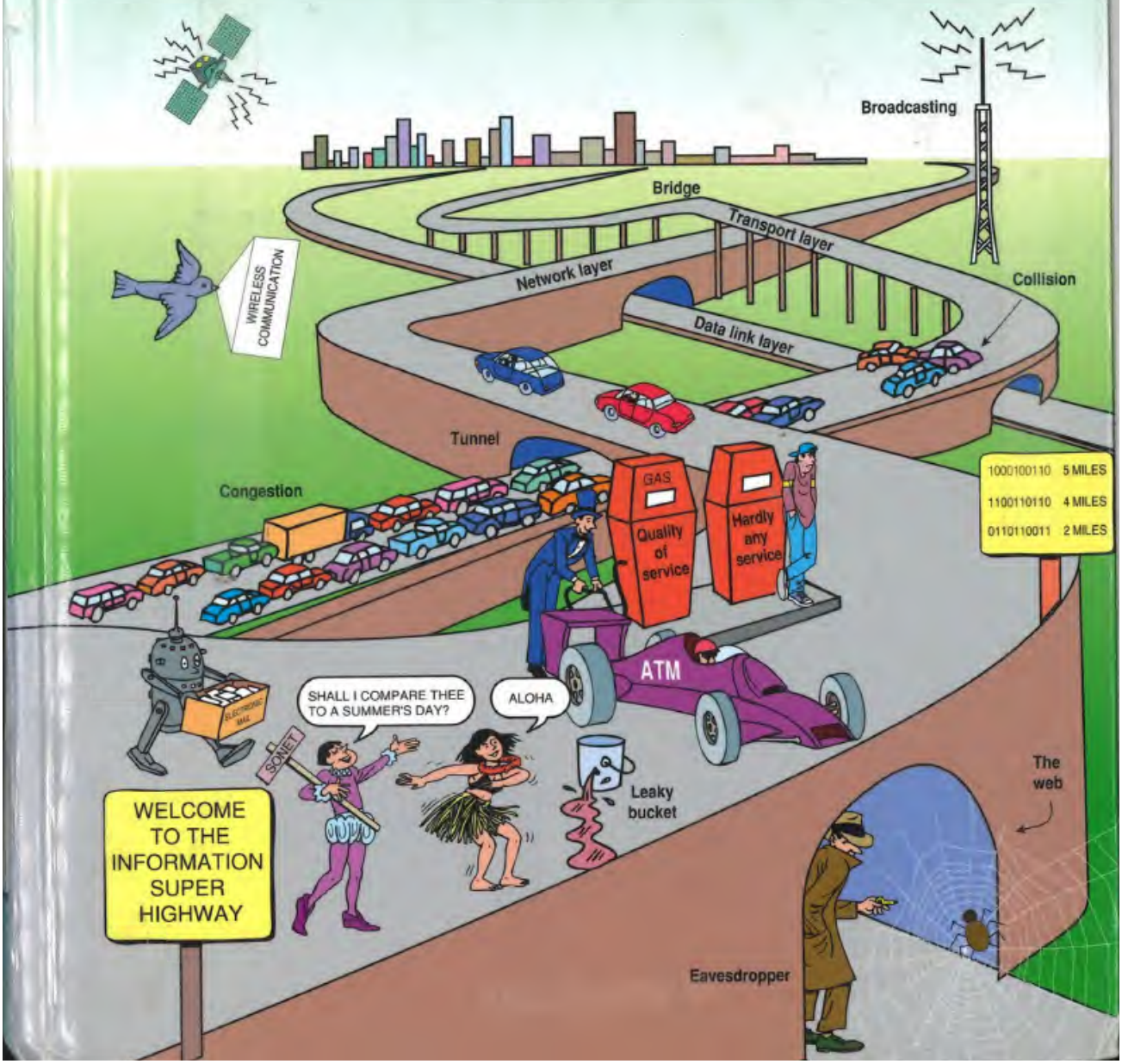
Kirkland & Ellis LLP Library
5 0628 01259222 3

THIRD EDITION

COMPUTER NETWORKS

ANDREW S. TANENBAUM



RED HAT 1057
RED HAT V. CAS

Computer Networks

Third Edition

RED HAT 1057
RED HAT V. CAS

Other bestselling titles by Andrew S. Tanenbaum:

Operating Systems: Design and Implementation, 2nd edition

This now-classic text on operating systems is the only book covering both the principles of operating systems and their application to a real system. All the traditional operating systems topics are covered in detail. In addition, the principles are carefully illustrated by MINIX, a free UNIX-like operating system for personal computers. The second edition, which is expected in late 1996, will deal with the new POSIX-based MINIX 2.0 system. As with the first edition, the book will include a listing of the MINIX source code. New to the second edition is a free CD-ROM in each book containing the complete MINIX system (binary and source code).

Modern Operating Systems

This bestselling text presents the basics of both single processor and distributed computer systems. Tanenbaum covers traditional topics including processes, memory management, and files systems as well as key issues in distributed systems including the client-server model, remote procedure call, threads and distributed files servers. This practical guide uses UNIX, MS-DOS, Mach and Amoeba to illustrate operating system concepts.

Distributed Operating Systems

This text covers the fundamental concepts of distributed operating systems. Key topics include communication and synchronization, processes and processors, distributed shared memory, distributed file systems, and distributed real-time systems. The principles of distributed computing are illustrated in four detailed case studies using Mach, Amoeba, Chorus and DCE operating environments.

Structured Computer Organization, 3rd edition

This leading text looks at computer architecture as a series of levels. At the bottom is the hardware: transistors, gates, registers, adders, and other circuits. Then comes the microprogramming level. After that is the conventional machine level, with its ADD, MOVE, JUMP, and other instructions. On top of that is the operating system, which adds new facilities such as file management and virtual memory. The final chapter deals with two advanced topics: parallel computers and the design of RISC machines.

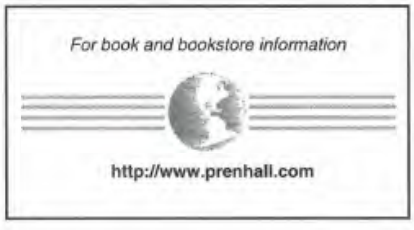
RED HAT 1057
RED HAT V. CAS

Computer Networks

Third Edition

Andrew S. Tanenbaum

Vrije Universiteit
Amsterdam, The Netherlands





Prentice Hall PTR
Upper Saddle River, New Jersey 07458

RED HAT 1057
RED HAT V. CAS

Library of Congress Cataloging in Publication Data

Tanenbaum, Andrew S. 1944-.
Computer networks / Andrew S. Tanenbaum. -- 3rd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-349945-6
1.Computer networks. I. Title.
TK5105.5.T36 1996 96-4121
004.6--dc20 CIP

Editorial/production manager: *Camille Trentacoste*
Interior design and composition: *Andrew S. Tanenbaum*
Cover design director: *Jerry Votta*
Cover designer: *Don Martinetti, DM Graphics, Inc.*
Cover concept: *Andrew S. Tanenbaum, from an idea by Marilyn Tremaine*
Interior graphics: *Hadel Studio*
Manufacturing manager: *Alexis R. Heydt*
Acquisitions editor: *Mary Franz*
Editorial Assistant: *Noreen Regina*



© 1996 by Prentice Hall PTR
Prentice-Hall, Inc.
A Simon & Schuster Company
Upper Saddle River, New Jersey 07458

The publisher offers discounts on this book when ordered in bulk quantities. For more information, contact:
Corporate Sales Department, Prentice Hall PTR, One Lake Street, Upper Saddle River, NJ 07458.
Phone: (800) 382-3419; Fax: (201) 236-7141. E-mail: corpsales@prenhall.com

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

All product names mentioned herein are the trademarks of their respective owners.

Printed in the United States of America
10 9 8

ISBN 0-13-349945-6

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

RED HAT 1057
RED HAT V. CAS

To Suzanne, Barbara, Marvin, and Little Bram

RED HAT 1057
RED HAT V. CAS

CONTENTS

PREFACE xv

1 INTRODUCTION 1

1.1 USES OF COMPUTER NETWORKS 3
- 1.1.1 Networks for Companies 3
- 1.1.2 Networks for People 4
- 1.1.3 Social Issues 6

1.2 NETWORK HARDWARE 7
- 1.2.1 Local Area Networks 9
- 1.2.2 Metropolitan Area Networks 10
- 1.2.3 Wide Area Networks 11
- 1.2.4 Wireless Networks 13
- 1.2.5 Internetworks 16

1.3 NETWORK SOFTWARE 16
- 1.3.1 Protocol Hierarchies 17
- 1.3.2 Design Issues for the Layers 21
- 1.3.3 Interfaces and Services 22
- 1.3.4 Connection-Oriented and Connectionless Services 23
- 1.3.5 Service Primitives 25
- 1.3.6 The Relationship of Services to Protocols 27

1.4 REFERENCE MODELS 28
- 1.4.1 The OSI Reference Model 28
- 1.4.2 The TCP/IP Reference Model 35
- 1.4.3 A Comparison of the OSI and TCP Reference Models 38
- 1.4.4 A Critique of the OSI Model and Protocols 40
- 1.4.5 A Critique of the TCP/IP Reference Model 43

1.5 EXAMPLE NETWORKS 44
- 1.5.1 Novell NetWare 45
- 1.5.2 The ARPANET 47
- 1.5.3 NSFNET 50
- 1.5.4 The Internet 52
- 1.5.5 Gigabit Testbeds 54

RED HAT 1057
RED HAT V. CAS

1.6 EXAMPLE DATA COMMUNICATION SERVICES 56
1.6.1 SMDS—Switched Multimegabit Data Service 57
1.6.2 X.25 Networks 59
1.6.3 Frame Relay 60
1.6.4 Broadband ISDN and ATM 61
1.6.5 Comparison of Services 66

1.7 NETWORK STANDARDIZATION 66
1.7.1 Who's Who in the Telecommunications World 67
1.7.2 Who's Who in the International Standards World 69
1.7.3 Who's Who in the Internet Standards World 70

1.8 OUTLINE OF THE REST OF THE BOOK 72

1.9. SUMMARY 73

2 THE PHYSICAL LAYER 77

2.1 THE THEORETICAL BASIS FOR DATA COMMUNICATION 77
2.1.1 Fourier Analysis 78
2.1.2 Bandwidth-Limited Signals 78
2.1.3 The Maximum Data Rate of a Channel 81

2.2 TRANSMISSION MEDIA 82
2.2.1 Magnetic Media 82
2.2.2 Twisted Pair 83
2.2.3 Baseband Coaxial Cable 84
2.2.4 Broadband Coaxial Cable 85
2.2.5 Fiber Optics 87

2.3 WIRELESS TRANSMISSION 94
2.3.1 The Electromagnetic Spectrum 94
2.3.2 Radio Transmission 97
2.3.3 Microwave Transmission 98
2.3.4 Infrared and Millimeter Waves 100
2.3.5 Lightwave Transmission 100

2.4 THE TELEPHONE SYSTEM 102
2.4.1 Structure of the Telephone System 103
2.4.2 The Politics of Telephones 106
2.4.3 The Local Loop 108
2.4.4 Trunks and Multiplexing 118
2.4.5 Switching 130

RED HAT 1057
RED HAT V. CAS

2.5 NARROWBAND ISDN 139
2.5.1 ISDN Services 140
2.5.2 ISDN System Architecture 140
2.5.3 The ISDN Interface 142
2.5.4 Perspective on N-ISDN 143

2.6 BROADBAND ISDN AND ATM 144
2.6.1 Virtual Circuits versus Circuit Switching 145
2.6.2 Transmission in ATM Networks 146
2.6.3 ATM Switches 147

2.7 CELLULAR RADIO 155
2.7.1 Paging Systems 155
2.7.2 Cordless Telephones 157
2.7.3 Analog Cellular Telephones 157
2.7.4 Digital Cellular Telephones 162
2.7.5 Personal Communications Services 162

2.8 COMMUNICATION SATELLITES 163
2.8.1 Geosynchronous Satellites 164
2.8.2 Low-Orbit Satellites 167
2.8.3 Satellites versus Fiber 168

2.9 SUMMARY 170

3 THE DATA LINK LAYER 175

3.1 DATA LINK LAYER DESIGN ISSUES 176
3.1.1 Services Provided to the Network Layer 176
3.1.2 Framing 179
3.1.3 Error Control 182
3.1.4 Flow Control 183

3.2 ERROR DETECTION AND CORRECTION 183
3.2.1 Error-Correcting Codes 184
3.2.2 Error-Detecting Codes 186

3.3 ELEMENTARY DATA LINK PROTOCOLS 190
3.3.1 An Unrestricted Simplex Protocol 195
3.3.2 A Simplex Stop-and-Wait Protocol 195
3.3.3 A Simplex Protocol for a Noisy Channel 197

RED HAT 1057
RED HAT V. CAS

3.4 SLIDING WINDOW PROTOCOLS 202
3.4.1 A One Bit Sliding Window Protocol 206
3.4.2 A Protocol Using Go Back n 207
3.4.3 A Protocol Using Selective Repeat 213

3.5 PROTOCOL SPECIFICATION AND VERIFICATION 219
3.5.1 Finite State Machine Models 219
3.5.2 Petri Net Models 223

3.6 EXAMPLE DATA LINK PROTOCOLS 225
3.6.1 HDLC—High-level Data Link Control 225
3.6.2 The Data Link Layer in the Internet 229
3.6.3 The Data Link Layer in ATM 235

3.7. SUMMARY 239

4 THE MEDIUM ACCESS SUBLAYER 243

4.1 THE CHANNEL ALLOCATION PROBLEM 244
4.1.1 Static Channel Allocation in LANs and MANs 244
4.1.2 Dynamic Channel Allocation in LANs and MANs 245

4.2 MULTIPLE ACCESS PROTOCOLS 246
4.2.1 ALOHA 246
4.2.2 Carrier Sense Multiple Access Protocols 250
4.2.3 Collision-Free Protocols 254
4.2.4 Limited-Contention Protocols 256
4.2.5 Wavelength Division Multiple Access Protocols 260
4.2.6 Wireless LAN Protocols 262
4.2.7 Digital Cellular Radio 266

