that makes organizational sense, without too much regard to geography.

Does it matter who is on which LAN? After all, in nearly all organizations, all the LANs are interconnected. In short, yes, it often matters. Network administrators like to group users on LANs to reflect the organizational structure rather than the physical layout of the building, for a variety of reasons. One issue is security. One LAN might host Web servers and other computers intended for public use. Another LAN might host computers containing the records of the Human Resources department that are not to be passed outside of the department. In such a situation, putting all the computers on a single LAN and not letting any of the servers be accessed from off the LAN makes sense. Management tends to frown when hearing that such an arrangement is impossible.

A second issue is load. Some LANs are more heavily used than others and it may be desirable to separate them. For example, if the folks in research are running all kinds of nifty experiments that sometimes get out of hand and saturate their LAN, the folks in management may not be enthusiastic about donating some of the capacity they were using for videoconferencing to help out. Then again, this might impress on management the need to install a faster network.

A third issue is broadcast traffic. Bridges broadcast traffic when the location of the destination is unknown, and upper-layer protocols use broadcasting as well. For example, when a user wants to send a packet to an IP address *x*, how does it know which MAC address to put in the frame? We will study this question in Chap. 5, but briefly summarized, the answer is that it broadcasts a frame containing the question "who owns IP address *x*?" Then it waits for an answer. As the number of computers in a LAN grows, so does the number of broadcasts. Each broadcast consumes more of the LAN capacity than a regular frame because it is delivered to every computer on the LAN. By keeping LANs no larger than they need to be, the impact of broadcast traffic is reduced.

Related to broadcasts is the problem that once in a while a network interface will break down or be misconfigured and begin generating an endless stream of broadcast frames. If the network is really unlucky, some of these frames will elicit responses that lead to ever more traffic. The result of this **broadcast storm** is that (1) the entire LAN capacity is occupied by these frames, and (2) all the machines on all the interconnected LANs are crippled just processing and discarding all the frames being broadcast.

At first it might appear that broadcast storms could be limited in scope by separating the LANs with bridges or switches, but if the goal is to achieve transparency (i.e., a machine can be moved to a different LAN across the bridge without anyone noticing it), then bridges have to forward broadcast frames.

Having seen why companies might want multiple LANs with restricted scopes, let us get back to the problem of decoupling the logical topology from the physical topology. Building a physical topology to reflect the organizational structure can add work and cost, even with centralized wiring and switches. For example, if two people in the same department work in different buildings, it may be easier to wire them to different switches that are part of different LANs. Even if this is not the case, a user might be shifted within the company from one department to another without changing offices, or might change offices without changing departments. This might result in the user being on the wrong LAN until an administrator changes the user's connector from one switch to another. Furthermore, the number of computers that belong to different departments may not be a good match for the number of ports on switches; some departments may be too small and others so big that they require multiple switches. This results in wasted switch ports that are not used.

In many companies, organizational changes occur all the time, meaning that system administrators spend a lot of time pulling out plugs and pushing them back

in somewhere else. Also, in some cases, the change cannot be made at all because the twisted pair from the user's machine is too far from the correct switch (e.g., in the wrong building), or the available switch ports are on the wrong LAN.

In response to customer requests for more flexibility, network vendors began working on a way to rewire buildings entirely in software. The resulting concept is called a **VLAN** (**Virtual LAN**). It has been standardized by the IEEE 802 committee and is now widely deployed in many organizations. Let us now take a look at it. For additional information about VLANs, see Seifert and Edwards (2008).

VLANs are based on VLAN-aware switches. To set up a VLAN-based network, the network administrator decides how many VLANs there will be, which computers will be on which VLAN, and what the VLANs will be called. Often the VLANs are (informally) named by colors, since it is then possible to print color diagrams showing the physical layout of the machines, with the members of the red LAN in red, members of the green LAN in green, and so on. In this way, both the physical and logical layouts are visible in a single view.

As an example, consider the bridged LAN of Fig. 4-47, in which nine of the machines belong to the G (gray) VLAN and five belong to the W (white) VLAN. Machines from the gray VLAN are spread across two switches, including two machines that connect to a switch via a hub.

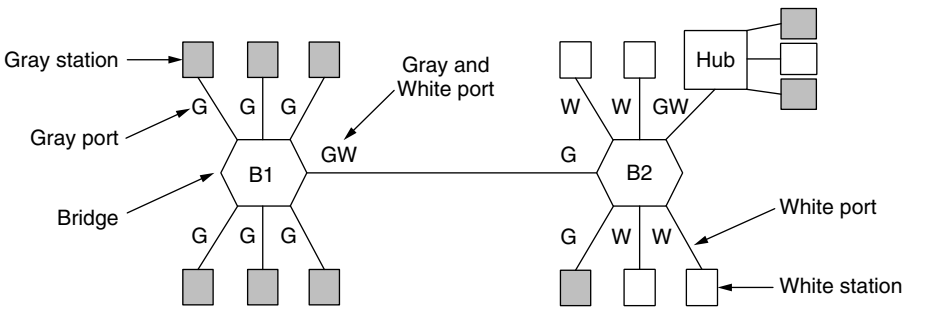


Figure 4-47. Two VLANs, gray and white, on a bridged LAN.

To make the VLANs function correctly, configuration tables have to be set up in the bridges. These tables tell which VLANs are accessible via which ports. When a frame comes in from, say, the gray VLAN, it must be forwarded on all the ports marked with a G. This holds for ordinary (i.e., unicast) traffic for which the bridges have not learned the location of the destination, as well as for multicast and broadcast traffic. Note that a port may be labeled with multiple VLAN colors.

As an example, suppose that one of the gray stations plugged into bridge *B1* in Fig. 4-47 sends a frame to a destination that has not been observed beforehand. Bridge *B1* will receive the frame and see that it came from a machine on the gray

VLAN, so it will flood the frame on all ports labeled G (except the incoming port). The frame will be sent to the five other gray stations attached to *B1* as well as over the link from *B1* to bridge *B2*. At bridge *B2*, the frame is similarly forwarded on all ports labeled G. This sends the frame to one further station and the hub (which will transmit the frame to all of its stations). The hub has both labels because it connects to machines from both VLANs. The frame is not sent on other ports without G in the label because the bridge knows that there are no machines on the gray VLAN that can be reached via these ports.

In our example, the frame is only sent from bridge *B1* to bridge *B2* because there are machines on the gray VLAN that are connected to *B2*. Looking at the white VLAN, we can see that the bridge *B2* port that connects to bridge *B1* is *not* labeled W. This means that a frame on the white VLAN will not be forwarded from bridge *B2* to bridge *B1*. This behavior is correct because no stations on the white VLAN are connected to *B1*.

The IEEE 802.1Q Standard

To implement this scheme, bridges need to know to which VLAN an incoming frame belongs. Without this information, for example, when bridge *B2* gets a frame from bridge *B1* in Fig. 4-47, it cannot know whether to forward the frame on the gray or white VLAN. If we were designing a new type of LAN, it would be easy enough to just add a VLAN field in the header. But what to do about Ethernet, which is the dominant LAN, and did not have any spare fields lying around for the VLAN identifier?

The IEEE 802 committee had this problem thrown into its lap in 1995. After much discussion, it did the unthinkable and changed the Ethernet header. The new format was published in IEEE standard **802.1Q**, issued in 1998. The new format contains a VLAN tag; we will examine it shortly. Not surprisingly, changing something as well established as the Ethernet header was not entirely trivial. A few questions that come to mind are:

1. Need we throw out several hundred million existing Ethernet cards?
2. If not, who generates the new fields?
3. What happens to frames that are already the maximum size?

Of course, the 802 committee was (only too painfully) aware of these problems and had to come up with solutions, which it did.

The key to the solution is to realize that the VLAN fields are only actually used by the bridges and switches and *not* by the user machines. Thus, in Fig. 4-47, it is not really essential that they are present on the lines going out to the end stations as long as they are on the line between the bridges. Also, to use VLANs, the bridges have to be VLAN aware. This fact makes the design feasible.

As to throwing out all existing Ethernet cards, the answer is no. Remember that the 802.3 committee could not even get people to change the *Type* field into a *Length* field. You can imagine the reaction to an announcement that all existing Ethernet cards had to be thrown out. However, new Ethernet cards are 802.1Q compliant and can correctly fill in the VLAN fields.

Because there can be computers (and switches) that are not VLAN aware, the first VLAN-aware bridge to touch a frame adds VLAN fields and the last one down the road removes them. An example of a mixed topology is shown in Fig. 4-48. In this figure, VLAN-aware computers generate tagged (i.e., 802.1Q) frames directly, and further switching uses these tags. The shaded symbols are VLAN-aware and the empty ones are not.

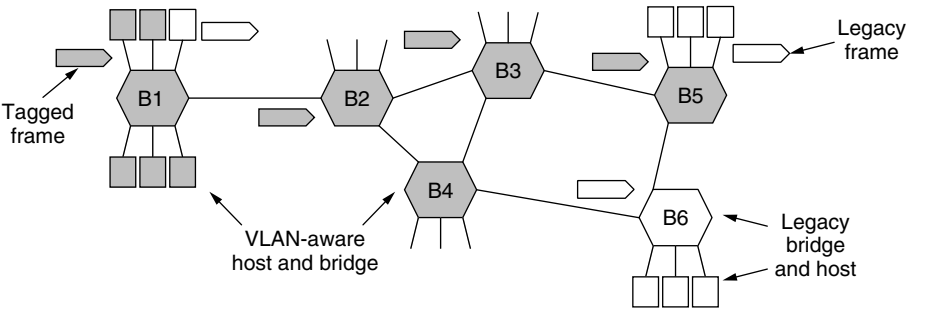


Figure 4-48. Bridged LAN that is only partly VLAN aware. The shaded symbols are VLAN aware. The empty ones are not.

With 802.1Q, frames are colored depending on the port on which they are received. For this method to work, all machines on a port must belong to the same VLAN, which reduces flexibility. For example, in Fig. 4-48, this property holds for all ports where an individual computer connects to a bridge, but not for the port where the hub connects to bridge *B2*.

Additionally, the bridge can use the higher-layer protocol to select the color. In this way, frames arriving on a port might be placed in different VLANs depending on whether they carry IP packets or PPP frames.

Other methods are possible, but they are not supported by 802.1Q. As one example, the MAC address can be used to select the VLAN color. This might be useful for frames coming in from a nearby 802.11 LAN in which laptops send frames via different ports as they move. One MAC address would then be mapped to a fixed VLAN regardless of which port it entered the LAN on.

As to the problem of frames longer than 1518 bytes, 802.1Q just raised the limit to 1522 bytes. Luckily, only VLAN-aware computers and switches must support these longer frames.

Now let us take a look at the 802.1Q frame format. It is shown in Fig. 4-49. The only change is the addition of a pair of 2-byte fields. The first one is the

VLAN protocol ID. It always has the value 0x8100. Since this number is greater than 1500, all Ethernet cards interpret it as a type rather than a length. What a legacy card does with such a frame is moot since such frames are not supposed to be sent to legacy cards.

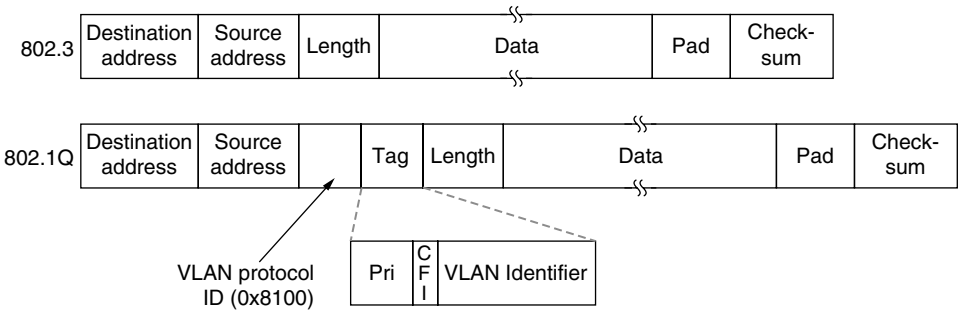


Figure 4-49. The 802.3 (legacy) and 802.1Q Ethernet frame formats.

The second 2-byte field contains three subfields. The main one is the *VLAN identifier*, occupying the low-order 12 bits. This is what the whole thing is about—the color of the VLAN to which the frame belongs. The 3-bit *Priority* field has nothing to do with VLANs at all, but since changing the Ethernet header is a once-in-a-decade event taking three years and featuring a hundred people, why not put in some other good things while you are at it? This field makes it possible to distinguish hard real-time traffic from soft real-time traffic from time-insensitive traffic in order to provide better quality of service over Ethernet. It is needed for voice over Ethernet (although in all fairness, IP has had a similar field for a quarter of a century and nobody ever used it).

The last field, *CFI* (*Canonical format indicator*), should have been called the *CEI* (*Corporate ego indicator*). It was originally intended to indicate the order of the bits in the MAC addresses (little-endian versus big-endian), but that use got lost in other controversies. Its presence now indicates that the payload contains a freeze-dried 802.5 frame that is hoping to find another 802.5 LAN at the destination while being carried by Ethernet in between. This whole arrangement, of course, has nothing whatsoever to do with VLANs. But standards' committee politics are not unlike regular politics: if you vote for my bit, I will vote for your bit.

As we mentioned above, when a tagged frame arrives at a VLAN-aware switch, the switch uses the VLAN identifier as an index into a table to find out which ports to send it on. But where does the table come from? If it is manually constructed, we are back to square zero: manual configuration of bridges. The beauty of the transparent bridge is that it is plug-and-play and does not require any manual configuration. It would be a terrible shame to lose that property. Fortunately, VLAN-aware bridges can also autoconfigure themselves based on observing the tags that come by. If a frame tagged as VLAN 4 comes in on port

3, apparently some machine on port 3 is on VLAN 4. The 802.1Q standard explains how to build the tables dynamically, mostly by referencing appropriate portions of the 802.1D standard.

Before leaving the subject of VLAN routing, it is worth making one last observation. Many people in the Internet and Ethernet worlds are fanatically in favor of connectionless networking and violently opposed to anything smacking of connections in the data link or network layers. Yet VLANs introduce something that is surprisingly similar to a connection. To use VLANs properly, each frame carries a new special identifier that is used as an index into a table inside the switch to look up where the frame is supposed to be sent. That is precisely what happens in connection-oriented networks. In connectionless networks, it is the destination address that is used for routing, not some kind of connection identifier. We will see more of this creeping connectionism in Chap. 5.

4.9 SUMMARY

Some networks have a single channel that is used for all communication. In these networks, the key design issue is the allocation of this channel among the competing stations wishing to use it. FDM and TDM are simple, efficient allocation schemes when the number of stations is small and fixed and the traffic is continuous. Both are widely used under these circumstances, for example, for dividing up the bandwidth on telephone trunks. However, when the number of stations is large and variable or the traffic is fairly bursty—the common case in computer networks—FDM and TDM are poor choices.

Numerous dynamic channel allocation algorithms have been devised. The ALOHA protocol, with and without slotting, is used in many derivatives in real systems, for example, cable modems and RFID. As an improvement when the state of the channel can be sensed, stations can avoid starting a transmission while another station is transmitting. This technique, carrier sensing, has led to a variety of CSMA protocols for LANs and MANs. It is the basis for classic Ethernet and 802.11 networks.

A class of protocols that eliminates contention altogether, or at least reduces it considerably, is well known. The bitmap protocol, topologies such as rings, and the binary countdown protocol completely eliminate contention. The tree walk protocol reduces it by dynamically dividing the stations into two disjoint groups of different sizes and allowing contention only within one group; ideally that group is chosen so that only one station is ready to send when it is permitted to do so.

Wireless LANs have the added problems that it is difficult to sense colliding transmissions, and that the coverage regions of stations may differ. In the dominant wireless LAN, IEEE 802.11, stations use CSMA/CA to mitigate the first problem by leaving small gaps to avoid collisions. The stations can also use the RTS/CTS protocol to combat hidden terminals that arise because of the second

problem. IEEE 802.11 is commonly used to connect laptops and other devices to wireless access points, but it can also be used between devices. Any of several physical layers can be used, including multichannel FDM with and without multiple antennas, and spread spectrum.

Like 802.11, RFID readers and tags use a random access protocol to communicate identifiers. Other wireless PANs and MANs have different designs. The Bluetooth system connects headsets and many kinds of peripherals to computers without wires. IEEE 802.16 provides a wide area wireless Internet data service for stationary and mobile computers. Both of these networks use a centralized, connection-oriented design in which the Bluetooth master and the WiMAX base station decide when each station may send or receive data. For 802.16, this design supports different quality of service for real-time traffic like telephone calls and interactive traffic like Web browsing. For Bluetooth, placing the complexity in the master leads to inexpensive slave devices.

Ethernet is the dominant form of wired LAN. Classic Ethernet used CSMA/CD for channel allocation on a yellow cable the size of a garden hose that snaked from machine to machine. The architecture has changed as speeds have risen from 10 Mbps to 10 Gbps and continue to climb. Now, point-to-point links such as twisted pair are attached to hubs and switches. With modern switches and full-duplex links, there is no contention on the links and the switch can forward frames between different ports in parallel.

With buildings full of LANs, a way is needed to interconnect them all. Plug-and-play bridges are used for this purpose. The bridges are built with a backward learning algorithm and a spanning tree algorithm. Since this functionality is built into modern switches, the terms ''bridge'' and ''switch'' are used interchangeably. To help with the management of bridged LANs, VLANs let the physical topology be divided into different logical topologies. The VLAN standard, IEEE 802.1Q, introduces a new format for Ethernet frames.

PROBLEMS

1. For this problem, use a formula from this chapter, but first state the formula. Frames arrive randomly at a 100-Mbps channel for transmission. If the channel is busy when a frame arrives, it waits its turn in a queue. Frame length is exponentially distributed with a mean of 10,000 bits/frame. For each of the following frame arrival rates, give the delay experienced by the average frame, including both queueing time and transmission time.

(a) 90 frames/sec.
(b) 900 frames/sec.
(c) 9000 frames/sec.

2. A group of N stations share a 56-kbps pure ALOHA channel. Each station outputs a 1000-bit frame on average once every 100 sec, even if the previous one has not yet been sent (e.g., the stations can buffer outgoing frames). What is the maximum value of N?

3. Consider the delay of pure ALOHA versus slotted ALOHA at low load. Which one is less? Explain your answer.

4. A large population of ALOHA users manages to generate 50 requests/sec, including both originals and retransmissions. Time is slotted in units of 40 msec.
(a) What is the chance of success on the first attempt?
(b) What is the probability of exactly k collisions and then a success?
(c) What is the expected number of transmission attempts needed?

5. In an infinite-population slotted ALOHA system, the mean number of slots a station waits between a collision and a retransmission is 4. Plot the delay versus throughput curve for this system.

6. What is the length of a contention slot in CSMA/CD for (a) a 2-km twin-lead cable (signal propagation speed is 82% of the signal propagation speed in vacuum)?, and (b) a 40-km multimode fiber optic cable (signal propagation speed is 65% of the signal propagation speed in vacuum)?

7. How long does a station, s, have to wait in the worst case before it can start transmitting its frame over a LAN that uses the basic bit-map protocol?

8. In the binary countdown protocol, explain how a lower-numbered station may be starved from sending a packet.

9. Sixteen stations, numbered 1 through 16, are contending for the use of a shared channel by using the adaptive tree walk protocol. If all the stations whose addresses are prime numbers suddenly become ready at once, how many bit slots are needed to resolve the contention?

10. Consider five wireless stations, A, B, C, D, and E. Station A can communicate with all other stations. B can communicate with A, C and E. C can communicate with A, B and D. D can communicate with A, C and E. E can communicate A, D and B.

(a) When A is sending to B, what other communications are possible?
(b) When B is sending to A, what other communications are possible?
(c) When B is sending to C, what other communications are possible?

11. Six stations, A through F, communicate using the MACA protocol. Is it possible for two transmissions to take place simultaneously? Explain your answer.

12. A seven-story office building has 15 adjacent offices per floor. Each office contains a wall socket for a terminal in the front wall, so the sockets form a rectangular grid in the vertical plane, with a separation of 4 m between sockets, both horizontally and vertically. Assuming that it is feasible to run a straight cable between any pair of sockets, horizontally, vertically, or diagonally, how many meters of cable are needed to connect all sockets using

(a) A star configuration with a single router in the middle?
(b) A classic 802.3 LAN?

Patent Owner's Ex. 2010
Page 199

13. What is the baud rate of classic 10-Mbps Ethernet?

14. Sketch the Manchester encoding on a classic Ethernet for the bit stream 0001110101.

15. A 1-km-long, 10-Mbps CSMA/CD LAN (not 802.3) has a propagation speed of 200 m/μsec. Repeaters are not allowed in this system. Data frames are 256 bits long, including 32 bits of header, checksum, and other overhead. The first bit slot after a successful transmission is reserved for the receiver to capture the channel in order to send a 32-bit acknowledgement frame. What is the effective data rate, excluding overhead, assuming that there are no collisions?

16. Two CSMA/CD stations are each trying to transmit long (multiframe) files. After each frame is sent, they contend for the channel, using the binary exponential backoff algorithm. What is the probability that the contention ends on round k, and what is the mean number of rounds per contention period?

17. An IP packet to be transmitted by Ethernet is 60 bytes long, including all its headers. If LLC is not in use, is padding needed in the Ethernet frame, and if so, how many bytes?

18. Ethernet frames must be at least 64 bytes long to ensure that the transmitter is still going in the event of a collision at the far end of the cable. Fast Ethernet has the same 64-byte minimum frame size but can get the bits out ten times faster. How is it possible to maintain the same minimum frame size?

19. Some books quote the maximum size of an Ethernet frame as 1522 bytes instead of 1500 bytes. Are they wrong? Explain your answer.

20. How many frames per second can gigabit Ethernet handle? Think carefully and take into account all the relevant cases. *Hint*: the fact that it is *gigabit* Ethernet matters.

21. Name two networks that allow frames to be packed back-to-back. Why is this feature worth having?

22. In Fig. 4-27, four stations, *A*, *B*, *C*, and *D*, are shown. Which of the last two stations do you think is closest to *A* and why?

23. Give an example to show that the RTS/CTS in the 802.11 protocol is a little different than in the MACA protocol.

24. A wireless LAN with one AP has 10 client stations. Four stations have data rates of 6 Mbps, four stations have data rates of 18 Mbps, and the last two stations have data rates of 54 Mbps. What is the data rate experienced by each station when all ten stations are sending data together, and

(a) TXOP is not used?
(b) TXOP is used?

25. Suppose that an 11-Mbps 802.11b LAN is transmitting 64-byte frames back-to-back over a radio channel with a bit error rate of 10^{-7}. How many frames per second will be damaged on average?

26. An 802.16 network has a channel width of 20 MHz. How many bits/sec can be sent to a subscriber station?

27. Give two reasons why networks might use an error-correcting code instead of error detection and retransmission.

28. List two ways in which WiMAX is similar to 802.11, and two ways in which it is different from 802.11.

29. From Fig. 4-34, we see that a Bluetooth device can be in two piconets at the same time. Is there any reason why one device cannot be the master in both of them at the same time?

30. What is the maximum size of the data field for a 3-slot Bluetooth frame at basic rate? Explain your answer.

31. Figure 4-24 shows several physical layer protocols. Which of these is closest to the Bluetooth physical layer protocol? What is the biggest difference between the two?

32. It is mentioned in Section 4.6.6 that the efficiency of a 1-slot frame with repetition encoding is about 13% at basic data rate. What will the efficiency be if a 5-slot frame with repetition encoding is used at basic data rate instead?

33. Beacon frames in the frequency hopping spread spectrum variant of 802.11 contain the dwell time. Do you think the analogous beacon frames in Bluetooth also contain the dwell time? Discuss your answer.

34. Suppose that there are 10 RFID tags around an RFID reader. What is the best value of Q? How likely is it that one tag responds with no collision in a given slot?

35. List some of the security concerns of an RFID system.

36. A switch designed for use with fast Ethernet has a backplane that can move 10 Gbps. How many frames/sec can it handle in the worst case?

37. Briefly describe the difference between store-and-forward and cut-through switches.

38. Consider the extended LAN connected using bridges *B1* and *B2* in Fig. 4-41(b). Suppose the hash tables in the two bridges are empty. List all ports on which a packet will be forwarded for the following sequence of data transmissions:

(a) *A* sends a packet to *C*.
(b) *E* sends a packet to *F*.
(c) *F* sends a packet to *E*.
(d) *G* sends a packet to *E*.
(e) *D* sends a packet to *A*.
(f) *B* sends a packet to *F*.

39. Store-and-forward switches have an advantage over cut-through switches with respect to damaged frames. Explain what it is.

40. It is mentioned in Section 4.8.3 that some bridges may not even be present in the spanning tree. Outline a scenario where a bridge may not be present in the spanning tree.

41. To make VLANs work, configuration tables are needed in the bridges. What if the VLANs of Fig. 4-47 used hubs rather than switches? Do the hubs need configuration tables, too? Why or why not?

Patent Owner's Ex. 2010
Page 201

42. In Fig. 4-48, the switch in the legacy end domain on the right is a VLAN-aware switch. Would it be possible to use a legacy switch there? If so, how would that work? If not, why not?

43. Write a program to simulate the behavior of the CSMA/CD protocol over Ethernet when there are *N* stations ready to transmit while a frame is being transmitted. Your program should report the times when each station successfully starts sending its frame. Assume that a clock tick occurs once every slot time (51.2 μsec) and a collision detection and sending of a jamming sequence takes one slot time. All frames are the maximum length allowed.

Patent Owner's Ex. 2010
Page 202

5

THE NETWORK LAYER

The network layer is concerned with getting packets from the source all the way to the destination. Getting to the destination may require making many hops at intermediate routers along the way. This function clearly contrasts with that of the data link layer, which has the more modest goal of just moving frames from one end of a wire to the other. Thus, the network layer is the lowest layer that deals with end-to-end transmission.

To achieve its goals, the network layer must know about the topology of the network (i.e., the set of all routers and links) and choose appropriate paths through it, even for large networks. It must also take care when choosing routes to avoid overloading some of the communication lines and routers while leaving others idle. Finally, when the source and destination are in different networks, new problems occur. It is up to the network layer to deal with them. In this chapter we will study all these issues and illustrate them, primarily using the Internet and its network layer protocol, IP.

5.1 NETWORK LAYER DESIGN ISSUES

In the following sections, we will give an introduction to some of the issues that the designers of the network layer must grapple with. These issues include the service provided to the transport layer and the internal design of the network.

Patent Owner's Ex. 2010
Page 203

5.1.1 Store-and-Forward Packet Switching

Before starting to explain the details of the network layer, it is worth restating the context in which the network layer protocols operate. This context can be seen in Fig. 5-1. The major components of the network are the ISP's equipment (routers connected by transmission lines), shown inside the shaded oval, and the customers' equipment, shown outside the oval. Host *H1* is directly connected to one of the ISP's routers, *A*, perhaps as a home computer that is plugged into a DSL modem. In contrast, *H2* is on a LAN, which might be an office Ethernet, with a router, *F*, owned and operated by the customer. This router has a leased line to the ISP's equipment. We have shown *F* as being outside the oval because it does not belong to the ISP. For the purposes of this chapter, however, routers on customer premises are considered part of the ISP network because they run the same algorithms as the ISP's routers (and our main concern here is algorithms).

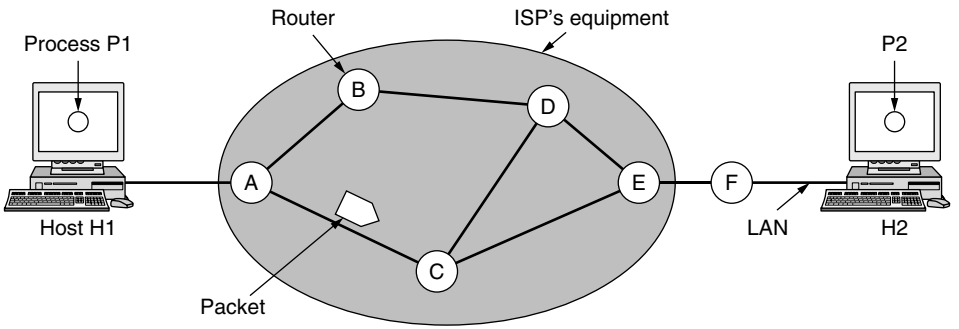


Figure 5-1. The environment of the network layer protocols.

This equipment is used as follows. A host with a packet to send transmits it to the nearest router, either on its own LAN or over a point-to-point link to the ISP. The packet is stored there until it has fully arrived and the link has finished its processing by verifying the checksum. Then it is forwarded to the next router along the path until it reaches the destination host, where it is delivered. This mechanism is store-and-forward packet switching, as we have seen in previous chapters.

5.1.2 Services Provided to the Transport Layer

The network layer provides services to the transport layer at the network layer/transport layer interface. An important question is precisely what kind of services the network layer provides to the transport layer. The services need to be carefully designed with the following goals in mind:

1. The services should be independent of the router technology.
2. The transport layer should be shielded from the number, type, and topology of the routers present.
3. The network addresses made available to the transport layer should use a uniform numbering plan, even across LANs and WANs.

Given these goals, the designers of the network layer have a lot of freedom in writing detailed specifications of the services to be offered to the transport layer. This freedom often degenerates into a raging battle between two warring factions. The discussion centers on whether the network layer should provide connection-oriented service or connectionless service.

One camp (represented by the Internet community) argues that the routers' job is moving packets around and nothing else. In this view (based on 40 years of experience with a real computer network), the network is inherently unreliable, no matter how it is designed. Therefore, the hosts should accept this fact and do error control (i.e., error detection and correction) and flow control themselves.

This viewpoint leads to the conclusion that the network service should be connectionless, with primitives SEND PACKET and RECEIVE PACKET and little else. In particular, no packet ordering and flow control should be done, because the hosts are going to do that anyway and there is usually little to be gained by doing it twice. This reasoning is an example of the **end-to-end argument**, a design principle that has been very influential in shaping the Internet (Saltzer et al., 1984). Furthermore, each packet must carry the full destination address, because each packet sent is carried independently of its predecessors, if any.

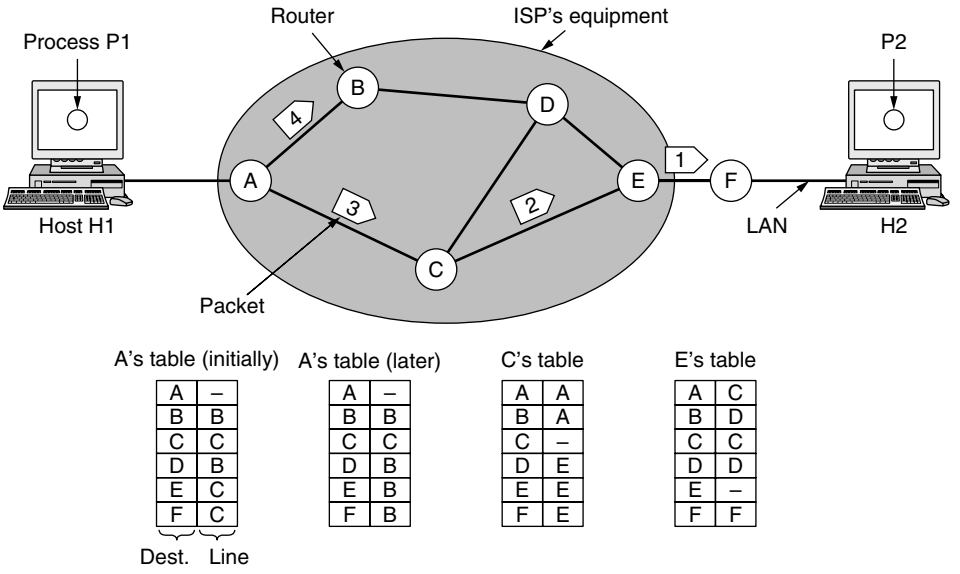
The other camp (represented by the telephone companies) argues that the network should provide a reliable, connection-oriented service. They claim that 100 years of successful experience with the worldwide telephone system is an excellent guide. In this view, quality of service is the dominant factor, and without connections in the network, quality of service is very difficult to achieve, especially for real-time traffic such as voice and video.

Even after several decades, this controversy is still very much alive. Early, widely used data networks, such as X.25 in the 1970s and its successor Frame Relay in the 1980s, were connection-oriented. However, since the days of the ARPANET and the early Internet, connectionless network layers have grown tremendously in popularity. The IP protocol is now an ever-present symbol of success. It was undeterred by a connection-oriented technology called ATM that was developed to overthrow it in the 1980s; instead, it is ATM that is now found in niche uses and IP that is taking over telephone networks. Under the covers, however, the Internet is evolving connection-oriented features as quality of service becomes more important. Two examples of connection-oriented technologies are MPLS (MultiProtocol Label Switching), which we will describe in this chapter, and VLANs, which we saw in Chap. 4. Both technologies are widely used.

5.1.3 Implementation of Connectionless Service

Having looked at the two classes of service the network layer can provide to its users, it is time to see how this layer works inside. Two different organizations are possible, depending on the type of service offered. If connectionless service is offered, packets are injected into the network individually and routed independently of each other. No advance setup is needed. In this context, the packets are frequently called **datagrams** (in analogy with telegrams) and the network is called a **datagram network**. If connection-oriented service is used, a path from the source router all the way to the destination router must be established before any data packets can be sent. This connection is called a **VC** (**virtual circuit**), in analogy with the physical circuits set up by the telephone system, and the network is called a **virtual-circuit network**. In this section, we will examine datagram networks; in the next one, we will examine virtual-circuit networks.

Let us now see how a datagram network works. Suppose that the process *P1* in Fig. 5-2 has a long message for *P2*. It hands the message to the transport layer, with instructions to deliver it to process *P2* on host *H2*. The transport layer code runs on *H1*, typically within the operating system. It prepends a transport header to the front of the message and hands the result to the network layer, probably just another procedure within the operating system.



A	–
B	B
C	C
D	B
E	C
F	C

A	–
B	B
C	C
D	B
E	B
F	B

A	A
B	A
C	–
D	E
E	E
F	E

A	C
B	D
C	C
D	D
E	–
F	F

Figure 5-2. Routing within a datagram network.

Let us assume for this example that the message is four times longer than the maximum packet size, so the network layer has to break it into four packets, 1, 2,

3, and 4, and send each of them in turn to router *A* using some point-to-point protocol, for example, PPP. At this point the ISP takes over. Every router has an internal table telling it where to send packets for each of the possible destinations. Each table entry is a pair consisting of a destination and the outgoing line to use for that destination. Only directly connected lines can be used. For example, in Fig. 5-2, *A* has only two outgoing lines—to *B* and to *C*—so every incoming packet must be sent to one of these routers, even if the ultimate destination is to some other router. *A*'s initial routing table is shown in the figure under the label "initially."

At *A*, packets 1, 2, and 3 are stored briefly, having arrived on the incoming link and had their checksums verified. Then each packet is forwarded according to *A*'s table, onto the outgoing link to *C* within a new frame. Packet 1 is then forwarded to *E* and then to *F*. When it gets to *F*, it is sent within a frame over the LAN to *H2*. Packets 2 and 3 follow the same route.

However, something different happens to packet 4. When it gets to *A* it is sent to router *B*, even though it is also destined for *F*. For some reason, *A* decided to send packet 4 via a different route than that of the first three packets. Perhaps it has learned of a traffic jam somewhere along the *ACE* path and updated its routing table, as shown under the label "later." The algorithm that manages the tables and makes the routing decisions is called the **routing algorithm**. Routing algorithms are one of the main topics we will study in this chapter. There are several different kinds of them, as we will see.

IP (Internet Protocol), which is the basis for the entire Internet, is the dominant example of a connectionless network service. Each packet carries a destination IP address that routers use to individually forward each packet. The addresses are 32 bits in IPv4 packets and 128 bits in IPv6 packets. We will describe IP in much detail later in this chapter.

5.1.4 Implementation of Connection-Oriented Service

For connection-oriented service, we need a virtual-circuit network. Let us see how that works. The idea behind virtual circuits is to avoid having to choose a new route for every packet sent, as in Fig. 5-2. Instead, when a connection is established, a route from the source machine to the destination machine is chosen as part of the connection setup and stored in tables inside the routers. That route is used for all traffic flowing over the connection, exactly the same way that the telephone system works. When the connection is released, the virtual circuit is also terminated. With connection-oriented service, each packet carries an identifier telling which virtual circuit it belongs to.

As an example, consider the situation shown in Fig. 5-3. Here, host *H1* has established connection 1 with host *H2*. This connection is remembered as the first entry in each of the routing tables. The first line of *A*'s table says that if a packet

bearing connection identifier 1 comes in from *H1*, it is to be sent to router *C* and given connection identifier 1. Similarly, the first entry at *C* routes the packet to *E*, also with connection identifier 1.

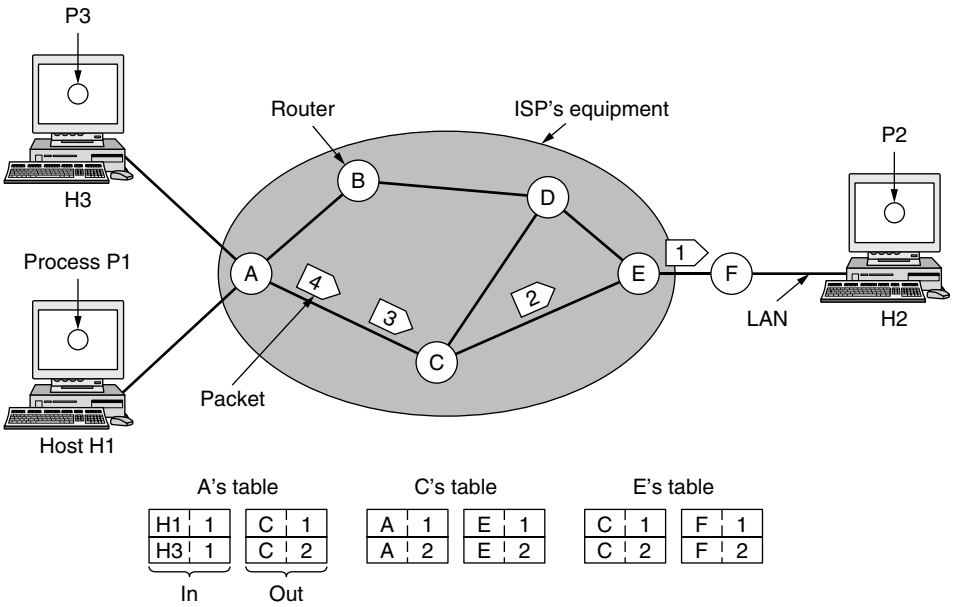


Figure 5-3. Routing within a virtual-circuit network.

Now let us consider what happens if *H3* also wants to establish a connection to *H2*. It chooses connection identifier 1 (because it is initiating the connection and this is its only connection) and tells the network to establish the virtual circuit. This leads to the second row in the tables. Note that we have a conflict here because although *A* can easily distinguish connection 1 packets from *H1* from connection 1 packets from *H3*, *C* cannot do this. For this reason, *A* assigns a different connection identifier to the outgoing traffic for the second connection. Avoiding conflicts of this kind is why routers need the ability to replace connection identifiers in outgoing packets.

In some contexts, this process is called **label switching**. An example of a connection-oriented network service is **MPLS** (**MultiProtocol Label Switching**). It is used within ISP networks in the Internet, with IP packets wrapped in an MPLS header having a 20-bit connection identifier or label. MPLS is often hidden from customers, with the ISP establishing long-term connections for large amounts of traffic, but it is increasingly being used to help when quality of service is important but also with other ISP traffic management tasks. We will have more to say about MPLS later in this chapter.

5.1.5 Comparison of Virtual-Circuit and Datagram Networks

Both virtual circuits and datagrams have their supporters and their detractors. We will now attempt to summarize both sets of arguments. The major issues are listed in Fig. 5-4, although purists could probably find a counterexample for everything in the figure.

Issue	Datagram network	Virtual-circuit network
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

Figure 5-4. Comparison of datagram and virtual-circuit networks.

Inside the network, several trade-offs exist between virtual circuits and datagrams. One trade-off is setup time versus address parsing time. Using virtual circuits requires a setup phase, which takes time and consumes resources. However, once this price is paid, figuring out what to do with a data packet in a virtual-circuit network is easy: the router just uses the circuit number to index into a table to find out where the packet goes. In a datagram network, no setup is needed but a more complicated lookup procedure is required to locate the entry for the destination.

A related issue is that the destination addresses used in datagram networks are longer than circuit numbers used in virtual-circuit networks because they have a global meaning. If the packets tend to be fairly short, including a full destination address in every packet may represent a significant amount of overhead, and hence a waste of bandwidth.

Yet another issue is the amount of table space required in router memory. A datagram network needs to have an entry for every possible destination, whereas a virtual-circuit network just needs an entry for each virtual circuit. However, this

advantage is somewhat illusory since connection setup packets have to be routed too, and they use destination addresses, the same as datagrams do.

Virtual circuits have some advantages in guaranteeing quality of service and avoiding congestion within the network because resources (e.g., buffers, bandwidth, and CPU cycles) can be reserved in advance, when the connection is established. Once the packets start arriving, the necessary bandwidth and router capacity will be there. With a datagram network, congestion avoidance is more difficult.

For transaction processing systems (e.g., stores calling up to verify credit card purchases), the overhead required to set up and clear a virtual circuit may easily dwarf the use of the circuit. If the majority of the traffic is expected to be of this kind, the use of virtual circuits inside the network makes little sense. On the other hand, for long-running uses such as VPN traffic between two corporate offices, permanent virtual circuits (that are set up manually and last for months or years) may be useful.

Virtual circuits also have a vulnerability problem. If a router crashes and loses its memory, even if it comes back up a second later, all the virtual circuits passing through it will have to be aborted. In contrast, if a datagram router goes down, only those users whose packets were queued in the router at the time need suffer (and probably not even then since the sender is likely to retransmit them shortly). The loss of a communication line is fatal to virtual circuits using it, but can easily be compensated for if datagrams are used. Datagrams also allow the routers to balance the traffic throughout the network, since routes can be changed partway through a long sequence of packet transmissions.

5.2 ROUTING ALGORITHMS

The main function of the network layer is routing packets from the source machine to the destination machine. In most networks, packets will require multiple hops to make the journey. The only notable exception is for broadcast networks, but even here routing is an issue if the source and destination are not on the same network segment. The algorithms that choose the routes and the data structures that they use are a major area of network layer design.

The **routing algorithm** is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the network uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the network uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the already established route. The latter case is sometimes called **session routing** because a route remains in force for an entire session (e.g., while logged in over a VPN).

It is sometimes useful to make a distinction between routing, which is making the decision which routes to use, and forwarding, which is what happens when a packet arrives. One can think of a router as having two processes inside it. One of them handles each packet as it arrives, looking up the outgoing line to use for it in the routing tables. This process is **forwarding**. The other process is responsible for filling in and updating the routing tables. That is where the routing algorithm comes into play.

Regardless of whether routes are chosen independently for each packet sent or only when new connections are established, certain properties are desirable in a routing algorithm: correctness, simplicity, robustness, stability, fairness, and efficiency. Correctness and simplicity hardly require comment, but the need for robustness may be less obvious at first. Once a major network comes on the air, it may be expected to run continuously for years without system-wide failures. During that period there will be hardware and software failures of all kinds. Hosts, routers, and lines will fail repeatedly, and the topology will change many times. The routing algorithm should be able to cope with changes in the topology and traffic without requiring all jobs in all hosts to be aborted. Imagine the havoc if the network needed to be rebooted every time some router crashed!

Stability is also an important goal for the routing algorithm. There exist routing algorithms that never converge to a fixed set of paths, no matter how long they run. A stable algorithm reaches equilibrium and stays there. It should converge quickly too, since communication may be disrupted until the routing algorithm has reached equilibrium.