4.3 IEEE STANDARD 802 FOR LANS AND MANS 275
4.3.1 IEEE Standard 802.3 and Ethernet 276
4.3.2 IEEE Standard 802.4: Token Bus 287
4.3.3 IEEE Standard 802.5: Token Ring 292
4.3.4 Comparison of 802.3, 802.4, and 802.5 299
4.3.5 IEEE Standard 802.6: Distributed Queue Dual Bus 301
4.3.6 IEEE Standard 802.2: Logical Link Control 302

RED HAT 1057
RED HAT V. CAS

4.4 BRIDGES 304
4.4.1 Bridges from 802.x to 802.y 307
4.4.2 Transparent Bridges 310
4.4.3 Source Routing Bridges 314
4.4.4 Comparison of 802 Bridges 316
4.4.5 Remote Bridges 317

4.5 HIGH-SPEED LANS 318
4.5.1 FDDI 319
4.5.2 Fast Ethernet 322
4.5.3 HIPPI—High-Performance Parallel Interface 325
4.5.4 Fibre Channel 326

4.6 SATELLITE NETWORKS 327
4.6.1 Polling 328
4.6.2 ALOHA 329
4.6.3 FDM 330
4.6.4 TDM 330
4.6.5 CDMA 333

4.7 SUMMARY 333

5 THE NETWORK LAYER 339

5.1 NETWORK LAYER DESIGN ISSUES 339
5.1.1 Services Provided to the Transport Layer 340
5.1.2 Internal Organization of the Network Layer 342
5.1.3 Comparison of Virtual Circuit and Datagram Subnets 344

5.2 ROUTING ALGORITHMS 345
5.2.1 The Optimality Principle 347
5.2.2 Shortest Path Routing 349
5.2.3 Flooding 351
5.2.4 Flow-Based Routing 353
5.2.5 Distance Vector Routing 355
5.2.6 Link State Routing 359
5.2.7 Hierarchical Routing 365
5.2.8 Routing for Mobile Hosts 367
5.2.9 Broadcast Routing 370
5.2.10 Multicast Routing 372

RED HAT 1057
RED HAT V. CAS

5.3 CONGESTION CONTROL ALGORITHMS 374
5.3.1 General Principles of Congestion Control 376
5.3.2 Congestion Prevention Policies 378
5.3.3 Traffic Shaping 379
5.3.4 Flow Specifications 384
5.3.5 Congestion Control in Virtual Circuit Subnets 386
5.3.6 Choke Packets 387
5.3.7 Load Shedding 390
5.3.8 Jitter Control 392
5.3.9 Congestion Control for Multicasting 393

5.4 INTERNETWORKING 396
5.4.1 How Networks Differ 399
5.4.2 Concatenated Virtual Circuits 401
5.4.3 Connectionless Internetworking 402
5.4.4 Tunneling 404
5.4.5 Internetwork Routing 405
5.4.6 Fragmentation 406
5.4.7 Firewalls 410

5.5 THE NETWORK LAYER IN THE INTERNET 412
5.5.1 The IP Protocol 413
5.5.2 IP Addresses 416
5.5.3 Subnets 417
5.5.4 Internet Control Protocols 419
5.5.5 The Interior Gateway Routing Protocol: OSPF 424
5.5.6 The Exterior Gateway Routing Protocol: BGP 429
5.5.7 Internet Multicasting 431
5.5.8 Mobile IP 432
5.5.9 CIDR—Classless InterDomain Routing 434
5.5.10 IPv6 437

5.6 THE NETWORK LAYER IN ATM NETWORKS 449
5.6.1 Cell Formats 450
5.6.2 Connection Setup 452
5.6.3 Routing and Switching 455
5.6.4 Service Categories 458
5.6.5 Quality of Service 460
5.6.6 Traffic Shaping and Policing 463
5.6.7 Congestion Control 467
5.6.8 ATM LANs 471

5.7 SUMMARY 473

RED HAT 1057
RED HAT V. CAS

6 THE TRANSPORT LAYER 479

6.1 THE TRANSPORT SERVICE 479
6.1.1 Services Provided to the Upper Layers 479
6.1.2 Quality of Service 481
6.1.3 Transport Service Primitives 483

6.2 ELEMENTS OF TRANSPORT PROTOCOLS 488
6.2.1 Addressing 489
6.2.2 Establishing a Connection 493
6.2.3 Releasing a Connection 498
6.2.4 Flow Control and Buffering 502
6.2.5 Multiplexing 506
6.2.6 Crash Recovery 508

6.3 A SIMPLE TRANSPORT PROTOCOL 510
6.3.1 The Example Service Primitives 510
6.3.2 The Example Transport Entity 512
6.3.3 The Example as a Finite State Machine 519

6.4 THE INTERNET TRANSPORT PROTOCOLS (TCP AND UDP) 521
6.4.1 The TCP Service Model 523
6.4.2 The TCP Protocol 524
6.4.3 The TCP Segment Header 526
6.4.4 TCP Connection Management 529
6.4.5 TCP Transmission Policy 533
6.4.6 TCP Congestion Control 536
6.4.7 TCP Timer Management 539
6.4.8 UDP 542
6.4.9 Wireless TCP and UDP 543

6.5 THE ATM AAL LAYER PROTOCOLS 545
6.5.1 Structure of the ATM Adaptation Layer 546
6.5.2 AAL 1 547
6.5.3 AAL 2 549
6.5.4 AAL 3/4 550
6.5.5 AAL 5 552
6.5.6 Comparison of AAL Protocols 554
6.5.7 SSCOP—Service Specific Connection-Oriented Protocol 555

6.6 PERFORMANCE ISSUES 555
6.6.1 Performance Problems in Computer Networks 556
6.6.2 Measuring Network Performance 559

RED HAT 1057
RED HAT V. CAS

6.6.3 System Design for Better Performance 561
6.6.4 Fast TPDU Processing 565
6.6.5 Protocols for Gigabit Networks 568

6.7 SUMMARY 572

7 THE APPLICATION LAYER 577

7.1 NETWORK SECURITY 577
7.1.1 Traditional Cryptography 580
7.1.2 Two Fundamental Cryptographic Principles 585
7.1.3 Secret-Key Algorithms 587
7.1.4 Public-Key Algorithms 597
7.1.5 Authentication Protocols 601
7.1.6 Digital Signatures 613
7.1.7 Social Issues 620

7.2 DNS—DOMAIN NAME SYSTEM 622
7.2.1 The DNS Name Space 622
7.2.2 Resource Records 624
7.2.3 Name Servers 628

7.3 SNMP—SIMPLE NETWORK MANAGEMENT PROTOCOL 630
7.3.1 The SNMP Model 631
7.3.2 ASN.1—Abstract Syntax Notation 1 633
7.3.3 SMI—Structure of Management Information 639
7.3.4 The MIB—Management Information Base 641
7.3.5 The SNMP Protocol 642

7.4 ELECTRONIC MAIL 643
7.4.1 Architecture and Services 645
7.4.2 The User Agent 646
7.4.3 Message Formats 650
7.4.4 Message Transfer 657
7.4.5 Email Privacy 663

7.5 USENET NEWS 669
7.5.1 The User View of USENET 670
7.5.2 How USENET is Implemented 675

RED HAT 1057
RED HAT V. CAS

7.6 THE WORLD WIDE WEB 681
7.6.1 The Client Side 682
7.6.2 The Server Side 685
7.6.3 Writing a Web Page in HTML 691
7.6.4 Java 706
7.6.5 Locating Information on the Web 720

7.7 MULTIMEDIA 723
7.7.1 Audio 724
7.7.2 Video 727
7.7.3 Data Compression 730
7.7.4 Video on Demand 744
7.7.5 MBone—Multicast Backbone 756

7.8 SUMMARY 760

8 READING LIST AND BIBLIOGRAPHY 767

8.1 SUGGESTIONS FOR FURTHER READING 767
8.1.1 Introduction and General Works 768
8.1.2 The Physical Layer 769
8.1.3 The Data Link Layer 770
8.1.4 The Medium Access Control Sublayer 770
8.1.5 The Network Layer 771
8.1.6 The Transport Layer 772
8.1.7 The Application Layer 772

8.2 ALPHABETICAL BIBLIOGRAPHY 775

INDEX 795

RED HAT 1057
RED HAT V. CAS

PREFACE

This book is now in its third edition. Each edition has corresponded to a different phase in the way computer networks were used. When the first edition appeared in 1980, networks were an academic curiosity. When the second edition appeared in 1988, networks were used by universities and large businesses. When the third edition appeared in 1996, computer networks, especially the worldwide Internet, had become a daily reality for millions of people.

Furthermore, the networking hardware and software have completely changed since the second edition appeared. In 1988, nearly all networks were based on copper wire. Now, many are based on fiber optics or wireless communication. Proprietary networks, such as SNA, have become far less important than public networks, especially the Internet. The OSI protocols have quietly vanished, and the TCP/IP protocol suite has become dominant. In fact, so much has changed, the book has almost been rewritten from scratch.

Although Chap. 1 has the same introductory function as it did in the second edition, the contents have been completely revised and brought up to date. For example, instead of basing the book on the seven-layer OSI model, a five-layer hybrid model (shown in Fig. 1-21) is now used and introduced in Chap. 1. While not exactly identical to the TCP/IP model, it is much closer to the TCP/IP model in spirit than it is to the OSI model used in the second edition. Also, the new running examples used throughout the book—the Internet and ATM networks— are introduced here, along with some gigabit networks and other popular networks.

RED HAT 1057
RED HAT V. CAS

In Chap. 2, the focus has moved from copper wire to fiber optics and wireless communication, since these are the technologies of the future. The telephone system has become almost entirely digital in the past decade, so the material on it has been largely rewritten, with new material on broadband ISDN added. The material on cellular radio has been greatly expanded, and new material on low-orbit satellites has been added to the chapter.

The order of discussion of the data link layer and the MAC sublayer has been reversed, since experience with students shows that they understand the MAC sublayer better after they have studied the data link layer. The example protocols there have been kept, as they have proven very popular, but they have been rewritten in C. New material on the Internet and ATM data link layers has been added.

The MAC sublayer principles of Chap. 4. have been revised to reflect new protocols, including wavelength division multiplexing, wireless LANs, and digital radio. The discussion of bridges has been revised, and new material has been added on high-speed LANs.

Most of the routing algorithms of Chap. 5 have been replaced by more modern ones, including distance vector and link state routing. The sections on congestion control have been completely redone, and material on the running examples, the Internet and ATM is all new.

Chap. 6 is still about the transport layer, but here, too, major changes have occurred, primarily, the addition of a large amount of new material about the Internet, ATM, and network performance.

Chap. 7, on the application layer, is now the longest chapter in the book. The material on network security has been doubled in length, and new material has been added on DNS, SNMP, email, USENET, the World Wide Web, HTML, Java, multimedia, video on demand, and the MBone.

Of the 395 figures in the third edition, 276 (70 percent) are completely new and some of the others have been revised. Of the 371 references to the literature, 282 (76 percent) are to books and papers that have appeared since the second edition was published. Of these, over 100 are to works published in 1995 and 1996 alone. All in all, probably 75 percent of the entire book is brand new, and parts of the remaining 25 percent have been heavily revised. Since this is effectively a new book, the cover was redesigned to avoid confusion with the second edition.

Computer books are full of acronyms. This one is no exception. By the time you are finished reading this one, all of the following should ring a bell: AAL, AMPS, ARP, ASN, ATM, BGP, CDMA, CDPD, CSMA, DQDB, DNS, FAQ, FDM, FTP, FTTC, FTTH, GSM, HDLC, HEC, HIPPI, IAB, ICMP, IDEA, IETF, IPv6, ISO, ITU, LATA, MAC, MACA, MAN, MIB, MIME, NAP, NNTP, NSA, NSAP, OSI, OSPF, PCM, PCN, PCS, PEM, PGP, PPP, PSTN, PTT, PVC, QAM, RARP, RFC, RSA, SABME, SAP, SAR, SDH, SDLC, SHA, SMI, SNA, SNMP, SNRME, SPX, TCP, UDP, VHF, VLF, VSAT, WARC, WDM, WWV, and WWW. But don't worry. Each one will be carefully defined before it is used.

RED HAT 1057
RED HAT V. CAS

To help instructors using this book as a text for course, the author has prepared three teaching aids:

- A problem solutions manual.
- PostScript files containing all the figures (for making overhead sheets).
- A simulator (written in C) for the example protocols of Chap. 3.

The solutions manual is available from Prentice Hall (but only to instructors). The file with the figures and the simulator are available via the World Wide Web. To get them, please see the author's home page: *http://www.cs.vu.nl/~ast/* .

The book was typeset in Times Roman using Troff, which, after all these years, is still the only way to go. While Troff is not as trendy as WYSIWYG systems, the reader is invited to compare the typesetting quality of this book with books produced by WYSIWYG systems. My only concession to PCs and desktop publishing is that for the first time, the art was produced using Adobe Illustrator, instead of being drawn on paper. Also for the first time, the book was produced entirely electronically. The PostScript output from Troff was sent over the Internet to the printer, where the film for making the offset plates was produced. No intermediate paper copy was printed and photographed, as is normally done.

Many people helped me during the course of the third edition. I would especially like to thank Chase Bailey, Saniya Ben Hassen, Nathaniel Borenstein, Ron Cocchi, Dave Crocker, Wiebren de Jonge, Carl Ellison, M. Rasit Eskicioglu, John Evans, Mario Gerla, Mike Goguen, Paul Green, Dick Grune, Wayne Hathaway, Franz Hauck, Jack Holtzman, Gerard Holzmann, Philip Homburg, Peter Honeyman, Raj Jain, Dave Johnson, Charlie Kaufman, Vinay Kumar, Jorg Liebeherr, Paul Mockapetris, Carol Orange, Craig Partridge, Charlie Perkins, Thomas Powell, Greg Sharp, Anne Steegstra, George Swallow, Mark Taylor, Peter van der Linden, Hans van Staveren, Maarten van Steen, Kees Verstoep, Stephen Walters, Michael Weintraub, Joseph Wilkes, and Stephen Wolff. Special thanks go to Radia Perlman for many helpful suggestions. My students have also helped in many ways. I would like to single out Martijn Bot, Wilbert de Graaf, Flavio del Pomo, and Arnold de Wit for their assistance.

My editor at Prentice Hall, Mary Franz, provided me with more reading material than I had consumed in the previous 10 years. She was also helpful in numerous other ways, small, medium, large, and jumbo. My production editor, Camille Trentacoste, taught me about people of snow, 8-up flats, fax [sic], and other important items, while performing yeoperson's service with a Picky Author and a tight schedule.

Finally, we come to the most important people. Suzanne, Barbara, Marvin, and even little Bram, have been through this routine before. They endure it with infinite patience and good grace. Thank you.

ANDREW S. TANENBAUM

RED HAT 1057
RED HAT V. CAS

1

INTRODUCTION

Each of the past three centuries has been dominated by a single technology. The 18th Century was the time 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 has been information gathering, processing, and distribution. Among other developments, we have seen the installation of worldwide telephone networks, the invention of radio and television, the birth and unprecedented growth of the computer industry, and the launching of communication satellites.

Due to rapid technological progress, these areas are rapidly converging, 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 even more sophisticated information processing grows even faster.

Although the computer industry is 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-size company or university might have had one or two

RED HAT 1057
RED HAT V. CAS

computers, while large institutions had at most a few dozen. The idea that within 20 years equally powerful computers smaller than postage stamps would be mass produced by the millions was pure science fiction.

The merging of computers and communications has had a profound influence on the way computer systems are organized. The concept of the "computer center" as a room with a large computer to which users bring their work for processing is now totally obsolete. 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 an *interconnected* collection of *autonomous* computers. 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, and communication satellites can also be used. By requiring the computers to be autonomous, we wish to exclude from our definition systems in which there is a clear master/slave relation. If one computer can forcibly start, stop, or control another one, the computers are not autonomous. A system with one control unit and many slaves is not a network; nor is a large computer with remote printers and terminals.

There is considerable confusion in the literature between a computer network and a **distributed system**. The key distinction is that in a distributed system, the existence of multiple autonomous computers is transparent (i.e., not visible) to the user. He† can type a command to run a program, and it runs. It is up to the operating system to select the best processor, find and transport all the input files to that processor, and put the results in the appropriate place.

In other words, the user of a distributed system is not aware that there are multiple processors; it looks like a virtual uniprocessor. Allocation of jobs to processors and files to disks, movement of files between where they are stored and where they are needed, and all other system functions must be automatic.

With a network, users must *explicitly* log onto one machine, *explicitly* submit jobs remotely, *explicitly* move files around and generally handle all the network management personally. With a distributed system, nothing has to be done explicitly; it is all automatically done by the system without the users' knowledge.

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.

RED HAT 1057
RED HAT V. CAS

Although this book primarily focuses on networks, many of the topics are also important in distributed systems. For more information about distributed systems, see (Coulouris et al., 1994; Mullender, 1993; and Tanenbaum, 1995).

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.

1.1.1. Networks for Companies

Many organizations have a substantial number of computers in operation, often located far apart. For example, a company with many factories may have a computer at each location to keep track of inventories, monitor productivity, and do the local payroll. Initially, each 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 extract and correlate information about the entire company.

Put in slightly more general form, the issue here is **resource sharing**, and 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 and the user. In other words, the mere fact that a user happens to be 1000 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."

A second goal is to provide **high reliability** by having alternative sources of supply. For example, all files could be replicated on two or three machines, so if one of them is unavailable (due to a hardware failure), the other copies could be used. In addition, the presence of multiple CPUs means that if one goes down, the others may be able to take over its work, although at reduced performance. For military, banking, air traffic control, nuclear reactor safety, and many other applications, the ability to continue operating in the face of hardware problems is of utmost importance.

Another goal is **saving money**. Small computers have a much better price/performance ratio than large ones. Mainframes (room-size computers) are roughly a factor of ten faster than personal computers, but they cost a thousand times more. This imbalance has caused many systems designers to build systems consisting of personal computers, one per user, with data kept on one or more shared **file server** machines. In this model, the users are called **clients**, and the whole arrangement is called the **client-server model**. It is illustrated in Fig. 1-1.

In the client-server model, communication generally takes the form of a request message from the client to the server asking for some work to be done.

RED HAT 1057
RED HAT V. CAS

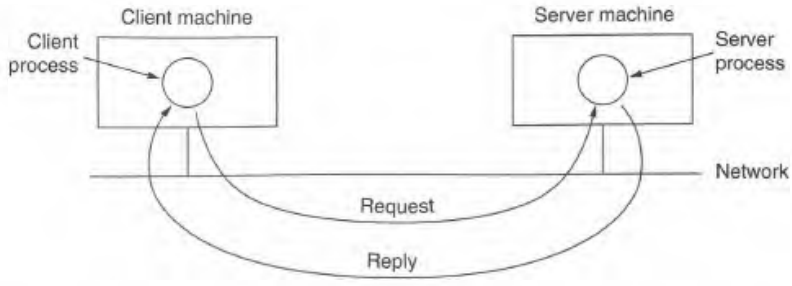


Fig. 1-1. The client-server model.

The server then does the work and sends back the reply. Usually, there are many clients using a small number of servers.

Another networking goal is scalability, the ability to increase system performance gradually as the workload grows just by adding more processors. With centralized mainframes, when the system is full, it must be replaced by a larger one, usually at great expense and even greater disruption to the users. With the client-server model, new clients and new servers can be added as needed.

Yet another goal of setting up a computer network has little to do with technology at all. A computer network can provide a powerful **communication medium** among widely separated employees. Using a network, it is easy for two or more people who live far apart to write a report together. When one worker makes a change to an on-line 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. In the long run, the use of networks to enhance human-to-human communication will probably prove more important than technical goals such as improved reliability.

1.1.2. Networks for People

The motivations given above for building computer networks are all essentially economic and technological in nature. If sufficiently large and powerful mainframes were available at acceptable prices, most companies would simply choose to keep all their data on them and give employees terminals connected to them. In the 1970s and early 1980s, most companies operated this way. Computer networks only became popular when networks of personal computers offered a huge price/performance advantage over mainframes.

Starting in the 1990s, computer networks began to start delivering services to private individuals at home. These services and the motivations for using them

RED HAT 1057
RED HAT V. CAS

are quite different than the "corporate efficiency" model described in the previous section. Below we will sketch three of the more exciting ones that are starting to happen:

1. Access to remote information.
2. Person-to-person communication.
3. Interactive entertainment.

Access to remote information will come in many forms. One area in which it is already happening is access to financial institutions. Many people pay their bills, manage their bank accounts, and handle their investments electronically. Home shopping is also becoming popular, with the ability to inspect the on-line catalogs of thousands of companies. Some of these catalogs will soon provide the ability to get an instant video on any product by just clicking on the product's name.

Newspapers will go on-line and be personalized. It will be possible to tell the newspaper that you want everything about corrupt politicians, big fires, scandals involving celebrities, and epidemics, but no football, thank you. At night while you sleep, the newspaper will be downloaded to your computer's disk or printed on your laser printer. On a small scale, this service already exists. The next step beyond newspapers (plus magazines and scientific journals) is the on-line digital library. Depending on the cost, size, and weight of book-sized notebook computers, printed books may become obsolete. Skeptics should take note of the effect the printing press had on the medieval illuminated manuscript.

Another application that falls in this category is access to information systems like the current World Wide Web, which contains information about the arts, business, cooking, government, health, history, hobbies, recreation, science, sports, travel, and too many other topics to even mention.

All of the above applications involve interactions between a person and a remote database. The second broad category of network use will be person-to-person interactions, basically the 21st Century's answer to the 19th Century's telephone. Electronic mail or **email** is already widely used by millions of people and will soon routinely contain audio and video as well as text. Smell in messages will take a bit longer to perfect.

Real-time email will allow remote users to communicate with no delay, possibly seeing and hearing each other as well. This technology makes it possible to have virtual meetings, called **videoconference**, among far-flung people. It is sometimes said that transportation and communication are having a race, and whichever wins will make the other obsolete. Virtual meetings could be used for remote school, getting medical opinions from distant specialists, and numerous other applications.

Worldwide newsgroups, with discussions on every conceivable topic are already commonplace among a select group of people, and this will grow to

RED HAT 1057
RED HAT V. CAS

include the population at large. These discussions, in which one person posts a message and all the other subscribers to the newsgroup can read it, run the gamut from humorous to impassioned.

Our third category is entertainment, which is a huge and growing industry. The killer application here (the one that may drive all the rest) is video on demand. A decade or so hence, it may be possible to select 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.

On the other hand, maybe the killer application will not be video on demand. Maybe it will be 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. If done with goggles and 3-dimensional real-time, photographic-quality moving images, we have a kind of worldwide shared virtual reality.

In short, the ability to merge information, communication, and entertainment will surely give rise to a massive new industry based on computer networking.

1.1.3. Social Issues

The widespread introduction of networking will introduce new social, ethical, political problems (Laudon, 1995). Let us just briefly mention a few of them; a thorough study would require a full book, at least. A popular feature of many networks are newsgroups or bulletin boards where people can exchange messages 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 when newsgroups are set up on topics that people actually care about, like politics, religion, or sex. Views posted to such groups may be deeply offensive to some people. Furthermore, messages need not be limited to text. High-resolution color photographs and even short video clips can now easily be transmitted over computer networks. Some people take a live-and-let-live view, but others feel that posting certain material (e.g., child pornography) is simply unacceptable. Thus the debate rages.

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. Stronger yet, having network operators censor messages would probably cause them to delete everything with even the slightest possibility of their being sued, and thus violate their users' rights to free speech. It is probably safe to say that this debate will go on for a while.

RED HAT 1057
RED HAT V. CAS

Another fun area is employee rights versus employer rights. Many people read and write email at work. Some employers have claimed the right to read and possibly censor employee messages, including messages sent from a home terminal after work. Not all employees agree with this (Sipior and Ward, 1995).

Even if employers have power over employees, does this relationship also govern universities and students? How about high schools and students? In 1994, Carnegie-Mellon University decided to turn off the incoming message stream for several newsgroups dealing with sex because the university felt the material was inappropriate for minors (i.e., those few students under 18). The fallout from this event will take years to settle.

Computer networks offer the potential for sending anonymous messages. In some situations, this capability may be desirable. For example, it provides 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. Anonymous accusations cannot be used as evidence.

In short, computer networks, like the printing press 500 years ago, allow ordinary citizens to distribute their views in different ways and to different audiences than were previously possible. This new-found freedom brings with it many unsolved social, political, and moral issues. The solution to these problems is left as an exercise for the reader.

1.2. NETWORK HARDWARE

It is now time to turn our attention from the applications and social aspects of networking to the technical issues involved in network design. 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:

1. Broadcast networks.
2. Point-to-point networks.

Broadcast networks have a single communication channel that is shared by all the machines on the network. Short messages, called **packets** in certain contexts, sent by any machine are received by all the others. An address field within the packet specifies for whom it is intended. Upon receiving a packet, a machine checks the address field. If the packet is intended for itself, it processes the packet; if the packet is intended for some other machine, it is just ignored.

As an analogy, consider someone standing at the end of a corridor with many rooms off it and shouting "Watson, come here. I want you." Although the packet

RED HAT 1057
RED HAT V. CAS

may actually be received (heard) by many people, only Watson responds. The others just ignore it. Another example is an airport announcement asking all flight 644 passengers to report to gate 12.

Broadcast systems generally 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, something known as **multicasting**. One possible scheme is to reserve one bit to indicate multicasting. The remaining $n - 1$ address bits can hold a group number. Each machine can "subscribe" to any or all of the groups. When a packet is sent to a certain group, it is delivered to all machines subscribing to that group.

In contrast, **point-to-point** networks consist of many connections between individual pairs of machines. To go from the source to the destination, a packet on this type of network may have to first visit one or more intermediate machines. Often multiple routes, of different lengths are possible, so routing algorithms play an important role in point-to-point networks. As a general rule (although there are many exceptions), smaller, geographically localized networks tend to use broadcasting, whereas larger networks usually are point-to-point.

Interprocessor distance	Processors located in same	Example
0.1 m	Circuit board	Data flow machine
1 m	System	Multicomputer
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1,000 km	Continent	
10,000 km	Planet	The Internet

Fig. 1-2. Classification of interconnected processors by scale.

An alternative criterion for classifying networks is their scale. In Fig. 1-2 we give a classification of multiple processor systems arranged by their physical size. At the top are **data flow machines**, highly parallel computers with many functional units all working on the same program. Next come the **multicomputers**, systems that communicate by sending messages over very short, very fast buses. Beyond the multicomputers are the true networks, computers that communicate

RED HAT 1057
RED HAT V. CAS

by exchanging messages over longer cables. These can be divided into local, metropolitan, and wide area networks. Finally, the connection of two or more networks is called an internetwork. The worldwide Internet is a well-known example of an internetwork. Distance is important as a classification metric because different techniques are used at different scales. In this book we will be concerned with only the true networks and their interconnection. Below we give a brief introduction to the subject of network hardware.

1.2.1. Local Area Networks

Local area networks, generally called **LANs**, are privately-owned networks within a single building or campus of up to a few kilometers in size. They are widely used to connect personal computers and workstations in company offices and factories to share resources (e.g., printers) and exchange information. LANs are distinguished from other kinds of networks by three characteristics: (1) their size, (2) their transmission technology, and (3) their topology.

LANs are restricted in size, which means that the worst-case transmission time is bounded and known in advance. Knowing this bound makes it possible to use certain kinds of designs that would not otherwise be possible. It also simplifies network management.

LANs often use a transmission technology consisting of a single cable to which all the machines are attached, like the telephone company party lines once used in rural areas. Traditional LANs run at speeds of 10 to 100 Mbps, have low delay (tens of microseconds), and make very few errors. Newer LANs may operate at higher speeds, up to hundreds of megabits/sec. In this book, we will adhere to tradition and measure line speeds in megabits/sec (Mbps), not megabytes/sec (MB/sec). A megabit is 1,000,000 bits, not 1,048,576 (2^{20}) bits.

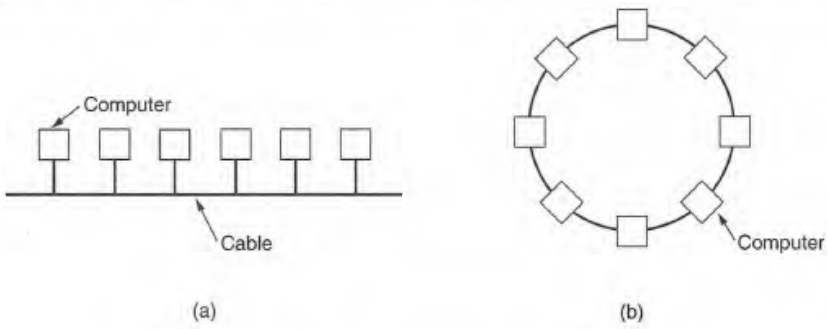


Fig. 1-3. Two broadcast networks. (a) Bus. (b) Ring.