Fairness and efficiency may sound obvious—surely no reasonable person would oppose them—but as it turns out, they are often contradictory goals. As a simple example of this conflict, look at Fig. 5-5. Suppose that there is enough traffic between A and A', between B and B', and between C and C' to saturate the horizontal links. To maximize the total flow, the X to X' traffic should be shut off altogether. Unfortunately, X and X' may not see it that way. Evidently, some compromise between global efficiency and fairness to individual connections is needed.

Before we can even attempt to find trade-offs between fairness and efficiency, we must decide what it is we seek to optimize. Minimizing the mean packet delay is an obvious candidate to send traffic through the network effectively, but so is maximizing total network throughput. Furthermore, these two goals are also in conflict, since operating any queueing system near capacity implies a long queueing delay. As a compromise, many networks attempt to minimize the distance a packet must travel, or simply reduce the number of hops a packet must make. Either choice tends to improve the delay and also reduce the amount of bandwidth consumed per packet, which tends to improve the overall network throughput as well.

Routing algorithms can be grouped into two major classes: nonadaptive and adaptive. **Nonadaptive algorithms** do not base their routing decisions on any

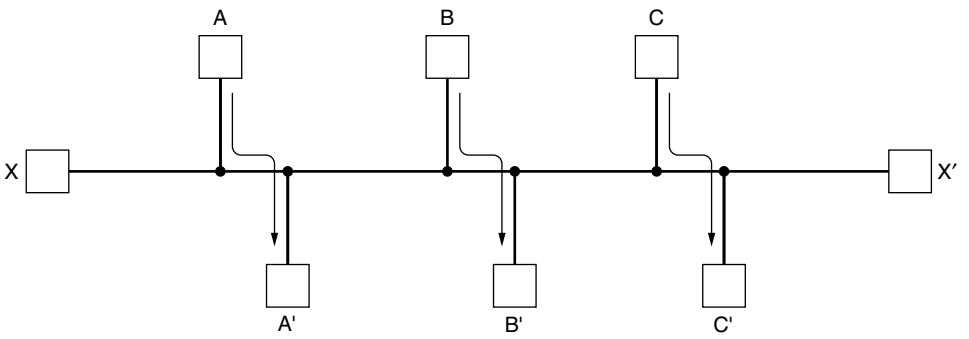


Figure 5-5. Network with a conflict between fairness and efficiency.

measurements or estimates of the current topology and traffic. Instead, the choice of the route to use to get from *I* to *J* (for all *I* and *J*) is computed in advance, off-line, and downloaded to the routers when the network is booted. This procedure is sometimes called **static routing**. Because it does not respond to failures, static routing is mostly useful for situations in which the routing choice is clear. For example, router *F* in Fig. 5-3 should send packets headed into the network to router *E* regardless of the ultimate destination.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and sometimes changes in the traffic as well. These **dynamic routing** algorithms differ in where they get their information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., when the topology changes, or every ΔT seconds as the load changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

In the following sections, we will discuss a variety of routing algorithms. The algorithms cover delivery models besides sending a packet from a source to a destination. Sometimes the goal is to send the packet to multiple, all, or one of a set of destinations. All of the routing algorithms we describe here make decisions based on the topology; we defer the possibility of decisions based on the traffic levels to Sec 5.3.

5.2.1 The Optimality Principle

Before we get into specific algorithms, it may be helpful to note that one can make a general statement about optimal routes without regard to network topology or traffic. This statement is known as the **optimality principle** (Bellman, 1957). It states that if router *J* is on the optimal path from router *I* to router *K*,

then the optimal path from J to K also falls along the same route. To see this, call the part of the route from I to J r_1 and the rest of the route r_2. If a route better than r_2 existed from J to K, it could be concatenated with r_1 to improve the route from I to K, contradicting our statement that $r_1 r_2$ is optimal.

As a direct consequence of the optimality principle, we can see that the set of optimal routes from all sources to a given destination form a tree rooted at the destination. Such a tree is called a **sink tree** and is illustrated in Fig. 5-6(b), where the distance metric is the number of hops. The goal of all routing algorithms is to discover and use the sink trees for all routers.

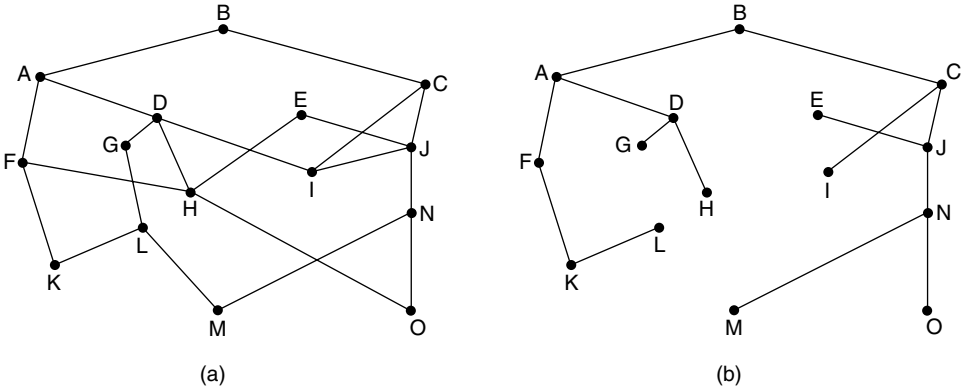


Figure 5-6. (a) A network. (b) A sink tree for router *B*.

Note that a sink tree is not necessarily unique; other trees with the same path lengths may exist. If we allow all of the possible paths to be chosen, the tree becomes a more general structure called a **DAG** (**Directed Acyclic Graph**). DAGs have no loops. We will use sink trees as a convenient shorthand for both cases. Both cases also depend on the technical assumption that the paths do not interfere with each other so, for example, a traffic jam on one path will not cause another path to divert.

Since a sink tree is indeed a tree, it does not contain any loops, so each packet will be delivered within a finite and bounded number of hops. In practice, life is not quite this easy. Links and routers can go down and come back up during operation, so different routers may have different ideas about the current topology. Also, we have quietly finessed the issue of whether each router has to individually acquire the information on which to base its sink tree computation or whether this information is collected by some other means. We will come back to these issues shortly. Nevertheless, the optimality principle and the sink tree provide a benchmark against which other routing algorithms can be measured.

5.2.2 Shortest Path Algorithm

Let us begin our study of routing algorithms with a simple technique for computing optimal paths given a complete picture of the network. These paths are the ones that we want a distributed routing algorithm to find, even though not all routers may know all of the details of the network.

The idea is to build a graph of the network, with each node of the graph representing a router and each edge of the graph representing a communication line, or link. To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

The concept of a **shortest path** deserves some explanation. One way of measuring path length is the number of hops. Using this metric, the paths *ABC* and *ABE* in Fig. 5-7 are equally long. Another metric is the geographic distance in kilometers, in which case *ABC* is clearly much longer than *ABE* (assuming the figure is drawn to scale).

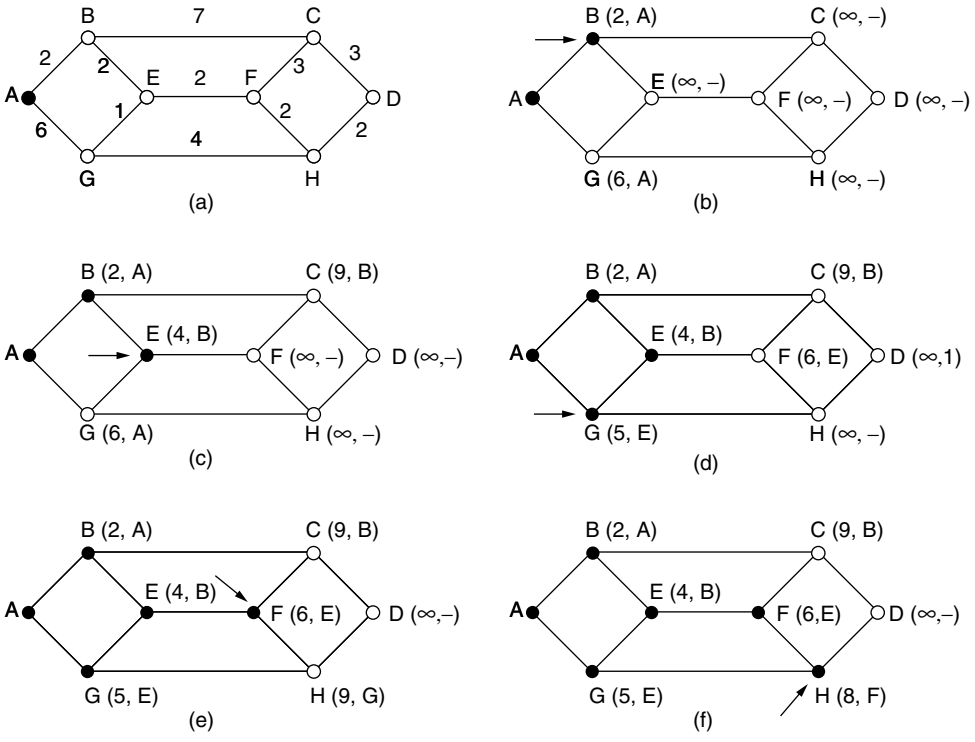


Figure 5-7. The first six steps used in computing the shortest path from *A* to *D*. The arrows indicate the working node.

However, many other metrics besides hops and physical distance are also possible. For example, each edge could be labeled with the mean delay of a standard test packet, as measured by hourly runs. With this graph labeling, the shortest path is the fastest path rather than the path with the fewest edges or kilometers.

In the general case, the labels on the edges could be computed as a function of the distance, bandwidth, average traffic, communication cost, measured delay, and other factors. By changing the weighting function, the algorithm would then compute the "shortest" path measured according to any one of a number of criteria or to a combination of criteria.

Several algorithms for computing the shortest path between two nodes of a graph are known. This one is due to Dijkstra (1959) and finds the shortest paths between a source and all destinations in the network. Each node is labeled (in parentheses) with its distance from the source node along the best known path. The distances must be non-negative, as they will be if they are based on real quantities like bandwidth and delay. Initially, no paths are known, so all nodes are labeled with infinity. As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may be either tentative or permanent. Initially, all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

To illustrate how the labeling algorithm works, look at the weighted, undirected graph of Fig. 5-7(a), where the weights represent, for example, distance. We want to find the shortest path from *A* to *D*. We start out by marking node *A* as permanent, indicated by a filled-in circle. Then we examine, in turn, each of the nodes adjacent to *A* (the working node), relabeling each one with the distance to *A*. Whenever a node is relabeled, we also label it with the node from which the probe was made so that we can reconstruct the final path later. If the network had more than one shortest path from *A* to *D* and we wanted to find all of them, we would need to remember all of the probe nodes that could reach a node with the same distance.

Having examined each of the nodes adjacent to *A*, we examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig. 5-7(b). This one becomes the new working node.

We now start at *B* and examine all nodes adjacent to it. If the sum of the label on *B* and the distance from *B* to the node being considered is less than the label on that node, we have a shorter path, so the node is relabeled.

After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively labeled node with the smallest value. This node is made permanent and becomes the working node for the next round. Figure 5-7 shows the first six steps of the algorithm.

To see why the algorithm works, look at Fig. 5-7(c). At this point we have just made *E* permanent. Suppose that there were a shorter path than *ABE*, say

AXYZE (for some *X* and *Y*). There are two possibilities: either node *Z* has already been made permanent, or it has not been. If it has, then *E* has already been probed (on the round following the one when *Z* was made permanent), so the *AXYZE* path has not escaped our attention and thus cannot be a shorter path.

Now consider the case where *Z* is still tentatively labeled. If the label at *Z* is greater than or equal to that at *E*, then *AXYZE* cannot be a shorter path than *ABE*. If the label is less than that of *E*, then *Z* and not *E* will become permanent first, allowing *E* to be probed from *Z*.

This algorithm is given in Fig. 5-8. The global variables *n* and *dist* describe the graph and are initialized before *shortest_path* is called. The only difference between the program and the algorithm described above is that in Fig. 5-8, we compute the shortest path starting at the terminal node, *t*, rather than at the source node, *s*.

Since the shortest paths from *t* to *s* in an undirected graph are the same as the shortest paths from *s* to *t*, it does not matter at which end we begin. The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When the final path is copied into the output variable, *path*, the path is thus reversed. The two reversal effects cancel, and the answer is produced in the correct order.

5.2.3 Flooding

When a routing algorithm is implemented, each router must make decisions based on local knowledge, not the complete picture of the network. A simple local technique is **flooding**, in which every incoming packet is sent out on every outgoing line except the one it arrived on.

Flooding obviously generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process. One such measure is to have a hop counter contained in the header of each packet that is decremented at each hop, with the packet being discarded when the counter reaches zero. Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely, the full diameter of the network.