Various topologies are possible for broadcast LANs. Figure 1-3 shows two of them. In a bus (i.e., a linear cable) network, at any instant one machine is the

RED HAT 1057
RED HAT V. CAS

master and is allowed to transmit. All other machines are required to refrain from sending. An arbitration mechanism is needed to resolve conflicts when two or more machines want to transmit simultaneously. The arbitration mechanism may be centralized or distributed. IEEE 802.3, popularly called **Ethernet**™, for example, is a bus-based broadcast network with decentralized control operating at 10 or 100 Mbps. Computers on an Ethernet can transmit whenever they want to; if two or more packets collide, each computer just waits a random time and tries again later.

A second type of broadcast system is the ring. In a ring, each bit propagates around on its own, not waiting for the rest of the packet to which it belongs. Typically, each bit circumnavigates the entire ring in the time it takes to transmit a few bits, often before the complete packet has even been transmitted. Like all other broadcast systems, some rule is needed for arbitrating simultaneous accesses to the ring. Various methods are in use and will be discussed later in this book. IEEE 802.5 (the IBM token ring), is a popular ring-based LAN operating at 4 and 16 Mbps.

Broadcast networks can be further divided into static and dynamic, depending on how the channel is allocated. A typical static allocation would be to divide up time into discrete intervals and run 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 a bus arbitration unit, which determines who goes next. It might do this by accepting requests and making a decision according to some internal algorithm. In the decentralized channel allocation method, there is no central entity; each machine must decide for itself whether or not to transmit. You might think that this always leads to chaos, but it does not. Later we will study many algorithms designed to bring order out of the potential chaos.

The other kind of LAN is built using point-to-point lines. Individual lines connect a specific machine with another specific machine. Such a LAN is really a miniature wide area network. We will look at these later.

1.2.2. Metropolitan Area Networks

A **metropolitan area network**, or **MAN** (plural: MANs, not MEN) is basically a bigger version of a LAN and normally uses similar technology. It might cover a group of nearby corporate offices or a city and might be either private or public. A MAN can support both data and voice, and might even be related to the local cable television network. A MAN just has one or two cables and does not contain switching elements, which shunt packets over one of several potential output lines. Not having to switch simplifies the design.

RED HAT 1057
RED HAT V. CAS

The main reason for even distinguishing MANs as a special category is that a standard has been adopted for them, and this standard is now being implemented. It is called **DQDB (Distributed Queue Dual Bus)** or for people who prefer numbers to letters, 802.6 (the number of the IEEE standard that defines it). DQDB consists of two unidirectional buses (cables) to which all the computers are connected, as shown in Fig. 1-4. Each bus has a head-end, a device that initiates transmission activity. Traffic that is destined for a computer to the right of the sender uses the upper bus. Traffic to the left uses the lower one.

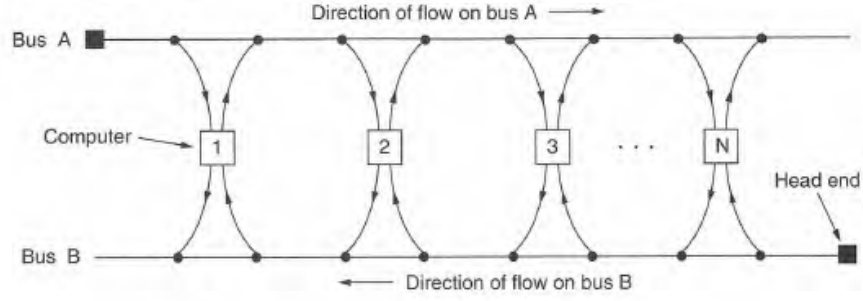


Fig. 1-4. Architecture of the DQDB metropolitan area network.

A key aspect of a MAN is that there is a broadcast medium (for 802.6, two cables) to which all the computers are attached. This greatly simplifies the design compared to other kinds of networks. We will discuss DQDB in more detail in Chap. 4.

1.2.3. Wide Area Networks

A **wide area network**, or **WAN**, spans a large geographical area, often a country or continent. It contains a collection of machines intended for running user (i.e., application) programs. We will follow traditional usage and call these machines **hosts**. The term **end system** is sometimes also used in the literature. The hosts are connected by a **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 from speaker to listener. By separating the pure communication aspects of the network (the subnet) from the application aspects (the hosts), the complete network design is greatly simplified.

In most wide area networks, the subnet consists of two distinct components: transmission lines and switching elements. Transmission lines (also called **circuits**, **channels**, or **trunks**) move bits between machines.

The switching elements are specialized computers used to connect two or more transmission lines. When data arrive on an incoming line, the switching

RED HAT 1057
RED HAT V. CAS

element must choose an outgoing line to forward them on. Unfortunately, there is no standard terminology used to name these computers. They are variously called **packet switching nodes**, **intermediate systems**, and **data switching exchanges**, among other things. As a generic term for the switching computers, we will use the word **router**, but the reader should be aware that no consensus on terminology exists here. In this model, shown in Fig. 1-5, each host is generally connected to a LAN on which a router is present, although in some cases a host can be connected directly to a router. The collection of communication lines and routers (but not the hosts) form the subnet.

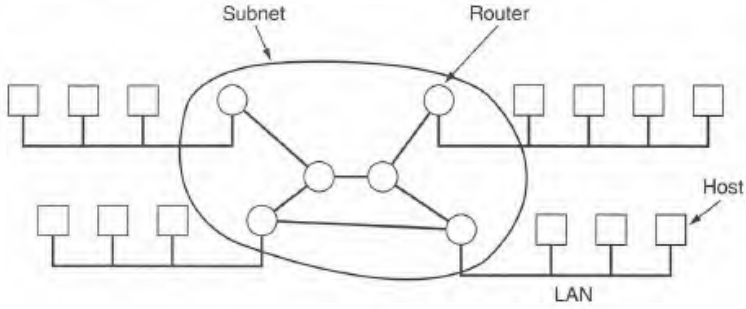


Fig. 1-5. Relation between hosts and the subnet.

An aside about the term "subnet" is worth making. Originally, its only meaning was the collection of routers and communication lines that moved packets from the source host to the destination host. However, some years later, it also acquired a second meaning in conjunction with network addressing (which we will discuss in Chap. 5). Hence the term has a certain ambiguity about it. Unfortunately, no widely-used alternative exists for its initial meaning, so with some hesitation we will use it in both senses. From the context, it will always be clear which is meant.

In most WANs, the network contains numerous cables or telephone lines, each one connecting a pair of routers. If two routers that do not share a cable nevertheless wish to communicate, they must do this indirectly, via other routers. When a packet is sent from one router to another via one or more intermediate routers, the packet is received at each intermediate router in its entirety, stored there until the required output line is free, and then forwarded. A subnet using this principle is called a **point-to-point**, **store-and-forward**, or **packet-switched** subnet. Nearly all wide area networks (except those using satellites) have store-and-forward subnets. When the packets are small and all the same size, they are often called **cells**.

When a point-to-point subnet is used, an important design issue is what the router interconnection topology should look like. Figure 1-6 shows several

RED HAT 1057
RED HAT V. CAS

possible topologies. Local networks that were designed as such usually have a symmetric topology. In contrast, wide area networks typically have irregular topologies.

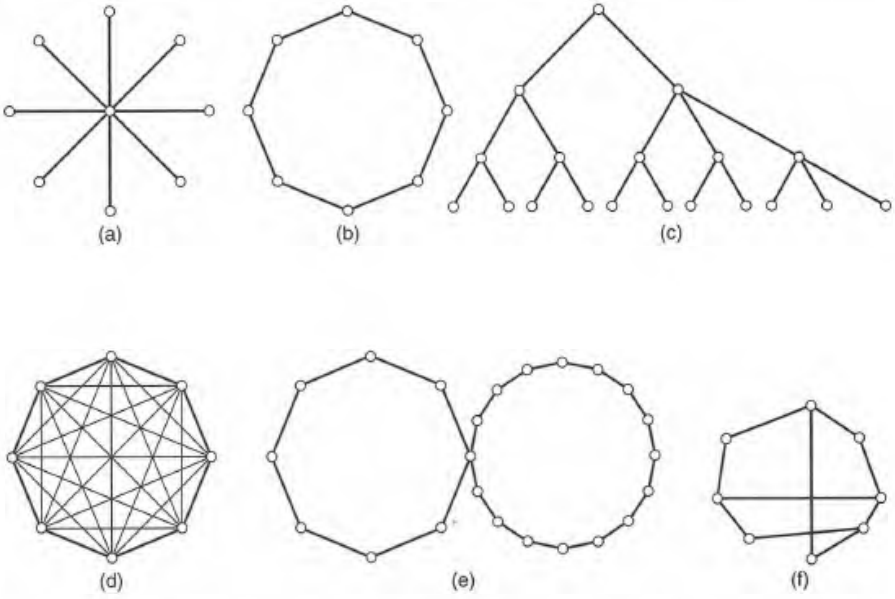


Fig. 1-6. Some possible topologies for a point-to-point subnet. (a) Star. (b) Ring. (c) Tree. (d) Complete. (e) Intersecting rings. (f) Irregular.

A second possibility for a WAN is a satellite or ground radio system. Each router has an antenna through which it can send and receive. All routers can hear the output *from* the satellite, and in some cases they can also hear the upward transmissions of their fellow routers *to* the satellite as well. Sometimes the routers are connected to a substantial point-to-point subnet, with only some of them having a satellite antenna. Satellite networks are inherently broadcast and are most useful when the broadcast property is important.

1.2.4. Wireless Networks

Mobile computers, such as notebook computers and personal digital assistants (PDAs), are the fastest-growing segment of the computer industry. Many of the owners of these computers have desktop machines on LANs and WANs back at the office and want to be connected to their home base even when away from home or en route. Since having a wired connection is impossible in cars and airplanes, there is a lot of interest in wireless networks. In this section we will

RED HAT 1057
RED HAT V. CAS

briefly introduce this topic. (Note: by section, we mean those portions of the book with a three-part number such as 1.2.4.)

Actually, digital wireless communication is not a new idea. As early as 1901, the Italian physicist Guglielmo Marconi demonstrated a ship-to-shore wireless telegraph using Morse Code (dots and dashes are binary, after all). Modern digital wireless systems have better performance, but the basic idea is the same. Additional information about these systems can be found in (Garg and Wilkes, 1996; and Pahlavan et al., 1995).

Wireless networks have many uses. A common one is the portable office. People on the road often want to use their portable electronic equipment to send and receive telephone calls, faxes, and electronic mail, read remote files, login on remote machines, and so on, and do this from anywhere on land, sea, or air.

Wireless networks are of great value to fleets of trucks, taxis, buses, and repairpersons for keeping in contact with home. Another use is for rescue workers at disaster sites (fires, floods, earthquakes, etc.) where the telephone system has been destroyed. Computers there can send messages, keep records, and so on.

Finally, wireless networks are important to the military. If you have to be able to fight a war anywhere on earth on 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. 1-7 shows. Portable computers are sometimes wired. For example, if a traveler plugs a portable computer into the telephone jack in a hotel, we have mobility without a wireless network. Another example is someone carrying a portable computer along as he inspects a train for technical problems. Here a long cord can trail along behind (vacuum cleaner model).

Wireless	Mobile	Applications
No	No	Stationary workstations in offices
No	Yes	Using a portable in a hotel; train maintenance
Yes	No	LANs in older, unwired buildings
Yes	Yes	Portable office; PDA for store inventory

Fig. 1-7. Combinations of wireless networks and mobile computing.

On the other hand, some wireless computers are not portable. An important example here is a company that owns an older building that does not have network cabling installed and wants to connect its computers. Installing a wireless LAN may require little more than buying a small box with some electronics and setting up some antennas. This solution may be cheaper than wiring the building.

Although wireless LANs are easy to install, they also have some disadvantages. Typically they have a capacity of 1–2 Mbps, which is much slower than

RED HAT 1057
RED HAT V. CAS

wired LANs. The error rates are often much higher, too, and the transmissions from different computers can interfere with one another.

But of course, there are also the true mobile, wireless applications, ranging from the portable office to people walking around a store with a PDA doing inventory. At many busy airports, car rental return clerks work out in the parking lot with wireless portable computers. They type in the license plate number of returning cars, and their portable, which has a built-in printer, calls the main computer, gets the rental information, and prints out the bill on the spot. True mobile computing is discussed further in (Forman and Zahorjan, 1994).

Wireless networks come in many forms. Some universities are already installing antennas all over campus to allow students to sit under the trees and consult the library's card catalog. Here the computers communicate directly with the wireless LAN in digital form. Another possibility is using a cellular (i.e., portable) telephone with a traditional analog modem. Direct digital cellular service, called **CDPD** (**Cellular Digital Packet Data**) is becoming available in many cities. We will study it in Chap. 4.

Finally, it is possible to have different combinations of wired and wireless networking. For example, in Fig. 1-8(a), we depict an airplane with a number of people using modems and seat-back telephones to call the office. Each call is independent of the other ones. A much more efficient option, however, is the flying LAN of Fig. 1-8(b). Here each seat comes equipped with an Ethernet connector into which passengers can plug their computers. A single router on the aircraft maintains a radio link with some router on the ground, changing routers as it flies along. This configuration is just a traditional LAN, except that its connection to the outside world happens to be a radio link instead of a hardwired line.

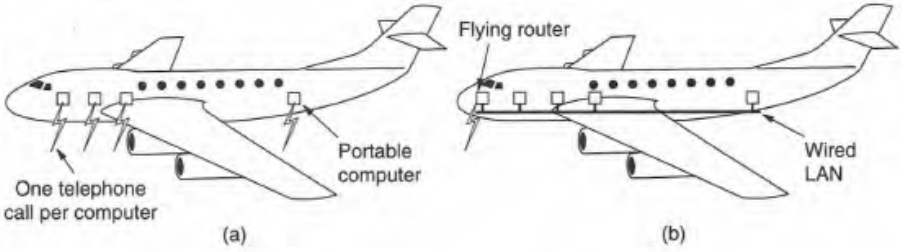


Fig. 1-8. (a) Individual mobile computers. (b) A flying LAN.

While many people believe that wireless portable computers are the wave of the future, at least one dissenting voice has been heard. Bob Metcalfe, the inventor of Ethernet, has written: "Mobile wireless computers are like mobile pipeless bathrooms—portapotties. They will be common on vehicles, and at construction sites, and rock concerts. My advice is to wire up your home and stay there" (Metcalfe, 1995). Will most people follow Metcalfe's advice? Time will tell.

RED HAT 1057
RED HAT V. CAS

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. This desire requires connecting together different, and frequently incompatible networks, sometimes by using machines called **gateways** to make the connection and provide the necessary translation, both in terms of hardware and software. A collection of interconnected networks is called an **internetwork** or just **internet**.

A common form of internet is a collection of LANs connected by a WAN. In fact, if we were to replace the label "subnet" in Fig. 1-5 by "WAN," nothing else in the figure would have to change. The only real distinction between a subnet and a WAN in this case is whether or not hosts are present. If the system within the closed curve contains only routers, it is a subnet. If it contains both routers and hosts with their own users, it is a WAN.

To avoid confusion, please note that the word "internet" will always be used in this book in a generic sense. In contrast, the **Internet** (note uppercase I) means a specific worldwide internet that is widely used to connect universities, government offices, companies, and of late, private individuals. We will have much to say about both internets and the Internet later in this book.

Subnets, networks, and internetworks are often confused. 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, for example, companies like America Online and CompuServe. As an analogy, the telephone system consists of telephone switching offices connected to each other 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. The combination of a subnet and its hosts forms a network. In the case of a LAN, the cable and the hosts form the network. There really is no subnet.

An internetwork is formed when distinct networks are connected together. In our view, connecting a LAN and a WAN or connecting two LANs forms an internetwork, but there is little agreement in the industry over terminology in this area.

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 method described here forms the keystone of the entire book and will occur repeatedly later on.

RED HAT 1057
RED HAT V. CAS

1.3.1. Protocol Hierarchies

To reduce their design complexity, most networks are organized as a series 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. However, in all networks, the purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

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 either to 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 impossible.

A five-layer network is illustrated in Fig. 1-9. The entities comprising the corresponding layers on different machines are called **peers**. In other words, it is the peers that communicate using the protocol.

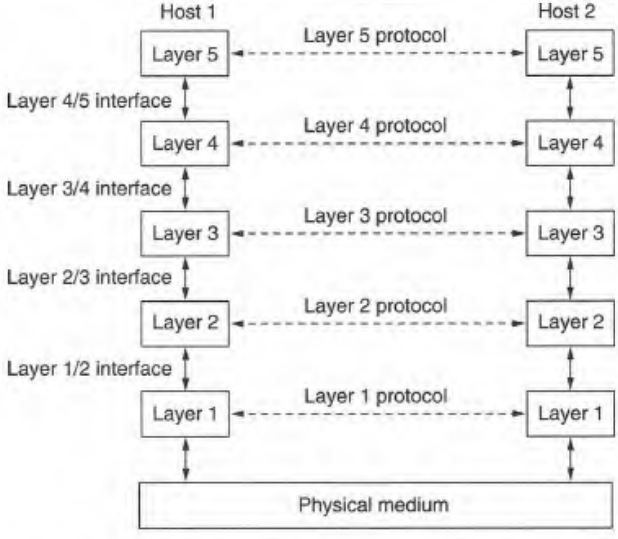


Fig. 1-9. Layers, protocols, and interfaces.

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

RED HAT 1057
RED HAT V. CAS

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. 1-9, virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers there is an **interface**. The interface defines which primitive operations and services the lower layer offers 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, clean-cut interfaces also make it simpler to replace the implementation of one layer with a completely different implementation (e.g., all the telephone lines are replaced by satellite channels), because all that is required of the new implementation is that it offers exactly the same set of services to its upstairs neighbor as the old implementation did.

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 are 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 protocols used by a certain system, one protocol per layer, is called a **protocol stack**. The subjects of network architectures, protocol stacks, and the protocols themselves are the principal topics 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. 1-10. The translators have agreed on a neutral language, Dutch, so the message is converted to "Ik hou van konijnen." The choice of 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, by, for example, fax (the layer 1 protocol). When the message arrives, it is translated into French and passed across the 2/3 interface to philosopher 2. 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

RED HAT 1057
RED HAT V. CAS

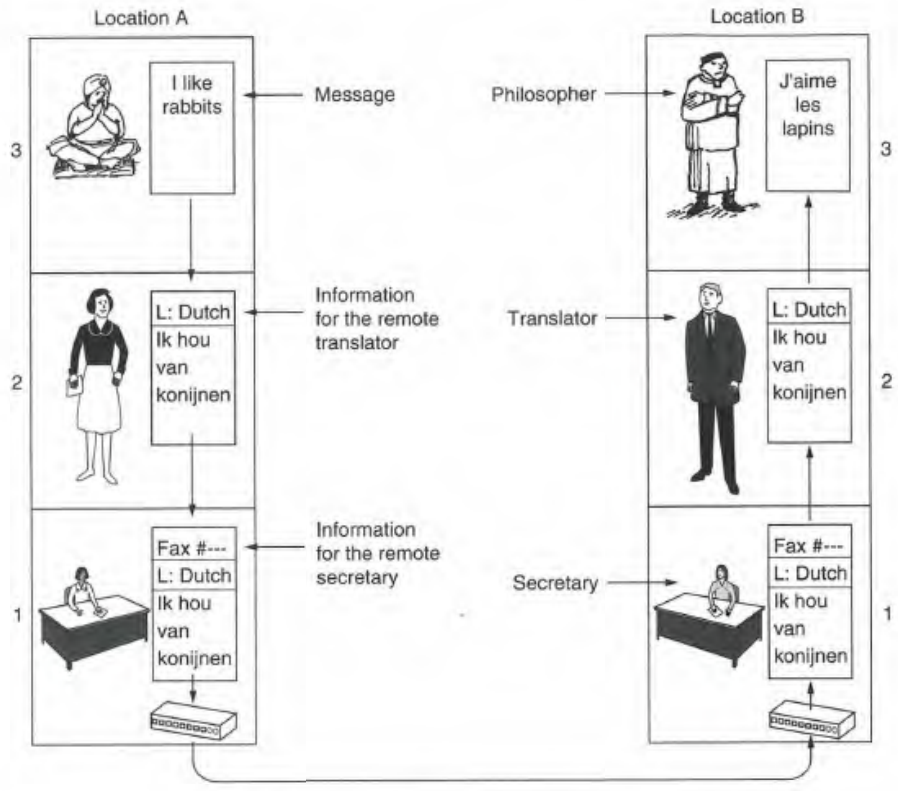


Fig. 1-10. The philosopher-translator-secretary architecture.

layer 3. Similarly the secretaries can switch from fax to email, or 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 upward 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. 1-11. 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 sequence numbers, to allow layer 4 on the destination machine to deliver messages in the right order if the lower layers do not maintain sequence. In some layers, headers also contain sizes, times, and other control fields.

In many networks, there is no limit to 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

RED HAT 1057
RED HAT V. CAS

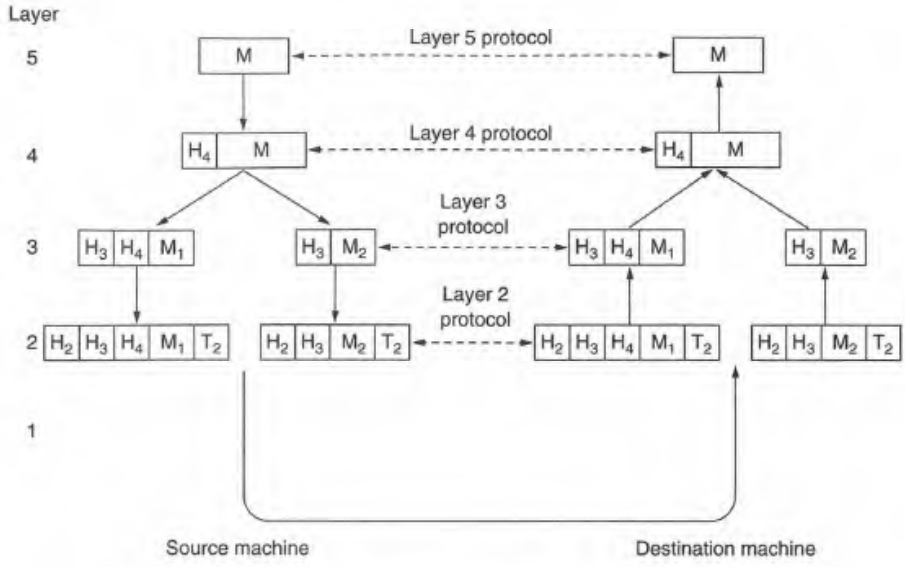


Fig. 1-11. Example information flow supporting virtual communication in layer 5.

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

Layer 3 decides which of the outgoing lines to use and passes the packets to layer 2. Layer 2 adds not only a header to each piece, 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. 1-11 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 a procedure called something like *SendToOtherSide* and *GetFromOtherSide*, even though these procedures actually communicate with lower layers across the 3/4 interface, not with the other side.

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 Section 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.

RED HAT 1057
RED HAT V. CAS

1.3.2. Design Issues for the Layers

Some of the key design issues that occur in computer networking are present in several layers. Below, we will briefly mention some of the more important ones.

Every layer needs a mechanism for identifying senders and receivers. Since a network normally has many computers, some of which have multiple processes, a means is needed for a process on one machine to specify with whom it wants to talk. As a consequence of having multiple destinations, some form of addressing is needed in order to specify a specific destination.

Another set of design decisions concerns the rules for data transfer. In some systems, data only travel in one direction (**simplex communication**). In others they can travel in either direction, but not simultaneously (**half-duplex communication**). In still others they travel in both directions at once (**full-duplex communication**). The protocol must also determine how many logical channels the connection corresponds to, and what their priorities are. Many networks provide at least two logical channels per connection, one for normal data and one for urgent data.

Error control is an important issue because physical communication circuits are not perfect. Many error-detecting and error-correcting codes are known, but both ends of the connection must agree on which one is being used. In addition, the receiver must have some way of telling the sender which messages have been correctly received and which have not.

Not all communication channels preserve the order of messages sent on them. To deal with a possible loss of sequencing, the protocol must make explicit provision for the receiver to allow the pieces to be put back together properly. An obvious solution is to number the pieces, but this solution still leaves open the question of what should be done with pieces that arrive out of order.

An issue that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Various solutions have been proposed and will be discussed later. Some of them involve some kind of feedback from the receiver to the sender, either directly or indirectly, about the receiver's current situation. Others limit the sender to an agreed upon transmission rate.

Another problem that must be solved at several levels is the inability of all processes to accept arbitrarily long messages. This property leads to mechanisms for disassembling, transmitting, and then reassembling messages. A related issue is what to do when processes insist upon transmitting data in units that are so small that sending each one separately is inefficient. Here the solution is to gather together several small messages heading toward a common destination into a single large message and dismember the large message at the other side.

When it is inconvenient or expensive to set up a separate connection for each pair of communicating processes, the underlying layer may decide to use the same connection for multiple, unrelated conversations. As long as this multiplexing and

RED HAT 1057
RED HAT V. CAS

demultiplexing is done transparently, it can be used by any layer. Multiplexing is needed in the physical layer, for example, where all the traffic for all connections has to be sent over at most a few physical circuits.

When there are multiple paths between source and destination, a route must be chosen. Sometimes this decision must be split over two or more layers. For example, to send data from London to Rome, a high-level decision might have to be made to go via France or Germany based on their respective privacy laws, and a low-level decision might have to be made to choose one of the many available circuits based on the current traffic load.

1.3.3. Interfaces and Services

The function of each layer is to provide services to the layer above it. In this section we will look at precisely what a service is in more detail, but first we will give some terminology.

The active elements in each layer are often called **entities**. An entity can be a software entity (such as a process), or a hardware entity (such as an intelligent I/O chip). Entities in the same layer on different machines are called **peer entities**. The entities in layer n implement a service used by layer $n + 1$. In this case layer n is called the **service provider** and layer $n + 1$ is called the **service user**. Layer n may use the services of layer $n - 1$ in order to provide its service. It may offer several classes of service, for example, fast, expensive communication and slow, cheap communication.

Services are available at **SAP**s (**Service Access Points**). The layer n SAPs are the places where layer $n + 1$ can access the services offered. Each SAP has an address that uniquely identifies it. To make this point clearer, the SAPs in the telephone system are the sockets into which modular telephones can be plugged, and the SAP addresses are the telephone numbers of these sockets. To call someone, you must know the callee's SAP address. Similarly, in the postal system, the SAP addresses are street addresses and post office box numbers. To send a letter, you must know the addressee's SAP address.

In order for two layers to exchange information, there has to be an agreed upon set of rules about the interface. At a typical interface, the layer $n + 1$ entity passes an **IDU** (**Interface Data Unit**) to the layer n entity through the SAP as shown in Fig. 1-12. The IDU consists of an **SDU** (**Service Data Unit**) and some control information. The SDU is the information passed across the network to the peer entity and then up to layer $n + 1$. The control information is needed to help the lower layer do its job (e.g., the number of bytes in the SDU) but is not part of the data itself.

In order to transfer the SDU, the layer n entity may have to fragment it into several pieces, each of which is given a header and sent as a separate **PDU** (**Protocol Data Unit**) such as a packet. The PDU headers are used by the peer entities

RED HAT 1057
RED HAT V. CAS

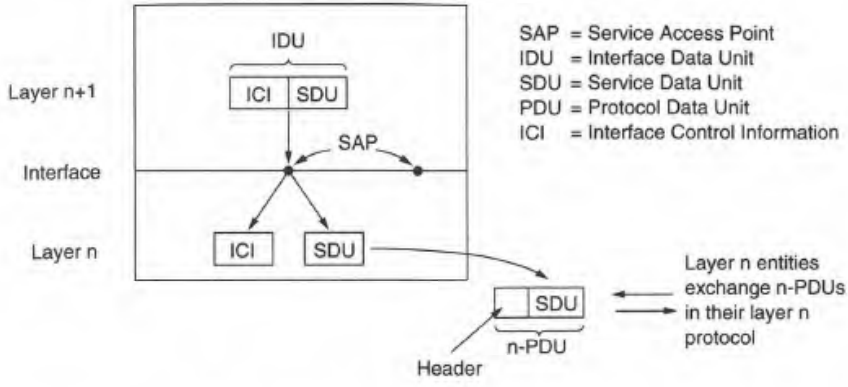


Fig. 1-12. Relation between layers at an interface.

to carry out their peer protocol. They identify which PDUs contain data and which contain control information, provide sequence numbers and counts, and so on.