Flooding with a hop count can produce an exponential number of duplicate packets as the hop count grows and routers duplicate packets they have seen before. A better technique for damming the flood is to have routers keep track of which packets have been flooded, to avoid sending them out a second time. One way to achieve this goal is to have the source router put a sequence number in each packet it receives from its hosts. Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen. If an incoming packet is on the list, it is not flooded.

```
#define MAX_NODES 1024                          /* maximum number of nodes */
#define INFINITY 1000000000                     /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];              /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                                /* the path being worked on */
      int predecessor;                          /* previous node */
      int length;                               /* length from source to this node */
      enum {permanent, tentative} label;        /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *p;

  for (p = &state[0]; p < &state[n]; p++) {     /* initialize state */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                                        /* k is the initial working node */
  do {                                          /* Is there a better path from k? */
      for (i = 0; i < n; i++)                   /* this graph has n nodes */
          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];
              }
          }

      /* Find the tentatively labeled node with the smallest label. */
      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);

  /* Copy the path into the output array. */
  i = 0;  k = s;
  do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Figure 5-8. Dijkstra's algorithm to compute the shortest path through a graph.

To prevent the list from growing without bound, each list should be augmented by a counter, *k*, meaning that all sequence numbers through *k* have been seen. When a packet comes in, it is easy to check if the packet has already been

flooded (by comparing its sequence number to *k*; if so, it is discarded. Furthermore, the full list below *k* is not needed, since *k* effectively summarizes it.

Flooding is not practical for sending most packets, but it does have some important uses. First, it ensures that a packet is delivered to every node in the network. This may be wasteful if there is a single destination that needs the packet, but it is effective for broadcasting information. In wireless networks, all messages transmitted by a station can be received by all other stations within its radio range, which is, in fact, flooding, and some algorithms utilize this property.

Second, flooding is tremendously robust. Even if large numbers of routers are blown to bits (e.g., in a military network located in a war zone), flooding will find a path if one exists, to get a packet to its destination. Flooding also requires little in the way of setup. The routers only need to know their neighbors. This means that flooding can be used as a building block for other routing algorithms that are more efficient but need more in the way of setup. Flooding can also be used as a metric against which other routing algorithms can be compared. Flooding always chooses the shortest path because it chooses every possible path in parallel. Consequently, no other algorithm can produce a shorter delay (if we ignore the overhead generated by the flooding process itself).

5.2.4 Distance Vector Routing

Computer networks generally use dynamic routing algorithms that are more complex than flooding, but more efficient because they find shortest paths for the current topology. Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular. In this section, we will look at the former algorithm. In the following section, we will study the latter algorithm.

A **distance vector routing** algorithm operates by having each router maintain a table (i.e., a vector) giving the best known distance to each destination and which link to use to get there. These tables are updated by exchanging information with the neighbors. Eventually, every router knows the best link to reach each destination.

The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for each router in the network. This entry has two parts: the preferred outgoing line to use for that destination and an estimate of the distance to that destination. The distance might be measured as the number of hops or using another metric, as we discussed for computing shortest paths.

The router is assumed to know the ''distance'' to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is propagation delay, the

router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can.

As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbors. Once every T msec, each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor X, with X_i being X's estimate of how long it takes to get to router i. If the router knows that the delay to X is m msec, it also knows that it can reach router i via X in $X_i + m$ msec. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding link in its new routing table. Note that the old routing table is not used in the calculation.

This updating process is illustrated in Fig. 5-9. Part (a) shows a network. The first four columns of part (b) show the delay vectors received from the neighbors of router J. A claims to have a 12-msec delay to B, a 25-msec delay to C, a 40-msec delay to D, etc. Suppose that J has measured or estimated its delay to its neighbors, A, I, H, and K, as 8, 10, 12, and 6 msec, respectively.

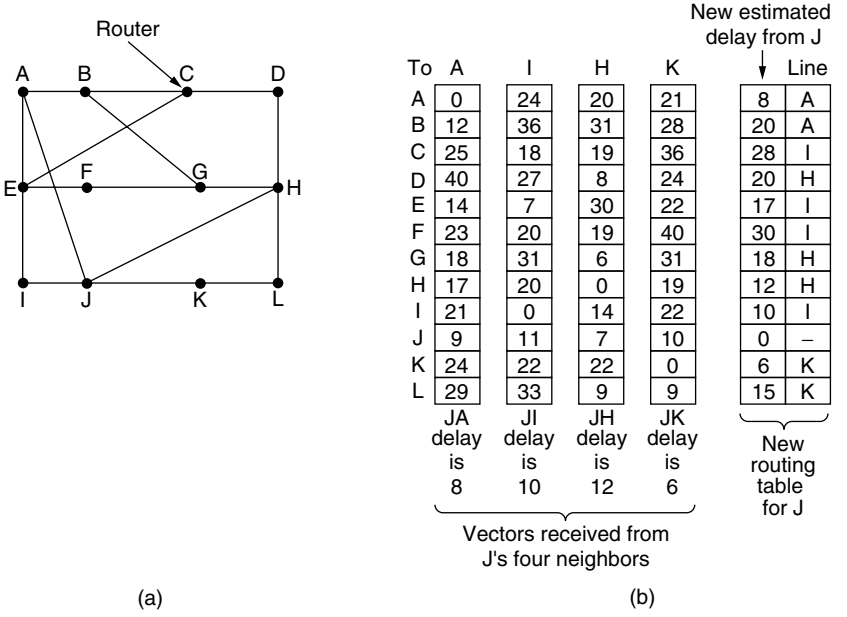


Figure 5-9. (a) A network. (b) Input from A, I, H, K, and the new routing table for J.

Consider how J computes its new route to router G. It knows that it can get to A in 8 msec, and furthermore A claims to be able to get to G in 18 msec, so J knows it can count on a delay of 26 msec to G if it forwards packets bound for G

to *A*. Similarly, it computes the delay to *G* via *I*, *H*, and *K* as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec, respectively. The best of these values is 18, so it makes an entry in its routing table that the delay to *G* is 18 msec and that the route to use is via *H*. The same calculation is performed for all the other destinations, with the new routing table shown in the last column of the figure.

The Count-to-Infinity Problem

The settling of routes to best paths across the network is called **convergence**. Distance vector routing is useful as a simple technique by which routers can collectively compute shortest paths, but it has a serious drawback in practice: although it converges to the correct answer, it may do so slowly. In particular, it reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination *X* is long. If, on the next exchange, neighbor *A* suddenly reports a short delay to *X*, the router just switches over to using the line to *A* to send traffic to *X*. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five-node (linear) network of Fig. 5-10, where the delay metric is the number of hops. Suppose *A* is down initially and all the other routers know this. In other words, they have all recorded the delay to *A* as infinity.

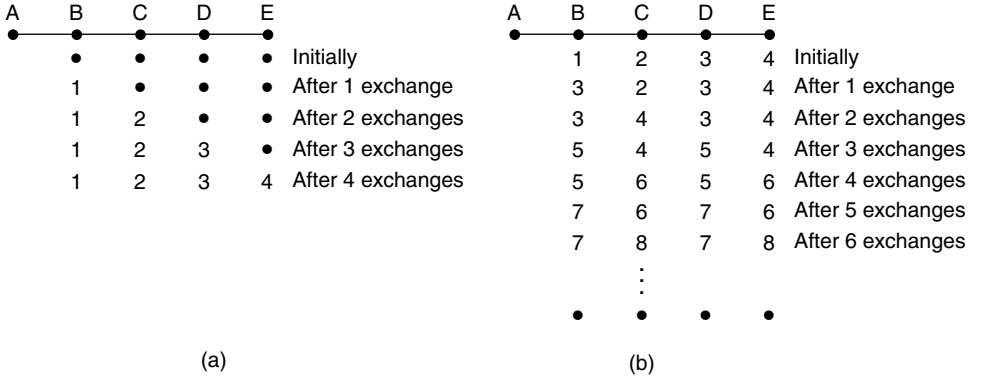


Figure 5-10. The count-to-infinity problem.

When *A* comes up, the other routers learn about it via the vector exchanges. For simplicity, we will assume that there is a gigantic gong somewhere that is struck periodically to initiate a vector exchange at all routers simultaneously. At the time of the first exchange, *B* learns that its left-hand neighbor has zero delay to *A*. *B* now makes an entry in its routing table indicating that *A* is one hop away to the left. All the other routers still think that *A* is down. At this point, the routing table entries for *A* are as shown in the second row of Fig. 5-10(a). On the next

exchange, *C* learns that *B* has a path of length 1 to *A*, so it updates its routing table to indicate a path of length 2, but *D* and *E* do not hear the good news until later. Clearly, the good news is spreading at the rate of one hop per exchange. In a network whose longest path is of length *N* hops, within *N* exchanges everyone will know about newly revived links and routers.

Now let us consider the situation of Fig. 5-10(b), in which all the links and routers are initially up. Routers *B*, *C*, *D*, and *E* have distances to *A* of 1, 2, 3, and 4 hops, respectively. Suddenly, either *A* goes down or the link between *A* and *B* is cut (which is effectively the same thing from *B*'s point of view).

At the first packet exchange, *B* does not hear anything from *A*. Fortunately, *C* says "Do not worry; I have a path to *A* of length 2." Little does *B* suspect that *C*'s path runs through *B* itself. For all *B* knows, *C* might have ten links all with separate paths to *A* of length 2. As a result, *B* thinks it can reach *A* via *C*, with a path length of 3. *D* and *E* do not update their entries for *A* on the first exchange.

On the second exchange, *C* notices that each of its neighbors claims to have a path to *A* of length 3. It picks one of them at random and makes its new distance to *A* 4, as shown in the third row of Fig. 5-10(b). Subsequent exchanges produce the history shown in the rest of Fig. 5-10(b).

From this figure, it should be clear why bad news travels slowly: no router ever has a value more than one higher than the minimum of all its neighbors. Gradually, all routers work their way up to infinity, but the number of exchanges required depends on the numerical value used for infinity. For this reason, it is wise to set infinity to the longest path plus 1.

Not entirely surprisingly, this problem is known as the **count-to-infinity** problem. There have been many attempts to solve it, for example, preventing routers from advertising their best paths back to the neighbors from which they heard them with the split horizon with poisoned reverse rule discussed in RFC 1058. However, none of these heuristics work well in practice despite the colorful names. The core of the problem is that when *X* tells *Y* that it has a path somewhere, *Y* has no way of knowing whether it itself is on the path.

5.2.5 Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. The primary problem that caused its demise was that the algorithm often took too long to converge after the network topology changed (due to the count-to-infinity problem). Consequently, it was replaced by an entirely new algorithm, now called **link state routing**. Variants of link state routing called IS-IS and OSPF are the routing algorithms that are most widely used inside large networks and the Internet today.

The idea behind link state routing is fairly simple and can be stated as five parts. Each router must do the following things to make it work:

1. Discover its neighbors and learn their network addresses.
2. Set the distance or cost metric to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to and receive packets from all other routers.
5. Compute the shortest path to every other router.

In effect, the complete topology is distributed to every router. Then Dijkstra's algorithm can be run at each router to find the shortest path to every other router. Below we will consider each of these five steps in more detail.

Learning about the Neighbors

When a router is booted, its first task is to learn who its neighbors are. It accomplishes this goal by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send back a reply giving its name. These names must be globally unique because when a distant router later hears that three routers are all connected to *F*, it is essential that it can determine whether all three mean the same *F*.

When two or more routers are connected by a broadcast link (e.g., a switch, ring, or classic Ethernet), the situation is slightly more complicated. Fig. 5-11(a) illustrates a broadcast LAN to which three routers, *A*, *C*, and *F*, are directly connected. Each of these routers is connected to one or more additional routers, as shown.

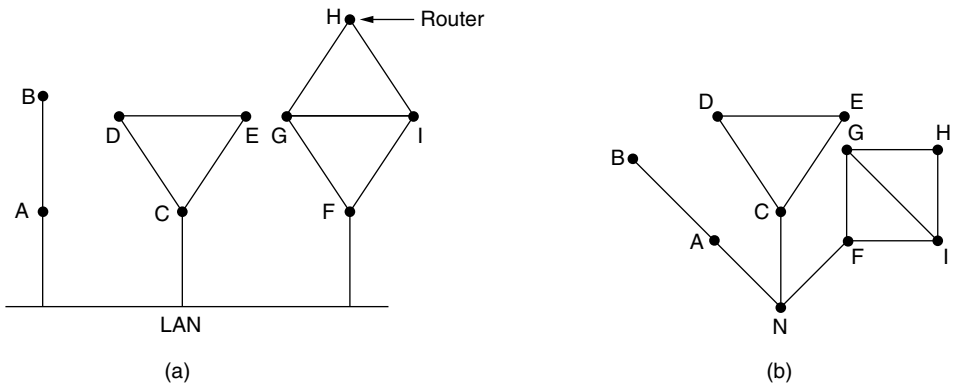


Figure 5-11. (a) Nine routers and a broadcast LAN. (b) A graph model of (a).

The broadcast LAN provides connectivity between each pair of attached routers. However, modeling the LAN as many point-to-point links increases the size

of the topology and leads to wasteful messages. A better way to model the LAN is to consider it as a node itself, as shown in Fig. 5-11(b). Here, we have introduced a new, artificial node, *N*, to which *A*, *C*, and *F* are connected. One **designated router** on the LAN is selected to play the role of *N* in the routing protocol. The fact that it is possible to go from *A* to *C* on the LAN is represented by the path *ANC* here.

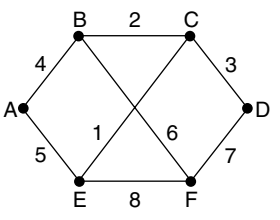
Setting Link Costs

The link state routing algorithm requires each link to have a distance or cost metric for finding shortest paths. The cost to reach neighbors can be set automatically, or configured by the network operator. A common choice is to make the cost inversely proportional to the bandwidth of the link. For example, 1-Gbps Ethernet may have a cost of 1 and 100-Mbps Ethernet a cost of 10. This makes higher-capacity paths better choices.

If the network is geographically spread out, the delay of the links may be factored into the cost so that paths over shorter links are better choices. The most direct way to determine this delay is to send over the line a special ECHO packet that the other side is required to send back immediately. By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of the delay.

Building Link State Packets

Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data. The packet starts with the identity of the sender, followed by a sequence number and age (to be described later) and a list of neighbors. The cost to each neighbor is also given. An example network is presented in Fig. 5-12(a) with costs shown as labels on the lines. The corresponding link state packets for all six routers are shown in Fig. 5-12(b).



(a)

Link State Packets

A		B		C		D		E		F	
Seq.		Seq.		Seq.		Seq.		Seq.		Seq.	
Age		Age		Age		Age		Age		Age	
B	4	A	4	B	2	C	3	A	5	B	6
E	5	C	2	D	3	F	7	C	1	D	7
		F	6	E	1			F	8	E	8

(b)

Figure 5-12. (a) A network. (b) The link state packets for this network.

Building the link state packets is easy. The hard part is determining when to build them. One possibility is to build them periodically, that is, at regular intervals. Another possibility is to build them when some significant event occurs, such as a line or neighbor going down or coming back up again or changing its properties appreciably.

Distributing the Link State Packets

The trickiest part of the algorithm is distributing the link state packets. All of the routers must get all of the link state packets quickly and reliably. If different routers are using different versions of the topology, the routes they compute can have inconsistencies such as loops, unreachable machines, and other problems.

First, we will describe the basic distribution algorithm. After that we will give some refinements. The fundamental idea is to use flooding to distribute the link state packets to all routers. To keep the flood in check, each packet contains a sequence number that is incremented for each new packet sent. Routers keep track of all the (source router, sequence) pairs they see. When a new link state packet comes in, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded. If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete as the router has more recent data.

This algorithm has a few problems, but they are manageable. First, if the sequence numbers wrap around, confusion will reign. The solution here is to use a 32-bit sequence number. With one link state packet per second, it would take 137 years to wrap around, so this possibility can be ignored.

Second, if a router ever crashes, it will lose track of its sequence number. If it starts again at 0, the next packet it sends will be rejected as a duplicate.

Third, if a sequence number is ever corrupted and 65,540 is received instead of 4 (a 1-bit error), packets 5 through 65,540 will be rejected as obsolete, since the current sequence number will be thought to be 65,540.

The solution to all these problems is to include the age of each packet after the sequence number and decrement it once per second. When the age hits zero, the information from that router is discarded. Normally, a new packet comes in, say, every 10 sec, so router information only times out when a router is down (or six consecutive packets have been lost, an unlikely event). The *Age* field is also decremented by each router during the initial flooding process, to make sure no packet can get lost and live for an indefinite period of time (a packet whose age is zero is discarded).

Some refinements to this algorithm make it more robust. When a link state packet comes in to a router for flooding, it is not queued for transmission immediately. Instead, it is put in a holding area to wait a short while in case more links are coming up or going down. If another link state packet from the same source comes in before the first packet is transmitted, their sequence numbers are

compared. If they are equal, the duplicate is discarded. If they are different, the older one is thrown out. To guard against errors on the links, all link state packets are acknowledged.

The data structure used by router *B* for the network shown in Fig. 5-12(a) is depicted in Fig. 5-13. Each row here corresponds to a recently arrived, but as yet not fully processed, link state packet. The table records where the packet originated, its sequence number and age, and the data. In addition, there are send and acknowledgement flags for each of *B*'s three links (to *A*, *C*, and *F*, respectively). The send flags mean that the packet must be sent on the indicated link. The acknowledgement flags mean that it must be acknowledged there.

			Send flags			ACK flags			
Source	Seq.	Age	A	C	F	A	C	F	Data
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	

Figure 5-13. The packet buffer for router *B* in Fig. 5-12(a).

In Fig. 5-13, the link state packet from *A* arrives directly, so it must be sent to *C* and *F* and acknowledged to *A*, as indicated by the flag bits. Similarly, the packet from *F* has to be forwarded to *A* and *C* and acknowledged to *F*.

However, the situation with the third packet, from *E*, is different. It arrives twice, once via *EAB* and once via *EFB*. Consequently, it has to be sent only to *C* but must be acknowledged to both *A* and *F*, as indicated by the bits.

If a duplicate arrives while the original is still in the buffer, bits have to be changed. For example, if a copy of *C*'s state arrives from *F* before the fourth entry in the table has been forwarded, the six bits will be changed to 100011 to indicate that the packet must be acknowledged to *F* but not sent there.

Computing the New Routes

Once a router has accumulated a full set of link state packets, it can construct the entire network graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The different directions may even have different costs. The shortest-path computations may then find different paths from router *A* to *B* than from router *B* to *A*.

Now Dijkstra's algorithm can be run locally to construct the shortest paths to all possible destinations. The results of this algorithm tell the router which link to

use to reach each destination. This information is installed in the routing tables, and normal operation is resumed.

Compared to distance vector routing, link state routing requires more memory and computation. For a network with n routers, each of which has k neighbors, the memory required to store the input data is proportional to kn, which is at least as large as a routing table listing all the destinations. Also, the computation time grows faster than kn, even with the most efficient data structures, an issue in large networks. Nevertheless, in many practical situations, link state routing works well because it does not suffer from slow convergence problems.

Link state routing is widely used in actual networks, so a few words about some example protocols are in order. Many ISPs use the **IS-IS** (**Intermediate System-Intermediate System**) link state protocol (Oran, 1990). It was designed for an early network called DECnet, later adopted by ISO for use with the OSI protocols and then modified to handle other protocols as well, most notably, IP. **OSPF** (**Open Shortest Path First**) is the other main link state protocol. It was designed by IETF several years after IS-IS and adopted many of the innovations designed for IS-IS. These innovations include a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics. As a consequence, there is very little difference between IS-IS and OSPF. The most important difference is that IS-IS can carry information about multiple network layer protocols at the same time (e.g., IP, IPX, and AppleTalk). OSPF does not have this feature, and it is an advantage in large multiprotocol environments. We will go over OSPF in Sec. 5.6.6.

A general comment on routing algorithms is also in order. Link state, distance vector, and other algorithms rely on processing at all the routers to compute routes. Problems with the hardware or software at even a small number of routers can wreak havoc across the network. For example, if a router claims to have a link it does not have or forgets a link it does have, the network graph will be incorrect. If a router fails to forward packets or corrupts them while forwarding them, the route will not work as expected. Finally, if it runs out of memory or does the routing calculation wrong, bad things will happen. As the network grows into the range of tens or hundreds of thousands of nodes, the probability of some router failing occasionally becomes nonnegligible. The trick is to try to arrange to limit the damage when the inevitable happens. Perlman (1988) discusses these problems and their possible solutions in detail.

5.2.6 Hierarchical Routing

As networks grow in size, the router routing tables grow proportionally. Not only is router memory consumed by ever-increasing tables, but more CPU time is needed to scan them and more bandwidth is needed to send status reports about them. At a certain point, the network may grow to the point where it is no longer

feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network.

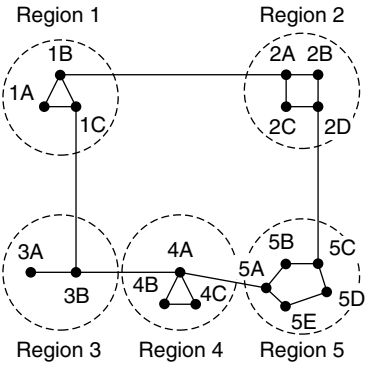
When hierarchical routing is used, the routers are divided into what we will call **regions**. Each router knows all the details about how to route packets to destinations within its own region but knows nothing about the internal structure of other regions. When different networks are interconnected, it is natural to regard each one as a separate region to free the routers in one network from having to know the topological structure of the other ones.

For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the regions into clusters, the clusters into zones, the zones into groups, and so on, until we run out of names for aggregations. As an example of a multilevel hierarchy, consider how a packet might be routed from Berkeley, California, to Malindi, Kenya. The Berkeley router would know the detailed topology within California but would send all out-of-state traffic to the Los Angeles router. The Los Angeles router would be able to route traffic directly to other domestic routers but would send all foreign traffic to New York. The New York router would be programmed to direct all traffic to the router in the destination country responsible for handling foreign traffic, say, in Nairobi. Finally, the packet would work its way down the tree in Kenya until it got to Malindi.

Figure 5-14 gives a quantitative example of routing in a two-level hierarchy with five regions. The full routing table for router *1A* has 17 entries, as shown in Fig. 5-14(b). When routing is done hierarchically, as in Fig. 5-14(c), there are entries for all the local routers, as before, but all other regions are condensed into a single router, so all traffic for region 2 goes via the *1B-2A* line, but the rest of the remote traffic goes via the *1C-3B* line. Hierarchical routing has reduced the table from 17 to 7 entries. As the ratio of the number of regions to the number of routers per region grows, the savings in table space increase.

Unfortunately, these gains in space are not free. There is a penalty to be paid: increased path length. For example, the best route from *1A* to *5C* is via region 2, but with hierarchical routing all traffic to region 5 goes via region 3, because that is better for most destinations in region 5.

When a single network becomes very large, an interesting question is "how many levels should the hierarchy have?" For example, consider a network with 720 routers. If there is no hierarchy, each router needs 720 routing table entries. If the network is partitioned into 24 regions of 30 routers each, each router needs 30 local entries plus 23 remote entries for a total of 53 entries. If a three-level hierarchy is chosen, with 8 clusters each containing 9 regions of 10 routers, each router needs 10 entries for local routers, 8 entries for routing to other regions within its own cluster, and 7 entries for distant clusters, for a total of 25 entries. Kamoun and Kleinrock (1979) discovered that the optimal number of levels for an N router network is $\ln N$, requiring a total of $e \ln N$ entries per router. They have also shown that the increase in effective mean path length caused by hierarchical routing is sufficiently small that it is usually acceptable.



(a)

Full table for 1A

Dest.	Line	Hops
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

(b)

Hierarchical table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Figure 5-14. Hierarchical routing.

5.2.7 Broadcast Routing

In some applications, hosts need to send messages to many or all other hosts. For example, a service distributing weather reports, stock market updates, or live radio programs might work best by sending to all machines and letting those that are interested read the data. Sending a packet to all destinations simultaneously is called **broadcasting**. Various methods have been proposed for doing it.

One broadcasting method that requires no special features from the network is for the source to simply send a distinct packet to each destination. Not only is the method wasteful of bandwidth and slow, but it also requires the source to have a complete list of all destinations. This method is not desirable in practice, even though it is widely applicable.

An improvement is **multidestination routing**, in which each packet contains either a list of destinations or a bit map indicating the desired destinations. When a packet arrives at a router, the router checks all the destinations to determine the set of output lines that will be needed. (An output line is needed if it is the best route to at least one of the destinations.) The router generates a new copy of the packet for each output line to be used and includes in each packet only those destinations that are to use the line. In effect, the destination set is partitioned among

the output lines. After a sufficient number of hops, each packet will carry only one destination like a normal packet. Multidestination routing is like using separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free. The network bandwidth is therefore used more efficiently. However, this scheme still requires the source to know all the destinations, plus it is as much work for a router to determine where to send one multidestination packet as it is for multiple distinct packets.

We have already seen a better broadcast routing technique: flooding. When implemented with a sequence number per source, flooding uses links efficiently with a decision rule at routers that is relatively simple. Although flooding is ill-suited for ordinary point-to-point communication, it rates serious consideration for broadcasting. However, it turns out that we can do better still once the shortest path routes for regular packets have been computed.

The idea for **reverse path forwarding** is elegant and remarkably simple once it has been pointed out (Dalal and Metcalfe, 1978). When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the link that is normally used for sending packets *toward* the source of the broadcast. If so, there is an excellent chance that the broadcast packet itself followed the best route from the router and is therefore the first copy to arrive at the router. This being the case, the router forwards copies of it onto all links except the one it arrived on. If, however, the broadcast packet arrived on a link other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

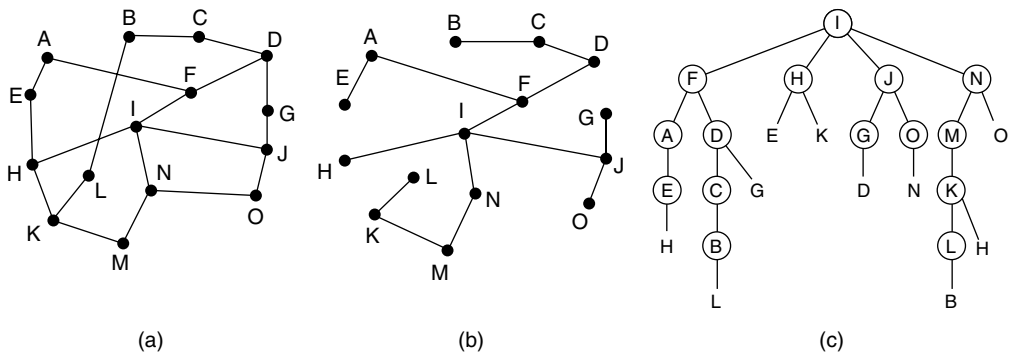


Figure 5-15. Reverse path forwarding. (a) A network. (b) A sink tree. (c) The tree built by reverse path forwarding.

An example of reverse path forwarding is shown in Fig. 5-15. Part (a) shows a network, part (b) shows a sink tree for router *I* of that network, and part (c) shows how the reverse path algorithm works. On the first hop, *I* sends packets to *F*, *H*, *J*, and *N*, as indicated by the second row of the tree. Each of these packets arrives on the preferred path to *I* (assuming that the preferred path falls along the sink tree) and is so indicated by a circle around the letter. On the second hop,

eight packets are generated, two by each of the routers that received a packet on the first hop. As it turns out, all eight of these arrive at previously unvisited routers, and five of these arrive along the preferred line. Of the six packets generated on the third hop, only three arrive on the preferred path (at *C*, *E*, and *K*); the others are duplicates. After five hops and 24 packets, the broadcasting terminates, compared with four hops and 14 packets had the sink tree been followed exactly.

The principal advantage of reverse path forwarding is that it is efficient while being easy to implement. It sends the broadcast packet over each link only once in each direction, just as in flooding, yet it requires only that routers know how to reach all destinations, without needing to remember sequence numbers (or use other mechanisms to stop the flood) or list all destinations in the packet.

Our last broadcast algorithm improves on the behavior of reverse path forwarding. It makes explicit use of the sink tree—or any other convenient spanning tree—for the router initiating the broadcast. A **spanning tree** is a subset of the network that includes all the routers but contains no loops. Sink trees are spanning trees. If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. In Fig. 5-15, for example, when the sink tree of part (b) is used as the spanning tree, the broadcast packet is sent with the minimum 14 packets. The only problem is that each router must have knowledge of some spanning tree for the method to be applicable. Sometimes this information is available (e.g., with link state routing, all routers know the complete topology, so they can compute a spanning tree) but sometimes it is not (e.g., with distance vector routing).

5.2.8 Multicast Routing

Some applications, such as a multiplayer game or live video of a sports event streamed to many viewing locations, send packets to multiple receivers. Unless the group is very small, sending a distinct packet to each receiver is expensive. On the other hand, broadcasting a packet is wasteful if the group consists of, say, 1000 machines on a million-node network, so that most receivers are not interested in the message (or worse yet, they are definitely interested but are not supposed to see it). Thus, we need a way to send messages to well-defined groups that are numerically large in size but small compared to the network as a whole.

Sending a message to such a group is called **multicasting**, and the routing algorithm used is called **multicast routing**. All multicasting schemes require some way to create and destroy groups and to identify which routers are members of a group. How these tasks are accomplished is not of concern to the routing algorithm. For now, we will assume that each group is identified by a multicast address and that routers know the groups to which they belong. We will revisit group membership when we describe the network layer of the Internet in Sec. 5.6.

Multicast routing schemes build on the broadcast routing schemes we have already studied, sending packets along spanning trees to deliver the packets to the members of the group while making efficient use of bandwidth. However, the best spanning tree to use depends on whether the group is dense, with receivers scattered over most of the network, or sparse, with much of the network not belonging to the group. In this section we will consider both cases.

If the group is dense, broadcast is a good start because it efficiently gets the packet to all parts of the network. But broadcast will reach some routers that are not members of the group, which is wasteful. The solution explored by Deering and Cheriton (1990) is to prune the broadcast spanning tree by removing links that do not lead to members. The result is an efficient multicast spanning tree.

As an example, consider the two groups, 1 and 2, in the network shown in Fig. 5-16(a). Some routers are attached to hosts that belong to one or both of these groups, as indicated in the figure. A spanning tree for the leftmost router is shown in Fig. 5-16(b). This tree can be used for broadcast but is overkill for multicast, as can be seen from the two pruned versions that are shown next. In Fig. 5-16(c), all the links that do not lead to hosts that are members of group 1 have been removed. The result is the multicast spanning tree for the leftmost router to send to group 1. Packets are forwarded only along this spanning tree, which is more efficient than the broadcast tree because there are 7 links instead of 10. Fig. 5-16(d) shows the multicast spanning tree after pruning for group 2. It is efficient too, with only five links this time. It also shows that different multicast groups have different spanning trees.

Various ways of pruning the spanning tree are possible. The simplest one can be used if link state routing is used and each router is aware of the complete topology, including which hosts belong to which groups. Each router can then construct its own pruned spanning tree for each sender to the group in question by constructing a sink tree for the sender as usual and then removing all links that do not connect group members to the sink node. **MOSPF** (**Multicast OSPF**) is an example of a link state protocol that works in this way (Moy, 1994).

With distance vector routing, a different pruning strategy can be followed. The basic algorithm is reverse path forwarding. However, whenever a router with no hosts interested in a particular group and no connections to other routers receives a multicast message for that group, it responds with a PRUNE message, telling the neighbor that sent the message not to send it any more multicasts from the sender for that group. When a router with no group members among its own hosts has received such messages on all the lines to which it sends the multicast, it, too, can respond with a PRUNE message. In this way, the spanning tree is recursively pruned. **DVMRP** (**Distance Vector Multicast Routing Protocol**) is an example of a multicast routing protocol that works this way (Waitzman et al., 1988).

Pruning results in efficient spanning trees that use only the links that are actually needed to reach members of the group. One potential disadvantage is that it is lots of work for routers, especially for large networks. Suppose that a network

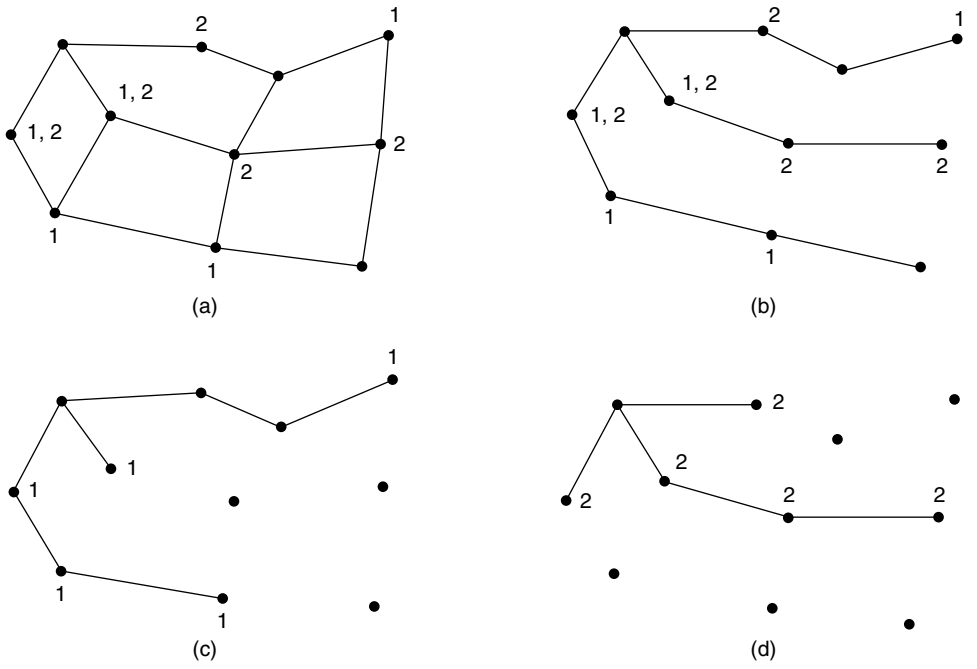


Figure 5-16. (a) A network. (b) A spanning tree for the leftmost router. (c) A multicast tree for group 1. (d) A multicast tree for group 2.

has *n* groups, each with an average of *m* nodes. At each router and for each group, *m* pruned spanning trees must be stored, for a total of *mn* trees. For example, Fig. 5-16(c) gives the spanning tree for the leftmost router to send to group 1. The spanning tree for the rightmost router to send to group 1 (not shown) will look quite different, as packets will head directly for group members rather than via the left side of the graph. This in turn means that routers must forward packets destined to group 1 in different directions depending on which node is sending to the group. When many large groups with many senders exist, considerable storage is needed to store all the trees.

An alternative design uses **core-based trees** to compute a single spanning tree for the group (Ballardie et al., 1993). All of the routers agree on a root (called the **core** or **rendezvous point**) and build the tree by sending a packet from each member to the root. The tree is the union of the paths traced by these packets. Fig. 5-17(a) shows a core-based tree for group 1. To send to this group, a sender sends a packet to the core. When the packet reaches the core, it is forwarded down the tree. This is shown in Fig. 5-17(b) for the sender on the righthand side of the network. As a performance optimization, packets destined for the group do not need to reach the core before they are multicast. As soon as a packet reaches the

tree, it can be forwarded up toward the root, as well as down all the other branches. This is the case for the sender at the top of Fig. 5-17(b).

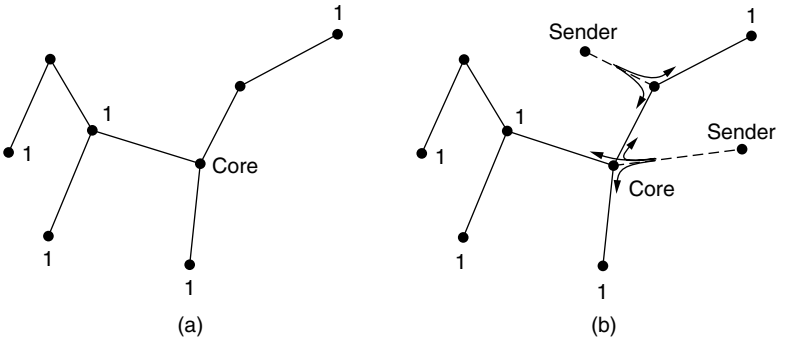


Figure 5-17. (a) Core-based tree for group 1. (b) Sending to group 1.

Having a shared tree is not optimal for all sources. For example, in Fig. 5-17(b), the packet from the sender on the righthand side reaches the top-right group member via the core in three hops, instead of directly. The inefficiency depends on where the core and senders are located, but often it is reasonable when the core is in the middle of the senders. When there is only a single sender, as in a video that is streamed to a group, using the sender as the core is optimal.

Also of note is that shared trees can be a major savings in storage costs, messages sent, and computation. Each router has to keep only one tree per group, instead of *m* trees. Further, routers that are not part of the tree do no work at all to support the group. For this reason, shared tree approaches like core-based trees are used for multicasting to sparse groups in the Internet as part of popular protocols such as **PIM** (**Protocol Independent Multicast**) (Fenner et al., 2006).

5.2.9 Anycast Routing

So far, we have covered delivery models in which a source sends to a single destination (called **unicast**), to all destinations (called broadcast), and to a group of destinations (called multicast). Another delivery model, called **anycast** is sometimes also useful. In anycast, a packet is delivered to the nearest member of a group (Partridge et al., 1993). Schemes that find these paths are called **anycast routing**.

Why would we want anycast? Sometimes nodes provide a service, such as time of day or content distribution for which it is getting the right information all that matters, not the node that is contacted; any node will do. For example, anycast is used in the Internet as part of DNS, as we will see in Chap. 7.

Luckily, we will not have to devise new routing schemes for anycast because regular distance vector and link state routing can produce anycast routes. Suppose

we want to anycast to the members of group 1. They will all be given the address "1," instead of different addresses. Distance vector routing will distribute vectors as usual, and nodes will choose the shortest path to destination 1. This will result in nodes sending to the nearest instance of destination 1. The routes are shown in Fig. 5-18(a). This procedure works because the routing protocol does not realize that there are multiple instances of destination 1. That is, it believes that all the instances of node 1 are the same node, as in the topology shown in Fig. 5-18(b).

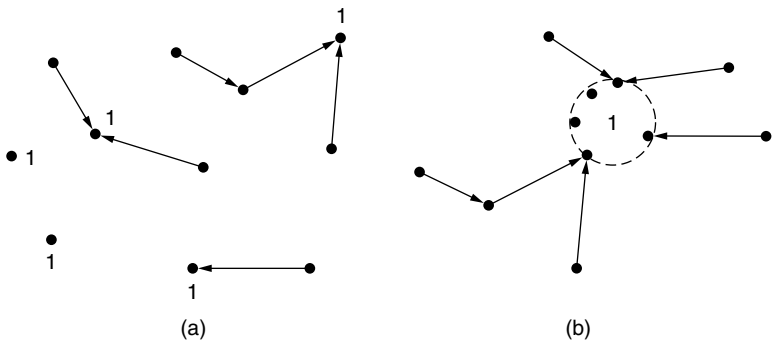


Figure 5-18. (a) Anycast routes to group 1. (b) Topology seen by the routing protocol.

This procedure works for link state routing as well, although there is the added consideration that the routing protocol must not find seemingly short paths that pass through node 1. This would result in jumps through hyperspace, since the instances of node 1 are really nodes located in different parts of the network. However, link state protocols already make this distinction between routers and hosts. We glossed over this fact earlier because it was not needed for our discussion.

5.2.10 Routing for Mobile Hosts

Millions of people use computers while on the go, from truly mobile situations with wireless devices in moving cars, to nomadic situations in which laptop computers are used in a series of different locations. We will use the term **mobile hosts** to mean either category, as distinct from stationary hosts that never move. Increasingly, people want to stay connected wherever in the world they may be, as easily as if they were at home. These mobile hosts introduce a new complication: to route a packet to a mobile host, the network first has to find it.

The model of the world that we will consider is one in which all hosts are assumed to have a permanent **home location** that never changes. Each hosts also has a permanent home address that can be used to determine its home location, analogous to the way the telephone number 1-212-5551212 indicates the United States (country code 1) and Manhattan (212). The routing goal in systems with

mobile hosts is to make it possible to send packets to mobile hosts using their fixed home addresses and have the packets efficiently reach them wherever they may be. The trick, of course, is to find them.

Some discussion of this model is in order. A different model would be to recompute routes as the mobile host moves and the topology changes. We could then simply use the routing schemes described earlier in this section. However, with a growing number of mobile hosts, this model would soon lead to the entire network endlessly computing new routes. Using the home addresses greatly reduces this burden.

Another alternative would be to provide mobility above the network layer, which is what typically happens with laptops today. When they are moved to new Internet locations, laptops acquire new network addresses. There is no association between the old and new addresses; the network does not know that they belonged to the same laptop. In this model, a laptop can be used to browse the Web, but other hosts cannot send packets to it (for example, for an incoming call), without building a higher layer location service, for example, signing into Skype *again* after moving. Moreover, connections cannot be maintained while the host is moving; new connections must be started up instead. Network-layer mobility is useful to fix these problems.

The basic idea used for mobile routing in the Internet and cellular networks is for the mobile host to tell a host at the home location where it is now. This host, which acts on behalf of the mobile host, is called the **home agent**. Once it knows where the mobile host is currently located, it can forward packets so that they are delivered.

Fig. 5-19 shows mobile routing in action. A sender in the northwest city of Seattle wants to send a packet to a host normally located across the United States in New York. The case of interest to us is when the mobile host is not at home. Instead, it is temporarily in San Diego.

The mobile host in San Diego must acquire a local network address before it can use the network. This happens in the normal way that hosts obtain network addresses; we will cover how this works for the Internet later in this chapter. The local address is called a **care of address**. Once the mobile host has this address, it can tell its home agent where it is now. It does this by sending a registration message to the home agent (step 1) with the care of address. The message is shown with a dashed line in Fig. 5-19 to indicate that it is a control message, not a data message.

Next, the sender sends a data packet to the mobile host using its permanent address (step 2). This packet is routed by the network to the host's home location because that is where the home address belongs. In New York, the home agent intercepts this packet because the mobile host is away from home. It then wraps or **encapsulates** the packet with a new header and sends this bundle to the care of address (step 3). This mechanism is called **tunneling**. It is very important in the Internet so we will look at it in more detail later.

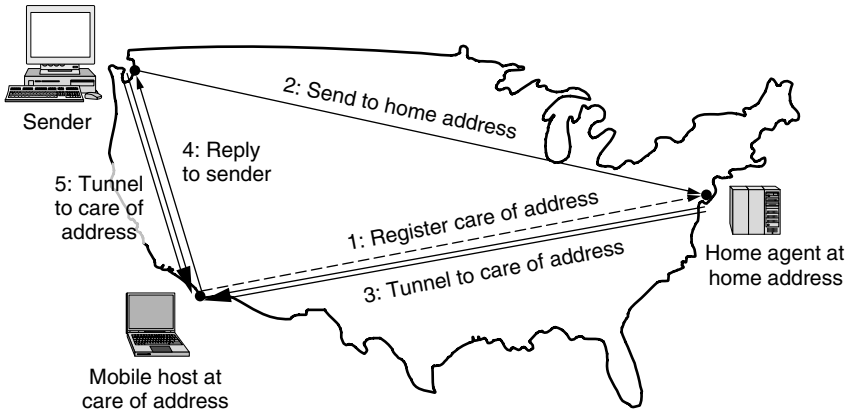


Figure 5-19. Packet routing for mobile hosts.

When the encapsulated packet arrives at the care of address, the mobile host unwraps it and retrieves the packet from the sender. The mobile host then sends its reply packet directly to the sender (step 4). The overall route is called **triangle routing** because it may be circuitous if the remote location is far from the home location. As part of step 4, the sender may learn the current care of address. Subsequent packets can be routed directly to the mobile host by tunneling them to the care of address (step 5), bypassing the home location entirely. If connectivity is lost for any reason as the mobile moves, the home address can always be used to reach the mobile.

An important aspect that we have omitted from this description is security. In general, when a host or router gets a message of the form "Starting right now, please send all of Stephany's mail to me," it might have a couple of questions about whom it is talking to and whether this is a good idea. Security information is included in the messages so that their validity can be checked with cryptographic protocols that we will study in Chap. 8.

There are many variations on mobile routing. The scheme above is modeled on IPv6 mobility, the form of mobility used in the Internet (Johnson et al., 2004) and as part of IP-based cellular networks such as UMTS. We showed the sender to be a stationary node for simplicity, but the designs let both nodes be mobile hosts. Alternatively, the host may be part of a mobile network, for example a computer in a plane. Extensions of the basic scheme support mobile networks with no work on the part of the hosts (Devarapalli et al., 2005).

Some schemes make use of a foreign (i.e., remote) agent, similar to the home agent but at the foreign location, or analogous to the VLR (Visitor Location Register) in cellular networks. However, in more recent schemes, the foreign agent is not needed; mobile hosts act as their own foreign agents. In either case, knowledge of the temporary location of the mobile host is limited to a small number of

hosts (e.g., the mobile, home agent, and senders) so that the many routers in a large network do not need to recompute routes.

For more information about mobile routing, see also Perkins (1998, 2002) and Snoeren and Balakrishnan (2000).

5.2.11 Routing in Ad Hoc Networks

We have now seen how to do routing when the hosts are mobile but the routers are fixed. An even more extreme case is one in which the routers themselves are mobile. Among the possibilities are emergency workers at an earthquake site, military vehicles on a battlefield, a fleet of ships at sea, or a gathering of people with laptop computers in an area lacking 802.11.

In all these cases, and others, each node communicates wirelessly and acts as both a host and a router. Networks of nodes that just happen to be near each other are called **ad hoc networks** or **MANETs** (**Mobile Ad hoc NETworks**). Let us now examine them briefly. More information can be found in Perkins (2001).

What makes ad hoc networks different from wired networks is that the topology is suddenly tossed out the window. Nodes can come and go or appear in new places at the drop of a bit. With a wired network, if a router has a valid path to some destination, that path continues to be valid barring failures, which are hopefully rare. With an ad hoc network, the topology may be changing all the time, so the desirability and even the validity of paths can change spontaneously without warning. Needless to say, these circumstances make routing in ad hoc networks more challenging than routing in their fixed counterparts.

Many, many routing algorithms for ad hoc networks have been proposed. However, since ad hoc networks have been little used in practice compared to mobile networks, it is unclear which of these protocols are most useful. As an example, we will look at one of the most popular routing algorithms, **AODV** (**Ad hoc On-demand Distance Vector**) (Perkins and Royer, 1999). It is a relative of the distance vector algorithm that has been adapted to work in a mobile environment, in which nodes often have limited bandwidth and battery lifetimes. Let us now see how it discovers and maintains routes.

Route Discovery

In AODV, routes to a destination are discovered on demand, that is, only when a somebody wants to send a packet to that destination. This saves much work that would otherwise be wasted when the topology changes before the route is used. At any instant, the topology of an ad hoc network can be described by a graph of connected nodes. Two nodes are connected (i.e., have an arc between them in the graph) if they can communicate directly using their radios. A basic but adequate model that is sufficient for our purposes is that each node can communicate with all other nodes that lie within its coverage circle. Real networks are

more complicated, with buildings, hills, and other obstacles that block communication, and nodes for which *A* is connected to *B* but *B* is not connected to *A* because *A* has a more powerful transmitter than *B*. However, for simplicity, we will assume all connections are symmetric.

To describe the algorithm, consider the newly formed ad hoc network of Fig. 5-20. Suppose that a process at node *A* wants to send a packet to node *I*. The AODV algorithm maintains a distance vector table at each node, keyed by destination, giving information about that destination, including the neighbor to which to send packets to reach the destination. First, *A* looks in its table and does not find an entry for *I*. It now has to discover a route to *I*. This property of discovering routes only when they are needed is what makes this algorithm "on demand."

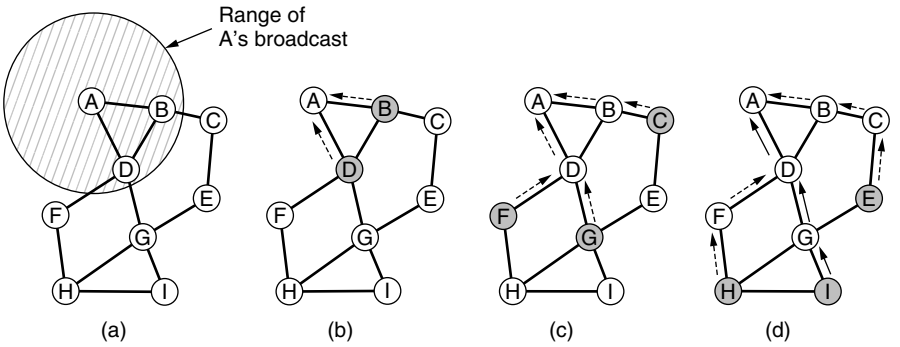


Figure 5-20. (a) Range of *A*'s broadcast. (b) After *B* and *D* receive it. (c) After *C*, *F*, and *G* receive it. (d) After *E*, *H*, and *I* receive it. The shaded nodes are new recipients. The dashed lines show possible reverse routes. The solid lines show the discovered route.

To locate *I*, *A* constructs a ROUTE REQUEST packet and broadcasts it using flooding, as described in Sec. 5.2.3. The transmission from *A* reaches *B* and *D*, as illustrated in Fig. 5-20(a). Each node rebroadcasts the request, which continues to reach nodes *F*, *G*, and *C* in Fig. 5-20(c) and nodes *H*, *E*, and *I* in Fig. 5-20(d). A sequence number set at the source is used to weed out duplicates during the flood. For example, *D* discards the transmission from *B* in Fig. 5-20(c) because it has already forwarded the request.

Eventually, the request reaches node *I*, which constructs a ROUTE REPLY packet. This packet is unicast to the sender along the reverse of the path followed by the request. For this to work, each intermediate node must remember the node that sent it the request. The arrows in Fig. 5-20(b)–(d) show the reverse route information that is stored. Each intermediate node also increments a hop count as it forwards the reply. This tells the nodes how far they are from the destination. The replies tell each intermediate node which neighbor to use to reach the destination: it is the node that sent them the reply. Intermediate nodes *G* and *D* put the

best route they hear into their routing tables as they process the reply. When the reply reaches *A*, a new route, *ADGI*, has been created.

In a large network, the algorithm generates many broadcasts, even for destinations that are close by. To reduce overhead, the scope of the broadcasts is limited using the IP packet's *Time to live* field. This field is initialized by the sender and decremented on each hop. If it hits 0, the packet is discarded instead of being broadcast. The route discovery process is then modified as follows. To locate a destination, the sender broadcasts a ROUTE REQUEST packet with *Time to live* set to 1. If no response comes back within a reasonable time, another one is sent, this time with *Time to live* set to 2. Subsequent attempts use 3, 4, 5, etc. In this way, the search is first attempted locally, then in increasingly wider rings.

Route Maintenance

Because nodes can move or be switched off, the topology can change spontaneously. For example, in Fig. 5-20, if *G* is switched off, *A* will not realize that the route it was using to *I* (*ADGI*) is no longer valid. The algorithm needs to be able to deal with this. Periodically, each node broadcasts a *Hello* message. Each of its neighbors is expected to respond to it. If no response is forthcoming, the broadcaster knows that that neighbor has moved out of range or failed and is no longer connected to it. Similarly, if it tries to send a packet to a neighbor that does not respond, it learns that the neighbor is no longer available.