1.3.4. Connection-Oriented and Connectionless Services

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 in the same order at the other end.

In contrast, **connectionless service** is modeled after the postal system. Each message (letter) carries the full destination address, and each one is routed through the system independent of all the others. 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. With a connection-oriented service this is impossible.

Each service can be characterized by a **quality of service**. 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

RED HAT 1057
RED HAT V. CAS

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, the message boundaries are preserved. When two 1-KB messages are sent, they arrive as two distinct 1-KB messages, never as one 2-KB message. (Note: KB means kilobytes; kb means kilobits.) In the latter, the connection is simply a stream of bytes, with no message boundaries. When 2K bytes arrive at the receiver, there is no way to tell if they were sent as one 2-KB message, two 1-KB 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, with a terminal logging into a remote timesharing system, a byte stream from the terminal to the computer is all that is needed.

As mentioned above, for some applications, the delays introduced by acknowledgements are unacceptable. One such application is digitized voice traffic. It is preferable for telephone users to hear a bit of noise on the line or a garbled word from time to time than to introduce a delay to wait for acknowledgements. Similarly, when transmitting a video film, having a few pixels wrong is no problem, but having the film jerk along as the flow stops to correct errors is very irritating.

Not all applications require connections. For example, as electronic mail becomes more common, can electronic junk mail be far behind? The electronic junk mail sender probably does not want to go to the trouble of setting up and later tearing down a connection just to send 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 provide an acknowledgement back to the sender.

In other situations, the convenience of not having to establish a connection to send one short 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.

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. For example, a query to the local library asking where Uighur is spoken falls into this category. Request-reply is commonly used to implement communication in the client-server model: the client issues a request and the server responds to it. Figure 1-13 summarizes the types of services discussed above.

RED HAT 1057
RED HAT V. CAS

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connectionless	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Registered mail
	Request-reply	Database query

Fig. 1-13. Six different types of service.

1.3.5. Service Primitives

A service is formally specified by a set of **primitives** (operations) available to a user or other entity to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. One way to classify the service primitives is to divide them into four classes as shown in Fig. 1-14.

Primitive	Meaning
Request	An entity wants the service to do some work
Indication	An entity is to be informed about an event
Response	An entity wants to respond to an event
Confirm	The response to an earlier request has come back

Fig. 1-14. Four classes of service primitives.

To illustrate the uses of the primitives, consider how a connection is established and released. The initiating entity does a CONNECT.request which results in a packet being sent. The receiver then gets a CONNECT.indication announcing that an entity somewhere wants to set up a connection to it. The entity getting the CONNECT.indication then uses the CONNECT.response primitive to tell whether it wants to accept or reject the proposed connection. Either way, the entity issuing the initial CONNECT.request finds out what happened via a CONNECT.confirm primitive.

Primitives can have parameters, and most of them do. The parameters to a CONNECT.request might specify the machine to connect to, the type of service desired, and the maximum message size to be used on the connection. The parameters to a CONNECT.indication might contain the caller's identity, the type of

RED HAT 1057
RED HAT V. CAS

service desired, and the proposed maximum message size. If the called entity did not agree to the proposed maximum message size, it could make a counterproposal in its *response* primitive, which would be made available to the original caller in the *confirm*. The details of this **negotiation** are part of the protocol. For example, in the case of two conflicting proposals about maximum message size, the protocol might specify that the smaller value is always chosen.

As an aside on terminology, we will carefully avoid the terms "open a connection" and "close a connection" because to electrical engineers, an "open circuit" is one with a gap or break in it. Electricity can only flow over "closed circuits." Computer scientists would never agree to having information flow over a closed circuit. To keep both camps pacified, we will use the terms "establish a connection" and "release a connection."

Services can be either **confirmed** or **unconfirmed**. In a confirmed service, there is a *request*, an *indication*, a *response*, and a *confirm*. In an unconfirmed service, there is just a *request* and an *indication*. CONNECT is always a confirmed service because the remote peer must agree to establish a connection. Data transfer, on the other hand, can be either confirmed or unconfirmed, depending on whether or not the sender needs an acknowledgement. Both kinds of services are used in networks.

To make the concept of a service more concrete, let us consider as an example a simple connection-oriented service with eight service primitives as follows:

1. CONNECT.request – Request a connection to be established.
2. CONNECT.indication – Signal the called party.
3. CONNECT.response – Used by the callee to accept/reject calls.
4. CONNECT.confirm – Tell the caller whether the call was accepted.
5. DATA.request – Request that data be sent.
6. DATA.indication – Signal the arrival of data.
7. DISCONNECT.request – Request that a connection be released.
8. DISCONNECT.indication – Signal the peer about the request.

In this example, CONNECT is a confirmed service (an explicit response is required), whereas DISCONNECT is unconfirmed (no response).

It may be helpful to make an analogy with the telephone system to see how these primitives are used. For this analogy, consider the steps required to call Aunt Millie on the telephone and invite her to your house for tea.

1. CONNECT.request – Dial Aunt Millie's phone number.
2. CONNECT.indication – Her phone rings.
3. CONNECT.response – She picks up the phone.

RED HAT 1057
RED HAT V. CAS

4. CONNECT.confirm – You hear the ringing stop.
5. DATA.request – You invite her to tea.
6. DATA.indication – She hears your invitation.
7. DATA.request – She says she would be delighted to come.
8. DATA.indication – You hear her acceptance.
9. DISCONNECT.request – You hang up the phone.
10. DISCONNECT.indication – She hears it and hangs up too.

Figure 1-15 shows this same sequence of steps as a series of service primitives, including the final confirmation of disconnection. Each step involves an interaction between two layers on one of the computers. Each *request* or *response* causes an *indication* or *confirm* at the other side a little later. In this example, the service users (you and Aunt Millie) are in layer $N + 1$ and the service provider (the telephone system) is in layer N.

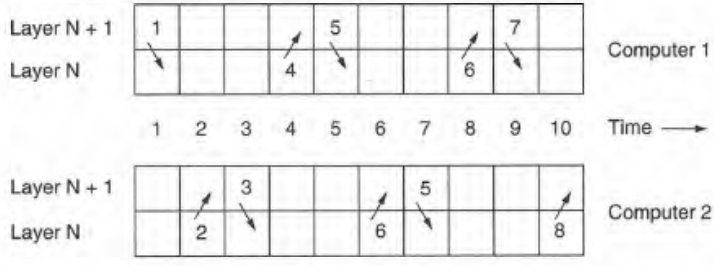


Fig. 1-15. How a computer would invite its Aunt Millie to tea. The numbers near the tail end of each arrow refer to the eight service primitives discussed in this section.

1.3.6. The Relationship of Services to Protocols

Services and protocols are distinct concepts, although they are frequently confused. This distinction is so important, however, 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 frames, packets, or messages that are exchanged by the peer entities within a layer. Entities use protocols in order to implement their service definitions. They

RED HAT 1057
RED HAT V. CAS

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.

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. 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.

1.4. REFERENCE MODELS

Now that we have discussed layered networks in the abstract, it is time to look at some examples. In the next two sections we will discuss two important network architectures, the OSI reference model and the TCP/IP reference model.

1.4.1. The OSI Reference Model

The OSI model is shown in Fig. 1-16 (minus the physical medium). 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). 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 usually 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 are as follows:

1. A layer should be created where a different level of 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.
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.

RED HAT 1057
RED HAT V. CAS

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.

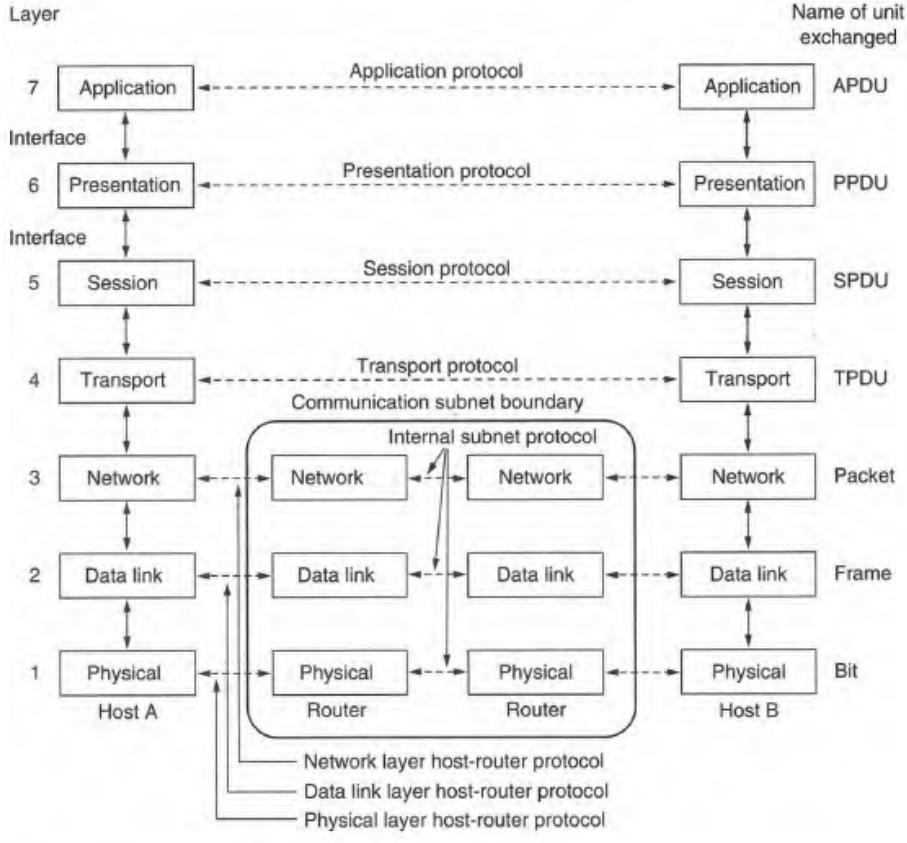


Fig. 1-16. The OSI reference model.

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

RED HAT 1057
RED HAT V. CAS

questions here are how many volts should be used to represent a 1 and how many for a 0, how many microseconds a bit lasts, whether transmission may proceed simultaneously in both directions, how the initial connection is established and how it is torn down when both sides are finished, and how many pins the network connector has and what each pin is used for. The design issues here largely deal with mechanical, electrical, and procedural interfaces, and the physical transmission medium, which lies below the physical layer.

The Data Link Layer

The main task of the **data link layer** is to take a raw transmission facility and transform it into a line that appears free of undetected transmission errors to the network layer. It accomplishes this task by having the sender break the input data up into **data frames** (typically a few hundred or a few thousand bytes), transmit the frames sequentially, and process the **acknowledgement frames** sent back by the receiver. Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is up to the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in the data, special care must be taken to make sure these patterns are not incorrectly interpreted as frame delimiters.

A noise burst on the line can destroy a frame completely. In this case, the data link layer software on the source machine can retransmit the frame. However, multiple transmissions of the same frame introduce the possibility of duplicate frames. A duplicate frame could be sent if the acknowledgement frame from the receiver back to the sender were lost. It is up to this layer to solve the problems caused by damaged, lost, and duplicate frames. The data link layer may offer several different service classes to the network layer, each of a different quality and with a different price.

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 must be employed to let the transmitter know how much buffer space the receiver has at the moment. Frequently, this flow regulation and the error handling are integrated.

If the line can be used to transmit data in both directions, this introduces a new complication that the data link layer software must deal with. The problem is that the acknowledgement frames for A to B traffic compete for the use of the line with data frames for the B to A traffic. A clever solution (piggybacking) has been devised; we will discuss it in detail later.

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 sublayer, deals with this problem.

RED HAT 1057
RED HAT V. CAS

The Network Layer

The **network layer** is concerned with controlling 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. They can also be determined at the start of each conversation, for example a terminal session. 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 each other's way, forming bottlenecks. The control of such congestion also belongs to the network layer.

Since the operators of the subnet may well expect remuneration for their efforts, there is often some accounting function built into the network layer. At the very least, the software must count how many packets or characters or bits are sent by each customer, to produce billing information. When a packet crosses a national border, with different rates on each side, the accounting can become complicated.

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 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 the session layer, 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.

Under normal conditions, the transport layer creates a distinct network connection for each transport connection required by the session layer. If the transport connection requires a high throughput, however, the transport layer might create multiple network connections, dividing the data among the network connections to improve throughput. On the other hand, if creating or maintaining a network connection is expensive, the transport layer might multiplex several transport connections onto the same network connection to reduce the cost. In all cases, the transport layer is required to make the multiplexing transparent to the session layer.

The transport layer also determines what type of service to provide the session

RED HAT 1057
RED HAT V. CAS

layer, and ultimately, 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 are transport of isolated messages with no guarantee about the order of delivery, and broadcasting of messages to multiple destinations. The type of service is determined when the connection is established.

The transport layer is a true end-to-end layer, from source to 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, the protocols are between each machine and its immediate neighbors, and not by 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. 1-16.

Many hosts are multiprogrammed, which implies that multiple connections will be entering and leaving each host. There needs to be some way to tell which message belongs to which connection. The transport header (H_4 in Fig. 1-11) is one place this information can be put.

In addition to multiplexing several message streams onto one channel, the transport layer must take care of establishing and deleting connections across the network. This requires some kind of naming mechanism, so that a process on one machine has a way of describing with whom it wishes to converse. There must also be a mechanism to regulate the flow of information, so that a fast host cannot overrun a slow one. Such a mechanism is called **flow control** and plays a key role in the transport layer (also in other layers). Flow control between hosts is distinct from flow control between routers, although we will later see that similar principles apply to both.

The Session Layer

The session layer allows users on different machines to establish **sessions** between them. A session allows ordinary data transport, as does the transport layer, but it also provides enhanced services useful in some applications. A session might be used to allow a user to log into a remote timesharing system or to transfer a file between two machines.

One of the services of the session layer is to manage dialogue control. Sessions can allow traffic to go in both directions at the same time, or in only one direction at a time. If traffic can only go one way at a time (analogous to a single railroad track), the session layer can help keep track of whose turn it is.

A related session service is **token management**. For some protocols, it is essential that both sides do not attempt the same operation at the same time. To manage these activities, the session layer provides tokens that can be exchanged. Only the side holding the token may perform the critical operation.

RED HAT 1057
RED HAT V. CAS

Another session service is **synchronization**. Consider the problems that might occur when trying to do a 2-hour file transfer between two machines with a 1-hour mean time between crashes. After each transfer was aborted, the whole transfer would have to start over again and would probably fail again the next time as well. To eliminate this problem, the session layer provides a way to insert checkpoints into the data stream, so that after a crash, only the data transferred after the last checkpoint have to be repeated.

The Presentation Layer

The **presentation layer** performs certain functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problems. In particular, unlike all the lower layers, which are just interested in moving bits reliably from here to there, the presentation layer is concerned with the syntax and semantics of the information transmitted.

A typical example of a presentation service is encoding data in a standard agreed upon way. Most user programs do not exchange random binary bit strings. They exchange things such as people's names, dates, amounts of money, and invoices. These items are represented as character strings, integers, floating-point numbers, and data structures composed of several simpler items. Different computers have different codes for representing character strings (e.g., ASCII and Unicode), integers (e.g., one's complement and two's complement), and so on. In order to make it possible for computers with different 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 converts from the representation used inside the computer to the network standard representation and back.

The Application Layer

The **application layer** contains a variety of protocols that are commonly needed. For example, there are hundreds of incompatible terminal types in the world. Consider the plight of a full screen editor that is supposed to work over a network with many different terminal types, each with different screen layouts, escape sequences for inserting and deleting text, moving the cursor, etc.

One way to solve this problem is to define an abstract **network virtual terminal** that editors and other programs can be written to deal with. To handle each terminal type, a piece of software must be written to map the functions of the network virtual terminal onto the real terminal. For example, when the editor moves the virtual terminal's cursor to the upper left-hand corner of the screen, this software must issue the proper command sequence to the real terminal to get its cursor there too. All the virtual terminal software is in the application layer.

Another application layer function is file transfer. Different file systems have

RED HAT 1057
RED HAT V. CAS

different file naming conventions, different ways of representing text lines, and so on. Transferring a file between two different systems requires handling these and other incompatibilities. This work, too, belongs to the application layer, as do electronic mail, remote job entry, directory lookup, and various other general-purpose and special-purpose facilities.

Data Transmission in the OSI Model

Figure 1-17 shows an example of how data can be transmitted using the OSI model. The sending process has some data it wants to send to the receiving process. It gives the data to the application layer, which then attaches the application header, *AH* (which may be null), to the front of it and gives the resulting item to the presentation layer.

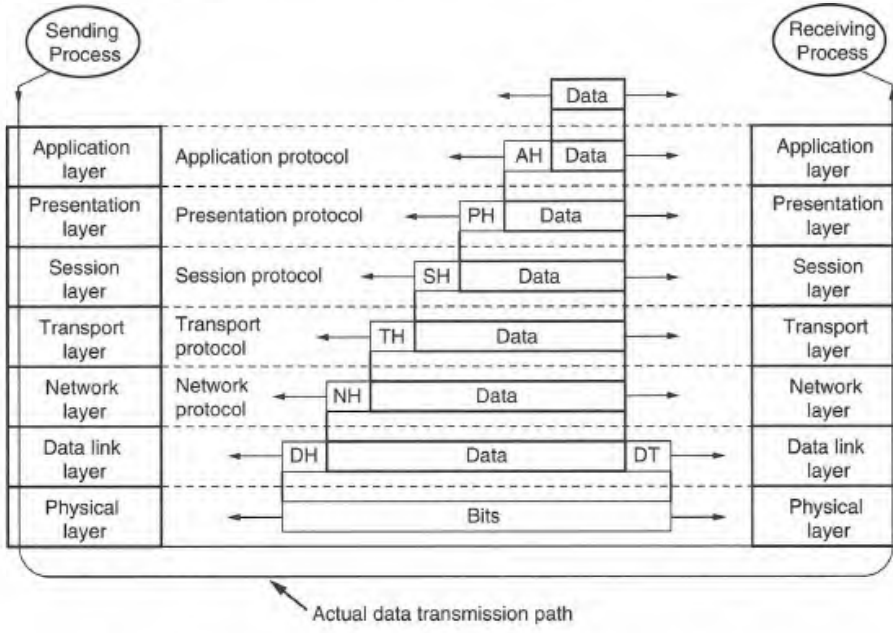


Fig. 1-17. An example of how the OSI model is used. Some of the headers may be null. (Source: H.C. Folts. Used with permission.)

The presentation layer may transform this item in various ways and possibly add a header to the front, giving the result to the session layer. It is important to realize that the presentation layer is not aware of which portion of the data given to it by the application layer is *AH*, if any, and which is true user data.

This process is repeated until the data reach the physical layer, where they are actually transmitted to the receiving machine. On that machine the various

RED HAT 1057
RED HAT V. CAS

headers are stripped off one by one as the message propagates up the layers until it finally arrives at the receiving process.

The key idea throughout is that although actual data transmission is vertical in Fig. 1-17, each layer is programmed as though it were horizontal. When the sending transport layer, for example, gets a message from the session layer, it attaches a transport header and sends it to the receiving transport layer. From its point of view, the fact that it must actually hand the message to the network layer on its own machine is an unimportant technicality. As an analogy, when a Tagalog-speaking diplomat is addressing the United Nations, he thinks of himself as addressing the other assembled diplomats. That, in fact, he is really only speaking to his translator is seen as a technical detail.

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 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 the ability to connect multiple networks together in a seamless way was one of the major design goals from the very beginning. This architecture later became known as the **TCP/IP Reference Model**, after its two primary protocols. It was first defined in (Cerf and Kahn, 1974). A later perspective is given in (Leiner et al., 1985). The design philosophy behind the model is discussed in (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, another major goal was that the network be able to survive loss of subnet hardware, with existing conversations not being broken off. In other words, 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, a flexible architecture was needed, since applications with divergent requirements were envisioned, ranging from transferring files to real-time speech transmission.

The Internet Layer

All these requirements led to the choice of a packet-switching network based on a connectionless internetwork layer. This layer, called the **internet layer**, is the linchpin that holds the whole architecture together. Its job is to permit hosts to

RED HAT 1057
RED HAT V. CAS

inject packets into any network and have them travel independently to the destination (potentially on a different network). They may even arrive in a 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.

The analogy here is with the (snail) mail system. A person can drop a sequence of international letters into a mail box in one country, and with a little luck, most of them will be delivered to the correct address in the destination country. Probably the letters will 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**). The job of the internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly the major issue here, as is avoiding congestion. For these reasons, it is reasonable to say that the TCP/IP internet layer is very similar in functionality to the OSI network layer. Figure 1-18 shows this correspondence.

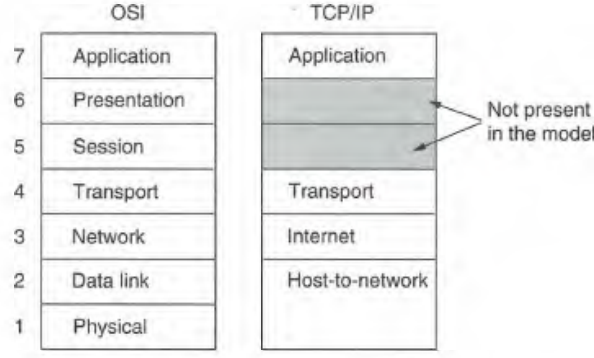


Fig. 1-18. The TCP/IP reference model.

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, the same as in the OSI transport layer. Two end-to-end 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

RED HAT 1057
RED HAT V. CAS

any other machine in the internet. It fragments the incoming byte stream into discrete messages and passes each one onto 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. 1-19. Since the model was developed, IP has been implemented on many other networks.

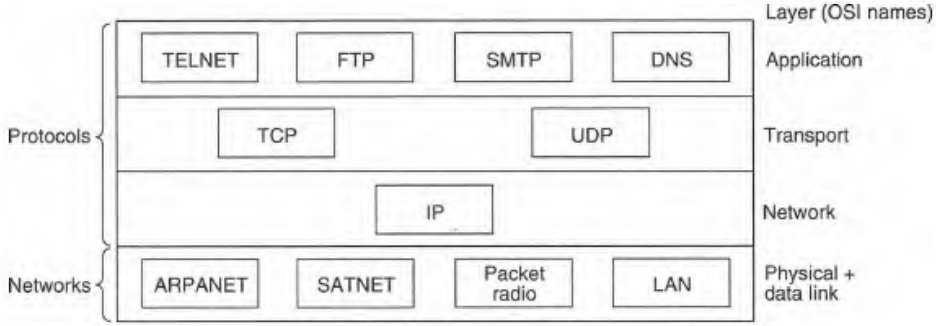


Fig. 1-19. Protocols and networks in the TCP/IP model initially.

The Application Layer

The TCP/IP model does not have session or presentation layers. No need for them was perceived, so they were not included. Experience with the OSI model has proven this view correct: they 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), as shown in Fig. 1-19. The virtual terminal protocol allows a user on one machine to log into a distant machine and work there. The file transfer protocol provides a way to move data efficiently from one machine to another. Electronic mail was originally just a kind of file transfer, but later a specialized protocol was developed for it. Many other protocols have been added to these over the years, such as the Domain Name Service (DNS) for mapping host names onto their network addresses, NNTP, the protocol used for moving news articles around, and HTTP, the protocol used for fetching pages on the World Wide Web, and many others.

RED HAT 1057
RED HAT V. CAS

The Host-to-Network Layer

Below the internet layer is a great void. The TCP/IP reference model does not really say much about what happens here, except to point out that the host has to connect to the network using some protocol so it can send IP packets over it. This protocol is not defined and varies from host to host and network to network. Books and papers about the TCP/IP model rarely discuss it.

1.4.3. A Comparison of the OSI and TCP 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 to make 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.

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

RED HAT 1057
RED HAT V. CAS

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 service, interface, and protocol, 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 is one of the main purposes of having layered protocols in the first place.

The OSI reference model was devised *before* the protocols were invented. This ordering means that the model was not biased toward one particular set of protocols, which made it quite general. The down side of this ordering is 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. When people started to build real networks using the OSI model and existing protocols, it was discovered that they 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 the 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 has four layers. 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 has only one mode in the network layer (connectionless) but supports both modes in the transport layer, giving the users a choice. This choice is especially important for simple request-response protocols.

RED HAT 1057
RED HAT V. CAS

1.4.4. 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 most 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 lessons may be useful. These lessons 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*, and which is illustrated in Fig. 1-20.

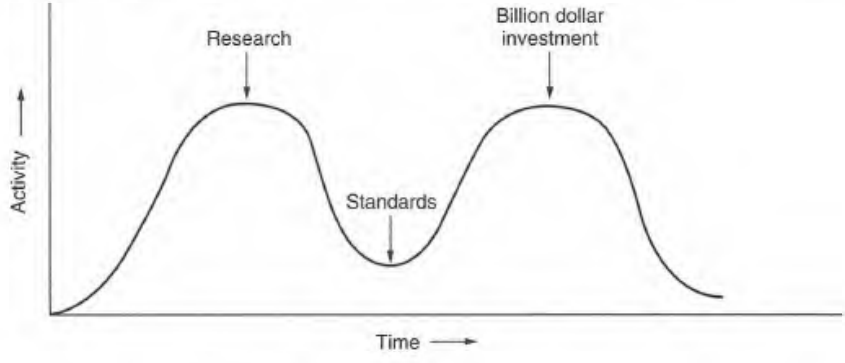


Fig. 1-20. The apocalypse of the two elephants.

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 subsides, corporations discover the subject, and the billion-dollar wave of investment hits.

RED HAT 1057
RED HAT V. CAS

It is essential that the standards be written in the trough between the two "elephants." If they are written too early, before the research is finished, the subject may still be poorly understood, which leads to bad standards. 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. Most discussions of the seven-layer model give the impression that the number and contents of the layers eventually chosen were the only way, or at least the obvious way. This is far from true. The session layer has little use in most applications, and the presentation layer is nearly empty. In fact, the British proposal to ISO only had five layers, not seven. In contrast to the session and presentation layers, the data link and network layers are so full that subsequent work has split them into multiple sublayers, each with different functions.

Although hardly anyone ever admits it in public, the real reason that the OSI model has seven layers is that at the time it was designed, IBM had a proprietary seven-layer protocol called **SNA**™ (**Systems Network Architecture**). At that time, IBM so dominated the computer industry that everyone else, including telephone companies, competing computer companies, and even major governments, were scared to death that IBM would use its market clout to effectively force everybody to use SNA, which it could change whenever it wished. The idea behind OSI was to produce an IBM-like reference model and protocol stack that would become the world standard, and controlled not by one company, but by a neutral organization, ISO.

The OSI model, along with the 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 in (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.

RED HAT 1057
RED HAT V. CAS

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.

Another issue is that the decision to place certain features in particular layers is not always obvious. The virtual terminal handling (now in the application layer) was in the presentation layer during much of the development of the standard. It was moved to the application layer because the committee had trouble deciding what the presentation layer was good for. Data security and encryption were so controversial that no one could agree which layer to put them in, so they were left out altogether. Network management was also omitted from the model for similar reasons.

Another criticism of the original standard is that it completely ignored connectionless services and connectionless protocols, even though most local area networks work that way. Subsequent addenda (known in the software world as bug fixes) corrected this problem.

Perhaps the most serious criticism is that the model is dominated by a communications mentality. The relationship of computing to communications is barely mentioned anywhere, and some of the choices made are wholly inappropriate to the way computers and software work. As an example, consider the OSI primitives, listed in Fig. 1-14. In particular, think carefully about the primitives and how one might use them in a programming language.