This information is used to purge routes that no longer work. For each possible destination, each node, *N*, keeps track of its active neighbors that have fed it a packet for that destination during the last ΔT seconds. When any of *N*'s neighbors becomes unreachable, it checks its routing table to see which destinations have routes using the now-gone neighbor. For each of these routes, the active neighbors are informed that their route via *N* is now invalid and must be purged from their routing tables. In our example, *D* purges its entries for *G* and *I* from its routing table and notifies *A*, which purges its entry for *I*. In the general case, the active neighbors tell their active neighbors, and so on, recursively, until all routes depending on the now-gone node are purged from all routing tables.

At this stage, the invalid routes have been purged from the network, and senders can find new, valid routes by using the discovery mechanism that we described. However, there is a complication. Recall that distance vector protocols can suffer from slow convergence or count-to-infinity problems after a topology change in which they confuse old, invalid routes with new, valid routes.

To ensure rapid convergence, routes include a sequence number that is controlled by the destination. The destination sequence number is like a logical clock. The destination increments it every time that it sends a fresh ROUTE REPLY. Senders ask for a fresh route by including in the ROUTE REQUEST the destination sequence number of the last route they used, which will either be the sequence number of the route that was just purged, or 0 as an initial value. The

request will be broadcast until a route with a higher sequence number is found. Intermediate nodes store the routes that have a higher sequence number, or the fewest hops for the current sequence number.

In the spirit of an on demand protocol, intermediate nodes only store the routes that are in use. Other route information learned during broadcasts is timed out after a short delay. Discovering and storing only the routes that are used helps to save bandwidth and battery life compared to a standard distance vector protocol that periodically broadcasts updates.

So far, we have considered only a single route, from *A* to *I*. To further save resources, route discovery and maintenance are shared when routes overlap. For instance, if *B* also wants to send packets to *I*, it will perform route discovery. However, in this case the request will first reach *D*, which already has a route to *I*. Node *D* can then generate a reply to tell *B* the route without any additional work being required.

There are many other ad hoc routing schemes. Another well-known on demand scheme is DSR (Dynamic Source Routing) (Johnson et al., 2001). A different strategy based on geography is explored by GPSR (Greedy Perimeter Stateless Routing) (Karp and Kung, 2000). If all nodes know their geographic positions, forwarding to a destination can proceed without route computation by simply heading in the right direction and circling back to escape any dead ends. Which protocols win out will depend on the kinds of ad hoc networks that prove useful in practice.

5.3 CONGESTION CONTROL ALGORITHMS

Too many packets present in (a part of) the network causes packet delay and loss that degrades performance. This situation is called **congestion**. The network and transport layers share the responsibility for handling congestion. Since congestion occurs within the network, it is the network layer that directly experiences it and must ultimately determine what to do with the excess packets. However, the most effective way to control congestion is to reduce the load that the transport layer is placing on the network. This requires the network and transport layers to work together. In this chapter we will look at the network aspects of congestion. In Chap. 6, we will complete the topic by covering the transport aspects of congestion.

Figure 5-21 depicts the onset of congestion. When the number of packets hosts send into the network is well within its carrying capacity, the number delivered is proportional to the number sent. If twice as many are sent, twice as many are delivered. However, as the offered load approaches the carrying capacity, bursts of traffic occasionally fill up the buffers inside routers and some packets are lost. These lost packets consume some of the capacity, so the number of delivered packets falls below the ideal curve. The network is now congested.

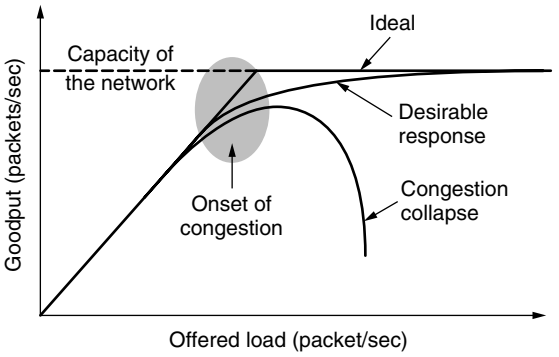


Figure 5-21. With too much traffic, performance drops sharply.

Unless the network is well designed, it may experience a **congestion collapse**, in which performance plummets as the offered load increases beyond the capacity. This can happen because packets can be sufficiently delayed inside the network that they are no longer useful when they leave the network. For example, in the early Internet, the time a packet spent waiting for a backlog of packets ahead of it to be sent over a slow 56-kbps link could reach the maximum time it was allowed to remain in the network. It then had to be thrown away. A different failure mode occurs when senders retransmit packets that are greatly delayed, thinking that they have been lost. In this case, copies of the same packet will be delivered by the network, again wasting its capacity. To capture these factors, the y-axis of Fig. 5-21 is given as **goodput**, which is the rate at which *useful* packets are delivered by the network.

We would like to design networks that avoid congestion where possible and do not suffer from congestion collapse if they do become congested. Unfortunately, congestion cannot wholly be avoided. If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up. If there is insufficient memory to hold all of them, packets will be lost. Adding more memory may help up to a point, but Nagle (1987) realized that if routers have an infinite amount of memory, congestion gets worse, not better. This is because by the time packets get to the front of the queue, they have already timed out (repeatedly) and duplicates have been sent. This makes matters worse, not better—it leads to congestion collapse.

Low-bandwidth links or routers that process packets more slowly than the line rate can also become congested. In this case, the situation can be improved by directing some of the traffic away from the bottleneck to other parts of the network. Eventually, however, all regions of the network will be congested. In this situation, there is no alternative but to shed load or build a faster network.

It is worth pointing out the difference between congestion control and flow control, as the relationship is a very subtle one. Congestion control has to do with

making sure the network is able to carry the offered traffic. It is a global issue, involving the behavior of all the hosts and routers. Flow control, in contrast, relates to the traffic between a particular sender and a particular receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver is able to absorb it.

To see the difference between these two concepts, consider a network made up of 100-Gbps fiber optic links on which a supercomputer is trying to force feed a large file to a personal computer that is capable of handling only 1 Gbps. Although there is no congestion (the network itself is not in trouble), flow control is needed to force the supercomputer to stop frequently to give the personal computer a chance to breathe.

At the other extreme, consider a network with 1-Mbps lines and 1000 large computers, half of which are trying to transfer files at 100 kbps to the other half. Here, the problem is not that of fast senders overpowering slow receivers, but that the total offered traffic exceeds what the network can handle.

The reason congestion control and flow control are often confused is that the best way to handle both problems is to get the host to slow down. Thus, a host can get a "slow down" message either because the receiver cannot handle the load or because the network cannot handle it. We will come back to this point in Chap. 6.

We will start our study of congestion control by looking at the approaches that can be used at different time scales. Then we will look at approaches to preventing congestion from occurring in the first place, followed by approaches for coping with it once it has set in.

5.3.1 Approaches to Congestion Control

The presence of congestion means that the load is (temporarily) greater than the resources (in a part of the network) can handle. Two solutions come to mind: increase the resources or decrease the load. As shown in Fig. 5-22, these solutions are usually applied on different time scales to either prevent congestion or react to it once it has occurred.

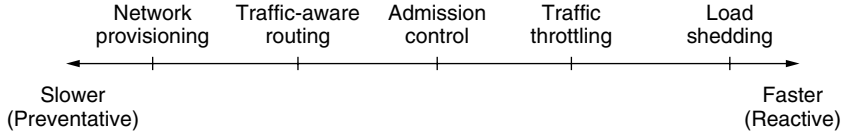


Figure 5-22. Timescales of approaches to congestion control.

The most basic way to avoid congestion is to build a network that is well matched to the traffic that it carries. If there is a low-bandwidth link on the path along which most traffic is directed, congestion is likely. Sometimes resources

can be added dynamically when there is serious congestion, for example, turning on spare routers or enabling lines that are normally used only as backups (to make the system fault tolerant) or purchasing bandwidth on the open market. More often, links and routers that are regularly heavily utilized are upgraded at the earliest opportunity. This is called **provisioning** and happens on a time scale of months, driven by long-term traffic trends.

To make the most of the existing network capacity, routes can be tailored to traffic patterns that change during the day as network users wake and sleep in different time zones. For example, routes may be changed to shift traffic away from heavily used paths by changing the shortest path weights. Some local radio stations have helicopters flying around their cities to report on road congestion to make it possible for their mobile listeners to route their packets (cars) around hotspots. This is called **traffic-aware routing**. Splitting traffic across multiple paths is also helpful.

However, sometimes it is not possible to increase capacity. The only way then to beat back the congestion is to decrease the load. In a virtual-circuit network, new connections can be refused if they would cause the network to become congested. This is called **admission control**.

At a finer granularity, when congestion is imminent the network can deliver feedback to the sources whose traffic flows are responsible for the problem. The network can request these sources to throttle their traffic, or it can slow down the traffic itself.

Two difficulties with this approach are how to identify the onset of congestion, and how to inform the source that needs to slow down. To tackle the first issue, routers can monitor the average load, queueing delay, or packet loss. In all cases, rising numbers indicate growing congestion.

To tackle the second issue, routers must participate in a feedback loop with the sources. For a scheme to work correctly, the time scale must be adjusted carefully. If every time two packets arrive in a row, a router yells STOP and every time a router is idle for 20 μsec, it yells GO, the system will oscillate wildly and never converge. On the other hand, if it waits 30 minutes to make sure before saying anything, the congestion-control mechanism will react too sluggishly to be of any use. Delivering timely feedback is a nontrivial matter. An added concern is having routers send more messages when the network is already congested.

Finally, when all else fails, the network is forced to discard packets that it cannot deliver. The general name for this is **load shedding**. A good policy for choosing which packets to discard can help to prevent congestion collapse.

5.3.2 Traffic-Aware Routing

The first approach we will examine is traffic-aware routing. The routing schemes we looked at in Sec 5.2 used fixed link weights. These schemes adapted to changes in topology, but not to changes in load. The goal in taking load into

account when computing routes is to shift traffic away from hotspots that will be the first places in the network to experience congestion.

The most direct way to do this is to set the link weight to be a function of the (fixed) link bandwidth and propagation delay plus the (variable) measured load or average queuing delay. Least-weight paths will then favor paths that are more lightly loaded, all else being equal.

Traffic-aware routing was used in the early Internet according to this model (Khanna and Zinky, 1989). However, there is a peril. Consider the network of Fig. 5-23, which is divided into two parts, East and West, connected by two links, *CF* and *EI*. Suppose that most of the traffic between East and West is using link *CF*, and, as a result, this link is heavily loaded with long delays. Including queueing delay in the weight used for the shortest path calculation will make *EI* more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over *EI*, loading this link. Consequently, in the next update, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems.

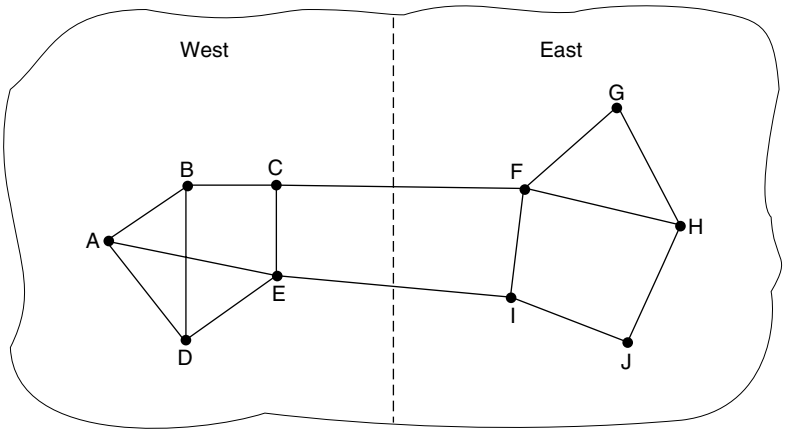


Figure 5-23. A network in which the East and West parts are connected by two links.

If load is ignored and only bandwidth and propagation delay are considered, this problem does not occur. Attempts to include load but change weights within a narrow range only slow down routing oscillations. Two techniques can contribute to a successful solution. The first is multipath routing, in which there can be multiple paths from a source to a destination. In our example this means that the traffic can be spread across both of the East to West links. The second one is for the routing scheme to shift traffic across routes slowly enough that it is able to converge, as in the scheme of Gallagher (1977).

Given these difficulties, in the Internet routing protocols do not generally adjust their routes depending on the load. Instead, adjustments are made outside the routing protocol by slowly changing its inputs. This is called **traffic engineering**.

5.3.3 Admission Control

One technique that is widely used in virtual-circuit networks to keep congestion at bay is **admission control**. The idea is simple: do not set up a new virtual circuit unless the network can carry the added traffic without becoming congested. Thus, attempts to set up a virtual circuit may fail. This is better than the alternative, as letting more people in when the network is busy just makes matters worse. By analogy, in the telephone system, when a switch gets overloaded it practices admission control by not giving dial tones.

The trick with this approach is working out when a new virtual circuit will lead to congestion. The task is straightforward in the telephone network because of the fixed bandwidth of calls (64 kbps for uncompressed audio). However, virtual circuits in computer networks come in all shapes and sizes. Thus, the circuit must come with some characterization of its traffic if we are to apply admission control.

Traffic is often described in terms of its rate and shape. The problem of how to describe it in a simple yet meaningful way is difficult because traffic is typically bursty—the average rate is only half the story. For example, traffic that varies while browsing the Web is more difficult to handle than a streaming movie with the same long-term throughput because the bursts of Web traffic are more likely to congest routers in the network. A commonly used descriptor that captures this effect is the **leaky bucket** or **token bucket**. A leaky bucket has two parameters that bound the average rate and the instantaneous burst size of traffic. Since leaky buckets are widely used for quality of service, we will go over them in detail in Sec. 5.4.

Armed with traffic descriptions, the network can decide whether to admit the new virtual circuit. One possibility is for the network to reserve enough capacity along the paths of each of its virtual circuits that congestion will not occur. In this case, the traffic description is a service agreement for what the network will guarantee its users. We have prevented congestion but veered into the related topic of quality of service a little too early; we will return to it in the next section.

Even without making guarantees, the network can use traffic descriptions for admission control. The task is then to estimate how many circuits will fit within the carrying capacity of the network without congestion. Suppose that virtual circuits that may blast traffic at rates up to 10 Mbps all pass through the same 100-Mbps physical link. How many circuits should be admitted? Clearly, 10 circuits can be admitted without risking congestion, but this is wasteful in the normal case since it may rarely happen that all 10 are transmitting full blast at the same time. In real networks, measurements of past behavior that capture the statistics of transmissions can be used to estimate the number of circuits to admit, to trade better performance for acceptable risk.

Admission control can also be combined with traffic-aware routing by considering routes around traffic hotspots as part of the setup procedure. For example,

consider the network illustrated in Fig. 5-24(a), in which two routers are congested, as indicated.

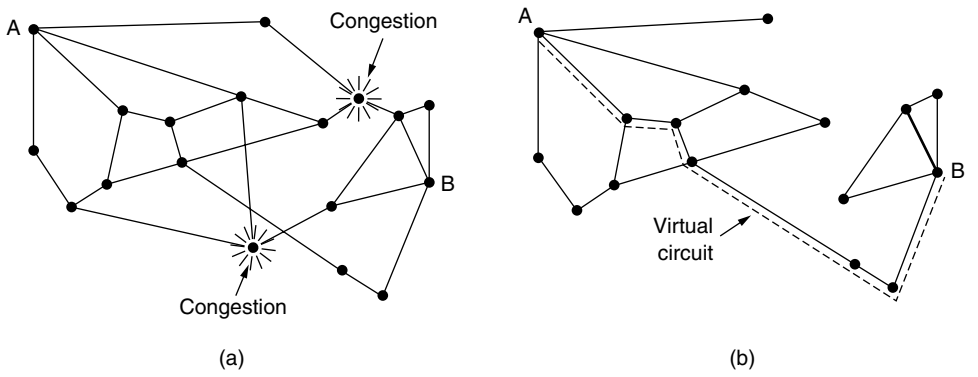


Figure 5-24. (a) A congested network. (b) The portion of the network that is not congested. A virtual circuit from *A* to *B* is also shown.

Suppose that a host attached to router *A* wants to set up a connection to a host attached to router *B*. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the network as shown in Fig. 5-24(b), omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids the congested routers. Shaikh et al. (1999) give a design for this kind of load-sensitive routing.

5.3.4 Traffic Throttling

In the Internet and many other computer networks, senders adjust their transmissions to send as much traffic as the network can readily deliver. In this setting, the network aims to operate just before the onset of congestion. When congestion is imminent, it must tell the senders to throttle back their transmissions and slow down. This feedback is business as usual rather than an exceptional situation. The term **congestion avoidance** is sometimes used to contrast this operating point with the one in which the network has become (overly) congested.

Let us now look at some approaches to throttling traffic that can be used in both datagram networks and virtual-circuit networks. Each approach must solve two problems. First, routers must determine when congestion is approaching, ideally before it has arrived. To do so, each router can continuously monitor the resources it is using. Three possibilities are the utilization of the output links, the buffering of queued packets inside the router, and the number of packets that are lost due to insufficient buffering. Of these possibilities, the second one is the most useful. Averages of utilization do not directly account for the burstiness of

most traffic—a utilization of 50% may be low for smooth traffic and too high for highly variable traffic. Counts of packet losses come too late. Congestion has already set in by the time that packets are lost.

The queueing delay inside routers directly captures any congestion experienced by packets. It should be low most of time, but will jump when there is a burst of traffic that generates a backlog. To maintain a good estimate of the queueing delay, d, a sample of the instantaneous queue length, s, can be made periodically and d updated according to

$$d_{\text{new}} = \alpha d_{\text{old}} + (1 - \alpha)s$$

where the constant α determines how fast the router forgets recent history. This is called an **EWMA** (**Exponentially Weighted Moving Average**). It smoothes out fluctuations and is equivalent to a low-pass filter. Whenever d moves above the threshold, the router notes the onset of congestion.

The second problem is that routers must deliver timely feedback to the senders that are causing the congestion. Congestion is experienced in the network, but relieving congestion requires action on behalf of the senders that are using the network. To deliver feedback, the router must identify the appropriate senders. It must then warn them carefully, without sending many more packets into the already congested network. Different schemes use different feedback mechanisms, as we will now describe.

Choke Packets

The most direct way to notify a sender of congestion is to tell it directly. In this approach, the router selects a congested packet and sends a **choke packet** back to the source host, giving it the destination found in the packet. The original packet may be tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and then forwarded in the usual way. To avoid increasing load on the network during a time of congestion, the router may only send choke packets at a low rate.

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination, for example, by 50%. In a datagram network, simply picking packets at random when there is congestion is likely to cause choke packets to be sent to fast senders, because they will have the most packets in the queue. The feedback implicit in this protocol can help prevent congestion yet not throttle any sender unless it causes trouble. For the same reason, it is likely that multiple choke packets will be sent to a given host and destination. The host should ignore these additional chokes for the fixed time interval until its reduction in traffic takes effect. After that period, further choke packets indicate that the network is still congested.

An example of a choke packet used in the early Internet is the SOURCE-QUENCH message (Postel, 1981). It never caught on, though, partly because the

circumstances in which it was generated and the effect it had were not clearly specified. The modern Internet uses an alternative notification design that we will describe next.

Explicit Congestion Notification

Instead of generating additional packets to warn of congestion, a router can tag any packet it forwards (by setting a bit in the packet's header) to signal that it is experiencing congestion. When the network delivers the packet, the destination can note that there is congestion and inform the sender when it sends a reply packet. The sender can then throttle its transmissions as before.

This design is called **ECN** (**Explicit Congestion Notification**) and is used in the Internet (Ramakrishnan et al., 2001). It is a refinement of early congestion signaling protocols, notably the binary feedback scheme of Ramakrishnan and Jain (1988) that was used in the DECNET architecture. Two bits in the IP packet header are used to record whether the packet has experienced congestion. Packets are unmarked when they are sent, as illustrated in Fig. 5-25. If any of the routers they pass through is congested, that router will then mark the packet as having experienced congestion as it is forwarded. The destination will then echo any marks back to the sender as an explicit congestion signal in its next reply packet. This is shown with a dashed line in the figure to indicate that it happens above the IP level (e.g., in TCP). The sender must then throttle its transmissions, as in the case of choke packets.

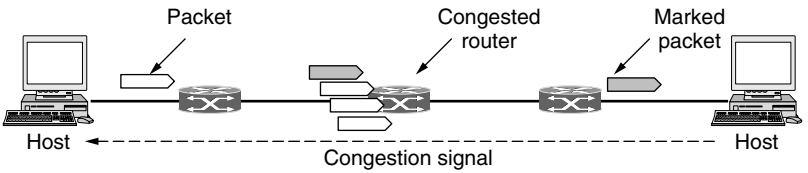


Figure 5-25. Explicit congestion notification

Hop-by-Hop Backpressure

At high speeds or over long distances, many new packets may be transmitted after congestion has been signaled because of the delay before the signal takes effect. Consider, for example, a host in San Francisco (router *A* in Fig. 5-26) that is sending traffic to a host in New York (router *D* in Fig. 5-26) at the OC-3 speed of 155 Mbps. If the New York host begins to run out of buffers, it will take about 40 msec for a choke packet to get back to San Francisco to tell it to slow down. An ECN indication will take even longer because it is delivered via the destination. Choke packet propagation is illustrated as the second, third, and fourth steps in

Fig. 5-26(a). In those 40 msec, another 6.2 megabits will have been sent. Even if the host in San Francisco completely shuts down immediately, the 6.2 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5-26(a) will the New York router notice a slower flow.

An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-26(b). Here, as soon as the choke packet reaches *F*, *F* is required to reduce the flow to *D*. Doing so will require *F* to devote more buffers to the connection, since the source is still sending away at full blast, but it gives *D* immediate relief, like a headache remedy in a television commercial. In the next step, the choke packet reaches *E*, which tells *E* to reduce the flow to *F*. This action puts a greater demand on *E*'s buffers but gives *F* immediate relief. Finally, the choke packet reaches *A* and the flow genuinely slows down.

The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion, at the price of using up more buffers upstream. In this way, congestion can be nipped in the bud without losing any packets. The idea is discussed in detail by Mishra et al. (1996).

5.3.5 Load Shedding

When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding. **Load shedding** is a fancy way of saying that when routers are being inundated by packets that they cannot handle, they just throw them away. The term comes from the world of electrical power generation, where it refers to the practice of utilities intentionally blacking out certain areas to save the entire grid from collapsing on hot summer days when the demand for electricity greatly exceeds the supply.

The key question for a router drowning in packets is which packets to drop. The preferred choice may depend on the type of applications that use the network. For a file transfer, an old packet is worth more than a new one. This is because dropping packet 6 and keeping packets 7 through 10, for example, will only force the receiver to do more work to buffer data that it cannot yet use. In contrast, for real-time media, a new packet is worth more than an old one. This is because packets become useless if they are delayed and miss the time at which they must be played out to the user.

The former policy (old is better than new) is often called **wine** and the latter (new is better than old) is often called **milk** because most people would rather drink new milk and old wine than the alternative.

More intelligent load shedding requires cooperation from the senders. An example is packets that carry routing information. These packets are more important than regular data packets because they establish routes; if they are lost, the network may lose connectivity. Another example is that algorithms for compressing video, like MPEG, periodically transmit an entire frame and then send subsequent

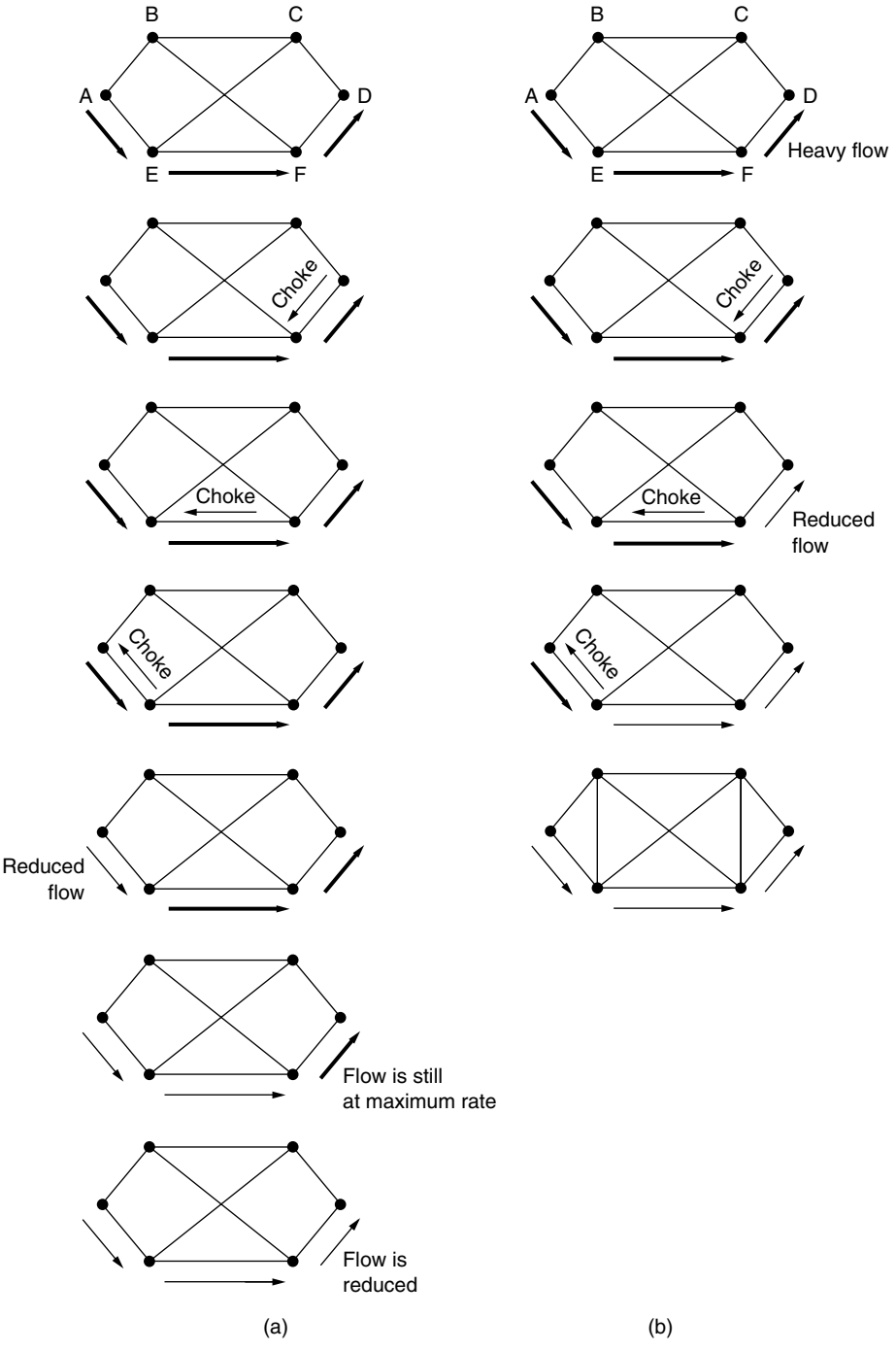


Figure 5-26. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

frames as differences from the last full frame. In this case, dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame because future packets depend on the full frame.

To implement an intelligent discard policy, applications must mark their packets to indicate to the network how important they are. Then, when packets have to be discarded, routers can first drop packets from the least important class, then the next most important class, and so on.

Of course, unless there is some significant incentive to avoid marking every packet as VERY IMPORTANT—NEVER, EVER DISCARD, nobody will do it. Often accounting and money are used to discourage frivolous marking. For example, the network might let senders send faster than the service they purchased allows if they mark excess packets as low priority. Such a strategy is actually not a bad idea because it makes more efficient use of idle resources, allowing hosts to use them as long as nobody else is interested, but without establishing a right to them when times get tough.

Random Early Detection

Dealing with congestion when it first starts is more effective than letting it gum up the works and then trying to deal with it. This observation leads to an interesting twist on load shedding, which is to discard packets before all the buffer space is really exhausted.

The motivation for this idea is that most Internet hosts do not yet get congestion signals from routers in the form of ECN. Instead, the only reliable indication of congestion that hosts get from the network is packet loss. After all, it is difficult to build a router that does not drop packets when it is overloaded. Transport protocols such as TCP are thus hardwired to react to loss as congestion, slowing down the source in response. The reasoning behind this logic is that TCP was designed for wired networks and wired networks are very reliable, so lost packets are mostly due to buffer overruns rather than transmission errors. Wireless links must recover transmission errors at the link layer (so they are not seen at the network layer) to work well with TCP.

This situation can be exploited to help reduce congestion. By having routers drop packets early, before the situation has become hopeless, there is time for the source to take action before it is too late. A popular algorithm for doing this is called **RED** (**Random Early Detection**) (Floyd and Jacobson, 1993). To determine when to start discarding, routers maintain a running average of their queue lengths. When the average queue length on some link exceeds a threshold, the link is said to be congested and a small fraction of the packets are dropped at random. Picking packets at random makes it more likely that the fastest senders will see a packet drop; this is the best option since the router cannot tell which source is causing the most trouble in a datagram network. The affected sender will notice the loss when there is no acknowledgement, and then the transport protocol

will slow down. The lost packet is thus delivering the same message as a choke packet, but implicitly, without the router sending any explicit signal.

RED routers improve performance compared to routers that drop packets only when their buffers are full, though they may require tuning to work well. For example, the ideal number of packets to drop depends on how many senders need to be notified of congestion. However, ECN is the preferred option if it is available. It works in exactly the same manner, but delivers a congestion signal explicitly rather than as a loss; RED is used when hosts cannot receive explicit signals.

5.4 QUALITY OF SERVICE

The techniques we looked at in the previous sections are designed to reduce congestion and improve network performance. However, there are applications (and customers) that demand stronger performance guarantees from the network than "the best that could be done under the circumstances.'' Multimedia applications in particular, often need a minimum throughput and maximum latency to work. In this section, we will continue our study of network performance, but now with a sharper focus on ways to provide quality of service that is matched to application needs. This is an area in which the Internet is undergoing a long-term upgrade.

An easy solution to provide good quality of service is to build a network with enough capacity for whatever traffic will be thrown at it. The name for this solution is **overprovisioning**. The resulting network will carry application traffic without significant loss and, assuming a decent routing scheme, will deliver packets with low latency. Performance doesn't get any better than this. To some extent, the telephone system is overprovisioned because it is rare to pick up a telephone and not get a dial tone instantly. There is simply so much capacity available that demand can almost always be met.

The trouble with this solution is that it is expensive. It is basically solving a problem by throwing money at it. Quality of service mechanisms let a network with less capacity meet application requirements just as well at a lower cost. Moreover, overprovisioning is based on expected traffic. All bets are off if the traffic pattern changes too much. With quality of service mechanisms, the network can honor the performance guarantees that it makes even when traffic spikes, at the cost of turning down some requests.

Four issues must be addressed to ensure quality of service:

1. What applications need from the network.
2. How to regulate the traffic that enters the network.
3. How to reserve resources at routers to guarantee performance.
4. Whether the network can safely accept more traffic.

No single technique deals efficiently with all these issues. Instead, a variety of techniques have been developed for use at the network (and transport) layer. Practical quality-of-service solutions combine multiple techniques. To this end, we will describe two versions of quality of service for the Internet called Integrated Services and Differentiated Services.

5.4.1 Application Requirements

A stream of packets from a source to a destination is called a **flow** (Clark, 1988). A flow might be all the packets of a connection in a connection-oriented network, or all the packets sent from one process to another process in a connectionless network. The needs of each flow can be characterized by four primary parameters: bandwidth, delay, jitter, and loss. Together, these determine the **QoS** (**Quality of Service**) the flow requires.

Several common applications and the stringency of their network requirements are listed in Fig. 5-27. Note that network requirements are less demanding than application requirements in those cases that the application can improve on the service provided by the network. In particular, networks do not need to be lossless for reliable file transfer, and they do not need to deliver packets with identical delays for audio and video playout. Some amount of loss can be repaired with retransmissions, and some amount of jitter can be smoothed by buffering packets at the receiver. However, there is nothing applications can do to remedy the situation if the network provides too little bandwidth or too much delay.

Application	Bandwidth	Delay	Jitter	Loss
Email	Low	Low	Low	Medium
File sharing	High	Low	Low	Medium
Web access	Medium	Medium	Low	Medium
Remote login	Low	Medium	Medium	Medium
Audio on demand	Low	Low	High	Low
Video on demand	High	Low	High	Low
Telephony	Low	High	High	Low
Videoconferencing	High	High	High	Low

Figure 5-27. Stringency of applications' quality-of-service requirements.

The applications differ in their bandwidth needs, with email, audio in all forms, and remote login not needing much, but file sharing and video in all forms needing a great deal.

More interesting are the delay requirements. File transfer applications, including email and video, are not delay sensitive. If all packets are delayed uniformly by a few seconds, no harm is done. Interactive applications, such as Web

surfing and remote login, are more delay sensitive. Real-time applications, such as telephony and videoconferencing, have strict delay requirements. If all the words in a telephone call are each delayed by too long, the users will find the connection unacceptable. On the other hand, playing audio or video files from a server does not require low delay.

The variation (i.e., standard deviation) in the delay or packet arrival times is called **jitter**. The first three applications in Fig. 5-27 are not sensitive to the packets arriving with irregular time intervals between them. Remote login is somewhat sensitive to that, since updates on the screen will appear in little bursts if the connection suffers much jitter. Video and especially audio are extremely sensitive to jitter. If a user is watching a video over the network and the frames are all delayed by exactly 2.000 seconds, no harm is done. But if the transmission time varies randomly between 1 and 2 seconds, the result will be terrible unless the application hides the jitter. For audio, a jitter of even a few milliseconds is clearly audible.

The first four applications have more stringent requirements on loss than audio and video because all bits must be delivered correctly. This goal is usually achieved with retransmissions of packets that are lost in the network by the transport layer. This is wasted work; it would be better if the network refused packets it was likely to lose in the first place. Audio and video applications can tolerate some lost packets without retransmission because people do not notice short pauses or occasional skipped frames.

To accommodate a variety of applications, networks may support different categories of QoS. An influential example comes from ATM networks, which were once part of a grand vision for networking but have since become a niche technology. They support:

1. Constant bit rate (e.g., telephony).
2. Real-time variable bit rate (e.g., compressed videoconferencing).
3. Non-real-time variable bit rate (e.g., watching a movie on demand).
4. Available bit rate (e.g., file transfer).

These categories are also useful for other purposes and other networks. Constant bit rate is an attempt to simulate a wire by providing a uniform bandwidth and a uniform delay. Variable bit rate occurs when video is compressed, with some frames compressing more than others. Sending a frame with a lot of detail in it may require sending many bits, whereas a shot of a white wall may compress extremely well. Movies on demand are not actually real time because a few seconds of video can easily be buffered at the receiver before playback starts, so jitter on the network merely causes the amount of stored-but-not-played video to vary. Available bit rate is for applications such as email that are not sensitive to delay or jitter and will take what bandwidth they can get.

5.4.2 Traffic Shaping

Before the network can make QoS guarantees, it must know what traffic is being guaranteed. In the telephone network, this characterization is simple. For example, a voice call (in uncompressed format) needs 64 kbps and consists of one 8-bit sample every 125 μsec. However, traffic in data networks is **bursty**. It typically arrives at nonuniform rates as the traffic rate varies (e.g., videoconferencing with compression), users interact with applications (e.g., browsing a new Web page), and computers switch between tasks. Bursts of traffic are more difficult to handle than constant-rate traffic because they can fill buffers and cause packets to be lost.

Traffic shaping is a technique for regulating the average rate and burstiness of a flow of data that enters the network. The goal is to allow applications to transmit a wide variety of traffic that suits their needs, including some bursts, yet have a simple and useful way to describe the possible traffic patterns to the network. When a flow is set up, the user and the network (i.e., the customer and the provider) agree on a certain traffic pattern (i.e., shape) for that flow. In effect, the customer says to the provider "My transmission pattern will look like this; can you handle it?"

Sometimes this agreement is called an **SLA** (**Service Level Agreement**), especially when it is made over aggregate flows and long periods of time, such as all of the traffic for a given customer. As long as the customer fulfills her part of the bargain and only sends packets according to the agreed-on contract, the provider promises to deliver them all in a timely fashion.

Traffic shaping reduces congestion and thus helps the network live up to its promise. However, to make it work, there is also the issue of how the provider can tell if the customer is following the agreement and what to do if the customer is not. Packets in excess of the agreed pattern might be dropped by the network, or they might be marked as having lower priority. Monitoring a traffic flow is called **traffic policing**.

Shaping and policing are not so important for peer-to-peer and other transfers that will consume any and all available bandwidth, but they are of great importance for real-time data, such as audio and video connections, which have stringent quality-of-service requirements.

Leaky and Token Buckets

We have already seen one way to limit the amount of data an application sends: the sliding window, which uses one parameter to limit how much data is in transit at any given time, which indirectly limits the rate. Now we will look at a more general way to characterize traffic, with the leaky bucket and token bucket algorithms. The formulations are slightly different but give an equivalent result.

Try to imagine a bucket with a small hole in the bottom, as illustrated in Fig. 5-28(b). No matter the rate at which water enters the bucket, the outflow is at a constant rate, *R*, when there is any water in the bucket and zero when the bucket is empty. Also, once the bucket is full to capacity *B*, any additional water entering it spills over the sides and is lost.

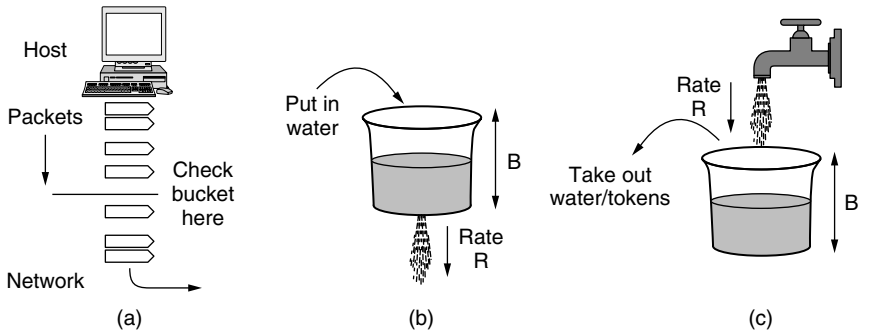


Figure 5-28. (a) Shaping packets. (b) A leaky bucket. (c) A token bucket.

This bucket can be used to shape or police packets entering the network, as shown in Fig. 5-28(a). Conceptually, each host is connected to the network by an interface containing a leaky bucket. To send a packet into the network, it must be possible to put more water into the bucket. If a packet arrives when the bucket is full, the packet must either be queued until enough water leaks out to hold it or be discarded. The former might happen at a host shaping its traffic for the network as part of the operating system. The latter might happen in hardware at a provider network interface that is policing traffic entering the network. This technique was proposed by Turner (1986) and is called the **leaky bucket algorithm**.

A different but equivalent formulation is to imagine the network interface as a bucket that is being filled, as shown in Fig. 5-28(c). The tap is running at rate *R* and the bucket has a capacity of *B*, as before. Now, to send a packet we must be able to take water, or tokens, as the contents are commonly called, out of the bucket (rather than putting water into the bucket). No more than a fixed number of tokens, *B*, can accumulate in the bucket, and if the bucket is empty, we must wait until more tokens arrive before we can send another packet. This algorithm is called the **token bucket algorithm**.

Leaky and token buckets limit the long-term rate of a flow but allow short-term bursts up to a maximum regulated length to pass through unaltered and without suffering any artificial delays. Large bursts will be smoothed by a leaky bucket traffic shaper to reduce congestion in the network. As an example, imagine that a computer can produce data at up to 1000 Mbps (125 million bytes/sec) and that the first link of the network also runs at this speed. The pattern of traffic the host generates is shown in Fig. 5-29(a). This pattern is bursty. The average

rate over one second is 200 Mbps, even though the host sends a burst of 16,000 KB at the top speed of 1000 Mbps (for 1/8 of the second).

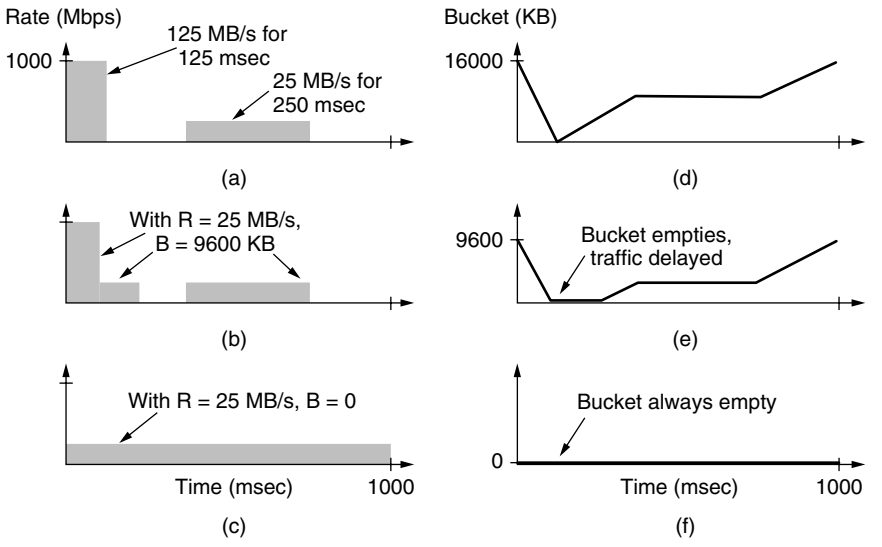


Figure 5-29. (a) Traffic from a host. Output shaped by a token bucket of rate 200 Mbps and capacity (b) 9600 KB and (c) 0 KB. Token bucket level for shaping with rate 200 Mbps and capacity (d) 16,000 KB, (e) 9600 KB, and (f) 0 KB.

Now suppose that the routers can accept data at the top speed only for short intervals, until their buffers fill up. The buffer size is 9600 KB, smaller than the traffic burst. For long intervals, the routers work best at rates not exceeding 200 Mbps (say, because this is all the bandwidth given to the customer). The implication is that if traffic is sent in this pattern, some of it will be dropped in the network because it does not fit into the buffers at routers.

To avoid this packet loss, we can shape the traffic at the host with a token bucket. If we use a rate, R, of 200 Mbps and a capacity, B, of 9600 KB, the traffic will fall within what the network can handle. The output of this token bucket is shown in Fig. 5-29(b). The host can send full throttle at 1000 Mbps for a short while until it has drained the bucket. Then it has to cut back to 200 Mbps until the burst has been sent. The effect is to spread out the burst over time because it was too large to handle all at once. The level of the token bucket is shown in Fig. 5-29(e). It starts off full and is depleted by the initial burst. When it reaches zero, new packets can be sent only at the rate at which the buffer is filling; there can be no more bursts until the bucket has recovered. The bucket fills when no traffic is being sent and stays flat when traffic is being sent at the fill rate.

We can also shape the traffic to be less bursty. Fig. 5-29(c) shows the output of a token bucket with R = 200 Mbps and a capacity of 0. This is the extreme case

in which the traffic has been completely smoothed. No bursts are allowed, and the traffic enters the network at a steady rate. The corresponding bucket level, shown in Fig. 5-29(f), is always empty. Traffic is being queued on the host for release into the network and there is always a packet waiting to be sent when it is allowed.

Finally, Fig. 5-29(d) shows the bucket level for a token bucket with $R = 200$ Mbps and a capacity of $B = 16{,}000$ KB. This is the smallest token bucket through which the traffic passes unaltered. It might be used at a router in the network to police the traffic that the host sends. If the host is sending traffic that conforms to the token bucket on which it has agreed with the network, the traffic will fit through that same token bucket run at the router at the edge of the network. If the host sends at a faster or burstier rate, the token bucket will run out of water. If this happens, a traffic policer will know that the traffic is not as described. It will then either drop the excess packets or lower their priority, depending on the design of the network. In our example, the bucket empties only momentarily, at the end of the initial burst, then recovers enough for the next burst.

Leaky and token buckets are easy to implement. We will now describe the operation of a token bucket. Even though we have described water flowing continuously into and out of the bucket, real implementations must work with discrete quantities. A token bucket is implemented with a counter for the level of the bucket. The counter is advanced by $R/\Delta T$ units at every clock tick of ΔT seconds. This would be 200 Kbit every 1 msec in our example above. Every time a unit of traffic is sent into the network, the counter is decremented, and traffic may be sent until the counter reaches zero.

When the packets are all the same size, the bucket level can just be counted in packets (e.g., 200 Mbit is 20 packets of 1250 bytes). However, often variable-sized packets are being used. In this case, the bucket level is counted in bytes. If the residual byte count is too low to send a large packet, the packet must wait until the next tick (or even longer, if the fill rate is small).

Calculating the length of the maximum burst (until the bucket empties) is slightly tricky. It is longer than just 9600 KB divided by 125 MB/sec because while the burst is being output, more tokens arrive. If we call the burst length S sec., the maximum output rate M bytes/sec, the token bucket capacity B bytes, and the token arrival rate R bytes/sec, we can see that an output burst contains a maximum of $B + RS$ bytes. We also know that the number of bytes in a maximum-speed burst of length S seconds is MS. Hence, we have

$$B + RS = MS$$

We can solve this equation to get $S = B/(M - R)$. For our parameters of $B = 9600$ KB, $M = 125$ MB/sec, and $R = 25$ MB/sec, we get a burst time of about 94 msec.

A potential problem with the token bucket algorithm is that it reduces large bursts down to the long-term rate R. It is frequently desirable to reduce the peak rate, but without going down to the long-term rate (and also without raising the

long-term rate to allow more traffic into the network). One way to get smoother traffic is to insert a second token bucket after the first one. The rate of the second bucket should be much higher than the first one. Basically, the first bucket characterizes the traffic, fixing its average rate but allowing some bursts. The second bucket reduces the peak rate at which the bursts are sent into the network. For example, if the rate of the second token bucket is set to be 500 Mbps and the capacity is set to 0, the initial burst will enter the network at a peak rate of 500 Mbps, which is lower than the 1000 Mbps rate we had previously.

Using all of these buckets can be a bit tricky. When token buckets are used for traffic shaping at hosts, packets are queued and delayed until the buckets permit them to be sent. When token buckets are used for traffic policing at routers in the network, the algorithm is simulated to make sure that no more packets are sent than permitted. Nevertheless, these tools provide ways to shape the network traffic into more manageable forms to assist in meeting quality-of-service requirements.

5.4.3 Packet Scheduling

Being able to regulate the shape of the offered traffic is a good start. However, to provide a performance guarantee, we must reserve sufficient resources along the route that the packets take through the network. To do this, we are assuming that the packets of a flow follow the same route. Spraying them over routers at random makes it hard to guarantee anything. As a consequence, something similar to a virtual circuit has to be set up from the source to the destination, and all the packets that belong to the flow must follow this route.

Algorithms that allocate router resources among the packets of a flow and between competing flows are called **packet scheduling algorithms**. Three different kinds of resources can potentially be reserved for different flows:

1. Bandwidth.
2. Buffer space.
3. CPU cycles.

The first one, bandwidth, is the most obvious. If a flow requires 1 Mbps and the outgoing line has a capacity of 2 Mbps, trying to direct three flows through that line is not going to work. Thus, reserving bandwidth means not oversubscribing any output line.

A second resource that is often in short supply is buffer space. When a packet arrives, it is buffered inside the router until it can be transmitted on the chosen outgoing line. The purpose of the buffer is to absorb small bursts of traffic as the flows contend with each other. If no buffer is available, the packet has to be discarded since there is no place to put it. For good quality of service, some buffers might be reserved for a specific flow so that flow does not have to compete for

buffers with other flows. Up to some maximum value, there will always be a buffer available when the flow needs one.

Finally, CPU cycles may also be a scarce resource. It takes router CPU time to process a packet, so a router can process only a certain number of packets per second. While modern routers are able to process most packets quickly, some kinds of packets require greater CPU processing, such as the ICMP packets we will describe in Sec. 5.6. Making sure that the CPU is not overloaded is needed to ensure timely processing of these packets.

Packet scheduling algorithms allocate bandwidth and other router resources by determining which of the buffered packets to send on the output line next. We already described the most straightforward scheduler when explaining how routers work. Each router buffers packets in a queue for each output line until they can be sent, and they are sent in the same order that they arrived. This algorithm is known as **FIFO** (**First-In First-Out**), or equivalently **FCFS** (**First-Come First-Serve**).

FIFO routers usually drop newly arriving packets when the queue is full. Since the newly arrived packet would have been placed at the end of the queue, this behavior is called **tail drop**. It is intuitive, and you may be wondering what alternatives exist. In fact, the RED algorithm we described in Sec. 5.3.5 chose a newly arriving packet to drop at random when the average queue length grew large. The other scheduling algorithms that we will describe also create other opportunities for deciding which packet to drop when the buffers are full.

FIFO scheduling is simple to implement, but it is not suited to providing good quality of service because when there are multiple flows, one flow can easily affect the performance of the other flows. If the first flow is aggressive and sends large bursts of packets, they will lodge in the queue. Processing packets in the order of their arrival means that the aggressive sender can hog most of the capacity of the routers its packets traverse, starving the other flows and reducing their quality of service. To add insult to injury, the packets of the other flows that do get through are likely to be delayed because they had to sit in the queue behind many packets from the aggressive sender.

Many packet scheduling algorithms have been devised that provide stronger isolation between flows and thwart attempts at interference (Bhatti and Crowcroft, 2000). One of the first ones was the **fair queueing** algorithm devised by Nagle (1987). The essence of this algorithm is that routers have separate queues, one for each flow for a given output line. When the line becomes idle, the router scans the queues round-robin, as shown in Fig. 5-30. It then takes the first packet on the next queue. In this way, with *n* hosts competing for the output line, each host gets to send one out of every *n* packets. It is fair in the sense that all flows get to send packets at the same rate. Sending more packets will not improve this rate.

Although a start, the algorithm has a flaw: it gives more bandwidth to hosts that use large packets than to hosts that use small packets. Demers et al. (1990) suggested an improvement in which the round-robin is done in such a way as to

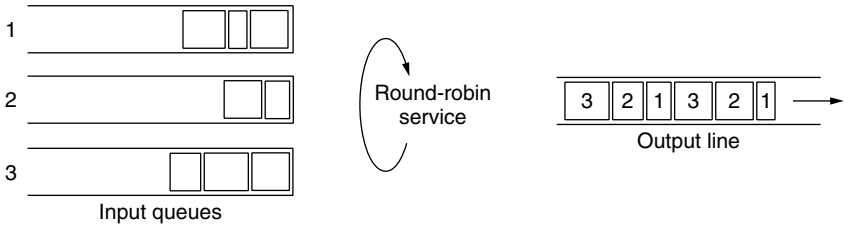
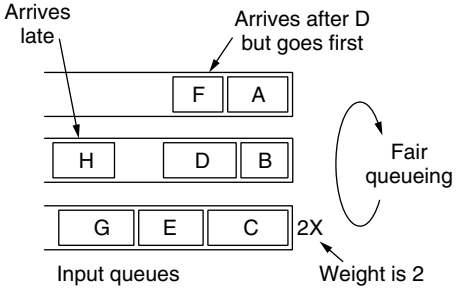


Figure 5-30. Round-robin fair queueing.

simulate a byte-by-byte round-robin, instead of a packet-by-packet round-robin. The trick is to compute a virtual time that is the number of the round at which each packet would finish being sent. Each round drains a byte from all of the queues that have data to send. The packets are then sorted in order of their finishing times and sent in that order.

This algorithm and an example of finish times for packets arriving in three flows are illustrated in Fig. 5-31. If a packet has length L, the round at which it will finish is simply L rounds after the start time. The start time is either the finish time of the previous packet, or the arrival time of the packet, if the queue is empty when it arrives.



Packet	Arrival time	Length	Finish time	Output order
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

(b)

Figure 5-31. (a) Weighted Fair Queueing. (b) Finishing times for the packets.

From the table in Fig. 5-32(b), and looking only at the first two packets in the top two queues, packets arrive in the order A, B, D, and F. Packet A arrives at round 0 and is 8 bytes long, so its finish time is round 8. Similarly the finish time for packet B is 11. Packet D arrives while B is being sent. Its finish time is 9 byte-rounds after it starts when B finishes, or 20. Similarly, the finish time for F is 16. In the absence of new arrivals, the relative sending order is A, B, F, D, even though F arrived after D. It is possible that another small packet will arrive on the top flow and obtain a finish time before D. It will only jump ahead of D if the

transmission of that packet has not started. Fair queueing does not preempt packets that are currently being transmitted. Because packets are sent in their entirety, fair queueing is only an approximation of the ideal byte-by-byte scheme. But it is a very good approximation, staying within one packet transmission of the ideal scheme at all times.

One shortcoming of this algorithm in practice is that it gives all hosts the same priority. In many situations, it is desirable to give, for example, video servers more bandwidth than, say, file servers. This is easily possible by giving the video server two or more bytes per round. This modified algorithm is called **WFQ** (**Weighted Fair Queueing**). Letting the number of bytes per round be the weight of a flow, W, we can now give the formula for computing the finish time:

$$F_i = \max(A_i, F_{i-1}) + L_i/W$$

where A_i is the arrival time, F_i is the finish time, and L_i is the length of packet i. The bottom queue of Fig. 5-31(a) has a weight of 2, so its packets are sent more quickly as you can see in the finish times given in Fig. 5-31(b).

Another practical consideration is implementation complexity. WFQ requires that packets be inserted by their finish time into a sorted queue. With N flows, this is at best an $O(\log N)$ operation per packet, which is difficult to achieve for many flows in high-speed routers. Shreedhar and Varghese (1995) describe an approximation called **deficit round robin** that can be implemented very efficiently, with only $O(1)$ operations per packet. WFQ is widely used given this approximation.

Other kinds of scheduling algorithms exist, too. A simple example is priority scheduling, in which each packet is marked with a priority. High-priority packets are always sent before any low-priority packets that are buffered. Within a priority, packets are sent in FIFO order. However, priority scheduling has the disadvantage that a burst of high-priority packets can starve low-priority packets, which may have to wait indefinitely. WFQ often provides a better alternative. By giving the high-priority queue a large weight, say 3, high-priority packets will often go through a short line (as relatively few packets should be high priority) yet some fraction of low priority packets will continue to be sent even when there is high priority traffic. A high and low priority system is essentially a two-queue WFQ system in which the high priority has infinite weight.

As a final example of a scheduler, packets might carry timestamps and be sent in timestamp order. Clark et al. (1992) describe a design in which the timestamp records how far the packet is behind or ahead of schedule as it is sent through a sequence of routers on the path. Packets that have been queued behind other packets at a router will tend to be behind schedule, and the packets that have been serviced first will tend to be ahead of schedule. Sending packets in order of their timestamps has the beneficial effect of speeding up slow packets while at the same time slowing down fast packets. The result is that all packets are delivered by the network with a more consistent delay.

5.4.4 Admission Control

We have now seen all the necessary elements for QoS and it is time to put them together to actually provide it. QoS guarantees are established through the process of admission control. We first saw admission control used to control congestion, which is a performance guarantee, albeit a weak one. The guarantees we are considering now are stronger, but the model is the same. The user offers a flow with an accompanying QoS requirement to the network. The network then decides whether to accept or reject the flow based on its capacity and the commitments it has made to other flows. If it accepts, the network reserves capacity in advance at routers to guarantee QoS when traffic is sent on the new flow.

The reservations must be made at all of the routers along the route that the packets take through the network. Any routers on the path without reservations might become congested, and a single congested router can break the QoS guarantee. Many routing algorithms find the single best path between each source and each destination and send all traffic over the best path. This may cause some flows to be rejected if there is not enough spare capacity along the best path. QoS guarantees for new flows may still be accommodated by choosing a different route for the flow that has excess capacity. This is called **QoS routing**. Chen and Nahrstedt (1998) give an overview of these techniques. It is also possible to split the traffic for each destination over multiple paths to more easily find excess capacity. A simple method is for routers to choose equal-cost paths and to divide the traffic equally or in proportion to the capacity of the outgoing links. However, more sophisticated algorithms are also available (Nelakuditi and Zhang, 2002).

Given a path, the decision to accept or reject a flow is not a simple matter of comparing the resources (bandwidth, buffers, cycles) requested by the flow with the router's excess capacity in those three dimensions. It is a little more complicated than that. To start with, although some applications may know about their bandwidth requirements, few know about buffers or CPU cycles, so at the minimum, a different way is needed to describe flows and translate this description to router resources. We will get to this shortly.

Next, some applications are far more tolerant of an occasional missed deadline than others. The applications must choose from the type of guarantees that the network can make, whether hard guarantees or behavior that will hold most of the time. All else being equal, everyone would like hard guarantees, but the difficulty is that they are expensive because they constrain worst case behavior. Guarantees for most of the packets are often sufficient for applications, and more flows with this guarantee can be supported for a fixed capacity.

Finally, some applications may be willing to haggle about the flow parameters and others may not. For example, a movie viewer that normally runs at 30 frames/sec may be willing to drop back to 25 frames/sec if there is not enough free bandwidth to support 30 frames/sec. Similarly, the number of pixels per frame, audio bandwidth, and other properties may be adjustable.

Because many parties may be involved in the flow negotiation (the sender, the receiver, and all the routers along the path between them), flows must be described accurately in terms of specific parameters that can be negotiated. A set of such parameters is called a **flow specification**. Typically, the sender (e.g., the video server) produces a flow specification proposing the parameters it would like to use. As the specification propagates along the route, each router examines it and modifies the parameters as need be. The modifications can only reduce the flow, not increase it (e.g., a lower data rate, not a higher one). When it gets to the other end, the parameters can be established.

As an example of what can be in a flow specification, consider the example of Fig. 5-32. This is based on RFCs 2210 and 2211 for Integrated Services, a QoS design we will cover in the next section. It has five parameters. The first two parameters, the *token bucket rate* and *token bucket size*, use a token bucket to give the maximum sustained rate the sender may transmit, averaged over a long time interval, and the largest burst it can send over a short time interval.

Parameter	Unit
Token bucket rate	Bytes/sec
Token bucket size	Bytes
Peak data rate	Bytes/sec
Minimum packet size	Bytes
Maximum packet size	Bytes

Figure 5-32. An example flow specification.

The third parameter, the *peak data rate*, is the maximum transmission rate tolerated, even for brief time intervals. The sender must never exceed this rate even for short bursts.

The last two parameters specify the minimum and maximum packet sizes, including the transport and network layer headers (e.g., TCP and IP). The minimum size is useful because processing each packet takes some fixed time, no matter how short. A router may be prepared to handle 10,000 packets/sec of 1 KB each, but not be prepared to handle 100,000 packets/sec of 50 bytes each, even though this represents a lower data rate. The maximum packet size is important due to internal network limitations that may not be exceeded. For example, if part of the path goes over an Ethernet, the maximum packet size will be restricted to no more than 1500 bytes no matter what the rest of the network can handle.

An interesting question is how a router turns a flow specification into a set of specific resource reservations. At first glance, it might appear that if a router has a link that runs at, say, 1 Gbps and the average packet is 1000 bits, it can process 1 million packets/sec. This observation is not the case, though, because there will always be idle periods on the link due to statistical fluctuations in the load. If the

link needs every bit of capacity to get its work done, idling for even a few bits creates a backlog it can never get rid of.

Even with a load slightly below the theoretical capacity, queues can build up and delays can occur. Consider a situation in which packets arrive at random with a mean arrival rate of λ packets/sec. The packets have random lengths and can be sent on the link with a mean service rate of μ packets/sec. Under the assumption that both the arrival and service distributions are Poisson distributions (what is called an M/M/1 queueing system, where "M" stands for Markov, i.e., Poisson), it can be proven using queueing theory that the mean delay experienced by a packet, T, is

$$T = \frac{1}{\mu} \times \frac{1}{1 - \lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1 - \rho}$$

where $\rho = \lambda/\mu$ is the CPU utilization. The first factor, $1/\mu$, is what the service time would be in the absence of competition. The second factor is the slowdown due to competition with other flows. For example, if $\lambda = 950{,}000$ packets/sec and $\mu = 1{,}000{,}000$ packets/sec, then $\rho = 0.95$ and the mean delay experienced by each packet will be 20 μsec instead of 1 μsec. This time accounts for both the queueing time and the service time, as can be seen when the load is very low ($\lambda/\mu \approx 0$). If there are, say, 30 routers along the flow's route, queueing delay alone will account for 600 μsec of delay.

One method of relating flow specifications to router resources that correspond to bandwidth and delay performance guarantees is given by Parekh and Gallagher (1993, 1994). It is based on traffic sources shaped by (R, B) token buckets and WFQ at routers. Each flow is given a WFQ weight W large enough to drain its token bucket rate R as shown in Fig. 5-33. For example, if the flow has a rate of 1 Mbps and the router and output link have a capacity of 1 Gbps, the weight for the flow must be greater than 1/1000th of the total of the weights for all of the flows at that router for the output link. This guarantees the flow a minimum bandwidth. If it cannot be given a large enough rate, the flow cannot be admitted.

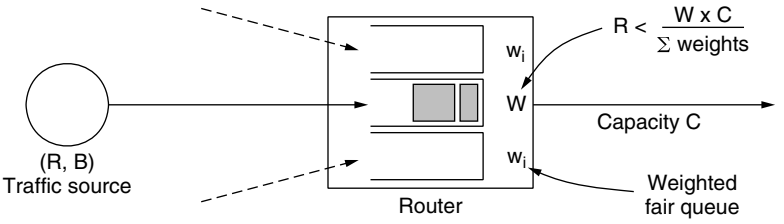


Figure 5-33. Bandwidth and delay guarantees with token buckets and WFQ.

The largest queueing delay the flow will see is a function of the burst size of the token bucket. Consider the two extreme cases. If the traffic is smooth, without

any bursts, packets will be drained from the router just as quickly as they arrive. There will be no queueing delay (ignoring packetization effects). On the other hand, if the traffic is saved up in bursts, then a maximum-size burst, *B*, may arrive at the router all at once. In this case the maximum queueing delay, *D*, will be the time taken to drain this burst at the guaranteed bandwidth, or *B/R* (again, ignoring packetization effects). If this delay is too large, the flow must request more bandwidth from the network.

These guarantees are hard. The token buckets bound the burstiness of the source, and fair queueing isolates the bandwidth given to different flows. This means that the flow will meet its bandwidth and delay guarantees regardless of how the other competing flows behave at the router. Those other flows cannot break the guarantee even by saving up traffic and all sending at once.

Moreover, the result holds for a path through multiple routers in any network topology. Each flow gets a minimum bandwidth because that bandwidth is guaranteed at each router. The reason each flow gets a maximum delay is more subtle. In the worst case that a burst of traffic hits the first router and competes with the traffic of other flows, it will be delayed up to the maximum delay of *D*. However, this delay will also smooth the burst. In turn, this means that the burst will incur no further queueing delays at later routers. The overall queueing delay will be at most *D*.

5.4.5 Integrated Services

Between 1995 and 1997, IETF put a lot of effort into devising an architecture for streaming multimedia. This work resulted in over two dozen RFCs, starting with RFCs 2205–2212. The generic name for this work is **integrated services**. It was aimed at both unicast and multicast applications. An example of the former is a single user streaming a video clip from a news site. An example of the latter is a collection of digital television stations broadcasting their programs as streams of IP packets to many receivers at various locations. Below we will concentrate on multicast, since unicast is a special case of multicast.

In many multicast applications, groups can change membership dynamically, for example, as people enter a video conference and then get bored and switch to a soap opera or the croquet channel. Under these conditions, the approach of having the senders reserve bandwidth in advance does not work well, since it would require each sender to track all entries and exits of its audience. For a system designed to transmit television with millions of subscribers, it would not work at all.

RSVP—The Resource reSerVation Protocol

The main part of the integrated services architecture that is visible to the users of the network is **RSVP**. It is described in RFCs 2205–2210. This protocol is used for making the reservations; other protocols are used for sending the data.

RSVP allows multiple senders to transmit to multiple groups of receivers, permits individual receivers to switch channels freely, and optimizes bandwidth use while at the same time eliminating congestion.

In its simplest form, the protocol uses multicast routing using spanning trees, as discussed earlier. Each group is assigned a group address. To send to a group, a sender puts the group's address in its packets. The standard multicast routing algorithm then builds a spanning tree covering all group members. The routing algorithm is not part of RSVP. The only difference from normal multicasting is a little extra information that is multicast to the group periodically to tell the routers along the tree to maintain certain data structures in their memories.

As an example, consider the network of Fig. 5-34(a). Hosts 1 and 2 are multicast senders, and hosts 3, 4, and 5 are multicast receivers. In this example, the senders and receivers are disjoint, but in general, the two sets may overlap. The multicast trees for hosts 1 and 2 are shown in Fig. 5-34(b) and Fig. 5-34(c), respectively.

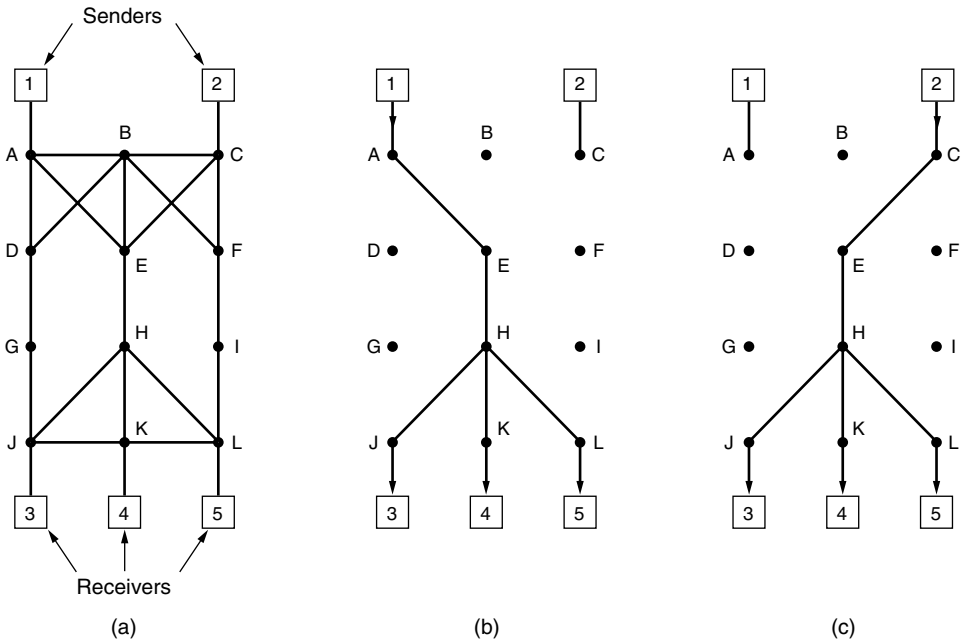


Figure 5-34. (a) A network. (b) The multicast spanning tree for host 1. (c) The multicast spanning tree for host 2.

To get better reception and eliminate congestion, any of the receivers in a group can send a reservation message up the tree to the sender. The message is propagated using the reverse path forwarding algorithm discussed earlier. At each

hop, the router notes the reservation and reserves the necessary bandwidth. We saw in the previous section how a weighted fair queueing scheduler can be used to make this reservation. If insufficient bandwidth is available, it reports back failure. By the time the message gets back to the source, bandwidth has been reserved all the way from the sender to the receiver making the reservation request along the spanning tree.

An example of such a reservation is shown in Fig. 5-35(a). Here host 3 has requested a channel to host 1. Once it has been established, packets can flow from 1 to 3 without congestion. Now consider what happens if host 3 next reserves a channel to the other sender, host 2, so the user can watch two television programs at once. A second path is reserved, as illustrated in Fig. 5-35(b). Note that two separate channels are needed from host 3 to router *E* because two independent streams are being transmitted.

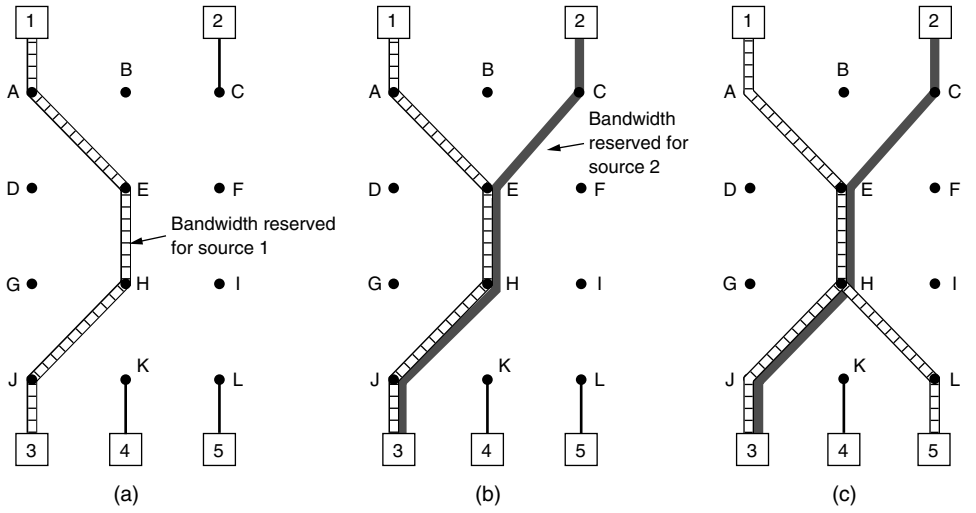


Figure 5-35. (a) Host 3 requests a channel to host 1. (b) Host 3 then requests a second channel, to host 2. (c) Host 5 requests a channel to host 1.

Finally, in Fig. 5-35(c), host 5 decides to watch the program being transmitted by host 1 and also makes a reservation. First, dedicated bandwidth is reserved as far as router *H*. However, this router sees that it already has a feed from host 1, so if the necessary bandwidth has already been reserved, it does not have to reserve any more. Note that hosts 3 and 5 might have asked for different amounts of bandwidth (e.g., if host 3 is playing on a small screen and only wants the low-resolution information), so the capacity reserved must be large enough to satisfy the greediest receiver.

When making a reservation, a receiver can (optionally) specify one or more sources that it wants to receive from. It can also specify whether these choices

are fixed for the duration of the reservation or whether the receiver wants to keep open the option of changing sources later. The routers use this information to optimize bandwidth planning. In particular, two receivers are only set up to share a path if they both agree not to change sources later on.

The reason for this strategy in the fully dynamic case is that reserved bandwidth is decoupled from the choice of source. Once a receiver has reserved bandwidth, it can switch to another source and keep that portion of the existing path that is valid for the new source. If host 2 is transmitting several video streams in real time, for example a TV broadcaster with multiple channels, host 3 may switch between them at will without changing its reservation: the routers do not care what program the receiver is watching.

5.4.6 Differentiated Services

Flow-based algorithms have the potential to offer good quality of service to one or more flows because they reserve whatever resources are needed along the route. However, they also have a downside. They require an advance setup to establish each flow, something that does not scale well when there are thousands or millions of flows. Also, they maintain internal per-flow state in the routers, making them vulnerable to router crashes. Finally, the changes required to the router code are substantial and involve complex router-to-router exchanges for setting up the flows. As a consequence, while work continues to advance integrated services, few deployments of it or anything like it exist yet.

For these reasons, IETF has also devised a simpler approach to quality of service, one that can be largely implemented locally in each router without advance setup and without having the whole path involved. This approach is known as **class-based** (as opposed to flow-based) quality of service. IETF has standardized an architecture for it, called **differentiated services**, which is described in RFCs 2474, 2475, and numerous others. We will now describe it.

Differentiated services can be offered by a set of routers forming an administrative domain (e.g., an ISP or a telco). The administration defines a set of service classes with corresponding forwarding rules. If a customer subscribes to differentiated services, customer packets entering the domain are marked with the class to which they belong. This information is carried in the *Differentiated services* field of IPv4 and IPv6 packets (described in Sec. 5.6). The classes are defined as **per hop behaviors** because they correspond to the treatment the packet will receive at each router, not a guarantee across the network. Better service is provided to packets with some per-hop behaviors (e.g., premium service) than to others (e.g., regular service). Traffic within a class may be required to conform to some specific shape, such as a leaky bucket with some specified drain rate. An operator with a nose for business might charge extra for each premium packet transported or might allow up to N premium packets per month for a fixed additional monthly fee. Note that this scheme requires no advance setup, no resource

reservation, and no time-consuming end-to-end negotiation for each flow, as with integrated services. This makes differentiated services relatively easy to implement.

Class-based service also occurs in other industries. For example, package delivery companies often offer overnight, two-day, and three-day service. Airlines offer first class, business class, and cattle-class service. Long-distance trains often have multiple service classes. Even the Paris subway has two different service classes. For packets, the classes may differ in terms of delay, jitter, and probability of being discarded in the event of congestion, among other possibilities (but probably not roomier Ethernet frames).

To make the difference between flow-based quality of service and class-based quality of service clearer, consider an example: Internet telephony. With a flow-based scheme, each telephone call gets its own resources and guarantees. With a class-based scheme, all the telephone calls together get the resources reserved for the class telephony. These resources cannot be taken away by packets from the Web browsing class or other classes, but no telephone call gets any private resources reserved for it alone.

Expedited Forwarding

The choice of service classes is up to each operator, but since packets are often forwarded between networks run by different operators, IETF has defined some network-independent service classes. The simplest class is **expedited forwarding**, so let us start with that one. It is described in RFC 3246.

The idea behind expedited forwarding is very simple. Two classes of service are available: regular and expedited. The vast majority of the traffic is expected to be regular, but a limited fraction of the packets are expedited. The expedited packets should be able to transit the network as though no other packets were present. In this way they will get low loss, low delay and low jitter service—just what is needed for VoIP. A symbolic representation of this "two-tube" system is given in Fig. 5-36. Note that there is still just one physical line. The two logical pipes shown in the figure represent a way to reserve bandwidth for different classes of service, not a second physical line.

One way to implement this strategy is as follows. Packets are classified as expedited or regular and marked accordingly. This step might be done on the sending host or in the ingress (first) router. The advantage of doing classification on the sending host is that more information is available about which packets belong to which flows. This task may be performed by networking software or even the operating system, to avoid having to change existing applications. For example, it is becoming common for VoIP packets to be marked for expedited service by hosts. If the packets pass through a corporate network or ISP that supports expedited service, they will receive preferential treatment. If the network does not support expedited service, no harm is done.

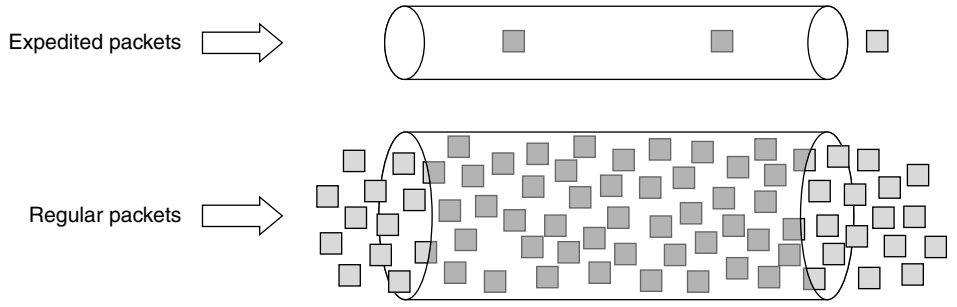


Figure 5-36. Expedited packets experience a traffic-free network.

Of course, if the marking is done by the host, the ingress router is likely to police the traffic to make sure that customers are not sending more expedited traffic than they have paid for. Within the network, the routers may have two output queues for each outgoing line, one for expedited packets and one for regular packets. When a packet arrives, it is queued accordingly. The expedited queue is given priority over the regular one, for example, by using a priority scheduler. In this way, expedited packets see an unloaded network, even when there is, in fact, a heavy load of regular traffic.

Assured Forwarding

A somewhat more elaborate scheme for managing the service classes is called **assured forwarding**. It is described in RFC 2597. Assured forwarding specifies that there shall be four priority classes, each class having its own resources. The top three classes might be called gold, silver, and bronze. In addition, it defines three discard classes for packets that are experiencing congestion: low, medium, and high. Taken together, these two factors define 12 service classes.

Figure 5-37 shows one way packets might be processed under assured forwarding. The first step is to classify the packets into one of the four priority classes. As before, this step might be done on the sending host (as shown in the figure) or in the ingress router, and the rate of higher-priority packets may be limited by the operator as part of the service offering.

The next step is to determine the discard class for each packet. This is done by passing the packets of each priority class through a traffic policer such as a token bucket. The policer lets all of the traffic through, but it identifies packets that fit within small bursts as low discard, packets that exceed small bursts as medium discard, and packets that exceed large bursts as high discard. The combination of priority and discard class is then encoded in each packet.

Finally, the packets are processed by routers in the network with a packet scheduler that distinguishes the different classes. A common choice is to use

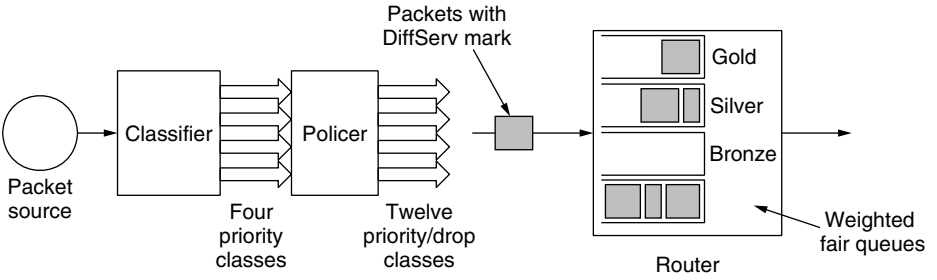


Figure 5-37. A possible implementation of assured forwarding.

weighted fair queueing for the four priority classes, with higher classes given higher weights. In this way, the higher classes will get most of the bandwidth, but the lower classes will not be starved of bandwidth entirely. For example, if the weights double from one class to the next higher class, the higher class will get twice the bandwidth. Within a priority class, packets with a higher discard class can be preferentially dropped by running an algorithm such as RED (Random Early Detection), which we saw in Sec. 5.3.5. RED will start to drop packets as congestion builds but before the router has run out of buffer space. At this stage, there is still buffer space with which to accept low discard packets while dropping high discard packets.

5.5 INTERNETWORKING

Until now, we have implicitly assumed that there is a single homogeneous network, with each machine using the same protocol in each layer. Unfortunately, this assumption is wildly optimistic. Many different networks exist, including PANs, LANs, MANs, and WANs. We have described Ethernet, Internet over cable, the fixed and mobile telephone networks, 802.11, 802.16, and more. Numerous protocols are in widespread use across these networks in every layer. In the following sections, we will take a careful look at the issues that arise when two or more networks are connected to form an **internetwork**, or more simply an **internet**.

It would be much simpler to join networks together if everyone used a single networking technology, and it is often the case that there is a dominant kind of network, such as Ethernet. Some pundits speculate that the multiplicity of technologies will go away as soon as everyone realizes how wonderful [fill in your favorite network] is. Do not count on it. History shows this to be wishful thinking. Different kinds of networks grapple with different problems, so, for example, Ethernet and satellite networks are always likely to differ. Reusing existing systems, such as running data networks on top of cable, the telephone network, and power

lines, adds constraints that cause the features of the networks to diverge. Heterogeneity is here to stay.

If there will always be different networks, it would be simpler if we did not need to interconnect them. This also is unlikely. Bob Metcalfe postulated that the value of a network with N nodes is the number of connections that may be made between the nodes, or N^2 (Gilder, 1993). This means that large networks are much more valuable than small networks because they allow many more connections, so there always will be an incentive to combine smaller networks.

The Internet is the prime example of this interconnection. (We will write Internet with a capital "I" to distinguish it from other internets, or connected networks.) The purpose of joining all these networks is to allow users on any of them to communicate with users on all the other ones. When you pay an ISP for Internet service, you may be charged depending on the bandwidth of your line, but what you are really paying for is the ability to exchange packets with any other host that is also connected to the Internet. After all, the Internet would not be very popular if you could only send packets to other hosts in the same city.

Since networks often differ in important ways, getting packets from one network to another is not always so easy. We must address problems of heterogeneity, and also problems of scale as the resulting internet grows very large. We will begin by looking at how networks can differ to see what we are up against. Then we shall see the approach used so successfully by IP (Internet Protocol), the network layer protocol of the Internet, including techniques for tunneling through networks, routing in internetworks, and packet fragmentation.

5.5.1 How Networks Differ

Networks can differ in many ways. Some of the differences, such as different modulation techniques or frame formats, are internal to the physical and data link layers. These differences will not concern us here. Instead, in Fig. 5-38 we list some of the differences that can be exposed to the network layer. It is papering over these differences that makes internetworking more difficult than operating within a single network.

When packets sent by a source on one network must transit one or more foreign networks before reaching the destination network, many problems can occur at the interfaces between networks. To start with, the source needs to be able to address the destination. What do we do if the source is on an Ethernet network and the destination is on a WiMAX network? Assuming we can even specify a WiMAX destination from an Ethernet network, packets would cross from a connectionless network to a connection-oriented one. This may require that a new connection be set up on short notice, which injects a delay, and much overhead if the connection is not used for many more packets.

Many specific differences may have to be accommodated as well. How do we multicast a packet to a group with some members on a network that does not

Item	Some Possibilities
Service offered	Connectionless versus connection oriented
Addressing	Different sizes, flat or hierarchical
Broadcasting	Present or absent (also multicast)
Packet size	Every network has its own maximum
Ordering	Ordered and unordered delivery
Quality of service	Present or absent; many different kinds
Reliability	Different levels of loss
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, packet, byte, or not at all

Figure 5-38. Some of the many ways networks can differ.

support multicast? The differing max packet sizes used by different networks can be a major nuisance, too. How do you pass an 8000-byte packet through a network whose maximum size is 1500 bytes? If packets on a connection-oriented network transit a connectionless network, they may arrive in a different order than they were sent. That is something the sender likely did not expect, and it might come as an (unpleasant) surprise to the receiver as well.

These kinds of differences can be papered over, with some effort. For example, a gateway joining two networks might generate separate packets for each destination in lieu of better network support for multicasting. A large packet might be broken up, sent in pieces, and then joined back together. Receivers might buffer packets and deliver them in order.

Networks also can differ in large respects that are more difficult to reconcile. The clearest example is quality of service. If one network has strong QoS and the other offers best effort service, it will be impossible to make bandwidth and delay guarantees for real-time traffic end to end. In fact, they can likely only be made while the best-effort network is operated at a low utilization, or hardly used, which is unlikely to be the goal of most ISPs. Security mechanisms are problematic, but at least encryption for confidentiality and data integrity can be layered on top of networks that do not already include it. Finally, differences in accounting can lead to unwelcome bills when normal usage suddenly becomes expensive, as roaming mobile phone users with data plans have discovered.

5.5.2 How Networks Can Be Connected

There are two basic choices for connecting different networks: we can build devices that translate or convert packets from each kind of network into packets for each other network, or, like good computer scientists, we can try to solve the

problem by adding a layer of indirection and building a common layer on top of the different networks. In either case, the devices are placed at the boundaries between networks.

Early on, Cerf and Kahn (1974) argued for a common layer to hide the differences of existing networks. This approach has been tremendously successful, and the layer they proposed was eventually separated into the TCP and IP protocols. Almost four decades later, IP is the foundation of the modern Internet. For this accomplishment, Cerf and Kahn were awarded the 2004 Turing Award, informally known as the Nobel Prize of computer science. IP provides a universal packet format that all routers recognize and that can be passed through almost every network. IP has extended its reach from computer networks to take over the telephone network. It also runs on sensor networks and other tiny devices that were once presumed too resource-constrained to support it.

We have discussed several different devices that connect networks, including repeaters, hubs, switches, bridges, routers, and gateways. Repeaters and hubs just move bits from one wire to another. They are mostly analog devices and do not understand anything about higher layer protocols. Bridges and switches operate at the link layer. They can be used to build networks, but only with minor protocol translation in the process, for example, between 10, 100 and 1000 Mbps Ethernet switches. Our focus in this section is interconnection devices that operate at the network layer, namely the routers. We will leave gateways, which are higher-layer interconnection devices, until later.

Let us first explore at a high level how interconnection with a common network layer can be used to interconnect dissimilar networks. An internet comprised of 802.11, MPLS, and Ethernet networks is shown in Fig. 5-39(a). Suppose that the source machine on the 802.11 network wants to send a packet to the destination machine on the Ethernet network. Since these technologies are different, and they are further separated by another kind of network (MPLS), some added processing is needed at the boundaries between the networks.

Because different networks may, in general, have different forms of addressing, the packet carries a network layer address that can identify any host across the three networks. The first boundary the packet reaches is when it transitions from an 802.11 network to an MPLS network. 802.11 provides a connectionless service, but MPLS provides a connection-oriented service. This means that a virtual circuit must be set up to cross that network. Once the packet has traveled along the virtual circuit, it will reach the Ethernet network. At this boundary, the packet may be too large to be carried, since 802.11 can work with larger frames than Ethernet. To handle this problem, the packet is divided into fragments, and each fragment is sent separately. When the fragments reach the destination, they are reassembled. Then the packet has completed its journey.

The protocol processing for this journey is shown in Fig. 5-39(b). The source accepts data from the transport layer and generates a packet with the common network layer header, which is IP in this example. The network header contains the

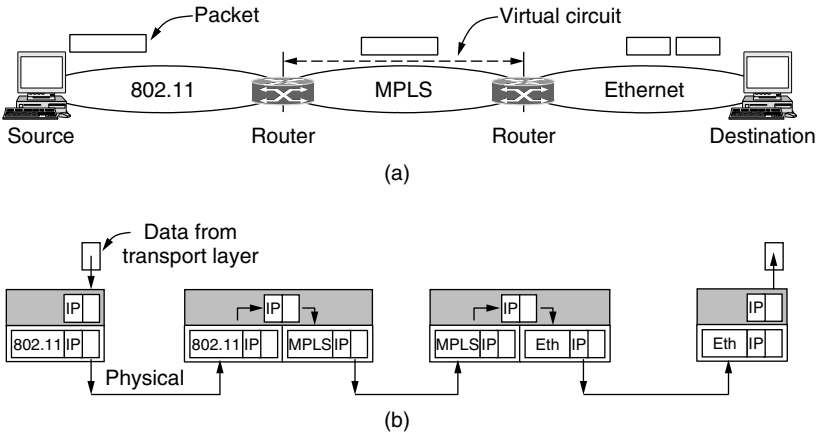


Figure 5-39. (a) A packet crossing different networks. (b) Network and link layer protocol processing.

ultimate destination address, which is used to determine that the packet should be sent via the first router. So the packet is encapsulated in an 802.11 frame whose destination is the first router and transmitted. At the router, the packet is removed from the frame's data field and the 802.11 frame header is discarded. The router now examines the IP address in the packet and looks up this address in its routing table. Based on this address, it decides to send the packet to the second router next. For this part of the path, an MPLS virtual circuit must be established to the second router and the packet must be encapsulated with MPLS headers that travel this circuit. At the far end, the MPLS header is discarded and the network address is again consulted to find the next network layer hop. It is the destination itself. Since the packet is too long to be sent over Ethernet, it is split into two portions. Each of these portions is put into the data field of an Ethernet frame and sent to the Ethernet address of the destination. At the destination, the Ethernet header is stripped from each of the frames, and the contents are reassembled. The packet has finally reached its destination.

Observe that there is an essential difference between the routed case and the switched (or bridged) case. With a router, the packet is extracted from the frame and the network address in the packet is used for deciding where to send it. With a switch (or bridge), the entire frame is transported on the basis of its MAC address. Switches do not have to understand the network layer protocol being used to switch packets. Routers do.

Unfortunately, internetworking is not as easy as we have made it sound. In fact, when bridges were introduced, it was intended that they would join different types of networks, or at least different types of LANs. They were to do this by translating frames from one LAN into frames from another LAN. However, this

did not work well, for the same reason that internetworking is difficult: the differences in the features of LANs, such as different maximum packet sizes and LANs with and without priority classes, are hard to mask. Today, bridges are predominantly used to connect the same kind of network at the link layer, and routers connect different networks at the network layer.

Internetworking has been very successful at building large networks, but it only works when there is a common network layer. There have, in fact, been many network protocols over time. Getting everybody to agree on a single format is difficult when companies perceive it to their commercial advantage to have a proprietary format that they control. Examples besides IP, which is now the near-universal network protocol, were IPX, SNA, and AppleTalk. None of these protocols are still in widespread use, but there will always be other protocols. The most relevant example now is probably IPv4 and IPv6. While these are both versions of IP, they are not compatible (or it would not have been necessary to create IPv6).

A router that can handle multiple network protocols is called a **multiprotocol router**. It must either translate the protocols, or leave connection for a higher protocol layer. Neither approach is entirely satisfactory. Connection at a higher layer, say, by using TCP, requires that all the networks implement TCP (which may not be the case). Then, it limits usage across the networks to applications that use TCP (which does not include many real-time applications).

The alternative is to translate packets between the networks. However, unless the packet formats are close relatives with the same information fields, such conversions will always be incomplete and often doomed to failure. For example, IPv6 addresses are 128 bits long. They will not fit in a 32-bit IPv4 address field, no matter how hard the router tries. Getting IPv4 and IPv6 to run in the same network has proven to be a major obstacle to the deployment of IPv6. (To be fair, so has getting customers to understand why they should want IPv6 in the first place.) Greater problems can be expected when translating between fundamentally different protocols, such as connectionless and connection-oriented network protocols. Given these difficulties, conversion is only rarely attempted. Arguably, even IP has only worked so well by serving as a kind of lowest common denominator. It requires little of the networks on which it runs, but offers only best-effort service as a result.

5.5.3 Tunneling

Handling the general case of making two different networks interwork is exceedingly difficult. However, there is a common special case that is manageable even for different network protocols. This case is where the source and destination hosts are on the same type of network, but there is a different network in between. As an example, think of an international bank with an IPv6 network

in Paris, an IPv6 network in London and connectivity between the offices via the IPv4 Internet. This situation is shown in Fig. 5-40.

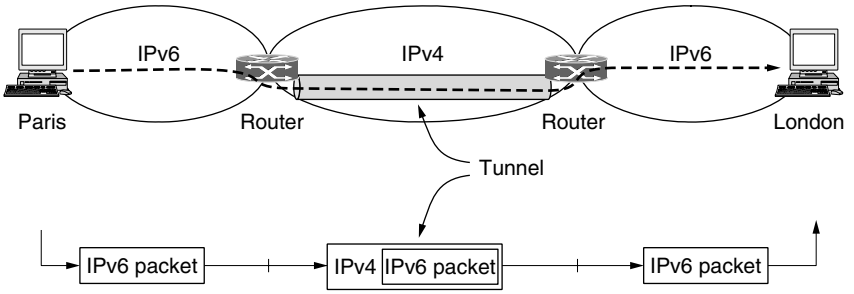


Figure 5-40. Tunneling a packet from Paris to London.

The solution to this problem is a technique called **tunneling**. To send an IP packet to a host in the London office, a host in the Paris office constructs the packet containing an IPv6 address in London, and sends it to the multiprotocol router that connects the Paris IPv6 network to the IPv4 Internet. When this router gets the IPv6 packet, it encapsulates the packet with an IPv4 header addressed to the IPv4 side of the multiprotocol router that connects to the London IPv6 network. That is, the router puts a (IPv6) packet inside a (IPv4) packet. When this wrapped packet arrives, the London router removes the original IPv6 packet and sends it onward to the destination host.

The path through the IPv4 Internet can be seen as a big tunnel extending from one multiprotocol router to the other. The IPv6 packet just travels from one end of the tunnel to the other, snug in its nice box. It does not have to worry about dealing with IPv4 at all. Neither do the hosts in Paris or London. Only the multiprotocol routers have to understand both IPv4 and IPv6 packets. In effect, the entire trip from one multiprotocol router to the other is like a hop over a single link.

An analogy may make tunneling clearer. Consider a person driving her car from Paris to London. Within France, the car moves under its own power, but when it hits the English Channel, it is loaded onto a high-speed train and transported to England through the Chunnel (cars are not permitted to drive through the Chunnel). Effectively, the car is being carried as freight, as depicted in Fig. 5-41. At the far end, the car is let loose on the English roads and once again continues to move under its own power. Tunneling of packets through a foreign network works the same way.

Tunneling is widely used to connect isolated hosts and networks using other networks. The network that results is called an **overlay** since it has effectively been overlaid on the base network. Deployment of a network protocol with a new feature is a common reason, as our "IPv6 over IPv4" example shows. The disadvantage of tunneling is that none of the hosts on the network that is tunneled over can be reached because the packets cannot escape in the middle of the tunnel.

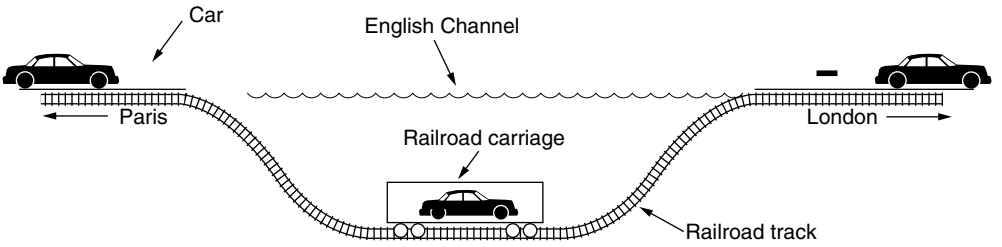


Figure 5-41. Tunneling a car from France to England.

However, this limitation of tunnels is turned into an advantage with **VPNs** (**Virtual Private Networks**). A VPN is simply an overlay that is used to provide a measure of security. We will explore VPNs when we get to Chap. 8.

5.5.4 Internetwork Routing

Routing through an internet poses the same basic problem as routing within a single network, but with some added complications. To start, the networks may internally use different routing algorithms. For example, one network may use link state routing and another distance vector routing. Since link state algorithms need to know the topology but distance vector algorithms do not, this difference alone would make it unclear how to find the shortest paths across the internet.

Networks run by different operators lead to bigger problems. First, the operators may have different ideas about what is a good path through the network. One operator may want the route with the least delay, while another may want the most inexpensive route. This will lead the operators to use different quantities to set the shortest-path costs (e.g., milliseconds of delay vs. monetary cost). The weights will not be comparable across networks, so shortest paths on the internet will not be well defined.

Worse yet, one operator may not want another operator to even know the details of the paths in its network, perhaps because the weights and paths may reflect sensitive information (such as the monetary cost) that represents a competitive business advantage.

Finally, the internet may be much larger than any of the networks that comprise it. It may therefore require routing algorithms that scale well by using a hierarchy, even if none of the individual networks need to use a hierarchy.

All of these considerations lead to a two-level routing algorithm. Within each network, an **intradomain** or **interior gateway protocol** is used for routing. (“Gateway’’ is an older term for “router.’’) It might be a link state protocol of the kind we have already described. Across the networks that make up the internet, an **interdomain** or **exterior gateway protocol** is used. The networks may all use different intradomain protocols, but they must use the same interdomain protocol.

In the Internet, the interdomain routing protocol is called **BGP** (**Border Gateway Protocol**). We will describe it in the next section.

There is one more important term to introduce. Since each network is operated independently of all the others, it is often referred to as an **AS** (**Autonomous System**). A good mental model for an AS is an ISP network. In fact, an ISP network may be comprised of more than one AS, if it is managed, or, has been acquired, as multiple networks. But the difference is usually not significant.

The two levels are usually not strictly hierarchical, as highly suboptimal paths might result if a large international network and a small regional network were both abstracted to be a single network. However, relatively little information about routes within the networks is exposed to find routes across the internetwork. This helps to address all of the complications. It improves scaling and lets operators freely select routes within their own networks using a protocol of their choosing. It also does not require weights to be compared across networks or expose sensitive information outside of networks.

However, we have said little so far about how the routes across the networks of the internet are determined. In the Internet, a large determining factor is the business arrangements between ISPs. Each ISP may charge or receive money from the other ISPs for carrying traffic. Another factor is that if internetwork routing requires crossing international boundaries, various laws may suddenly come into play, such as Sweden's strict privacy laws about exporting personal data about Swedish citizens from Sweden. All of these nontechnical factors are wrapped up in the concept of a **routing policy** that governs the way autonomous networks select the routes that they use. We will return to routing policies when we describe BGP.

5.5.5 Packet Fragmentation

Each network or link imposes some maximum size on its packets. These limits have various causes, among them

1. Hardware (e.g., the size of an Ethernet frame).
2. Operating system (e.g., all buffers are 512 bytes).
3. Protocols (e.g., the number of bits in the packet length field).
4. Compliance with some (inter)national standard.
5. Desire to reduce error-induced retransmissions to some level.
6. Desire to prevent one packet from occupying the channel too long.

The result of all these factors is that the network designers are not free to choose any old maximum packet size they wish. Maximum payloads for some common

technologies are 1500 bytes for Ethernet and 2272 bytes for 802.11. IP is more generous, allows for packets as big as 65,515 bytes.

Hosts usually prefer to transmit large packets because this reduces packet overheads such as bandwidth wasted on header bytes. An obvious internetworking problem appears when a large packet wants to travel through a network whose maximum packet size is too small. This nuisance has been a persistent issue, and solutions to it have evolved along with much experience gained on the Internet.

One solution is to make sure the problem does not occur in the first place. However, this is easier said than done. A source does not usually know the path a packet will take through the network to a destination, so it certainly does not know how small packets must be to get there. This packet size is called the **Path MTU** (**Path Maximum Transmission Unit**). Even if the source did know the path MTU, packets are routed independently in a connectionless network such as the Internet. This routing means that paths may suddenly change, which can unexpectedly change the path MTU.

The alternative solution to the problem is to allow routers to break up packets into **fragments**, sending each fragment as a separate network layer packet. However, as every parent of a small child knows, converting a large object into small fragments is considerably easier than the reverse process. (Physicists have even given this effect a name: the second law of thermodynamics.) Packet-switching networks, too, have trouble putting the fragments back together again.

Two opposing strategies exist for recombining the fragments back into the original packet. The first strategy is to make fragmentation caused by a ''small-packet'' network transparent to any subsequent networks through which the packet must pass on its way to the ultimate destination. This option is shown in Fig. 5-42(a). In this approach, when an oversized packet arrives at *G1*, the router breaks it up into fragments. Each fragment is addressed to the same exit router, *G2*, where the pieces are recombined. In this way, passage through the small-packet network is made transparent. Subsequent networks are not even aware that fragmentation has occurred.

Transparent fragmentation is straightforward but has some problems. For one thing, the exit router must know when it has received all the pieces, so either a count field or an ''end of packet'' bit must be provided. Also, because all packets must exit via the same router so that they can be reassembled, the routes are constrained. By not allowing some fragments to follow one route to the ultimate destination and other fragments a disjoint route, some performance may be lost. More significant is the amount of work that the router may have to do. It may need to buffer the fragments as they arrive, and decide when to throw them away if not all of the fragments arrive. Some of this work may be wasteful, too, as the packet may pass through a series of small packet networks and need to be repeatedly fragmented and reassembled.

The other fragmentation strategy is to refrain from recombining fragments at any intermediate routers. Once a packet has been fragmented, each fragment is

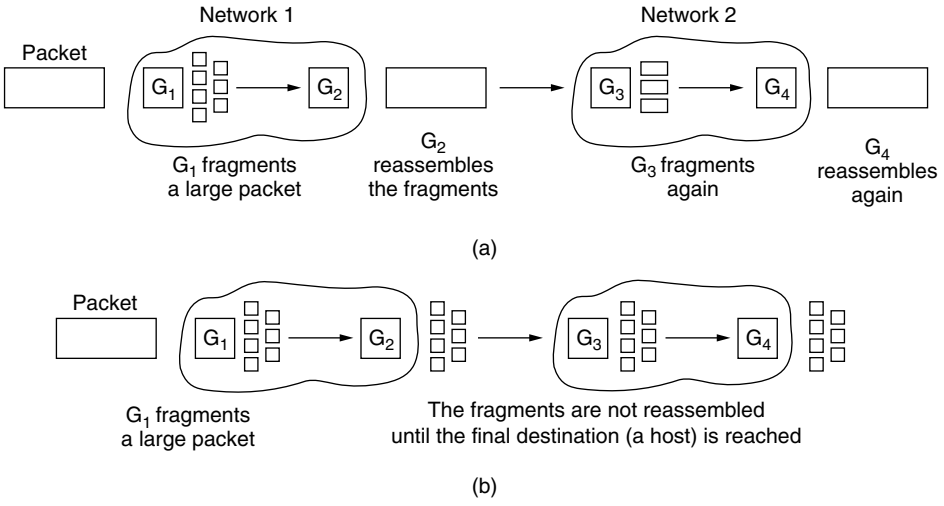


Figure 5-42. (a) Transparent fragmentation. (b) Nontransparent fragmentation.

treated as though it were an original packet. The routers pass the fragments, as shown in Fig. 5-42(b), and reassembly is performed only at the destination host.

The main advantage of nontransparent fragmentation is that it requires routers to do less work. IP works this way. A complete design requires that the fragments be numbered in such a way that the original data stream can be reconstructed. The design used by IP is to give every fragment a packet number (carried on all packets), an absolute byte offset within the packet, and a flag indicating whether it is the end of the packet. An example is shown in Fig. 5-43. While simple, this design has some attractive properties. Fragments can be placed in a buffer at the destination in the right place for reassembly, even if they arrive out of order. Fragments can also be fragmented if they pass over a network with a yet smaller MTU. This is shown in Fig. 5-43(c). Retransmissions of the packet (if all fragments were not received) can be fragmented into different pieces. Finally, fragments can be of arbitrary size, down to a single byte plus the packet header. In all cases, the destination simply uses the packet number and fragment offset to place the data in the right position, and the end-of-packet flag to determine when it has the complete packet.

Unfortunately, this design still has problems. The overhead can be higher than with transparent fragmentation because fragment headers are now carried over some links where they may not be needed. But the real problem is the existence of fragments in the first place. Kent and Mogul (1987) argued that fragmentation is detrimental to performance because, as well as the header overheads, a whole packet is lost if any of its fragments are lost, and because fragmentation is more of a burden for hosts than was originally realized.

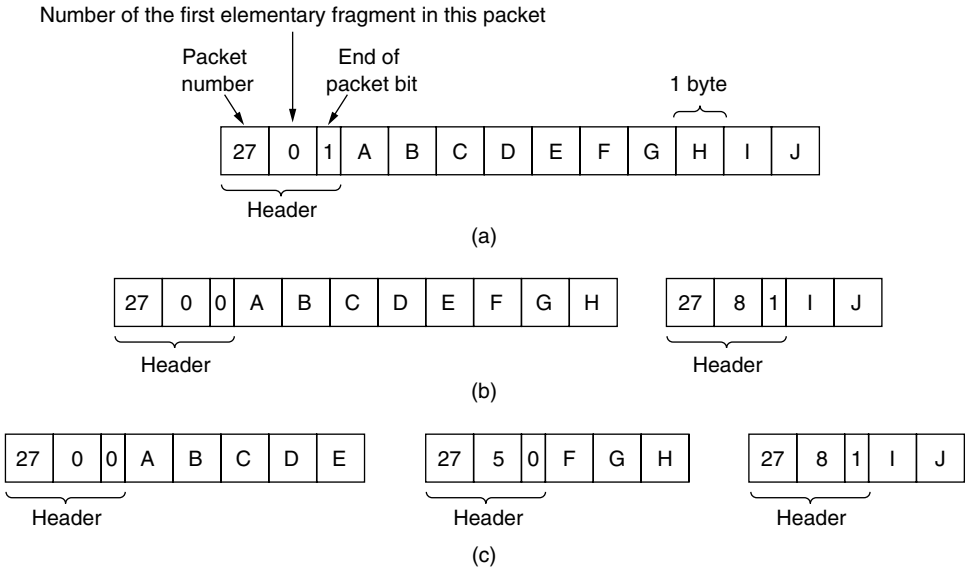


Figure 5-43. Fragmentation when the elementary data size is 1 byte. (a) Original packet, containing 10 data bytes. (b) Fragments after passing through a network with maximum packet size of 8 payload bytes plus header. (c) Fragments after passing through a size 5 gateway.

This leads us back to the original solution of getting rid of fragmentation in the network, the strategy used in the modern Internet. The process is called **path MTU discovery** (Mogul and Deering, 1990). It works as follows. Each IP packet is sent with its header bits set to indicate that no fragmentation is allowed to be performed. If a router receives a packet that is too large, it generates an error packet, returns it to the source, and drops the packet. This is shown in Fig. 5-44. When the source receives the error packet, it uses the information inside to refragment the packet into pieces that are small enough for the router to handle. If a router further down the path has an even smaller MTU, the process is repeated.

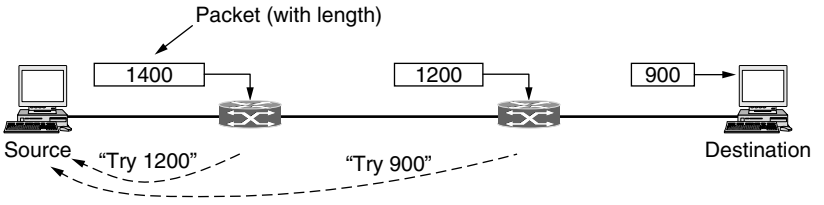


Figure 5-44. Path MTU discovery.

The advantage of path MTU discovery is that the source now knows what length packet to send. If the routes and path MTU change, new error packets will be triggered and the source will adapt to the new path. However, fragmentation is still needed between the source and the destination unless the higher layers learn the path MTU and pass the right amount of data to IP. TCP and IP are typically implemented together (as "TCP/IP") to be able to pass this sort of information. Even if this is not done for other protocols, fragmentation has still been moved out of the network and into the hosts.

The disadvantage of path MTU discovery is that there may be added startup delays simply to send a packet. More than one round-trip delay may be needed to probe the path and find the MTU before any data is delivered to the destination. This begs the question of whether there are better designs. The answer is probably "Yes." Consider the design in which each router simply truncates packets that exceed its MTU. This would ensure that the destination learns the MTU as rapidly as possible (from the amount of data that was delivered) and receives some of the data.

5.6 THE NETWORK LAYER IN THE INTERNET

It is now time to discuss the network layer of the Internet in detail. But before getting into specifics, it is worth taking a look at the principles that drove its design in the past and made it the success that it is today. All too often, nowadays, people seem to have forgotten them. These principles are enumerated and discussed in RFC 1958, which is well worth reading (and should be mandatory for all protocol designers—with a final exam at the end). This RFC draws heavily on ideas put forth by Clark (1988) and Saltzer et al. (1984). We will now summarize what we consider to be the top 10 principles (from most important to least important).

1. **Make sure it works.** Do not finalize the design or standard until multiple prototypes have successfully communicated with each other. All too often, designers first write a 1000-page standard, get it approved, then discover it is deeply flawed and does not work. Then they write version 1.1 of the standard. This is not the way to go.
2. **Keep it simple.** When in doubt, use the simplest solution. William of Occam stated this principle (Occam's razor) in the 14th century. Put in modern terms: fight features. If a feature is not absolutely essential, leave it out, especially if the same effect can be achieved by combining other features.
3. **Make clear choices.** If there are several ways of doing the same thing, choose one. Having two or more ways to do the same thing is looking for trouble. Standards often have multiple options or modes

or parameters because several powerful parties insist that their way is best. Designers should strongly resist this tendency. Just say no.

4. **Exploit modularity.** This principle leads directly to the idea of having protocol stacks, each of whose layers is independent of all the other ones. In this way, if circumstances require one module or layer to be changed, the other ones will not be affected.
5. **Expect heterogeneity.** Different types of hardware, transmission facilities, and applications will occur on any large network. To handle them, the network design must be simple, general, and flexible.
6. **Avoid static options and parameters.** If parameters are unavoidable (e.g., maximum packet size), it is best to have the sender and receiver negotiate a value rather than defining fixed choices.
7. **Look for a good design; it need not be perfect.** Often, the designers have a good design but it cannot handle some weird special case. Rather than messing up the design, the designers should go with the good design and put the burden of working around it on the people with the strange requirements.
8. **Be strict when sending and tolerant when receiving.** In other words, send only packets that rigorously comply with the standards, but expect incoming packets that may not be fully conformant and try to deal with them.
9. **Think about scalability.** If the system is to handle millions of hosts and billions of users effectively, no centralized databases of any kind are tolerable and load must be spread as evenly as possible over the available resources.
10. **Consider performance and cost.** If a network has poor performance or outrageous costs, nobody will use it.

Let us now leave the general principles and start looking at the details of the Internet's network layer. In the network layer, the Internet can be viewed as a collection of networks or **ASes** (**Autonomous Systems**) that are interconnected. There is no real structure, but several major backbones exist. These are constructed from high-bandwidth lines and fast routers. The biggest of these backbones, to which everyone else connects to reach the rest of the Internet, are called **Tier 1 networks**. Attached to the backbones are ISPs (Internet Service Providers) that provide Internet access to homes and businesses, data centers and colocation facilities full of server machines, and regional (mid-level) networks. The data centers serve much of the content that is sent over the Internet. Attached

to the regional networks are more ISPs, LANs at many universities and companies, and other edge networks. A sketch of this quasihierarchical organization is given in Fig. 5-45.

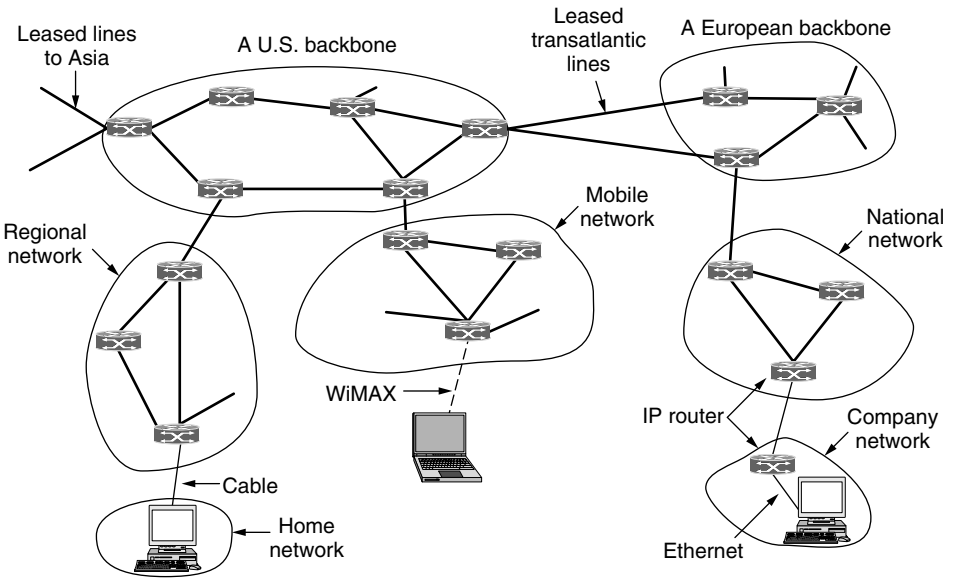


Figure 5-45. The Internet is an interconnected collection of many networks.

The glue that holds the whole Internet together is the network layer protocol, **IP** (**Internet Protocol**). Unlike most older network layer protocols, IP was designed from the beginning with internetworking in mind. A good way to think of the network layer is this: its job is to provide a best-effort (i.e., not guaranteed) way to transport packets from source to destination, without regard to whether these machines are on the same network or whether there are other networks in between them.

Communication in the Internet works as follows. The transport layer takes data streams and breaks them up so that they may be sent as IP packets. In theory, packets can be up to 64 KB each, but in practice they are usually not more than 1500 bytes (so they fit in one Ethernet frame). IP routers forward each packet through the Internet, along a path from one router to the next, until the destination is reached. At the destination, the network layer hands the data to the transport layer, which gives it to the receiving process. When all the pieces finally get to the destination machine, they are reassembled by the network layer into the original datagram. This datagram is then handed to the transport layer.

In the example of Fig. 5-45, a packet originating at a host on the home network has to traverse four networks and a large number of IP routers before even getting to the company network on which the destination host is located. This is

not unusual in practice, and there are many longer paths. There is also much redundant connectivity in the Internet, with backbones and ISPs connecting to each other in multiple locations. This means that there are many possible paths between two hosts. It is the job of the IP routing protocols to decide which paths to use.

5.6.1 The IP Version 4 Protocol

An appropriate place to start our study of the network layer in the Internet is with the format of the IP datagrams themselves. An IPv4 datagram consists of a header part and a body or payload part. The header has a 20-byte fixed part and a variable-length optional part. The header format is shown in Fig. 5-46. The bits are transmitted from left to right and top to bottom, with the high-order bit of the *Version* field going first. (This is a "big-endian" network byte order. On little-endian machines, such as Intel x86 computers, a software conversion is required on both transmission and reception.) In retrospect, little endian would have been a better choice, but at the time IP was designed, no one knew it would come to dominate computing.

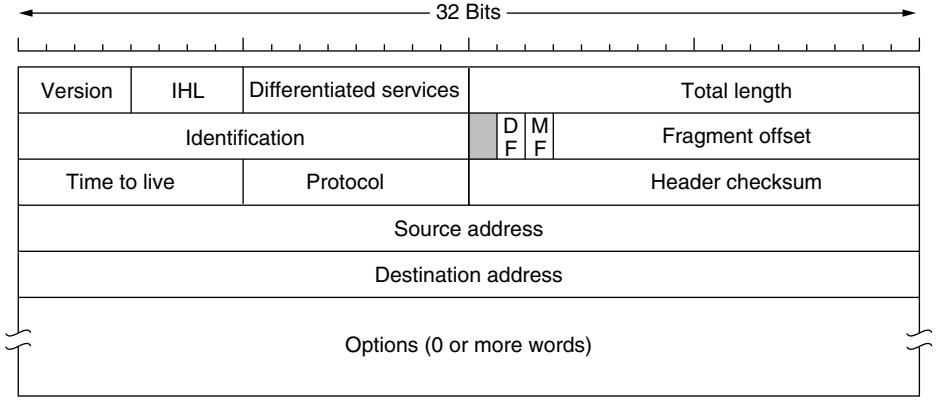


Figure 5-46. The IPv4 (Internet Protocol) header.

The *Version* field keeps track of which version of the protocol the datagram belongs to. Version 4 dominates the Internet today, and that is where we have started our discussion. By including the version at the start of each datagram, it becomes possible to have a transition between versions over a long period of time. In fact, IPv6, the next version of IP, was defined more than a decade ago, yet is only just beginning to be deployed. We will describe it later in this section. Its use will eventually be forced when each of China's almost 2^{31} people has a desktop PC, a laptop, and an IP phone. As an aside on numbering, IPv5 was an experimental real-time stream protocol that was never widely used.

Since the header length is not constant, a field in the header, *IHL*, is provided to tell how long the header is, in 32-bit words. The minimum value is 5, which applies when no options are present. The maximum value of this 4-bit field is 15, which limits the header to 60 bytes, and thus the *Options* field to 40 bytes. For some options, such as one that records the route a packet has taken, 40 bytes is far too small, making those options useless.

The *Differentiated services* field is one of the few fields that has changed its meaning (slightly) over the years. Originally, it was called the *Type of service* field. It was and still is intended to distinguish between different classes of service. Various combinations of reliability and speed are possible. For digitized voice, fast delivery beats accurate delivery. For file transfer, error-free transmission is more important than fast transmission. The *Type of service* field provided 3 bits to signal priority and 3 bits to signal whether a host cared more about delay, throughput, or reliability. However, no one really knew what to do with these bits at routers, so they were left unused for many years. When differentiated services were designed, IETF threw in the towel and reused this field. Now, the top 6 bits are used to mark the packet with its service class; we described the expedited and assured services earlier in this chapter. The bottom 2 bits are used to carry explicit congestion notification information, such as whether the packet has experienced congestion; we described explicit congestion notification as part of congestion control earlier in this chapter.

The *Total length* includes everything in the datagram—both header and data. The maximum length is 65,535 bytes. At present, this upper limit is tolerable, but with future networks, larger datagrams may be needed.

The *Identification* field is needed to allow the destination host to determine which packet a newly arrived fragment belongs to. All the fragments of a packet contain the same *Identification* value.

Next comes an unused bit, which is surprising, as available real estate in the IP header is extremely scarce. As an April Fool's joke, Bellovin (2003) proposed using this bit to detect malicious traffic. This would greatly simplify security, as packets with the "evil" bit set would be known to have been sent by attackers and could just be discarded. Unfortunately, network security is not this simple.

Then come two 1-bit fields related to fragmentation. *DF* stands for Don't Fragment. It is an order to the routers not to fragment the packet. Originally, it was intended to support hosts incapable of putting the pieces back together again. Now it is used as part of the process to discover the path MTU, which is the largest packet that can travel along a path without being fragmented. By marking the datagram with the *DF* bit, the sender knows it will either arrive in one piece, or an error message will be returned to the sender.

MF stands for More Fragments. All fragments except the last one have this bit set. It is needed to know when all fragments of a datagram have arrived.

The *Fragment offset* tells where in the current packet this fragment belongs. All fragments except the last one in a datagram must be a multiple of 8 bytes, the

elementary fragment unit. Since 13 bits are provided, there is a maximum of 8192 fragments per datagram, supporting a maximum packet length up to the limit of the *Total length* field. Working together, the *Identification*, *MF*, and *Fragment offset* fields are used to implement fragmentation as described in Sec. 5.5.5.

The *TtL (Time to live)* field is a counter used to limit packet lifetimes. It was originally supposed to count time in seconds, allowing a maximum lifetime of 255 sec. It must be decremented on each hop and is supposed to be decremented multiple times when a packet is queued for a long time in a router. In practice, it just counts hops. When it hits zero, the packet is discarded and a warning packet is sent back to the source host. This feature prevents packets from wandering around forever, something that otherwise might happen if the routing tables ever become corrupted.

When the network layer has assembled a complete packet, it needs to know what to do with it. The *Protocol* field tells it which transport process to give the packet to. TCP is one possibility, but so are UDP and some others. The numbering of protocols is global across the entire Internet. Protocols and other assigned numbers were formerly listed in RFC 1700, but nowadays they are contained in an online database located at *www.iana.org*.

Since the header carries vital information such as addresses, it rates its own checksum for protection, the *Header checksum*. The algorithm is to add up all the 16-bit halfwords of the header as they arrive, using one's complement arithmetic, and then take the one's complement of the result. For purposes of this algorithm, the *Header checksum* is assumed to be zero upon arrival. Such a checksum is useful for detecting errors while the packet travels through the network. Note that it must be recomputed at each hop because at least one field always changes (the *Time to live* field), but tricks can be used to speed up the computation.

The *Source address* and *Destination address* indicate the IP address of the source and destination network interfaces. We will discuss Internet addresses in the next section.

The *Options* field was designed to provide an escape to allow subsequent versions of the protocol to include information not present in the original design, to permit experimenters to try out new ideas, and to avoid allocating header bits to information that is rarely needed. The options are of variable length. Each begins with a 1-byte code identifying the option. Some options are followed by a 1-byte option length field, and then one or more data bytes. The *Options* field is padded out to a multiple of 4 bytes. Originally, the five options listed in Fig. 5-47 were defined.

The *Security* option tells how secret the information is. In theory, a military router might use this field to specify not to route packets through certain countries the military considers to be "bad guys." In practice, all routers ignore it, so its only practical function is to help spies find the good stuff more easily.

The *Strict source routing* option gives the complete path from source to destination as a sequence of IP addresses. The datagram is required to follow that

Option	Description
Security	Specifies how secret the datagram is
Strict source routing	Gives the complete path to be followed
Loose source routing	Gives a list of routers not to be missed
Record route	Makes each router append its IP address
Timestamp	Makes each router append its address and timestamp

Figure 5-47. Some of the IP options.

exact route. It is most useful for system managers who need to send emergency packets when the routing tables have been corrupted, or for making timing measurements.

The *Loose source routing* option requires the packet to traverse the list of routers specified, in the order specified, but it is allowed to pass through other routers on the way. Normally, this option will provide only a few routers, to force a particular path. For example, to force a packet from London to Sydney to go west instead of east, this option might specify routers in New York, Los Angeles, and Honolulu. This option is most useful when political or economic considerations dictate passing through or avoiding certain countries.

The *Record route* option tells each router along the path to append its IP address to the *Options* field. This allows system managers to track down bugs in the routing algorithms ("Why are packets from Houston to Dallas visiting Tokyo first?"). When the ARPANET was first set up, no packet ever passed through more than nine routers, so 40 bytes of options was plenty. As mentioned above, now it is too small.

Finally, the *Timestamp* option is like the *Record route* option, except that in addition to recording its 32-bit IP address, each router also records a 32-bit timestamp. This option, too, is mostly useful for network measurement.

Today, IP options have fallen out of favor. Many routers ignore them or do not process them efficiently, shunting them to the side as an uncommon case. That is, they are only partly supported and they are rarely used.

5.6.2 IP Addresses

A defining feature of IPv4 is its 32-bit addresses. Every host and router on the Internet has an IP address that can be used in the *Source address* and *Destination address* fields of IP packets. It is important to note that an IP address does not actually refer to a host. It really refers to a network interface, so if a host is on two networks, it must have two IP addresses. However, in practice, most hosts are on one network and thus have one IP address. In contrast, routers have multiple interfaces and thus multiple IP addresses.

Prefixes

IP addresses are hierarchical, unlike Ethernet addresses. Each 32-bit address is comprised of a variable-length network portion in the top bits and a host portion in the bottom bits. The network portion has the same value for all hosts on a single network, such as an Ethernet LAN. This means that a network corresponds to a contiguous block of IP address space. This block is called a **prefix**.

IP addresses are written in **dotted decimal notation**. In this format, each of the 4 bytes is written in decimal, from 0 to 255. For example, the 32-bit hexadecimal address 80D00297 is written as 128.208.2.151. Prefixes are written by giving the lowest IP address in the block and the size of the block. The size is determined by the number of bits in the network portion; the remaining bits in the host portion can vary. This means that the size must be a power of two. By convention, it is written after the prefix IP address as a slash followed by the length in bits of the network portion. In our example, if the prefix contains 2^8 addresses and so leaves 24 bits for the network portion, it is written as 128.208.0.0/24.

Since the prefix length cannot be inferred from the IP address alone, routing protocols must carry the prefixes to routers. Sometimes prefixes are simply described by their length, as in a "/16" which is pronounced "slash 16." The length of the prefix corresponds to a binary mask of 1s in the network portion. When written out this way, it is called a **subnet mask**. It can be ANDed with the IP address to extract only the network portion. For our example, the subnet mask is 255.255.255.0. Fig. 5-48 shows a prefix and a subnet mask.

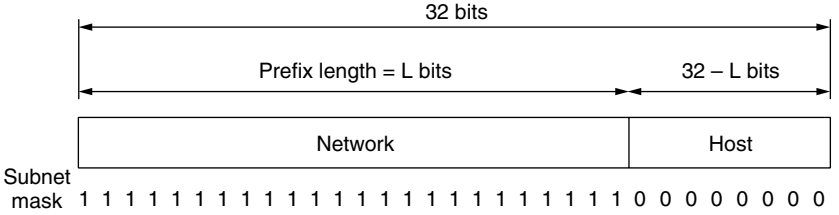


Figure 5-48. An IP prefix and a subnet mask.

Hierarchical addresses have significant advantages and disadvantages. The key advantage of prefixes is that routers can forward packets based on only the network portion of the address, as long as each of the networks has a unique address block. The host portion does not matter to the routers because all hosts on the same network will be sent in the same direction. It is only when the packets reach the network for which they are destined that they are forwarded to the correct host. This makes the routing tables much smaller than they would otherwise be. Consider that the number of hosts on the Internet is approaching one billion. That would be a very large table for every router to keep. However, by using a hierarchy, routers need to keep routes for only around 300,000 prefixes.

While using a hierarchy lets Internet routing scale, it has two disadvantages. First, the IP address of a host depends on where it is located in the network. An Ethernet address can be used anywhere in the world, but every IP address belongs to a specific network, and routers will only be able to deliver packets destined to that address to the network. Designs such as mobile IP are needed to support hosts that move between networks but want to keep the same IP addresses.

The second disadvantage is that the hierarchy is wasteful of addresses unless it is carefully managed. If addresses are assigned to networks in (too) large blocks, there will be (many) addresses that are allocated but not in use. This allocation would not matter much if there were plenty of addresses to go around. However, it was realized more than two decades ago that the tremendous growth of the Internet was rapidly depleting the free address space. IPv6 is the solution to this shortage, but until it is widely deployed there will be great pressure to allocate IP addresses so that they are used very efficiently.

Subnets

Network numbers are managed by a nonprofit corporation called **ICANN** (**Internet Corporation for Assigned Names and Numbers**), to avoid conflicts. In turn, ICANN has delegated parts of the address space to various regional authorities, which dole out IP addresses to ISPs and other companies. This is the process by which a company is allocated a block of IP addresses.

However, this process is only the start of the story, as IP address assignment is ongoing as companies grow. We have said that routing by prefix requires all the hosts in a network to have the same network number. This property can cause problems as networks grow. For example, consider a university that started out with our example /16 prefix for use by the Computer Science Dept. for the computers on its Ethernet. A year later, the Electrical Engineering Dept. wants to get on the Internet. The Art Dept. soon follows suit. What IP addresses should these departments use? Getting further blocks requires going outside the university and may be expensive or inconvenient. Moreover, the /16 already allocated has enough addresses for over 60,000 hosts. It might be intended to allow for significant growth, but until that happens, it is wasteful to allocate further blocks of IP addresses to the same university. A different organization is required.

The solution is to allow the block of addresses to be split into several parts for internal use as multiple networks, while still acting like a single network to the outside world. This is called **subnetting** and the networks (such as Ethernet LANs) that result from dividing up a larger network are called **subnets**. As we mentioned in Chap. 1, you should be aware that this new usage of the term conflicts with older usage of "subnet" to mean the set of all routers and communication lines in a network.

Fig. 5-49 shows how subnets can help with our example. The single /16 has been split into pieces. This split does not need to be even, but each piece must be

aligned so that any bits can be used in the lower host portion. In this case, half of the block (a /17) is allocated to the Computer Science Dept, a quarter is allocated to the Electrical Engineering Dept. (a /18), and one eighth (a /19) to the Art Dept. The remaining eighth is unallocated. A different way to see how the block was divided is to look at the resulting prefixes when written in binary notation:

Computer Science:	10000000	11010000	1lxxxxxxx	xxxxxxxx
Electrical Eng.:	10000000	11010000	00lxxxxxx	xxxxxxxx
Art:	10000000	11010000	011lxxxxx	xxxxxxxx

Here, the vertical bar (|) shows the boundary between the subnet number and the host portion.

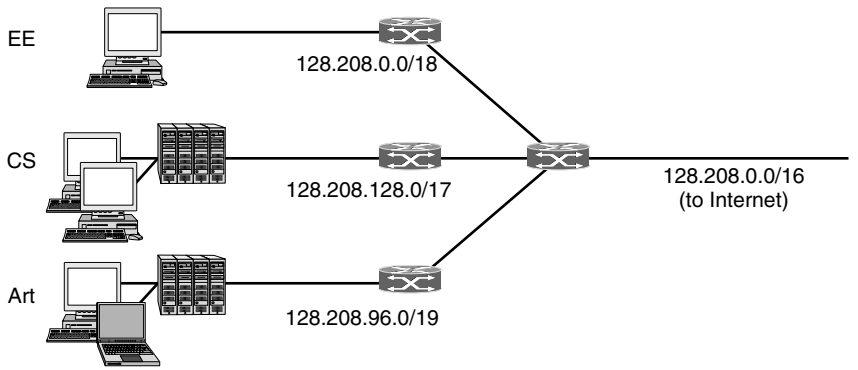


Figure 5-49. Splitting an IP prefix into separate networks with subnetting.

When a packet comes into the main router, how does the router know which subnet to give it to? This is where the details of our prefixes come in. One way would be for each router to have a table with 65,536 entries telling it which outgoing line to use for each host on campus. But this would undermine the main scaling benefit we get from using a hierarchy. Instead, the routers simply need to know the subnet masks for the networks on campus.

When a packet arrives, the router looks at the destination address of the packet and checks which subnet it belongs to. The router can do this by ANDing the destination address with the mask for each subnet and checking to see if the result is the corresponding prefix. For example, consider a packet destined for IP address 128.208.2.151. To see if it is for the Computer Science Dept., we AND with 255.255.128.0 to take the first 17 bits (which is 128.208.0.0) and see if they match the prefix address (which is 128.208.128.0). They do not match. Checking the first 18 bits for the Electrical Engineering Dept., we get 128.208.0.0 when ANDing with the subnet mask. This does match the prefix address, so the packet is forwarded onto the interface which leads to the Electrical Engineering network.

The subnet divisions can be changed later if necessary, by updating all subnet masks at routers inside the university. Outside the network, the subnetting is not visible, so allocating a new subnet does not require contacting ICANN or changing any external databases.

CIDR—Classless InterDomain Routing

Even if blocks of IP addresses are allocated so that the addresses are used efficiently, there is still a problem that remains: routing table explosion.

Routers in organizations at the edge of a network, such as a university, need to have an entry for each of their subnets, telling the router which line to use to get to that network. For routes to destinations outside of the organization, they can use the simple default rule of sending the packets on the line toward the ISP that connects the organization to the rest of the Internet. The other destination addresses must all be out there somewhere.

Routers in ISPs and backbones in the middle of the Internet have no such luxury. They must know which way to go to get to every network and no simple default will work. These core routers are said to be in the **default-free zone** of the Internet. No one really knows how many networks are connected to the Internet any more, but it is a large number, probably at least a million. This can make for a very large table. It may not sound large by computer standards, but realize that routers must perform a lookup in this table to forward every packet, and routers at large ISPs may forward up to millions of packets per second. Specialized hardware and fast memory are needed to process packets at these rates, not a general-purpose computer.

In addition, routing algorithms require each router to exchange information about the addresses it can reach with other routers. The larger the tables, the more information needs to be communicated and processed. The processing grows at least linearly with the table size. Greater communication increases the likelihood that some parts will get lost, at least temporarily, possibly leading to routing instabilities.

The routing table problem could have been solved by going to a deeper hierarchy, like the telephone network. For example, having each IP address contain a country, state/province, city, network, and host field might work. Then, each router would only need to know how to get to each country, the states or provinces in its own country, the cities in its state or province, and the networks in its city. Unfortunately, this solution would require considerably more than 32 bits for IP addresses and would use addresses inefficiently (and Liechtenstein would have as many bits in its addresses as the United States).

Fortunately, there is something we can do to reduce routing table sizes. We can apply the same insight as subnetting: routers at different locations can know about a given IP address as belonging to prefixes of different sizes. However, instead of splitting an address block into subnets, here we combine multiple small

prefixes into a single larger prefix. This process is called **route aggregation**. The resulting larger prefix is sometimes called a **supernet**, to contrast with subnets as the division of blocks of addresses.

With aggregation, IP addresses are contained in prefixes of varying sizes. The same IP address that one router treats as part of a /22 (a block containing 2^{10} addresses) may be treated by another router as part of a larger /20 (which contains 2^{12} addresses). It is up to each router to have the corresponding prefix information. This design works with subnetting and is called **CIDR** (**Classless Inter-Domain Routing**), which is pronounced "cider," as in the drink. The most recent version of it is specified in RFC 4632 (Fuller and Li, 2006). The name highlights the contrast with addresses that encode hierarchy with classes, which we will describe shortly.

To make CIDR easier to understand, let us consider an example in which a block of 8192 IP addresses is available starting at 194.24.0.0. Suppose that Cambridge University needs 2048 addresses and is assigned the addresses 194.24.0.0 through 194.24.7.255, along with mask 255.255.248.0. This is a /21 prefix. Next, Oxford University asks for 4096 addresses. Since a block of 4096 addresses must lie on a 4096-byte boundary, Oxford cannot be given addresses starting at 194.24.8.0. Instead, it gets 194.24.16.0 through 194.24.31.255, along with subnet mask 255.255.240.0. Finally, the University of Edinburgh asks for 1024 addresses and is assigned addresses 194.24.8.0 through 194.24.11.255 and mask 255.255.252.0. These assignments are summarized in Fig. 5-50.

University	First address	Last address	How many	Prefix
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Available)	194.24.12.0	194.24.15.255	1024	194.24.12.0/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

Figure 5-50. A set of IP address assignments.

All of the routers in the default-free zone are now told about the IP addresses in the three networks. Routers close to the universities may need to send on a different outgoing line for each of the prefixes, so they need an entry for each of the prefixes in their routing tables. An example is the router in London in Fig. 5-51.

Now let us look at these three universities from the point of view of a distant router in New York. All of the IP addresses in the three prefixes should be sent from New York (or the U.S. in general) to London. The routing process in London notices this and combines the three prefixes into a single aggregate entry for the prefix 194.24.0.0/19 that it passes to the New York router. This prefix contains 8K addresses and covers the three universities and the otherwise unallocated 1024 addresses. By using aggregation, three prefixes have been reduced to one, reducing

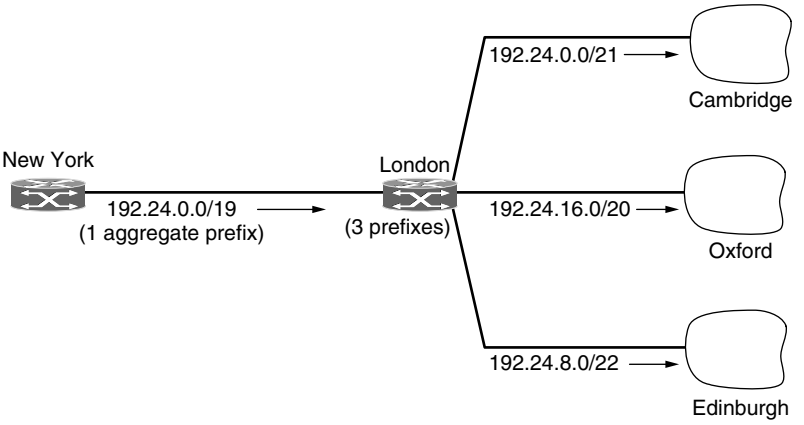


Figure 5-51. Aggregation of IP prefixes.

the prefixes that the New York router must be told about and the routing table entries in the New York router.

When aggregation is turned on, it is an automatic process. It depends on which prefixes are located where in the Internet not on the actions of an administrator assigning addresses to networks. Aggregation is heavily used throughout the Internet and can reduce the size of router tables to around 200,000 prefixes.

As a further twist, prefixes are allowed to overlap. The rule is that packets are sent in the direction of the most specific route, or the **longest matching prefix** that has the fewest IP addresses. Longest matching prefix routing provides a useful degree of flexibility, as seen in the behavior of the router at New York in Fig. 5-52. This router still uses a single aggregate prefix to send traffic for the three universities to London. However, the previously available block of addresses within this prefix has now been allocated to a network in San Francisco. One possibility is for the New York router to keep four prefixes, sending packets for three of them to London and packets for the fourth to San Francisco. Instead, longest matching prefix routing can handle this forwarding with the two prefixes that are shown. One overall prefix is used to direct traffic for the entire block to London. One more specific prefix is also used to direct a portion of the larger prefix to San Francisco. With the longest matching prefix rule, IP addresses within the San Francisco network will be sent on the outgoing line to San Francisco, and all other IP addresses in the larger prefix will be sent to London.

Conceptually, CIDR works as follows. When a packet comes in, the routing table is scanned to determine if the destination lies within the prefix. It is possible that multiple entries with different prefix lengths will match, in which case the entry with the longest prefix is used. Thus, if there is a match for a /20 mask and a /24 mask, the /24 entry is used to look up the outgoing line for the packet. However, this process would be tedious if the table were really scanned entry by entry.

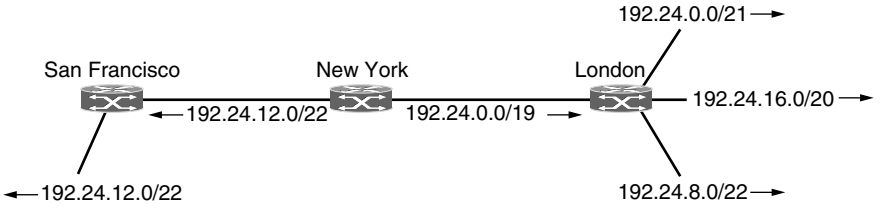


Figure 5-52. Longest matching prefix routing at the New York router.

Instead, complex algorithms have been devised to speed up the address matching process (Ruiz-Sanchez et al., 2001). Commercial routers use custom VLSI chips with these algorithms embedded in hardware.

Classful and Special Addressing

To help you better appreciate why CIDR is so useful, we will briefly relate the design that predated it. Before 1993, IP addresses were divided into the five categories listed in Fig. 5-53. This allocation has come to be called **classful addressing**.

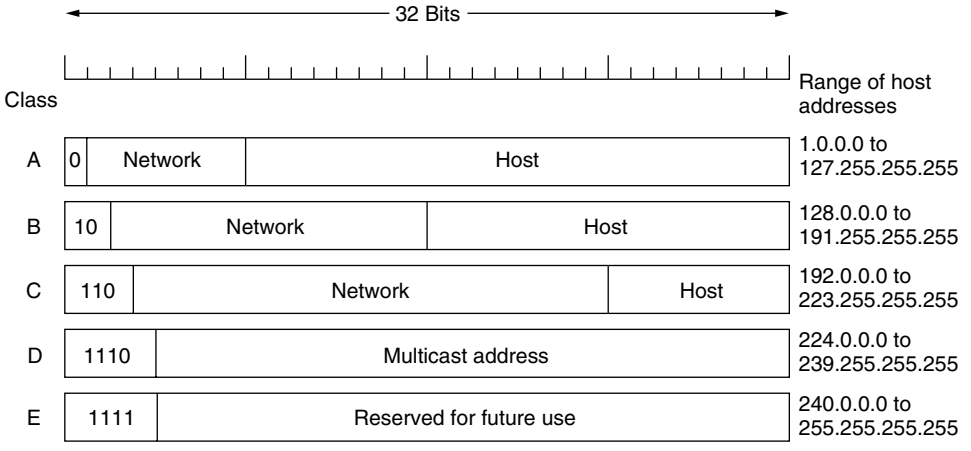


Figure 5-53. IP address formats.

The class A, B, and C formats allow for up to 128 networks with 16 million hosts each, 16,384 networks with up to 65,536 hosts each, and 2 million networks (e.g., LANs) with up to 256 hosts each (although a few of these are special). Also supported is multicast (the class D format), in which a datagram is directed to multiple hosts. Addresses beginning with 1111 are reserved for use in the future. They would be valuable to use now given the depletion of the IPv4 address space.

Unfortunately, many hosts will not accept these addresses as valid because they have been off-limits for so long and it is hard to teach old hosts new tricks.

This is a hierarchical design, but unlike CIDR the sizes of the address blocks are fixed. Over 2 billion addresses exist, but organizing the address space by classes wastes millions of them. In particular, the real villain is the class B network. For most organizations, a class A network, with 16 million addresses, is too big, and a class C network, with 256 addresses is too small. A class B network, with 65,536, is just right. In Internet folklore, this situation is known as the **three bears problem** [as in *Goldilocks and the Three Bears* (Southey, 1848)].

In reality, though, a class B address is far too large for most organizations. Studies have shown that more than half of all class B networks have fewer than 50 hosts. A class C network would have done the job, but no doubt every organization that asked for a class B address thought that one day it would outgrow the 8-bit host field. In retrospect, it might have been better to have had class C networks use 10 bits instead of 8 for the host number, allowing 1022 hosts per network. Had this been the case, most organizations would probably have settled for a class C network, and there would have been half a million of them (versus only 16,384 class B networks).

It is hard to fault the Internet's designers for not having provided more (and smaller) class B addresses. At the time the decision was made to create the three classes, the Internet was a research network connecting the major research universities in the U.S. (plus a very small number of companies and military sites doing networking research). No one then perceived the Internet becoming a mass-market communication system rivaling the telephone network. At the time, someone no doubt said: "The U.S. has about 2000 colleges and universities. Even if all of them connect to the Internet and many universities in other countries join, too, we are never going to hit 16,000, since there are not that many universities in the whole world. Furthermore, having the host number be an integral number of bytes speeds up packet processing'' (which was then done entirely in software). Perhaps some day people will look back and fault the folks who designed the telephone number scheme and say: "What idiots. Why didn't they include the planet number in the phone number?'' But at the time, it did not seem necessary.

To handle these problems, subnets were introduced to flexibly assign blocks of addresses within an organization. Later, CIDR was added to reduce the size of the global routing table. Today, the bits that indicate whether an IP address belongs to class A, B, or C network are no longer used, though references to these classes in the literature are still common.

To see how dropping the classes made forwarding more complicated, consider how simple it was in the old classful system. When a packet arrived at a router, a copy of the IP address was shifted right 28 bits to yield a 4-bit class number. A 16-way branch then sorted packets into A, B, C (and D and E) classes, with eight of the cases for class A, four of the cases for class B, and two of the cases for class C. The code for each class then masked off the 8-, 16-, or 24-bit network

number and right aligned it in a 32-bit word. The network number was then looked up in the A, B, or C table, usually by indexing for A and B networks and hashing for C networks. Once the entry was found, the outgoing line could be looked up and the packet forwarded. This is much simpler than the longest matching prefix operation, which can no longer use a simple table lookup because an IP address may have any length prefix.

Class D addresses continue to be used in the Internet for multicast. Actually, it might be more accurate to say that they are starting to be used for multicast, since Internet multicast has not been widely deployed in the past.

There are also several other addresses that have special meanings, as shown in Fig. 5-54. The IP address 0.0.0.0, the lowest address, is used by hosts when they are being booted. It means "this network" or "this host." IP addresses with 0 as the network number refer to the current network. These addresses allow machines to refer to their own network without knowing its number (but they have to know the network mask to know how many 0s to include). The address consisting of all 1s, or 255.255.255.255—the highest address—is used to mean all hosts on the indicated network. It allows broadcasting on the local network, typically a LAN. The addresses with a proper network number and all 1s in the host field allow machines to send broadcast packets to distant LANs anywhere in the Internet. However, many network administrators disable this feature as it is mostly a security hazard. Finally, all addresses of the form 127.*xx.yy.zz* are reserved for loopback testing. Packets sent to that address are not put out onto the wire; they are processed locally and treated as incoming packets. This allows packets to be sent to the host without the sender knowing its number, which is useful for testing.

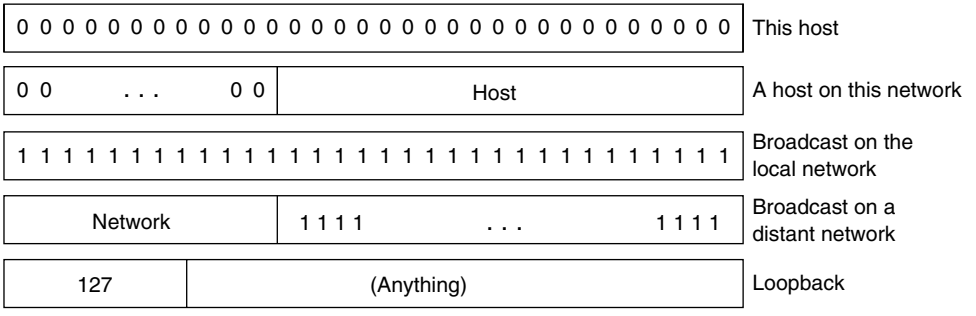


Figure 5-54. Special IP addresses.

NAT—Network Address Translation

IP addresses are scarce. An ISP might have a /16 address, giving it 65,534 usable host numbers. If it has more customers than that, it has a problem.

This scarcity has led to techniques to use IP addresses sparingly. One approach is to dynamically assign an IP address to a computer when it is on and using the network, and to take the IP address back when the host becomes inactive. The IP address can then be assigned to another computer that becomes active. In this way, a single /16 address can handle up to 65,534 active users.

This strategy works well in some cases, for example, for dialup networking and mobile and other computers that may be temporarily absent or powered off. However, it does not work very well for business customers. Many PCs in businesses are expected to be on continuously. Some are employee machines, backed up at night, and some are servers that may have to serve a remote request at a moment's notice. These businesses have an access line that always provides connectivity to the rest of the Internet.

Increasingly, this situation also applies to home users subscribing to ADSL or Internet over cable, since there is no connection charge (just a monthly flat rate charge). Many of these users have two or more computers at home, often one for each family member, and they all want to be online all the time. The solution is to connect all the computers into a home network via a LAN and put a (wireless) router on it. The router then connects to the ISP. From the ISP's point of view, the family is now the same as a small business with a handful of computers. Welcome to Jones, Inc. With the techniques we have seen so far, each computer must have its own IP address all day long. For an ISP with many thousands of customers, particularly business customers and families that are just like small businesses, the demand for IP addresses can quickly exceed the block that is available.

The problem of running out of IP addresses is not a theoretical one that might occur at some point in the distant future. It is happening right here and right now. The long-term solution is for the whole Internet to migrate to IPv6, which has 128-bit addresses. This transition is slowly occurring, but it will be years before the process is complete. To get by in the meantime, a quick fix was needed. The quick fix that is widely used today came in the form of **NAT** (**Network Address Translation**), which is described in RFC 3022 and which we will summarize below. For additional information, see Dutcher (2001).

The basic idea behind NAT is for the ISP to assign each home or business a single IP address (or at most, a small number of them) for Internet traffic. *Within* the customer network, every computer gets a unique IP address, which is used for routing intramural traffic. However, just before a packet exits the customer network and goes to the ISP, an address translation from the unique internal IP address to the shared public IP address takes place. This translation makes use of three ranges of IP addresses that have been declared as private. Networks may use them internally as they wish. The only rule is that no packets containing these addresses may appear on the Internet itself. The three reserved ranges are:

10.0.0.0	– 10.255.255.255/8	(16,777,216 hosts)
172.16.0.0	– 172.31.255.255/12	(1,048,576 hosts)
192.168.0.0	– 192.168.255.255/16	(65,536 hosts)

The first range provides for 16,777,216 addresses (except for all 0s and all 1s, as usual) and is the usual choice, even if the network is not large.

The operation of NAT is shown in Fig. 5-55. Within the customer premises, every machine has a unique address of the form 10.*x.y.z*. However, before a packet leaves the customer premises, it passes through a **NAT box** that converts the internal IP source address, 10.0.0.1 in the figure, to the customer's true IP address, 198.60.42.12 in this example. The NAT box is often combined in a single device with a firewall, which provides security by carefully controlling what goes into the customer network and what comes out of it. We will study firewalls in Chap. 8. It is also possible to integrate the NAT box into a router or ADSL modem.

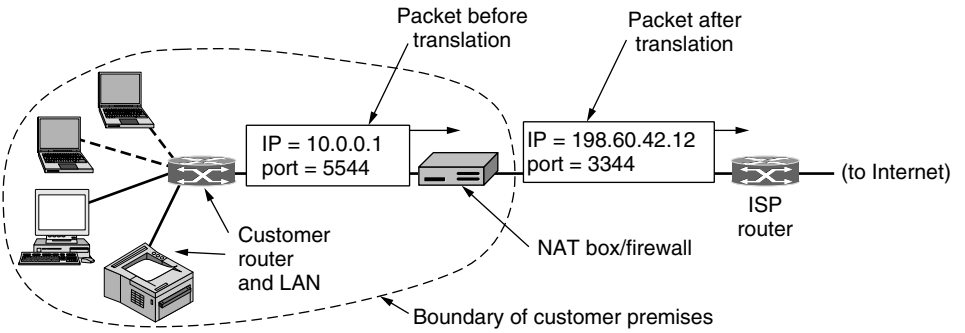


Figure 5-55. Placement and operation of a NAT box.

So far, we have glossed over one tiny but crucial detail: when the reply comes back (e.g., from a Web server), it is naturally addressed to 198.60.42.12, so how does the NAT box know which internal address to replace it with? Herein lies the problem with NAT. If there were a spare field in the IP header, that field could be used to keep track of who the real sender was, but only 1 bit is still unused. In principle, a new option could be created to hold the true source address, but doing so would require changing the IP code on all the machines on the entire Internet to handle the new option. This is not a promising alternative for a quick fix.

What actually happens is as follows. The NAT designers observed that most IP packets carry either TCP or UDP payloads. When we study TCP and UDP in Chap. 6, we will see that both of these have headers containing a source port and a destination port. Below we will just discuss TCP ports, but exactly the same story holds for UDP ports. The ports are 16-bit integers that indicate where the TCP connection begins and ends. These ports provide the field needed to make NAT work.

When a process wants to establish a TCP connection with a remote process, it attaches itself to an unused TCP port on its own machine. This is called the **source port** and tells the TCP code where to send incoming packets belonging to this connection. The process also supplies a **destination port** to tell who to give

the packets to on the remote side. Ports 0–1023 are reserved for well-known services. For example, port 80 is the port used by Web servers, so remote clients can locate them. Each outgoing TCP message contains both a source port and a destination port. Together, these ports serve to identify the processes using the connection on both ends.

An analogy may make the use of ports clearer. Imagine a company with a single main telephone number. When people call the main number, they reach an operator who asks which extension they want and then puts them through to that extension. The main number is analogous to the customer's IP address and the extensions on both ends are analogous to the ports. Ports are effectively an extra 16 bits of addressing that identify which process gets which incoming packet.

Using the *Source port* field, we can solve our mapping problem. Whenever an outgoing packet enters the NAT box, the 10.*x.y.z* source address is replaced by the customer's true IP address. In addition, the TCP *Source port* field is replaced by an index into the NAT box's 65,536-entry translation table. This table entry contains the original IP address and the original source port. Finally, both the IP and TCP header checksums are recomputed and inserted into the packet. It is necessary to replace the *Source port* because connections from machines 10.0.0.1 and 10.0.0.2 may both happen to use port 5000, for example, so the *Source port* alone is not enough to identify the sending process.

When a packet arrives at the NAT box from the ISP, the *Source port* in the TCP header is extracted and used as an index into the NAT box's mapping table. From the entry located, the internal IP address and original TCP *Source port* are extracted and inserted into the packet. Then, both the IP and TCP checksums are recomputed and inserted into the packet. The packet is then passed to the customer router for normal delivery using the 10.*x.y.z* address.

Although this scheme sort of solves the problem, networking purists in the IP community have a tendency to regard it as an abomination-on-the-face-of-the-earth. Briefly summarized, here are some of the objections. First, NAT violates the architectural model of IP, which states that every IP address uniquely identifies a single machine worldwide. The whole software structure of the Internet is built on this fact. With NAT, thousands of machines may (and do) use address 10.0.0.1.

Second, NAT breaks the end-to-end connectivity model of the Internet, which says that any host can send a packet to any other host at any time. Since the mapping in the NAT box is set up by outgoing packets, incoming packets cannot be accepted until after outgoing ones. In practice, this means that a home user with NAT can make TCP/IP connections to a remote Web server, but a remote user cannot make connections to a game server on the home network. Special configuration or **NAT traversal** techniques are needed to support this kind of situation.

Third, NAT changes the Internet from a connectionless network to a peculiar kind of connection-oriented network. The problem is that the NAT box must maintain information (i.e., the mapping) for each connection passing through it.

Having the network maintain connection state is a property of connection-oriented networks, not connectionless ones. If the NAT box crashes and its mapping table is lost, all its TCP connections are destroyed. In the absence of NAT, a router can crash and restart with no long-term effect on TCP connections. The sending process just times out within a few seconds and retransmits all unacknowledged packets. With NAT, the Internet becomes as vulnerable as a circuit-switched network.

Fourth, NAT violates the most fundamental rule of protocol layering: layer k may not make any assumptions about what layer $k + 1$ has put into the payload field. This basic principle is there to keep the layers independent. If TCP is later upgraded to TCP-2, with a different header layout (e.g., 32-bit ports), NAT will fail. The whole idea of layered protocols is to ensure that changes in one layer do not require changes in other layers. NAT destroys this independence.

Fifth, processes on the Internet are not required to use TCP or UDP. If a user on machine *A* decides to use some new transport protocol to talk to a user on machine *B* (for example, for a multimedia application), introduction of a NAT box will cause the application to fail because the NAT box will not be able to locate the TCP *Source port* correctly.

A sixth and related problem is that some applications use multiple TCP/IP connections or UDP ports in prescribed ways. For example, **FTP**, the standard **File Transfer Protocol**, inserts IP addresses in the body of packet for the receiver to extract and use. Since NAT knows nothing about these arrangements, it cannot rewrite the IP addresses or otherwise account for them. This lack of understanding means that FTP and other applications such as the H.323 Internet telephony protocol (which we will study in Chap. 7) will fail in the presence of NAT unless special precautions are taken. It is often possible to patch NAT for these cases, but having to patch the code in the NAT box every time a new application comes along is not a good idea.

Finally, since the TCP *Source port* field is 16 bits, at most 65,536 machines can be mapped onto an IP address. Actually, the number is slightly less because the first 4096 ports are reserved for special uses. However, if multiple IP addresses are available, each one can handle up to 61,440 machines.

A view of these and other problems with NAT is given in RFC 2993. Despite the issues, NAT is widely used in practice, especially for home and small business networks, as the only expedient technique to deal with the IP address shortage. It has become wrapped up with firewalls and privacy because it blocks unsolicited incoming packets by default. For this reason, it is unlikely to go away even when IPv6 is widely deployed.

5.6.3 IP Version 6

IP has been in heavy use for decades. It has worked extremely well, as demonstrated by the exponential growth of the Internet. Unfortunately, IP has become a victim of its own popularity: it is close to running out of addresses. Even

with CIDR and NAT using addresses more sparingly, the last IPv4 addresses are expected to be assigned by ICANN before the end of 2012. This looming disaster was recognized almost two decades ago, and it sparked a great deal of discussion and controversy within the Internet community about what to do about it.

In this section, we will describe both the problem and several proposed solutions. The only long-term solution is to move to larger addresses. **IPv6** (**IP version 6**) is a replacement design that does just that. It uses 128-bit addresses; a shortage of these addresses is not likely any time in the foreseeable future. However, IPv6 has proved very difficult to deploy. It is a different network layer protocol that does not really interwork with IPv4, despite many similarities. Also, companies and users are not really sure why they should want IPv6 in any case. The result is that IPv6 is deployed and used on only a tiny fraction of the Internet (estimates are 1%) despite having been an Internet Standard since 1998. The next several years will be an interesting time, as the few remaining IPv4 addresses are allocated. Will people start to auction off their IPv4 addresses on eBay? Will a black market in them spring up? Who knows.

In addition to the address problems, other issues loom in the background. In its early years, the Internet was largely used by universities, high-tech industries, and the U.S. Government (especially the Dept. of Defense). With the explosion of interest in the Internet starting in the mid-1990s, it began to be used by a different group of people, often with different requirements. For one thing, numerous people with smart phones use it to keep in contact with their home bases. For another, with the impending convergence of the computer, communication, and entertainment industries, it may not be that long before every telephone and television set in the world is an Internet node, resulting in a billion machines being used for audio and video on demand. Under these circumstances, it became apparent that IP had to evolve and become more flexible.

Seeing these problems on the horizon, in 1990 IETF started work on a new version of IP, one that would never run out of addresses, would solve a variety of other problems, and be more flexible and efficient as well. Its major goals were:

1. Support billions of hosts, even with inefficient address allocation.
2. Reduce the size of the routing tables.
3. Simplify the protocol, to allow routers to process packets faster.
4. Provide better security (authentication and privacy).
5. Pay more attention to the type of service, particularly for real-time data.
6. Aid multicasting by allowing scopes to be specified.
7. Make it possible for a host to roam without changing its address.
8. Allow the protocol to evolve in the future.
9. Permit the old and new protocols to coexist for years.

The design of IPv6 presented a major opportunity to improve all of the features in IPv4 that fall short of what is now wanted. To develop a protocol that met all these requirements, IETF issued a call for proposals and discussion in RFC 1550. Twenty-one responses were initially received. By December 1992, seven serious proposals were on the table. They ranged from making minor patches to IP, to throwing it out altogether and replacing it with a completely different protocol.

One proposal was to run TCP over CLNP, the network layer protocol designed for OSI. With its 160-bit addresses, CLNP would have provided enough address space forever as it could give every molecule of water in the oceans enough addresses (roughly 2^5) to set up a small network. This choice would also have unified two major network layer protocols. However, many people felt that this would have been an admission that something in the OSI world was actually done right, a statement considered Politically Incorrect in Internet circles. CLNP was patterned closely on IP, so the two are not really that different. In fact, the protocol ultimately chosen differs from IP far more than CLNP does. Another strike against CLNP was its poor support for service types, something required to transmit multimedia efficiently.

Three of the better proposals were published in *IEEE Network* (Deering, 1993; Francis, 1993; and Katz and Ford, 1993). After much discussion, revision, and jockeying for position, a modified combined version of the Deering and Francis proposals, by now called **SIPP** (**Simple Internet Protocol Plus**) was selected and given the designation **IPv6.**

IPv6 meets IETF's goals fairly well. It maintains the good features of IP, discards or deemphasizes the bad ones, and adds new ones where needed. In general, IPv6 is not compatible with IPv4, but it is compatible with the other auxiliary Internet protocols, including TCP, UDP, ICMP, IGMP, OSPF, BGP, and DNS, with small modifications being required to deal with longer addresses. The main features of IPv6 are discussed below. More information about it can be found in RFCs 2460 through 2466.

First and foremost, IPv6 has longer addresses than IPv4. They are 128 bits long, which solves the problem that IPv6 set out to solve: providing an effectively unlimited supply of Internet addresses. We will have more to say about addresses shortly.

The second major improvement of IPv6 is the simplification of the header. It contains only seven fields (versus 13 in IPv4). This change allows routers to process packets faster and thus improves throughput and delay. We will discuss the header shortly, too.

The third major improvement is better support for options. This change was essential with the new header because fields that previously were required are now optional (because they are not used so often). In addition, the way options are represented is different, making it simple for routers to skip over options not intended for them. This feature speeds up packet processing time.

A fourth area in which IPv6 represents a big advance is in security. IETF had its fill of newspaper stories about precocious 12-year-olds using their personal computers to break into banks and military bases all over the Internet. There was a strong feeling that something had to be done to improve security. Authentication and privacy are key features of the new IP. These were later retrofitted to IPv4, however, so in the area of security the differences are not so great any more.

Finally, more attention has been paid to quality of service. Various half-hearted efforts to improve QoS have been made in the past, but now, with the growth of multimedia on the Internet, the sense of urgency is greater.

The Main IPv6 Header

The IPv6 header is shown in Fig. 5-56. The *Version* field is always 6 for IPv6 (and 4 for IPv4). During the transition period from IPv4, which has already taken more than a decade, routers will be able to examine this field to tell what kind of packet they have. As an aside, making this test wastes a few instructions in the critical path, given that the data link header usually indicates the network protocol for demultiplexing, so some routers may skip the check. For example, the Ethernet *Type* field has different values to indicate an IPv4 or an IPv6 payload. The discussions between the "Do it right" and "Make it fast" camps will no doubt be lengthy and vigorous.

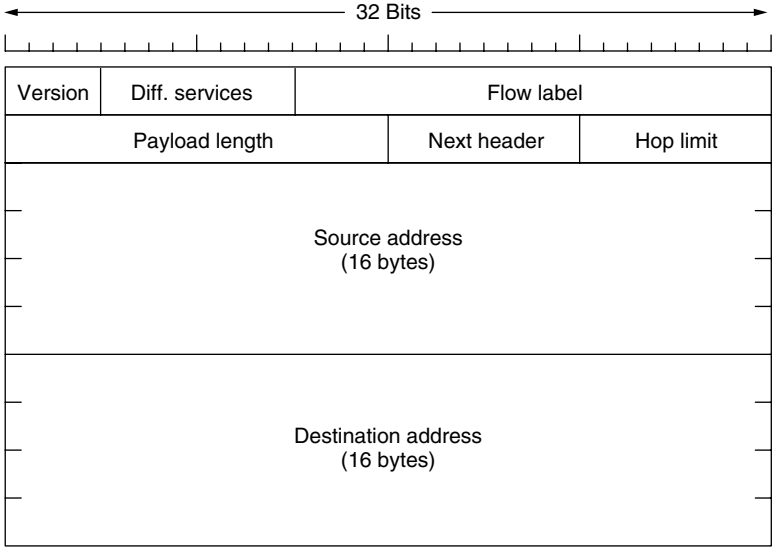


Figure 5-56. The IPv6 fixed header (required).

The *Differentiated services* field (originally called *Traffic class*) is used to distinguish the class of service for packets with different real-time delivery

requirements. It is used with the differentiated service architecture for quality of service in the same manner as the field of the same name in the IPv4 packet. Also, the low-order 2 bits are used to signal explicit congestion indications, again in the same way as with IPv4.

The *Flow label* field provides a way for a source and destination to mark groups of packets that have the same requirements and should be treated in the same way by the network, forming a pseudoconnection. For example, a stream of packets from one process on a certain source host to a process on a specific destination host might have stringent delay requirements and thus need reserved bandwidth. The flow can be set up in advance and given an identifier. When a packet with a nonzero *Flow label* shows up, all the routers can look it up in internal tables to see what kind of special treatment it requires. In effect, flows are an attempt to have it both ways: the flexibility of a datagram network and the guarantees of a virtual-circuit network.

Each flow for quality of service purposes is designated by the source address, destination address, and flow number. This design means that up to 2^{20} flows may be active at the same time between a given pair of IP addresses. It also means that even if two flows coming from different hosts but with the same flow label pass through the same router, the router will be able to tell them apart using the source and destination addresses. It is expected that flow labels will be chosen randomly, rather than assigned sequentially starting at 1, so routers are expected to hash them.

The *Payload length* field tells how many bytes follow the 40-byte header of Fig. 5-56. The name was changed from the IPv4 *Total length* field because the meaning was changed slightly: the 40 header bytes are no longer counted as part of the length (as they used to be). This change means the payload can now be 65,535 bytes instead of a mere 65,515 bytes.

The *Next header* field lets the cat out of the bag. The reason the header could be simplified is that there can be additional (optional) extension headers. This field tells which of the (currently) six extension headers, if any, follow this one. If this header is the last IP header, the *Next header* field tells which transport protocol handler (e.g., TCP, UDP) to pass the packet to.

The *Hop limit* field is used to keep packets from living forever. It is, in practice, the same as the *Time to live* field in IPv4, namely, a field that is decremented on each hop. In theory, in IPv4 it was a time in seconds, but no router used it that way, so the name was changed to reflect the way it is actually used.

Next come the *Source address* and *Destination address* fields. Deering's original proposal, SIP, used 8-byte addresses, but during the review process many people felt that with 8-byte addresses IPv6 would run out of addresses within a few decades, whereas with 16-byte addresses it would never run out. Other people argued that 16 bytes was overkill, whereas still others favored using 20-byte addresses to be compatible with the OSI datagram protocol. Still another faction wanted variable-sized addresses. After much debate and more than a few words

unprintable in an academic textbook, it was decided that fixed-length 16-byte addresses were the best compromise.

A new notation has been devised for writing 16-byte addresses. They are written as eight groups of four hexadecimal digits with colons between the groups, like this:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Since many addresses will have many zeros inside them, three optimizations have been authorized. First, leading zeros within a group can be omitted, so 0123 can be written as 123. Second, one or more groups of 16 zero bits can be replaced by a pair of colons. Thus, the above address now becomes

8000::123:4567:89AB:CDEF

Finally, IPv4 addresses can be written as a pair of colons and an old dotted decimal number, for example:

::192.31.20.46

Perhaps it is unnecessary to be so explicit about it, but there are a lot of 16-byte addresses. Specifically, there are 2^{128} of them, which is approximately 3×10^{38}. If the entire earth, land and water, were covered with computers, IPv6 would allow 7×10^{23} IP addresses per square meter. Students of chemistry will notice that this number is larger than Avogadro's number. While it was not the intention to give every molecule on the surface of the earth its own IP address, we are not that far off.

In practice, the address space will not be used efficiently, just as the telephone number address space is not (the area code for Manhattan, 212, is nearly full, but that for Wyoming, 307, is nearly empty). In RFC 3194, Durand and Huitema calculated that, using the allocation of telephone numbers as a guide, even in the most pessimistic scenario there will still be well over 1000 IP addresses per square meter of the entire earth's surface (land and water). In any likely scenario, there will be trillions of them per square meter. In short, it seems unlikely that we will run out in the foreseeable future.

It is instructive to compare the IPv4 header (Fig. 5-46) with the IPv6 header (Fig. 5-56) to see what has been left out in IPv6. The *IHL* field is gone because the IPv6 header has a fixed length. The *Protocol* field was taken out because the *Next header* field tells what follows the last IP header (e.g., a UDP or TCP segment).

All the fields relating to fragmentation were removed because IPv6 takes a different approach to fragmentation. To start with, all IPv6-conformant hosts are expected to dynamically determine the packet size to use. They do this using the path MTU discovery procedure we described in Sec. 5.5.5. In brief, when a host sends an IPv6 packet that is too large, instead of fragmenting it, the router that is unable to forward it drops the packet and sends an error message back to the

sending host. This message tells the host to break up all future packets to that destination. Having the host send packets that are the right size in the first place is ultimately much more efficient than having the routers fragment them on the fly. Also, the minimum-size packet that routers must be able to forward has been raised from 576 to 1280 bytes to allow 1024 bytes of data and many headers.

Finally, the *Checksum* field is gone because calculating it greatly reduces performance. With the reliable networks now used, combined with the fact that the data link layer and transport layers normally have their own checksums, the value of yet another checksum was deemed not worth the performance price it extracted. Removing all these features has resulted in a lean and mean network layer protocol. Thus, the goal of IPv6—a fast, yet flexible, protocol with plenty of address space—is met by this design.

Extension Headers

Some of the missing IPv4 fields are occasionally still needed, so IPv6 introduces the concept of (optional) **extension headers**. These headers can be supplied to provide extra information, but encoded in an efficient way. Six kinds of extension headers are defined at present, as listed in Fig. 5-57. Each one is optional, but if more than one is present they must appear directly after the fixed header, and preferably in the order listed.

Extension header	Description
Hop-by-hop options	Miscellaneous information for routers
Destination options	Additional information for the destination
Routing	Loose list of routers to visit
Fragmentation	Management of datagram fragments
Authentication	Verification of the sender's identity
Encrypted security payload	Information about the encrypted contents

Figure 5-57. IPv6 extension headers.

Some of the headers have a fixed format; others contain a variable number of variable-length options. For these, each item is encoded as a (*Type, Length, Value*) tuple. The *Type* is a 1-byte field telling which option this is. The *Type* values have been chosen so that the first 2 bits tell routers that do not know how to process the option what to do. The choices are: skip the option; discard the packet; discard the packet and send back an ICMP packet; and discard the packet but do not send ICMP packets for multicast addresses (to prevent one bad multicast packet from generating millions of ICMP reports).

The *Length* is also a 1-byte field. It tells how long the value is (0 to 255 bytes). The *Value* is any information required, up to 255 bytes.

The hop-by-hop header is used for information that all routers along the path must examine. So far, one option has been defined: support of datagrams exceeding 64 KB. The format of this header is shown in Fig. 5-58. When it is used, the *Payload length* field in the fixed header is set to 0.

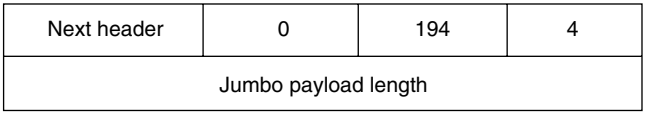


Figure 5-58. The hop-by-hop extension header for large datagrams (jumbograms).

As with all extension headers, this one starts with a byte telling what kind of header comes next. This byte is followed by one telling how long the hop-by-hop header is in bytes, excluding the first 8 bytes, which are mandatory. All extensions begin this way.

The next 2 bytes indicate that this option defines the datagram size (code 194) and that the size is a 4-byte number. The last 4 bytes give the size of the datagram. Sizes less than 65,536 bytes are not permitted and will result in the first router discarding the packet and sending back an ICMP error message. Datagrams using this header extension are called **jumbograms**. The use of jumbograms is important for supercomputer applications that must transfer gigabytes of data efficiently across the Internet.

The destination options header is intended for fields that need only be interpreted at the destination host. In the initial version of IPv6, the only options defined are null options for padding this header out to a multiple of 8 bytes, so initially it will not be used. It was included to make sure that new routing and host software can handle it, in case someone thinks of a destination option some day.

The routing header lists one or more routers that must be visited on the way to the destination. It is very similar to the IPv4 loose source routing in that all addresses listed must be visited in order, but other routers not listed may be visited in between. The format of the routing header is shown in Fig. 5-59.

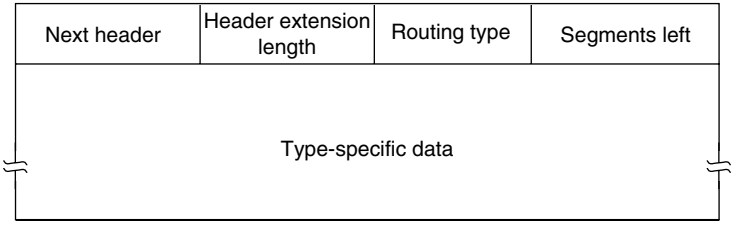


Figure 5-59. The extension header for routing.

The first 4 bytes of the routing extension header contain four 1-byte integers. The *Next header* and *Header extension length* fields were described above. The *Routing type* field gives the format of the rest of the header. Type 0 says that a reserved 32-bit word follows the first word, followed by some number of IPv6 addresses. Other types may be invented in the future, as needed. Finally, the *Segments left* field keeps track of how many of the addresses in the list have not yet been visited. It is decremented every time one is visited. When it hits 0, the packet is on its own with no more guidance about what route to follow. Usually, at this point it is so close to the destination that the best route is obvious.

The fragment header deals with fragmentation similarly to the way IPv4 does. The header holds the datagram identifier, fragment number, and a bit telling whether more fragments will follow. In IPv6, unlike in IPv4, only the source host can fragment a packet. Routers along the way may not do this. This change is a major philosophical break with the original IP, but in keeping with current practice for IPv4. Plus, it simplifies the routers' work and makes routing go faster. As mentioned above, if a router is confronted with a packet that is too big, it discards the packet and sends an ICMP error packet back to the source. This information allows the source host to fragment the packet into smaller pieces using this header and try again.

The authentication header provides a mechanism by which the receiver of a packet can be sure of who sent it. The encrypted security payload makes it possible to encrypt the contents of a packet so that only the intended recipient can read it. These headers use the cryptographic techniques that we will describe in Chap. 8 to accomplish their missions.

Controversies

Given the open design process and the strongly held opinions of many of the people involved, it should come as no surprise that many choices made for IPv6 were highly controversial, to say the least. We will summarize a few of these briefly below. For all the gory details, see the RFCs.

We have already mentioned the argument about the address length. The result was a compromise: 16-byte fixed-length addresses.

Another fight developed over the length of the *Hop limit* field. One camp felt strongly that limiting the maximum number of hops to 255 (implicit in using an 8-bit field) was a gross mistake. After all, paths of 32 hops are common now, and 10 years from now much longer paths may be common. These people argued that using a huge address size was farsighted but using a tiny hop count was shortsighted. In their view, the greatest sin a computer scientist can commit is to provide too few bits somewhere.

The response was that arguments could be made to increase every field, leading to a bloated header. Also, the function of the *Hop limit* field is to keep packets from wandering around for too long a time and 65,535 hops is far, far too long.

Finally, as the Internet grows, more and more long-distance links will be built, making it possible to get from any country to any other country in half a dozen hops at most. If it takes more than 125 hops to get from the source and the destination to their respective international gateways, something is wrong with the national backbones. The 8-bitters won this one.

Another hot potato was the maximum packet size. The supercomputer community wanted packets in excess of 64 KB. When a supercomputer gets started transferring, it really means business and does not want to be interrupted every 64 KB. The argument against large packets is that if a 1-MB packet hits a 1.5-Mbps T1 line, that packet will tie the line up for over 5 seconds, producing a very noticeable delay for interactive users sharing the line. A compromise was reached here: normal packets are limited to 64 KB, but the hop-by-hop extension header can be used to permit jumbograms.

A third hot topic was removing the IPv4 checksum. Some people likened this move to removing the brakes from a car. Doing so makes the car lighter so it can go faster, but if an unexpected event happens, you have a problem.

The argument against checksums was that any application that really cares about data integrity has to have a transport layer checksum anyway, so having another one in IP (in addition to the data link layer checksum) is overkill. Furthermore, experience showed that computing the IP checksum was a major expense in IPv4. The antichecksum camp won this one, and IPv6 does not have a checksum.

Mobile hosts were also a point of contention. If a portable computer flies halfway around the world, can it continue operating there with the same IPv6 address, or does it have to use a scheme with home agents? Some people wanted to build explicit support for mobile hosts into IPv6. That effort failed when no consensus could be found for any specific proposal.

Probably the biggest battle was about security. Everyone agreed it was essential. The war was about where to put it and how. First where. The argument for putting it in the network layer is that it then becomes a standard service that all applications can use without any advance planning. The argument against it is that really secure applications generally want nothing less than end-to-end encryption, where the source application does the encryption and the destination application undoes it. With anything less, the user is at the mercy of potentially buggy network layer implementations over which he has no control. The response to this argument is that these applications can just refrain from using the IP security features and do the job themselves. The rejoinder to that is that the people who do not trust the network to do it right do not want to pay the price of slow, bulky IP implementations that have this capability, even if it is disabled.

Another aspect of where to put security relates to the fact that many (but not all) countries have very stringent export laws concerning cryptography. Some, notably France and Iraq, also restrict its use domestically, so that people cannot have secrets from the government. As a result, any IP implementation that used a

cryptographic system strong enough to be of much value could not be exported from the United States (and many other countries) to customers worldwide. Having to maintain two sets of software, one for domestic use and one for export, is something most computer vendors vigorously oppose.

One point on which there was no controversy is that no one expects the IPv4 Internet to be turned off on a Sunday evening and come back up as an IPv6 Internet Monday morning. Instead, isolated ''islands'' of IPv6 will be converted, initially communicating via tunnels, as we showed in Sec. 5.5.3. As the IPv6 islands grow, they will merge into bigger islands. Eventually, all the islands will merge, and the Internet will be fully converted.

At least, that was the plan. Deployment has proved the Achilles heel of IPv6. It remains little used, even though all major operating systems fully support it. Most deployments are new situations in which a network operator—for example, a mobile phone operator— needs a large number of IP addresses. Many strategies have been defined to help ease the transition. Among them are ways to automatically configure the tunnels that carry IPv6 over the IPv4 Internet, and ways for hosts to automatically find the tunnel endpoints. Dual-stack hosts have an IPv4 and an IPv6 implementation so that they can select which protocol to use depending on the destination of the packet. These strategies will streamline the substantial deployment that seems inevitable when IPv4 addresses are exhausted. For more information about IPv6, see Davies (2008).

5.6.4 Internet Control Protocols

In addition to IP, which is used for data transfer, the Internet has several companion control protocols that are used in the network layer. They include ICMP, ARP, and DHCP. In this section, we will look at each of these in turn, describing the versions that correspond to IPv4 because they are the protocols that are in common use. ICMP and DHCP have similar versions for IPv6; the equivalent of ARP is called NDP (Neighbor Discovery Protocol) for IPv6.

IMCP—The Internet Control Message Protocol

The operation of the Internet is monitored closely by the routers. When something unexpected occurs during packet processing at a router, the event is reported to the sender by the **ICMP** (**Internet Control Message Protocol**). ICMP is also used to test the Internet. About a dozen types of ICMP messages are defined. Each ICMP message type is carried encapsulated in an IP packet. The most important ones are listed in Fig. 5-60.

The DESTINATION UNREACHABLE message is used when the router cannot locate the destination or when a packet with the *DF* bit cannot be delivered because a ''small-packet'' network stands in the way.

Message type	Description
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo and echo reply	Check if a machine is alive
Timestamp request/reply	Same as Echo, but with timestamp
Router advertisement/solicitation	Find a nearby router

Figure 5-60. The principal ICMP message types.

The TIME EXCEEDED message is sent when a packet is dropped because its *TtL (Time to live)* counter has reached zero. This event is a symptom that packets are looping, or that the counter values are being set too low.

One clever use of this error message is the **traceroute** utility that was developed by Van Jacobson in 1987. Traceroute finds the routers along the path from the host to a destination IP address. It finds this information without any kind of privileged network support. The method is simply to send a sequence of packets to the destination, first with a TtL of 1, then a TtL of 2, 3, and so on. The counters on these packets will reach zero at successive routers along the path. These routers will each obediently send a TIME EXCEEDED message back to the host. From those messages, the host can determine the IP addresses of the routers along the path, as well as keep statistics and timings on parts of the path. It is not what the TIME EXCEEDED message was intended for, but it is perhaps the most useful network debugging tool of all time.

The PARAMETER PROBLEM message indicates that an illegal value has been detected in a header field. This problem indicates a bug in the sending host's IP software or possibly in the software of a router transited.

The SOURCE QUENCH message was long ago used to throttle hosts that were sending too many packets. When a host received this message, it was expected to slow down. It is rarely used anymore because when congestion occurs, these packets tend to add more fuel to the fire and it is unclear how to respond to them. Congestion control in the Internet is now done largely by taking action in the transport layer, using packet losses as a congestion signal; we will study it in detail in Chap. 6.

The REDIRECT message is used when a router notices that a packet seems to be routed incorrectly. It is used by the router to tell the sending host to update to a better route.

The ECHO and ECHO REPLY messages are sent by hosts to see if a given destination is reachable and currently alive. Upon receiving the ECHO message,

the destination is expected to send back an ECHO REPLY message. These messages are used in the **ping** utility that checks if a host is up and on the Internet.

The TIMESTAMP REQUEST and TIMESTAMP REPLY messages are similar, except that the arrival time of the message and the departure time of the reply are recorded in the reply. This facility can be used to measure network performance.

The ROUTER ADVERTISEMENT and ROUTER SOLICITATION messages are used to let hosts find nearby routers. A host needs to learn the IP address of at least one router to be able to send packets off the local network.

In addition to these messages, others have been defined. The online list is now kept at *www.iana.org/assignments/icmp-parameters*.

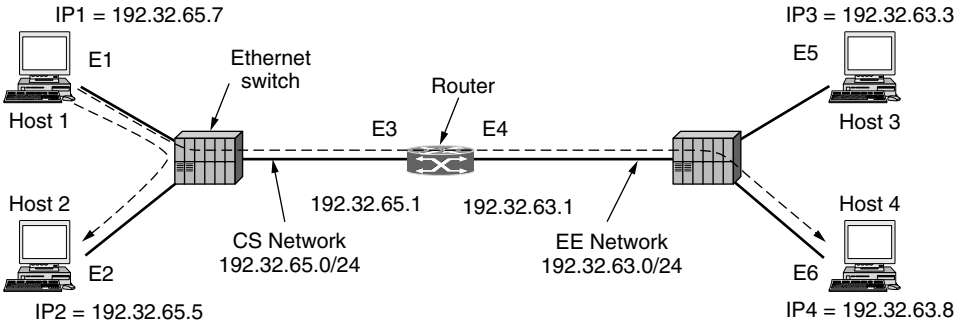
ARP—The Address Resolution Protocol

Although every machine on the Internet has one or more IP addresses, these addresses are not sufficient for sending packets. Data link layer NICs (Network Interface Cards) such as Ethernet cards do not understand Internet addresses. In the case of Ethernet, every NIC ever manufactured comes equipped with a unique 48-bit Ethernet address. Manufacturers of Ethernet NICs request a block of Ethernet addresses from IEEE to ensure that no two NICs have the same address (to avoid conflicts should the two NICs ever appear on the same LAN). The NICs send and receive frames based on 48-bit Ethernet addresses. They know nothing at all about 32-bit IP addresses.

The question now arises, how do IP addresses get mapped onto data link layer addresses, such as Ethernet? To explain how this works, let us use the example of Fig. 5-61, in which a small university with two /24 networks is illustrated. One network (CS) is a switched Ethernet in the Computer Science Dept. It has the prefix 192.32.65.0/24. The other LAN (EE), also switched Ethernet, is in Electrical Engineering and has the prefix 192.32.63.0/24. The two LANs are connected by an IP router. Each machine on an Ethernet and each interface on the router has a unique Ethernet address, labeled *E1* through *E6*, and a unique IP address on the CS or EE network.

Let us start out by seeing how a user on host 1 sends a packet to a user on host 2 on the CS network. Let us assume the sender knows the name of the intended receiver, possibly something like *eagle.cs.uni.edu*. The first step is to find the IP address for host 2. This lookup is performed by DNS, which we will study in Chap. 7. For the moment, we will just assume that DNS returns the IP address for host 2 (192.32.65.5).

The upper layer software on host 1 now builds a packet with 192.32.65.5 in the *Destination address* field and gives it to the IP software to transmit. The IP software can look at the address and see that the destination is on the CS network, (i.e., its own network). However, it still needs some way to find the destination's Ethernet address to send the frame. One solution is to have a configuration file somewhere in the system that maps IP addresses onto Ethernet addresses. While



Frame	Source IP	Source Eth.	Destination IP	Destination Eth.
Host 1 to 2, on CS net	IP1	E1	IP2	E2
Host 1 to 4, on CS net	IP1	E1	IP4	E3
Host 1 to 4, on EE net	IP1	E4	IP4	E6

Figure 5-61. Two switched Ethernet LANs joined by a router.

this solution is certainly possible, for organizations with thousands of machines keeping all these files up to date is an error-prone, time-consuming job.

A better solution is for host 1 to output a broadcast packet onto the Ethernet asking who owns IP address 192.32.65.5. The broadcast will arrive at every machine on the CS Ethernet, and each one will check its IP address. Host 2 alone will respond with its Ethernet address (*E2*). In this way host 1 learns that IP address 192.32.65.5 is on the host with Ethernet address *E2*. The protocol used for asking this question and getting the reply is called **ARP** (**Address Resolution Protocol**). Almost every machine on the Internet runs it. ARP is defined in RFC 826.

The advantage of using ARP over configuration files is the simplicity. The system manager does not have to do much except assign each machine an IP address and decide about subnet masks. ARP does the rest.

At this point, the IP software on host 1 builds an Ethernet frame addressed to *E2*, puts the IP packet (addressed to 192.32.65.5) in the payload field, and dumps it onto the Ethernet. The IP and Ethernet addresses of this packet are given in Fig. 5-61. The Ethernet NIC of host 2 detects this frame, recognizes it as a frame for itself, scoops it up, and causes an interrupt. The Ethernet driver extracts the IP packet from the payload and passes it to the IP software, which sees that it is correctly addressed and processes it.

Various optimizations are possible to make ARP work more efficiently. To start with, once a machine has run ARP, it caches the result in case it needs to contact the same machine shortly. Next time it will find the mapping in its own cache, thus eliminating the need for a second broadcast. In many cases, host 2

will need to send back a reply, forcing it, too, to run ARP to determine the sender's Ethernet address. This ARP broadcast can be avoided by having host 1 include its IP-to-Ethernet mapping in the ARP packet. When the ARP broadcast arrives at host 2, the pair (192.32.65.7, *E1*) is entered into host 2's ARP cache. In fact, all machines on the Ethernet can enter this mapping into their ARP caches.

To allow mappings to change, for example, when a host is configured to use a new IP address (but keeps its old Ethernet address), entries in the ARP cache should time out after a few minutes. A clever way to help keep the cached information current and to optimize performance is to have every machine broadcast its mapping when it is configured. This broadcast is generally done in the form of an ARP looking for its own IP address. There should not be a response, but a side effect of the broadcast is to make or update an entry in everyone's ARP cache. This is known as a **gratuitous ARP**. If a response does (unexpectedly) arrive, two machines have been assigned the same IP address. The error must be resolved by the network manager before both machines can use the network.

Now let us look at Fig. 5-61 again, only this time assume that host 1 wants to send a packet to host 4 (192.32.63.8) on the EE network. Host 1 will see that the destination IP address is not on the CS network. It knows to send all such off-network traffic to the router, which is also known as the **default gateway**. By convention, the default gateway is the lowest address on the network (198.31.65.1). To send a frame to the router, host 1 must still know the Ethernet address of the router interface on the CS network. It discovers this by sending an ARP broadcast for 198.31.65.1, from which it learns *E3*. It then sends the frame. The same lookup mechanisms are used to send a packet from one router to the next over a sequence of routers in an Internet path.

When the Ethernet NIC of the router gets this frame, it gives the packet to the IP software. It knows from the network masks that the packet should be sent onto the EE network where it will reach host 4. If the router does not know the Ethernet address for host 4, then it will use ARP again. The table in Fig. 5-61 lists the source and destination Ethernet and IP addresses that are present in the frames as observed on the CS and EE networks. Observe that the Ethernet addresses change with the frame on each network while the IP addresses remain constant (because they indicate the endpoints across all of the interconnected networks).

It is also possible to send a packet from host 1 to host 4 without host 1 knowing that host 4 is on a different network. The solution is to have the router answer ARPs on the CS network for host 4 and give its Ethernet address, *E3*, as the response. It is not possible to have host 4 reply directly because it will not see the ARP request (as routers do not forward Ethernet-level broadcasts). The router will then receive frames sent to 192.32.63.8 and forward them onto the EE network. This solution is called **proxy ARP**. It is used in special cases in which a host wants to appear on a network even though it actually resides on another network. A common situation, for example, is a mobile computer that wants some other node to pick up packets for it when it is not on its home network.

DHCP—The Dynamic Host Configuration Protocol

ARP (as well as other Internet protocols) makes the assumption that hosts are configured with some basic information, such as their own IP addresses. How do hosts get this information? It is possible to manually configure each computer, but that is tedious and error-prone. There is a better way, and it is called **DHCP** (**Dynamic Host Configuration Protocol**).

With DHCP, every network must have a DHCP server that is responsible for configuration. When a computer is started, it has a built-in Ethernet or other link layer address embedded in the NIC, but no IP address. Much like ARP, the computer broadcasts a request for an IP address on its network. It does this by using a DHCP DISCOVER packet. This packet must reach the DHCP server. If that server is not directly attached to the network, the router will be configured to receive DHCP broadcasts and relay them to the DHCP server, wherever it is located.

When the server receives the request, it allocates a free IP address and sends it to the host in a DHCP OFFER packet (which again may be relayed via the router). To be able to do this work even when hosts do not have IP addresses, the server identifies a host using its Ethernet address (which is carried in the DHCP DISCOVER packet)

An issue that arises with automatic assignment of IP addresses from a pool is for how long an IP address should be allocated. If a host leaves the network and does not return its IP address to the DHCP server, that address will be permanently lost. After a period of time, many addresses may be lost. To prevent that from happening, IP address assignment may be for a fixed period of time, a technique called **leasing**. Just before the lease expires, the host must ask for a DHCP renewal. If it fails to make a request or the request is denied, the host may no longer use the IP address it was given earlier.

DHCP is described in RFCs 2131 and 2132. It is widely used in the Internet to configure all sorts of parameters in addition to providing hosts with IP addresses. As well as in business and home networks, DHCP is used by ISPs to set the parameters of devices over the Internet access link, so that customers do not need to phone their ISPs to get this information. Common examples of the information that is configured include the network mask, the IP address of the default gateway, and the IP addresses of DNS and time servers. DHCP has largely replaced earlier protocols (called RARP and BOOTP) with more limited functionality.

5.6.5 Label Switching and MPLS

So far, on our tour of the network layer of the Internet, we have focused exclusively on packets as datagrams that are forwarded by IP routers. There is also another kind of technology that is starting to be widely used, especially by ISPs, in order to move Internet traffic across their networks. This technology is

called **MPLS** (**MultiProtocol Label Switching**) and it is perilously close to circuit switching. Despite the fact that many people in the Internet community have an intense dislike for connection-oriented networking, the idea seems to keep coming back. As Yogi Berra once put it, it is like deja vu all over again. However, there are essential differences between the way the Internet handles route construction and the way connection-oriented networks do it, so the technique is certainly not traditional circuit switching.

MPLS adds a label in front of each packet, and forwarding is based on the label rather than on the destination address. Making the label an index into an internal table makes finding the correct output line just a matter of table lookup. Using this technique, forwarding can be done very quickly. This advantage was the original motivation behind MPLS, which began as proprietary technology known by various names including **tag switching**. Eventually, IETF began to standardize the idea. It is described in RFC 3031 and many other RFCs. The main benefits over time have come to be routing that is flexible and forwarding that is suited to quality of service as well as fast.

The first question to ask is where does the label go? Since IP packets were not designed for virtual circuits, there is no field available for virtual-circuit numbers within the IP header. For this reason, a new MPLS header had to be added in front of the IP header. On a router-to-router line using PPP as the framing protocol, the frame format, including the PPP, MPLS, IP, and TCP headers, is as shown in Fig. 5-62.

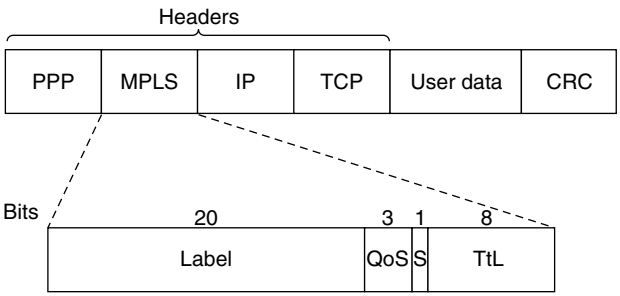


Figure 5-62. Transmitting a TCP segment using IP, MPLS, and PPP.

The generic MPLS header is 4 bytes long and has four fields. Most important is the *Label* field, which holds the index. The *QoS* field indicates the class of service. The *S* field relates to stacking multiple labels (which is discussed below). The *TtL* field indicates how many more times the packet may be forwarded. It is decremented at each router, and if it hits 0, the packet is discarded. This feature prevents infinite looping in the case of routing instability.

MPLS falls between the IP network layer protocol and the PPP link layer protocol. It is not really a layer 3 protocol because it depends on IP or other network

layer addresses to set up label paths. It is not really a layer 2 protocol either because it forwards packets across multiple hops, not a single link. For this reason, MPLS is sometimes described as a layer 2.5 protocol. It is an illustration that real protocols do not always fit neatly into our ideal layered protocol model.

On the brighter side, because the MPLS headers are not part of the network layer packet or the data link layer frame, MPLS is to a large extent independent of both layers. Among other things, this property means it is possible to build MPLS switches that can forward both IP packets and non-IP packets, depending on what shows up. This feature is where the "multiprotocol" in the name MPLS came from. MPLS can also carry IP packets over non-IP networks.

When an MPLS-enhanced packet arrives at a **LSR** (**Label Switched Router**), the label is used as an index into a table to determine the outgoing line to use and also the new label to use. This label swapping is used in all virtual-circuit networks. Labels have only local significance and two different routers can feed unrelated packets with the same label into another router for transmission on the same outgoing line. To be distinguishable at the other end, labels have to be remapped at every hop. We saw this mechanism in action in Fig. 5-3. MPLS uses the same technique.

As an aside, some people distinguish between *forwarding* and *switching*. Forwarding is the process of finding the best match for a destination address in a table to decide where to send packets. An example is the longest matching prefix algorithm used for IP forwarding. In contrast, switching uses a label taken from the packet as an index into a forwarding table. It is simpler and faster. These definitions are far from universal, however.

Since most hosts and routers do not understand MPLS, we should also ask when and how the labels are attached to packets. This happens when an IP packet reaches the edge of an MPLS network. The **LER** (**Label Edge Router**) inspects the destination IP address and other fields to see which MPLS path the packet should follow, and puts the right label on the front of the packet. Within the MPLS network, this label is used to forward the packet. At the other edge of the MPLS network, the label has served its purpose and is removed, revealing the IP packet again for the next network. This process is shown in Fig. 5-63. One difference from traditional virtual circuits is the level of aggregation. It is certainly possible for each flow to have its own set of labels through the MPLS network. However, it is more common for routers to group multiple flows that end at a particular router or LAN and use a single label for them. The flows that are grouped together under a single label are said to belong to the same **FEC** (**Forwarding Equivalence Class**). This class covers not only where the packets are going, but also their service class (in the differentiated services sense) because all the packets are treated the same way for forwarding purposes.

With traditional virtual-circuit routing, it is not possible to group several distinct paths with different endpoints onto the same virtual-circuit identifier because there would be no way to distinguish them at the final destination. With MPLS,

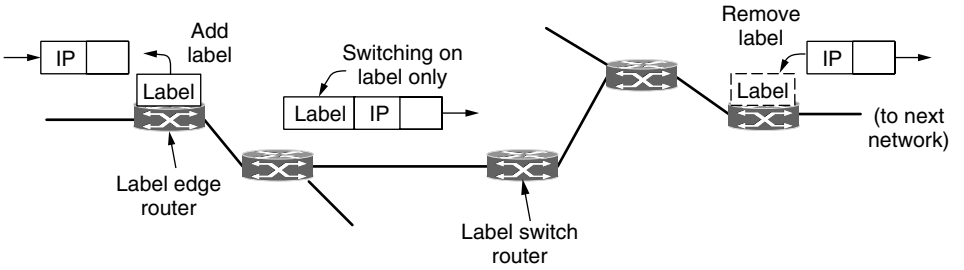


Figure 5-63. Forwarding an IP packet through an MPLS network.

the packets still contain their final destination address, in addition to the label. At the end of the labeled route, the label header can be removed and forwarding can continue the usual way, using the network layer destination address.

Actually, MPLS goes even further. It can operate at multiple levels at once by adding more than one label to the front of a packet. For example, suppose that there are many packets that already have different labels (because we want to treat the packets differently somewhere in the network) that should follow a common path to some destination. Instead of setting up many label switching paths, one for each of the different labels, we can set up a single path. When the already-labeled packets reach the start of this path, another label is added to the front. This is called a stack of labels. The outermost label guides the packets along the path. It is removed at the end of the path, and the labels revealed, if any, are used to forward the packet further. The *S* bit in Fig. 5-62 allows a router removing a label to know if there are any additional labels left. It is set to 1 for the bottom label and 0 for all the other labels.

The final question we will ask is how the label forwarding tables are set up so that packets follow them. This is one area of major difference between MPLS and conventional virtual-circuit designs. In traditional virtual-circuit networks, when a user wants to establish a connection, a setup packet is launched into the network to create the path and make the forwarding table entries. MPLS does not involve users in the setup phase. Requiring users to do anything other than send a datagram would break too much existing Internet software.

Instead, the forwarding information is set up by protocols that are a combination of routing protocols and connection setup protocols. These control protocols are cleanly separated from label forwarding, which allows multiple, different control protocols to be used. One of the variants works like this. When a router is booted, it checks to see which routes it is the final destination for (e.g., which prefixes belong to its interfaces). It then creates one or more FECs for them, allocates a label for each one, and passes the labels to its neighbors. They, in turn, enter the labels in their forwarding tables and send new labels to their neighbors, until all the routers have acquired the path. Resources can also be reserved as the

path is constructed to guarantee an appropriate quality of service. Other variants can set up different paths, such as traffic engineering paths that take unused capacity into account, and create paths on-demand to support service offerings such as quality of service.

Although the basic ideas behind MPLS are straightforward, the details are complicated, with many variations and use cases that are being actively developed. For more information, see Davie and Farrel (2008) and Davie and Rekhter (2000).

5.6.6 OSPF—An Interior Gateway Routing Protocol

We have now finished our study of how packets are forwarded in the Internet. It is time to move on to the next topic: routing in the Internet. As we mentioned earlier, the Internet is made up of a large number of independent networks or **ASes** (**Autonomous Systems**) that are operated by different organizations, usually a company, university, or ISP. Inside of its own network, an organization can use its own algorithm for internal routing, or **intradomain routing**, as it is more commonly known. Nevertheless, there are only a handful of standard protocols that are popular. In this section, we will study the problem of intradomain routing and look at the OSPF protocol that is widely used in practice. An intradomain routing protocol is also called an **interior gateway protocol**. In the next section, we will study the problem of routing between independently operated networks, or **interdomain routing**. For that case, all networks must use the same interdomain routing protocol or **exterior gateway protocol**. The protocol that is used in the Internet is BGP (Border Gateway Protocol).

Early intradomain routing protocols used a distance vector design, based on the distributed Bellman-Ford algorithm inherited from the ARPANET. RIP (Routing Information Protocol) is the main example that is used to this day. It works well in small systems, but less well as networks get larger. It also suffers from the count-to-infinity problem and generally slow convergence. The ARPANET switched over to a link state protocol in May 1979 because of these problems, and in 1988 IETF began work on a link state protocol for intradomain routing. That protocol, called **OSPF** (**Open Shortest Path First**), became a standard in 1990. It drew on a protocol called **IS-IS** (**Intermediate-System to Intermediate-System**), which became an ISO standard. Because of their shared heritage, the two protocols are much more alike than different. For the complete story, see RFC 2328. They are the dominant intradomain routing protocols, and most router vendors now support both of them. OSPF is more widely used in company networks, and IS-IS is more widely used in ISP networks. Of the two, we will give a sketch of how OSPF works.

Given the long experience with other routing protocols, the group designing OSPF had a long list of requirements that had to be met. First, the algorithm had to be published in the open literature, hence the "O" in OSPF. A proprietary

solution owned by one company would not do. Second, the new protocol had to support a variety of distance metrics, including physical distance, delay, and so on. Third, it had to be a dynamic algorithm, one that adapted to changes in the topology automatically and quickly.

Fourth, and new for OSPF, it had to support routing based on type of service. The new protocol had to be able to route real-time traffic one way and other traffic a different way. At the time, IP had a *Type of service* field, but no existing routing protocol used it. This field was included in OSPF but still nobody used it, and it was eventually removed. Perhaps this requirement was ahead of its time, as it preceded IETF's work on differentiated services, which has rejuvenated classes of service.

Fifth, and related to the above, OSPF had to do load balancing, splitting the load over multiple lines. Most previous protocols sent all packets over a single best route, even if there were two routes that were equally good. The other route was not used at all. In many cases, splitting the load over multiple routes gives better performance.

Sixth, support for hierarchical systems was needed. By 1988, some networks had grown so large that no router could be expected to know the entire topology. OSPF had to be designed so that no router would have to.

Seventh, some modicum of security was required to prevent fun-loving students from spoofing routers by sending them false routing information. Finally, provision was needed for dealing with routers that were connected to the Internet via a tunnel. Previous protocols did not handle this well.

OSPF supports both point-to-point links (e.g., SONET) and broadcast networks (e.g., most LANs). Actually, it is able to support networks with multiple routers, each of which can communicate directly with the others (called **multiaccess networks**) even if they do not have broadcast capability. Earlier protocols did not handle this case well.

An example of an autonomous system network is given in Fig. 5-64(a). Hosts are omitted because they do not generally play a role in OSPF, while routers and networks (which may contain hosts) do. Most of the routers in Fig. 5-64(a) are connected to other routers by point-to-point links, and to networks to reach the hosts on those networks. However, routers *R3*, *R4*, and *R5* are connected by a broadcast LAN such as switched Ethernet.

OSPF operates by abstracting the collection of actual networks, routers, and links into a directed graph in which each arc is assigned a weight (distance, delay, etc.). A point-to-point connection between two routers is represented by a pair of arcs, one in each direction. Their weights may be different. A broadcast network is represented by a node for the network itself, plus a node for each router. The arcs from that network node to the routers have weight 0. They are important nonetheless, as without them there is no path through the network. Other networks, which have only hosts, have only an arc reaching them and not one returning. This structure gives routes to hosts, but not through them.

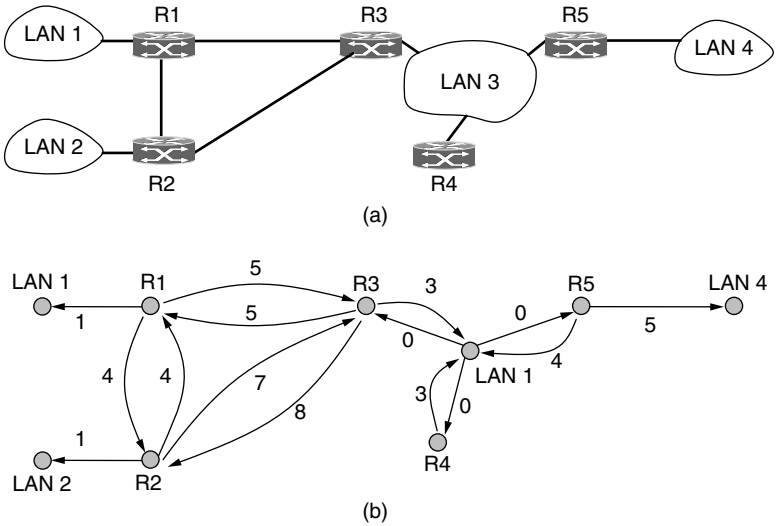


Figure 5-64. (a) An autonomous system. (b) A graph representation of (a).

Figure 5-64(b) shows the graph representation of the network of Fig. 5-64(a). What OSPF fundamentally does is represent the actual network as a graph like this and then use the link state method to have every router compute the shortest path from itself to all other nodes. Multiple paths may be found that are equally short. In this case, OSPF remembers the set of shortest paths and during packet forwarding, traffic is split across them. This helps to balance load. It is called **ECMP** (**Equal Cost MultiPath**).

Many of the ASes in the Internet are themselves large and nontrivial to manage. To work at this scale, OSPF allows an AS to be divided into numbered **areas**, where an area is a network or a set of contiguous networks. Areas do not overlap but need not be exhaustive, that is, some routers may belong to no area. Routers that lie wholly within an area are called **internal routers**. An area is a generalization of an individual network. Outside an area, its destinations are visible but not its topology. This characteristic helps routing to scale.

Every AS has a **backbone area**, called area 0. The routers in this area are called **backbone routers**. All areas are connected to the backbone, possibly by tunnels, so it is possible to go from any area in the AS to any other area in the AS via the backbone. A tunnel is represented in the graph as just another arc with a cost. As with other areas, the topology of the backbone is not visible outside the backbone.

Each router that is connected to two or more areas is called an **area border router**. It must also be part of the backbone. The job of an area border router is to summarize the destinations in one area and to inject this summary into the other

areas to which it is connected. This summary includes cost information but not all the details of the topology within an area. Passing cost information allows hosts in other areas to find the best area border router to use to enter an area. Not passing topology information reduces traffic and simplifies the shortest-path computations of routers in other areas. However, if there is only one border router out of an area, even the summary does not need to be passed. Routes to destinations out of the area always start with the instruction "Go to the border router." This kind of area is called a **stub area**.

The last kind of router is the **AS boundary router**. It injects routes to external destinations on other ASes into the area. The external routes then appear as destinations that can be reached via the AS boundary router with some cost. An external route can be injected at one or more AS boundary routers. The relationship between ASes, areas, and the various kinds of routers is shown in Fig. 5-65. One router may play multiple roles, for example, a border router is also a backbone router.

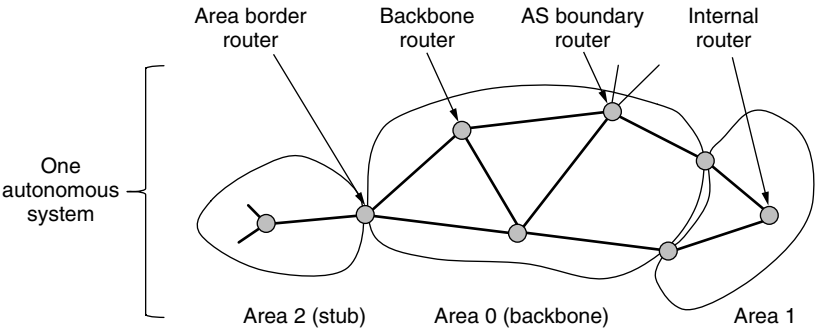


Figure 5-65. The relation between ASes, backbones, and areas in OSPF.

During normal operation, each router within an area has the same link state database and runs the same shortest path algorithm. Its main job is to calculate the shortest path from itself to every other router and network in the entire AS. An area border router needs the databases for all the areas to which it is connected and must run the shortest path algorithm for each area separately.

For a source and destination in the same area, the best intra-area route (that lies wholly within the area) is chosen. For a source and destination in different areas, the inter-area route must go from the source to the backbone, across the backbone to the destination area, and then to the destination. This algorithm forces a star configuration on OSPF, with the backbone being the hub and the other areas being spokes. Because the route with the lowest cost is chosen, routers in different parts of the network may use different area border routers to enter the backbone and destination area. Packets are routed from source to destination "as is." They are not encapsulated or tunneled (unless going to an area whose

only connection to the backbone is a tunnel). Also, routes to external destinations may include the external cost from the AS boundary router over the external path, if desired, or just the cost internal to the AS.

When a router boots, it sends HELLO messages on all of its point-to-point lines and multicasts them on LANs to the group consisting of all the other routers. From the responses, each router learns who its neighbors are. Routers on the same LAN are all neighbors.

OSPF works by exchanging information between adjacent routers, which is not the same as between neighboring routers. In particular, it is inefficient to have every router on a LAN talk to every other router on the LAN. To avoid this situation, one router is elected as the **designated router**. It is said to be **adjacent** to all the other routers on its LAN, and exchanges information with them. In effect, it is acting as the single node that represents the LAN. Neighboring routers that are not adjacent do not exchange information with each other. A backup designated router is always kept up to date to ease the transition should the primary designated router crash and need to be replaced immediately.

During normal operation, each router periodically floods LINK STATE UPDATE messages to each of its adjacent routers. These messages gives its state and provide the costs used in the topological database. The flooding messages are acknowledged, to make them reliable. Each message has a sequence number, so a router can see whether an incoming LINK STATE UPDATE is older or newer than what it currently has. Routers also send these messages when a link goes up or down or its cost changes.

DATABASE DESCRIPTION messages give the sequence numbers of all the link state entries currently held by the sender. By comparing its own values with those of the sender, the receiver can determine who has the most recent values. These messages are used when a link is brought up.

Either partner can request link state information from the other one by using LINK STATE REQUEST messages. The result of this algorithm is that each pair of adjacent routers checks to see who has the most recent data, and new information is spread throughout the area this way. All these messages are sent directly in IP packets. The five kinds of messages are summarized in Fig. 5-66.

Message type	Description
Hello	Used to discover who the neighbors are
Link state update	Provides the sender's costs to its neighbors
Link state ack	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Requests information from the partner

Figure 5-66. The five types of OSPF messages.

Finally, we can put all the pieces together. Using flooding, each router informs all the other routers in its area of its links to other routers and networks and the cost of these links. This information allows each router to construct the graph for its area(s) and compute the shortest paths. The backbone area does this work, too. In addition, the backbone routers accept information from the area border routers in order to compute the best route from each backbone router to every other router. This information is propagated back to the area border routers, which advertise it within their areas. Using this information, internal routers can select the best route to a destination outside their area, including the best exit router to the backbone.

5.6.7 BGP—The Exterior Gateway Routing Protocol

Within a single AS, OSPF and IS-IS are the protocols that are commonly used. Between ASes, a different protocol, called **BGP** (**Border Gateway Protocol**), is used. A different protocol is needed because the goals of an intradomain protocol and an interdomain protocol are not the same. All an intradomain protocol has to do is move packets as efficiently as possible from the source to the destination. It does not have to worry about politics.

In contrast, interdomain routing protocols have to worry about politics a great deal (Metz, 2001). For example, a corporate AS might want the ability to send packets to any Internet site and receive packets from any Internet site. However, it might be unwilling to carry transit packets originating in a foreign AS and ending in a different foreign AS, even if its own AS is on the shortest path between the two foreign ASes (“That’s their problem, not ours”). On the other hand, it might be willing to carry transit traffic for its neighbors, or even for specific other ASes that paid it for this service. Telephone companies, for example, might be happy to act as carriers for their customers, but not for others. Exterior gateway protocols in general, and BGP in particular, have been designed to allow many kinds of routing policies to be enforced in the interAS traffic.

Typical policies involve political, security, or economic considerations. A few examples of possible routing constraints are:

1. Do not carry commercial traffic on the educational network.
2. Never send traffic from the Pentagon on a route through Iraq.
3. Use TeliaSonera instead of Verizon because it is cheaper.
4. Don’t use AT&T in Australia because performance is poor.
5. Traffic starting or ending at Apple should not transit Google.

As you might imagine from this list, routing policies can be highly individual. They are often proprietary because they contain sensitive business information.

However, we can describe some patterns that capture the reasoning of the company above and that are often used as a starting point.

A routing policy is implemented by deciding what traffic can flow over which of the links between ASes. One common policy is that a customer ISP pays another provider ISP to deliver packets to any other destination on the Internet and receive packets sent from any other destination. The customer ISP is said to buy **transit service** from the provider ISP. This is just like a customer at home buying Internet access service from an ISP. To make it work, the provider should advertise routes to all destinations on the Internet to the customer over the link that connects them. In this way, the customer will have a route to use to send packets anywhere. Conversely, the customer should advertise routes only to the destinations on its network to the provider. This will let the provider send traffic to the customer only for those addresses; the customer does not want to handle traffic intended for other destinations.

We can see an example of transit service in Fig. 5-67. There are four ASes that are connected. The connection is often made with a link at **IXPs** (**Internet eXchange Points**), facilities to which many ISPs have a link for the purpose of connecting with other ISPs. *AS2*, *AS3*, and *AS4* are customers of *AS1*. They buy transit service from it. Thus, when source *A* sends to destination *C*, the packets travel from *AS2* to *AS1* and finally to *AS4*. The routing advertisements travel in the opposite direction to the packets. *AS4* advertises *C* as a destination to its transit provider, *AS1*, to let sources reach *C* via *AS1*. Later, *AS1* advertises a route to *C* to its other customers, including *AS2*, to let the customers know that they can send traffic to *C* via *AS1*.

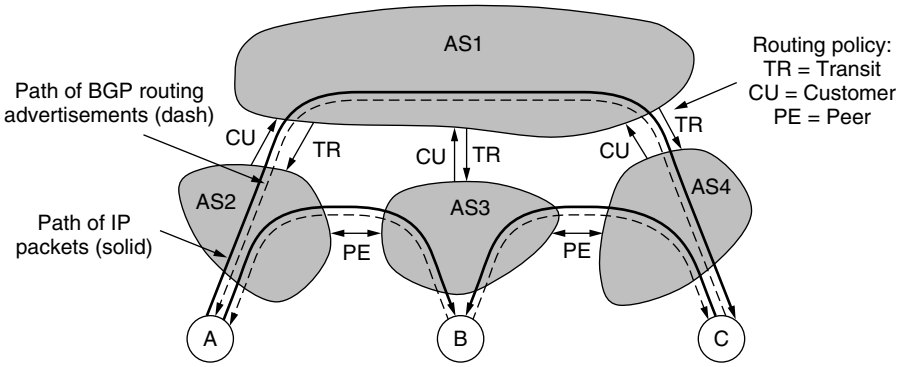


Figure 5-67. Routing policies between four autonomous systems.

In Fig. 5-67, all of the other ASes buy transit service from *AS1*. This provides them with connectivity so they can interact with any host on the Internet. However, they have to pay for this privilege. Suppose that *AS2* and *AS3* exchange a lot of traffic. Given that their networks are connected already, if they want to, they

can use a different policy—they can send traffic directly to each other for free. This will reduce the amount of traffic they must have *AS1* deliver on their behalf, and hopefully it will reduce their bills. This policy is called **peering**.

To implement peering, two ASes send routing advertisements to each other for the addresses that reside in their networks. Doing so makes it possible for *AS2* to send *AS3* packets from *A* destined to *B* and vice versa. However, note that peering is not transitive. In Fig. 5-67, *AS3* and *AS4* also peer with each other. This peering allows traffic from *C* destined for *B* to be sent directly to *AS4*. What happens if *C* sends a packet to *A*? *AS3* is only advertising a route to *B* to *AS4*. It is not advertising a route to *A*. The consequence is that traffic will not pass from *AS4* to *AS3* to *AS2*, even though a physical path exists. This restriction is exactly what *AS3* wants. It peers with *AS4* to exchange traffic, but does not want to carry traffic from *AS4* to other parts of the Internet since it is not being paid to so do. Instead, *AS4* gets transit service from *AS1*. Thus, it is *AS1* who will carry the packet from *C* to *A*.

Now that we know about transit and peering, we can also see that *A*, *B*, and *C* have transit arrangements. For example, *A* must buy Internet access from *AS2*. *A* might be a single home computer or a company network with many LANs. However, it does not need to run BGP because it is a **stub network** that is connected to the rest of the Internet by only one link. So the only place for it to send packets destined outside of the network is over the link to *AS2*. There is nowhere else to go. This path can be arranged simply by setting up a default route. For this reason, we have not shown *A*, *B*, and *C* as ASes that participate in interdomain routing.

On the other hand, some company networks are connected to multiple ISPs. This technique is used to improve reliability, since if the path through one ISP fails, the company can use the path via the other ISP. This technique is called **multihoming**. In this case, the company network is likely to run an interdomain routing protocol (e.g., BGP) to tell other ASes which addresses should be reached via which ISP links.

Many variations on these transit and peering policies are possible, but they already illustrate how business relationships and control over where route advertisements go can implement different kinds of policies. Now we will consider in more detail how routers running BGP advertise routes to each other and select paths over which to forward packets.

BGP is a form of distance vector protocol, but it is quite unlike intradomain distance vector protocols such as RIP. We have already seen that policy, instead of minimum distance, is used to pick which routes to use. Another large difference is that instead of maintaining just the cost of the route to each destination, each BGP router keeps track of the path used. This approach is called a **path vector protocol**. The path consists of the next hop router (which may be on the other side of the ISP, not adjacent) and the sequence of ASes, or **AS path**, that the route has followed (given in reverse order). Finally, pairs of BGP routers communicate

with each other by establishing TCP connections. Operating this way provides reliable communication and also hides all the details of the network being passed through.

An example of how BGP routes are advertised is shown in Fig. 5-68. There are three ASes and the middle one is providing transit to the left and right ISPs. A route advertisement to prefix *C* starts in *AS3*. When it is propagated across the link to *R2c* at the top of the figure, it has the AS path of simply *AS3* and the next hop router of *R3a*. At the bottom, it has the same AS path but a different next hop because it came across a different link. This advertisement continues to propagate and crosses the boundary into *AS1*. At router *R1a*, at the top of the figure, the AS path is *AS2*, *AS3* and the next hop is *R2a*.

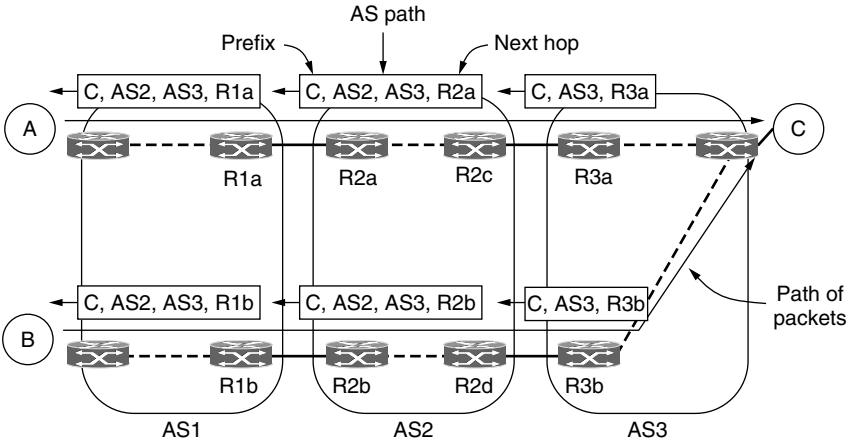


Figure 5-68. Propagation of BGP route advertisements.

Carrying the complete path with the route makes it easy for the receiving router to detect and break routing loops. The rule is that each router that sends a route outside of the AS prepends its own AS number to the route. (This is why the list is in reverse order.) When a router receives a route, it checks to see if its own AS number is already in the AS path. If it is, a loop has been detected and the advertisement is discarded. However, and somewhat ironically, it was realized in the late 1990s that despite this precaution BGP suffers from a version of the count-to-infinity problem (Labovitz et al., 2001). There are no long-lived loops, but routes can sometimes be slow to converge and have transient loops.

Giving a list of ASes is a very coarse way to specify a path. An AS might be a small company, or an international backbone network. There is no way of telling from the route. BGP does not even try because different ASes may use different intradomain protocols whose costs cannot be compared. Even if they could be compared, an AS may not want to reveal its internal metrics. This is one of the ways that interdomain routing protocols differ from intradomain protocols.

So far we have seen how a route advertisement is sent across the link between two ISPs. We still need some way to propagate BGP routes from one side of the ISP to the other, so they can be sent on to the next ISP. This task could be handled by the intradomain protocol, but because BGP is very good at scaling to large networks, a variant of BGP is often used. It is called **iBGP** (**internal BGP**) to distinguish it from the regular use of BGP as **eBGP** (**external BGP**).

The rule for propagating routes inside an ISP is that every router at the boundary of the ISP learns of all the routes seen by all the other boundary routers, for consistency. If one boundary router on the ISP learns of a prefix to IP 128.208.0.0/16, all the other routers will learn of this prefix. The prefix will then be reachable from all parts of the ISP, no matter how packets enter the ISP from other ASes.

We have not shown this propagation in Fig. 5-68 to avoid clutter, but, for example, router *R2b* will know that it can reach *C* via either router *R2c* at top or router *R2d* at bottom. The next hop is updated as the route crosses within the ISP so that routers on the far side of the ISP know which router to use to exit the ISP on the other side. This can be seen in the leftmost routes in which the next hop points to a router in the same ISP and not a router in the next ISP.

We can now describe the key missing piece, which is how BGP routers choose which route to use for each destination. Each BGP router may learn a route for a given destination from the router it is connected to in the next ISP and from all of the other boundary routers (which have heard different routes from the routers they are connected to in other ISPs). Each router must decide which route in this set of routes is the best one to use. Ultimately the answer is that it is up to the ISP to write some policy to pick the preferred route. However, this explanation is very general and not at all satisfying, so we can at least describe some common strategies.

The first strategy is that routes via peered networks are chosen in preference to routes via transit providers. The former are free; the latter cost money. A similar strategy is that customer routes are given the highest preference. It is only good business to send traffic directly to the paying customers.

A different kind of strategy is the default rule that shorter AS paths are better. This is debatable given that an AS could be a network of any size, so a path through three small ASes could actually be shorter than a path through one big AS. However, shorter tends to be better on average, and this rule is a common tiebreaker.

The final strategy is to prefer the route that has the lowest cost within the ISP. This is the strategy implemented in Fig. 5-68. Packets sent from *A* to *C* exit *AS1* at the top router, *R1a*. Packets sent from *B* exit via the bottom router, *R1b*. The reason is that both *A* and *B* are taking the lowest-cost path or quickest route out of *AS1*. Because they are located in different parts of the ISP, the quickest exit for each one is different. The same thing happens as the packets pass through *AS2*. On the last leg, *AS3* has to carry the packet from *B* through its own network.

This strategy is known as **early exit** or **hot-potato routing**. It has the curious side effect of tending to make routes asymmetric. For example, consider the path taken when *C* sends a packet back to *B*. The packet will exit *AS3* quickly, at the top router, to avoid wasting its resources. Similarly, it will stay at the top when *AS2* passes it to *AS1* as quickly as possible. Then the packet will have a longer journey in *AS1*. This is a mirror image of the path taken from *B* to *C*.

The above discussion should make clear that each BGP router chooses its own best route from the known possibilities. It is not the case, as might naively be expected, that BGP chooses a path to follow at the AS level and OSPF chooses paths within each of the ASes. BGP and the interior gateway protocol are integrated much more deeply. This means that, for example, BGP can find the best exit point from one ISP to the next and this point will vary across the ISP, as in the case of the hot-potato policy. It also means that BGP routers in different parts of one AS may choose different AS paths to reach the same destination. Care must be exercised by the ISP to configure all of the BGP routers to make compatible choices given all of this freedom, but this can be done in practice.

Amazingly, we have only scratched the surface of BGP. For more information, see the BGP version 4 specification in RFC 4271 and related RFCs. However, realize that much of its complexity lies with policies, which are not described in the specification of the BGP protocol.

5.6.8 Internet Multicasting

Normal IP communication is between one sender and one receiver. However, for some applications, it is useful for a process to be able to send to a large number of receivers simultaneously. Examples are streaming a live sports event to many viewers, delivering program updates to a pool of replicated servers, and handling digital conference (i.e., multiparty) telephone calls.

IP supports one-to-many communication, or multicasting, using class D IP addresses. Each class D address identifies a group of hosts. Twenty-eight bits are available for identifying groups, so over 250 million groups can exist at the same time. When a process sends a packet to a class D address, a best-effort attempt is made to deliver it to all the members of the group addressed, but no guarantees are given. Some members may not get the packet.

The range of IP addresses 224.0.0.0/24 is reserved for multicast on the local network. In this case, no routing protocol is needed. The packets are multicast by simply broadcasting them on the LAN with a multicast address. All hosts on the LAN receive the broadcasts, and hosts that are members of the group process the packet. Routers do not forward the packet off the LAN. Some examples of local multicast addresses are:

224.0.0.1 All systems on a LAN
224.0.0.2 All routers on a LAN
224.0.0.5 All OSPF routers on a LAN
224.0.0.251 All DNS servers on a LAN

Other multicast addresses may have members on different networks. In this case, a routing protocol is needed. But first the multicast routers need to know which hosts are members of a group. A process asks its host to join in a specific group. It can also ask its host to leave the group. Each host keeps track of which groups its processes currently belong to. When the last process on a host leaves a group, the host is no longer a member of that group. About once a minute, each multicast router sends a query packet to all the hosts on its LAN (using the local multicast address of 224.0.0.1, of course) asking them to report back on the groups to which they currently belong. The multicast routers may or may not be colocated with the standard routers. Each host sends back responses for all the class D addresses it is interested in. These query and response packets use a protocol called **IGMP** (**Internet Group Management Protocol**). It is described in RFC 3376.

Any of several multicast routing protocols may be used to build multicast spanning trees that give paths from senders to all of the members of the group. The algorithms that are used are the ones we described in Sec. 5.2.8. Within an AS, the main protocol used is **PIM** (**Protocol Independent Multicast**). PIM comes in several flavors. In Dense Mode PIM, a pruned reverse path forwarding tree is created. This is suited to situations in which members are everywhere in the network, such as distributing files to many servers within a data center network. In Sparse Mode PIM, spanning trees that are built are similar to core-based trees. This is suited to situations such as a content provider multicasting TV to subscribers on its IP network. A variant of this design, called Source-Specific Multicast PIM, is optimized for the case that there is only one sender to the group. Finally, multicast extensions to BGP or tunnels need to be used to create multicast routes when the group members are in more than one AS.

5.6.9 Mobile IP

Many users of the Internet have mobile computers and want to stay connected when they are away from home and even on the road in between. Unfortunately, the IP addressing system makes working far from home easier said than done, as we will describe shortly. When people began demanding the ability anyway, IETF set up a Working Group to find a solution. The Working Group quickly formulated a number of goals considered desirable in any solution. The major ones were:

1. Each mobile host must be able to use its home IP address anywhere.
2. Software changes to the fixed hosts were not permitted.
3. Changes to the router software and tables were not permitted.
4. Most packets for mobile hosts should not make detours on the way.
5. No overhead should be incurred when a mobile host is at home.

The solution chosen was the one described in Sec. 5.2.10. In brief, every site that wants to allow its users to roam has to create a helper at the site called a **home agent**. When a mobile host shows up at a foreign site, it obtains a new IP address (called a care-of address) at the foreign site. The mobile then tells the home agent where it is now by giving it the care-of address. When a packet for the mobile arrives at the home site and the mobile is elsewhere, the home agent grabs the packet and tunnels it to the mobile at the current care-of address. The mobile can send reply packets directly to whoever it is communicating with, but still using its home address as the source address. This solution meets all the requirements stated above except that packets for mobile hosts do make detours.

Now that we have covered the network layer of the Internet, we can go into the solution in more detail. The need for mobility support in the first place comes from the IP addressing scheme itself. Every IP address contains a network number and a host number. For example, consider the machine with IP address 160.80.40.20/16. The 160.80 gives the network number; the 40.20 is the host number. Routers all over the world have routing tables telling which link to use to get to network 160.80. Whenever a packet comes in with a destination IP address of the form 160.80.xxx.yyy, it goes out on that line. If all of a sudden, the machine with that address is carted off to some distant site, the packets for it will continue to be routed to its home LAN (or router).

At this stage, there are two options—both unattractive. The first is that we could create a route to a more specific prefix. That is, if the distant site advertises a route to 160.80.40.20/32, packets sent to the destination will start arriving in the right place again. This option depends on the longest matching prefix algorithm that is used at routers. However, we have added a route to an IP prefix with a single IP address in it. All ISPs in the world will learn about this prefix. If everyone changes global IP routes in this way when they move their computer, each router would have millions of table entries, at astronomical cost to the Internet. This option is not workable.

The second option is to change the IP address of the mobile. True, packets sent to the home IP address will no longer be delivered until all the relevant people, programs, and databases are informed of the change. But the mobile can still use the Internet at the new location to browse the Web and run other applications. This option handles mobility at a higher layer. It is what typically happens when a user takes a laptop to a coffee store and uses the Internet via the local wireless network. The disadvantage is that it breaks some applications, and it does not keep connectivity as the mobile moves around.

As an aside, mobility can also be handled at a lower layer, the link layer. This is what happens when using a laptop on a single 802.11 wireless network. The IP address of the mobile does not change and the network path remains the same. It is the wireless link that is providing mobility. However, the degree of mobility is limited. If the laptop moves too far, it will have to connect to the Internet via another network with a different IP address.

The mobile IP solution for IPv4 is given in RFC 3344. It works with the existing Internet routing and allows hosts to stay connected with their own IP addresses as they move about. For it to work, the mobile must be able to discover when it has moved. This is accomplished with ICMP router advertisement and solicitation messages. Mobiles listen for periodic router advertisements or send a solicitation to discover the nearest router. If this router is not the usual address of the router when the mobile is at home, it must be on a foreign network. If this router has changed since last time, the mobile has moved to another foreign network. This same mechanism lets mobile hosts find their home agents.

To get a care-of IP address on the foreign network, a mobile can simply use DHCP. Alternatively, if IPv4 addresses are in short supply, the mobile can send and receive packets via a foreign agent that already has an IP address on the network. The mobile host finds a foreign agent using the same ICMP mechanism used to find the home agent. After the mobile obtains an IP address or finds a foreign agent, it is able to use the network to send a message to its home agent, informing the home agent of its current location.

The home agent needs a way to intercept packets sent to the mobile only when the mobile is not at home. ARP provides a convenient mechanism. To send a packet over an Ethernet to an IP host, the router needs to know the Ethernet address of the host. The usual mechanism is for the router to send an ARP query to ask, for example, what is the Ethernet address of 160.80.40.20. When the mobile is at home, it answers ARP queries for its IP address with its own Ethernet address. When the mobile is away, the home agent responds to this query by giving its Ethernet address. The router then sends packets for 160.80.40.20 to the home agent. Recall that this is called a proxy ARP.

To quickly update ARP mappings back and forth when the mobile leaves home or arrives back home, another ARP technique called a **gratuitous ARP** can be used. Basically, the mobile or home agent send themselves an ARP query for the mobile IP address that supplies the right answer so that the router notices and updates its mapping.

Tunneling to send a packet between the home agent and the mobile host at the care-of address is done by encapsulating the packet with another IP header destined for the care-of address. When the encapsulated packet arrives at the care-of address, the outer IP header is removed to reveal the packet.

As with many Internet protocols, the devil is in the details, and most often the details of compatibility with other protocols that are deployed. There are two complications. First, NAT boxes depend on peeking past the IP header to look at the TCP or UDP header. The original form of tunneling for mobile IP did not use these headers, so it did not work with NAT boxes. The solution was to change the encapsulation to include a UDP header.

The second complication is that some ISPs check the source IP addresses of packets to see that they match where the routing protocol believes the source should be located. This technique is called **ingress filtering**, and it is a security

measure intended to discard traffic with seemingly incorrect addresses that may be malicious. However, packets sent from the mobile to other Internet hosts when it is on a foreign network will have a source IP address that is out of place, so they will be discarded. To get around this problem, the mobile can use the care-of address as a source to tunnel the packets back to the home agent. From here, they are sent into the Internet from what appears to be the right location. The cost is that the route is more roundabout.

Another issue we have not discussed is security. When a home agent gets a message asking it to please forward all of Roberta's packets to some IP address, it had better not comply unless it is convinced that Roberta is the source of this request, and not somebody trying to impersonate her. Cryptographic authentication protocols are used for this purpose. We will study such protocols in Chap. 8.

Mobility protocols for IPv6 build on the IPv4 foundation. The scheme above suffers from the triangle routing problem in which packets sent to the mobile take a dogleg through a distant home agent. In IPv6, route optimization is used to follow a direct path between the mobile and other IP addresses after the initial packets have followed the long route. Mobile IPv6 is defined in RFC 3775.

There is another kind of mobility that is also being defined for the Internet. Some airplanes have built-in wireless networking that passengers can use to connect their laptops to the Internet. The plane has a router that connects to the rest of the Internet via a wireless link. (Did you expect a wired link?) So now we have a flying router, which means that the whole network is mobile. Network mobility designs support this situation without the laptops realizing that the plane is mobile. As far as they are concerned, it is just another network. Of course, some of the laptops may be using mobile IP to keep their home addresses while they are on the plane, so we have two levels of mobility. Network mobility is defined for IPv6 in RFC 3963.

5.7 SUMMARY

The network layer provides services to the transport layer. It can be based on either datagrams or virtual circuits. In both cases, its main job is routing packets from the source to the destination. In datagram networks, a routing decision is made on every packet. In virtual-circuit networks, it is made when the virtual circuit is set up.

Many routing algorithms are used in computer networks. Flooding is a simple algorithm to send a packet along all paths. Most algorithms find the shortest path and adapt to changes in the network topology. The main algorithms are distance vector routing and link state routing. Most actual networks use one of these. Other important routing topics are the use of hierarchy in large networks, routing for mobile hosts, and broadcast, multicast, and anycast routing.

Networks can easily become congested, leading to increased delay and lost packets. Network designers attempt to avoid congestion by designing the network to have enough capacity, choosing uncongested routes, refusing to accept more traffic, signaling sources to slow down, and shedding load.

The next step beyond just dealing with congestion is to actually try to achieve a promised quality of service. Some applications care more about throughput whereas others care more about delay and jitter. The methods that can be used to provide different qualities of service include a combination of traffic shaping, reserving resources at routers, and admission control. Approaches that have been designed for good quality of service include IETF integrated services (including RSVP) and differentiated services.

Networks differ in various ways, so when multiple networks are interconnected, problems can occur. When different networks have different maximum packet sizes, fragmentation may be needed. Different networks may run different routing protocols internally but need to run a common protocol externally. Sometimes the problems can be finessed by tunneling a packet through a hostile network, but if the source and destination networks are different, this approach fails.

The Internet has a rich variety of protocols related to the network layer. These include the datagram protocol, IP, and associated control protocols such as ICMP, ARP, and DHCP. A connection-oriented protocol called MPLS carries IP packets across some networks. One of the main routing protocols used within networks is OSPF, and the routing protocol used across networks is BGP. The Internet is rapidly running out of IP addresses, so a new version of IP, IPv6, has been developed and is ever-so-slowly being deployed.

PROBLEMS

1. Give two example computer applications for which connection-oriented service is appropriate. Now give two examples for which connectionless service is best.
2. Datagram networks route each packet as a separate unit, independent of all others. Virtual-circuit networks do not have to do this, since each data packet follows a predetermined route. Does this observation mean that virtual-circuit networks do not need the capability to route isolated packets from an arbitrary source to an arbitrary destination? Explain your answer.
3. Give three examples of protocol parameters that might be negotiated when a connection is set up.
4. Assuming that all routers and hosts are working properly and that all software in both is free of all errors, is there any chance, however small, that a packet will be delivered to the wrong destination?

5. Give a simple heuristic for finding two paths through a network from a given source to a given destination that can survive the loss of any communication line (assuming two such paths exist). The routers are considered reliable enough, so it is not necessary to worry about the possibility of router crashes.

6. Consider the network of Fig. 5-12(a). Distance vector routing is used, and the following vectors have just come in to router *C*: from *B*: (5, 0, 8, 12, 6, 2); from *D*: (16, 12, 6, 0, 9, 10); and from *E*: (7, 6, 3, 9, 0, 4). The cost of the links from *C* to *B*, *D*, and *E*, are 6, 3, and 5, respectively. What is *C*'s new routing table? Give both the outgoing line to use and the cost.

7. If costs are recorded as 8-bit numbers in a 50-router network, and distance vectors are exchanged twice a second, how much bandwidth per (full-duplex) line is chewed up by the distributed routing algorithm? Assume that each router has three lines to other routers.

8. In Fig. 5-13 the Boolean OR of the two sets of ACF bits are 111 in every row. Is this just an accident here, or does it hold for all networks under all circumstances?

9. For hierarchical routing with 4800 routers, what region and cluster sizes should be chosen to minimize the size of the routing table for a three-layer hierarchy? A good starting place is the hypothesis that a solution with *k* clusters of *k* regions of *k* routers is close to optimal, which means that *k* is about the cube root of 4800 (around 16). Use trial and error to check out combinations where all three parameters are in the general vicinity of 16.

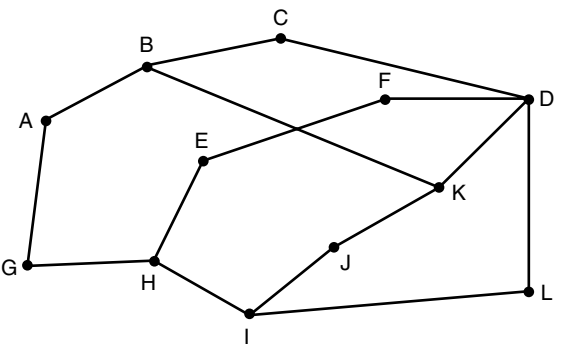
10. In the text it was stated that when a mobile host is not at home, packets sent to its home LAN are intercepted by its home agent on that LAN. For an IP network on an 802.3 LAN, how does the home agent accomplish this interception?

11. Looking at the network of Fig. 5-6, how many packets are generated by a broadcast from *B*, using

(a) reverse path forwarding?
(b) the sink tree?

12. Consider the network of Fig. 5-15(a). Imagine that one new line is added, between *F* and *G*, but the sink tree of Fig. 5-15(b) remains unchanged. What changes occur to Fig. 5-15(c)?

13. Compute a multicast spanning tree for router *C* in the following network for a group with members at routers *A*, *B*, *C*, *D*, *E*, *F*, *I*, and *K*.



14. Suppose that node *B* in Fig. 5-20 has just rebooted and has no routing information in its tables. It suddenly needs a route to *H*. It sends out broadcasts with *TtL* set to 1, 2, 3, and so on. How many rounds does it take to find a route?

15. As a possible congestion control mechanism in a network using virtual circuits internally, a router could refrain from acknowledging a received packet until (1) it knows its last transmission along the virtual circuit was received successfully and (2) it has a free buffer. For simplicity, assume that the routers use a stop-and-wait protocol and that each virtual circuit has one buffer dedicated to it for each direction of traffic. If it takes *T* sec to transmit a packet (data or acknowledgement) and there are *n* routers on the path, what is the rate at which packets are delivered to the destination host? Assume that transmission errors are rare and that the host-router connection is infinitely fast.

16. A datagram network allows routers to drop packets whenever they need to. The probability of a router discarding a packet is *p*. Consider the case of a source host connected to the source router, which is connected to the destination router, and then to the destination host. If either of the routers discards a packet, the source host eventually times out and tries again. If both host-router and router-router lines are counted as hops, what is the mean number of

(a) hops a packet makes per transmission?
(b) transmissions a packet makes?
(c) hops required per received packet?

17. Describe two major differences between the ECN method and the RED method of congestion avoidance.

18. A token bucket scheme is used for traffic shaping. A new token is put into the bucket every 5 μsec. Each token is good for one short packet, which contains 48 bytes of data. What is the maximum sustainable data rate?

19. A computer on a 6-Mbps network is regulated by a token bucket. The token bucket is filled at a rate of 1 Mbps. It is initially filled to capacity with 8 megabits. How long can the computer transmit at the full 6 Mbps?

20. The network of Fig. 5-34 uses RSVP with multicast trees for hosts 1 and 2 as shown. Suppose that host 3 requests a channel of bandwidth 2 MB/sec for a flow from host 1 and another channel of bandwidth 1 MB/sec for a flow from host 2. At the same time, host 4 requests a channel of bandwidth 2 MB/sec for a flow from host 1 and host 5 requests a channel of bandwidth 1 MB/sec for a flow from host 2. How much total bandwidth will be reserved for these requests at routers *A*, *B*, *C*, *E*, *H*, *J*, *K*, and *L*?

21. A router can process 2 million packets/sec. The load offered to it is 1.5 million packets/sec on average. If a route from source to destination contains 10 routers, how much time is spent being queued and serviced by the router?

22. Consider the user of differentiated services with expedited forwarding. Is there a guarantee that expedited packets experience a shorter delay than regular packets? Why or why not?

Patent Owner's Ex. 2010
Page 339

23. Suppose that host *A* is connected to a router *R*1, *R*1 is connected to another router, *R*2, and *R*2 is connected to host *B*. Suppose that a TCP message that contains 900 bytes of data and 20 bytes of TCP header is passed to the IP code at host *A* for delivery to *B*. Show the *Total length, Identification*, *DF*, *MF*, and *Fragment offset* fields of the IP header in each packet transmitted over the three links. Assume that link *A-R1* can support a maximum frame size of 1024 bytes including a 14-byte frame header, link *R1-R2* can support a maximum frame size of 512 bytes, including an 8-byte frame header, and link *R2-B* can support a maximum frame size of 512 bytes including a 12-byte frame header.

24. A router is blasting out IP packets whose total length (data plus header) is 1024 bytes. Assuming that packets live for 10 sec, what is the maximum line speed the router can operate at without danger of cycling through the IP datagram ID number space?

25. An IP datagram using the *Strict source routing* option has to be fragmented. Do you think the option is copied into each fragment, or is it sufficient to just put it in the first fragment? Explain your answer.

26. Suppose that instead of using 16 bits for the network part of a class B address originally, 20 bits had been used. How many class B networks would there have been?

27. Convert the IP address whose hexadecimal representation is C22F1582 to dotted decimal notation.

28. A network on the Internet has a subnet mask of 255.255.240.0. What is the maximum number of hosts it can handle?

29. While IP addresses are tried to specific networks, Ethernet addresses are not. Can you think of a good reason why they are not?

30. A large number of consecutive IP addresses are available starting at 198.16.0.0. Suppose that four organizations, *A*, *B*, *C*, and *D*, request 4000, 2000, 4000, and 8000 addresses, respectively, and in that order. For each of these, give the first IP address assigned, the last IP address assigned, and the mask in the *w.x.y.z/s* notation.

31. A router has just received the following new IP addresses: 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21, and 57.6.120.0/21. If all of them use the same outgoing line, can they be aggregated? If so, to what? If not, why not?

32. The set of IP addresses from 29.18.0.0 to 19.18.128.255 has been aggregated to 29.18.0.0/17. However, there is a gap of 1024 unassigned addresses from 29.18.60.0 to 29.18.63.255 that are now suddenly assigned to a host using a different outgoing line. Is it now necessary to split up the aggregate address into its constituent blocks, add the new block to the table, and then see if any reaggregation is possible? If not, what can be done instead?

33. A router has the following (CIDR) entries in its routing table:

Address/mask	**Next hop**
135.46.56.0/22	Interface 0
135.46.60.0/22	Interface 1
192.53.40.0/23	Router 1
default	Router 2

For each of the following IP addresses, what does the router do if a packet with that address arrives?

(a) 135.46.63.10
(b) 135.46.57.14
(c) 135.46.52.2
(d) 192.53.40.7
(e) 192.53.56.7

34. Many companies have a policy of having two (or more) routers connecting the company to the Internet to provide some redundancy in case one of them goes down. Is this policy still possible with NAT? Explain your answer.

35. You have just explained the ARP protocol to a friend. When you are all done, he says: "I've got it. ARP provides a service to the network layer, so it is part of the data link layer." What do you say to him?

36. Describe a way to reassemble IP fragments at the destination.

37. Most IP datagram reassembly algorithms have a timer to avoid having a lost fragment tie up reassembly buffers forever. Suppose that a datagram is fragmented into four fragments. The first three fragments arrive, but the last one is delayed. Eventually, the timer goes off and the three fragments in the receiver's memory are discarded. A little later, the last fragment stumbles in. What should be done with it?

38. In IP, the checksum covers only the header and not the data. Why do you suppose this design was chosen?

39. A person who lives in Boston travels to Minneapolis, taking her portable computer with her. To her surprise, the LAN at her destination in Minneapolis is a wireless IP LAN, so she does not have to plug in. Is it still necessary to go through the entire business with home agents and foreign agents to make email and other traffic arrive correctly?

40. IPv6 uses 16-byte addresses. If a block of 1 million addresses is allocated every picosecond, how long will the addresses last?

41. The *Protocol* field used in the IPv4 header is not present in the fixed IPv6 header. Why not?

42. When the IPv6 protocol is introduced, does the ARP protocol have to be changed? If so, are the changes conceptual or technical?

43. Write a program to simulate routing using flooding. Each packet should contain a counter that is decremented on each hop. When the counter gets to zero, the packet is discarded. Time is discrete, with each line handling one packet per time interval. Make three versions of the program: all lines are flooded, all lines except the input line are flooded, and only the (statically chosen) best k lines are flooded. Compare flooding with deterministic routing ($k = 1$) in terms of both delay and the bandwidth used.

44. Write a program that simulates a computer network using discrete time. The first packet on each router queue makes one hop per time interval. Each router has only a finite number of buffers. If a packet arrives and there is no room for it, it is discarded

and not retransmitted. Instead, there is an end-to-end protocol, complete with time-outs and acknowledgement packets, that eventually regenerates the packet from the source router. Plot the throughput of the network as a function of the end-to-end time-out interval, parameterized by error rate.

45. Write a function to do forwarding in an IP router. The procedure has one parameter, an IP address. It also has access to a global table consisting of an array of triples. Each triple contains three integers: an IP address, a subnet mask, and the outline line to use. The function looks up the IP address in the table using CIDR and returns the line to use as its value.

46. Use the *traceroute* (UNIX) or *tracert* (Windows) programs to trace the route from your computer to various universities on other continents. Make a list of transoceanic links you have discovered. Some sites to try are

www.berkeley.edu (California)
www.mit.edu (Massachusetts)
www.vu.nl (Amsterdam)
www.ucl.ac.uk (London)
www.usyd.edu.au (Sydney)
www.u-tokyo.ac.jp (Tokyo)
www.uct.ac.za (Cape Town)