The CONNECT.request primitive is simple. One can imagine a library procedure, *connect*, that programs can call to establish a connection. Now think about CONNECT.indication. When a message arrives, the destination process has to be signaled. In effect, it has to get an interrupt—hardly an appropriate concept for programs written in any modern high-level language. Of course, in the lowest layer, an indication (interrupt) does occur.

If the program were expecting an incoming call, it could call a library procedure *receive* to block itself. But if this were the case, why was *receive* not the primitive instead of *indication*? *Receive* is clearly oriented toward the way computers work, whereas *indication* is equally clearly oriented toward the way telephones work. Computers are different from telephones. Telephones ring. Computers do not ring. In short, the semantic model of an interrupt-driven system is conceptually a poor idea and totally at odds with all modern ideas of structured programming. This and similar problems are discussed by Langsford (1984).

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

RED HAT 1057
RED HAT V. CAS

"OSI" with "poor quality." While the products got better 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 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 help much. 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 DoD correcting this later by announcing that it was actually Ada®.

Despite the fact that the OSI model and protocols have been less than a resounding success, there are still a few organizations interested in it, mostly European PTTs that still have a monopoly on telecommunication. Consequently a feeble effort has been made to update OSI, resulting in a revised model published in 1994. For what was changed (little) and what should have been changed (a lot), see (Day, 1995).

1.4.5. 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 service, interface, and protocol. Good software engineering practice requires differentiating between the specification and the implementation, something that OSI does very carefully, and 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 describe SNA using the TCP/IP model would be nearly impossible, for example.

Third, the host-to-network layer is not really a layer at all in the normal sense that the term is 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 a crucial one and one should not be sloppy about it.

RED HAT 1057
RED HAT V. CAS

Fourth, the TCP/IP model does not distinguish (or even mention) 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, 25 years later, it is still in widespread use.

In summary, despite its problems, the OSI *model* (minus the session and presentation layers) has proven to be exceptionally useful for discussing computer networks. In contrast, the OSI *protocols* have not become popular. The reverse is true of TCP/IP: the *model* is practically nonexistent, but the *protocols* are widely used. Since computer scientists like to have their cake and eat it, too, in this book we will use a modified OSI model but concentrate primarily on the TCP/IP and related protocols, as well as newer ones such as SMDS, frame relay, SONET, and ATM. In effect, we will use the hybrid model of Fig. 1-21 as the framework for this book.

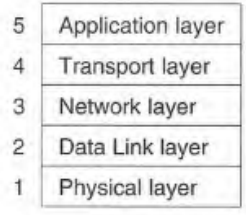


Fig. 1-21. The hybrid reference model to be used in this book.

1.5. EXAMPLE NETWORKS

Numerous networks are currently operating around the world. Some of these are public networks run by common carriers or PTTs, others are research networks, yet others are cooperative networks run by their users, and still others are commercial or corporate networks. In the following sections we will take a look

RED HAT 1057
RED HAT V. CAS

at a few current and historical networks to get an idea of what they are (or were) like and how they differ from one another.

Networks differ in their history, administration, facilities offered, technical design, and user communities. The history and administration can vary from a network carefully planned by a single organization with a well-defined goal, to an ad hoc collection of machines that have been connected to one another over the years without any master plan or central administration at all. The facilities available range from arbitrary process-to-process communication to electronic mail, file transfer, remote login, and remote execution. The technical designs can differ in the transmission media used, the naming and routing algorithms employed, the number and contents of the layers present, and the protocols used. Finally, the user community can vary from a single corporation to all the academic computer scientists in the industrialized world.

In the following sections we will look at a few examples. These are the popular commercial LAN networking package, Novell NetWare®, the worldwide Internet (including its predecessors, the ARPANET and NSFNET), and the first gigabit networks.

1.5.1. Novell NetWare

The most popular network system in the PC world is **Novell NetWare**. It was designed to be used by companies downsizing from a mainframe to a network of PCs. In such systems, each user has a desktop PC functioning as a client. In addition, some number of powerful PCs operate as servers, providing file services, database services, and other services to a collection of clients. In other words, Novell NetWare is based on the client-server model.

NetWare uses a proprietary protocol stack illustrated in Fig. 1-22. It is based on the old Xerox Network System, XNS™ but with various modifications. Novell NetWare predates OSI and is not based on it. If anything, it looks more like TCP/IP than like OSI.

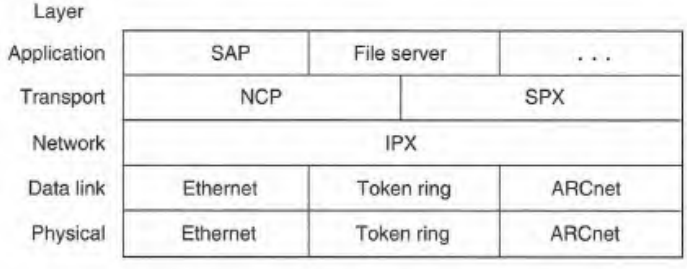


Fig. 1-22. The Novell NetWare reference model.

The physical and data link layers can be chosen from among various industry standards, including Ethernet, IBM token ring, and ARCnet. The network layer

RED HAT 1057
RED HAT V. CAS

runs an unreliable connectionless internetwork protocol called **IPX** (**Internet Packet eXchange**). It passes packets transparently from source to destination, even if the source and destination are on different networks. IPX is functionally similar to IP, except that it uses 12-byte addresses instead of 4-byte addresses. The wisdom of this choice will become apparent in Chap. 5.

Above IPX comes a connection-oriented transport protocol called **NCP** (**Network Core Protocol**). NCP also provides various other services besides user data transport and is really the heart of NetWare. A second protocol, **SPX** (**Sequenced Packet eXchange**), is also available, but provides only transport. TCP is another option. Applications can choose any of them. The file system uses NCP and Lotus Notes® uses SPX, for example. The session and presentation layers do not exist. Various application protocols are present in the application layer.

As in TCP/IP, the key to the entire architecture is the internet datagram packet on top of which everything else is built. The format of an IPX packet is shown in Fig. 1-23. The *Checksum* field is rarely used, since the underlying data link layer also provides a checksum. The *Packet length* field tells how long the entire packet is, header plus data. The *Transport control* field counts how many networks the packet has traversed. When this exceeds a maximum, the packet is discarded. The *Packet type* field is used to mark various control packets. The two addresses each contain a 32-bit network number, a 48-bit machine number (the 802 LAN address), and 16-bit local address (socket) on that machine. Finally, we have the data, which occupy the rest of the packet, with the maximum size being determined by the underlying network.

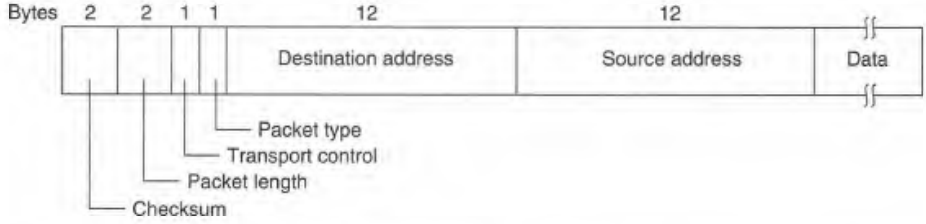


Fig. 1-23. A Novell NetWare IPX packet.

About once a minute, each server broadcasts a packet giving its address and telling what services it offers. These broadcasts use the **SAP** (**Service Advertising Protocol**) protocol. The packets are seen and collected by special agent processes running on the router machines. The agents use the information contained in them to construct databases of which servers are running where.

When a client machine is booted, it broadcasts a request asking where the nearest server is. The agent on the local router machine sees this request, looks in its database of servers, and matches up the request with the best server. The choice of server to use is then sent back to the client. The client can now establish

RED HAT 1057
RED HAT V. CAS

an NCP connection with the server. Using this connection, the client and server negotiate the maximum packet size. From this point on, the client can access the file system and other services using this connection. It can also query the server's database to look for other (more distant) servers.

1.5.2. The ARPANET

Let us now switch gears from LANs to WANs. In the mid-1960s, at the height of the Cold War, the DoD wanted a command and control network that could survive a nuclear war. Traditional circuit-switched telephone networks were considered too vulnerable, since the loss of one line or switch would certainly terminate all conversations using them and might even partition the network. To solve this problem, DoD turned to its research arm, ARPA (later DARPA, now ARPA again), the (periodically Defense) Advanced Research Projects Agency.

ARPA was created in response to the Soviet Union's launching Sputnik in 1957 and had the mission of advancing technology that might be useful to the military. 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.

Several early grants went to universities for investigating the then-radical idea of packet switching, something that had been suggested by Paul Baran in a series of RAND Corporation reports published in the early 1960s. After some discussions with various experts, ARPA decided that the network the DoD needed should be a packet-switched network, consisting of a subnet and host computers.

The subnet would consist of minicomputers called **IMPs** (**Interface Message Processors**) connected by 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 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

RED HAT 1057
RED HAT V. CAS

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.

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. 1-24.

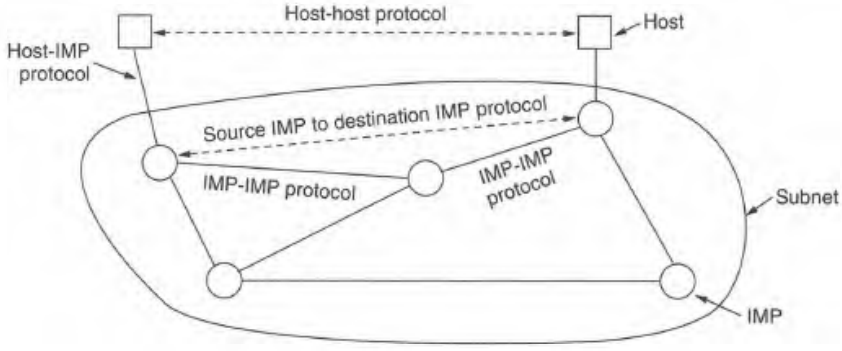


Fig. 1-24. The original ARPANET design.

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 felt 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.

To deal with problem of host software, Larry Roberts of ARPA 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 design of the network and its software to them and then to 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 on the air 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 network grew quickly as more IMPs were delivered and installed; it soon spanned the United States. Figure 1-25 shows how rapidly the ARPANET grew in the first 3 years.

Later the IMP software was changed to allow terminals to connect directly to a special IMP, called a **TIP** (**Terminal Interface Processor**), without having to go through a host. Subsequent changes included having multiple hosts per IMP (to save money), hosts talking to multiple IMPs (to protect against IMP failures),

RED HAT 1057
RED HAT V. CAS

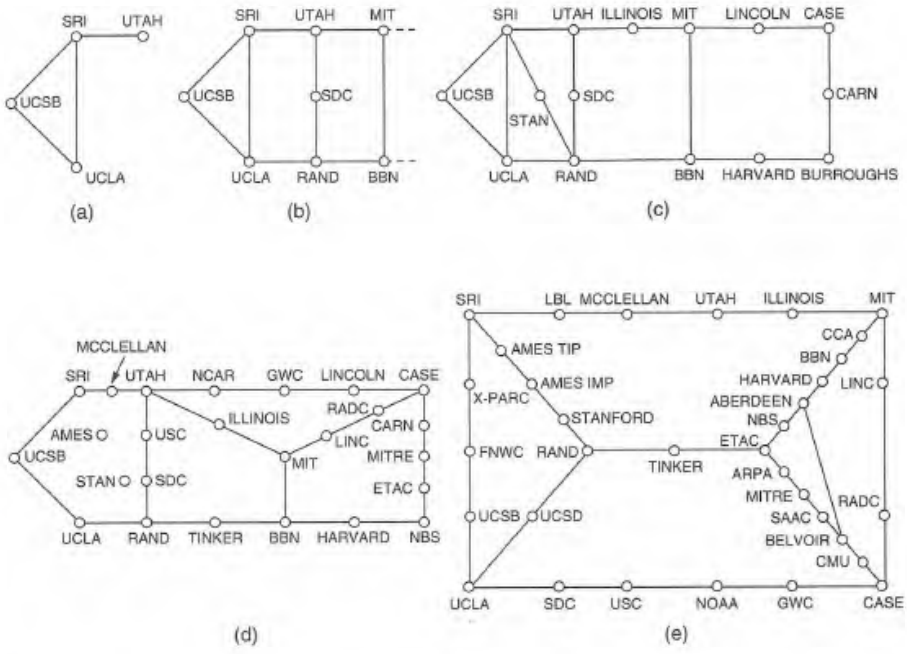


Fig. 1-25. Growth of the ARPANET. (a) Dec. 1969. (b) July 1970. (c) March 1971. (d) April 1972. (e) Sept. 1972.

and hosts and IMPs separated by a large distance (to accommodate hosts far from the subnet).

In addition to helping the fledgling ARPANET grow, ARPA also funded research on satellite networks and mobile packet radio networks. In one 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 multiple 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 being hooked up to the ARPANET.

To encourage adoption of these new protocols, ARPA awarded several contracts to BBN and the University of California at Berkeley to integrate them into Berkeley UNIX. Researchers at Berkeley developed a convenient program

RED HAT 1057
RED HAT V. CAS

interface to the network (sockets) and wrote many application, utility, and management programs to make networking easier.

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.

By 1983, the ARPANET was stable and successful, with over 200 IMPs and hundreds of hosts. At this point, ARPA turned the management of the network over to the Defense Communications Agency (DCA), to run it as an operational network. The first thing DCA did was to separate the military portion (about 160 IMPs, of which 110 in the United States and 50 abroad) into a separate subnet, **MILNET**, with stringent gateways between MILNET and the remaining research subnet.

During the 1980s, additional networks, especially LANs, were connected to the ARPANET. As the scale increased, finding hosts became increasingly expensive, so **DNS (Domain Naming 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.

By 1990, the ARPANET had been overtaken by newer networks that it itself had spawned, so it was shut down and dismantled, but it lives on in the hearts and minds of network researchers everywhere. MILNET continues to operate, however.

1.5.3. 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, which many did not have. This lack of universal access prompted NSF to set up a virtual network, **CSNET**, centered around a single machine at BBN that supported dial-up lines and had connections to the ARPANET and other networks. Using CSNET, academic researchers could call up and leave email for other people to pick up later. It was simple, but it worked.

By 1984 NSF began designing a high-speed 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 as the

RED HAT 1057
RED HAT V. CAS

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 the 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. 1-26.

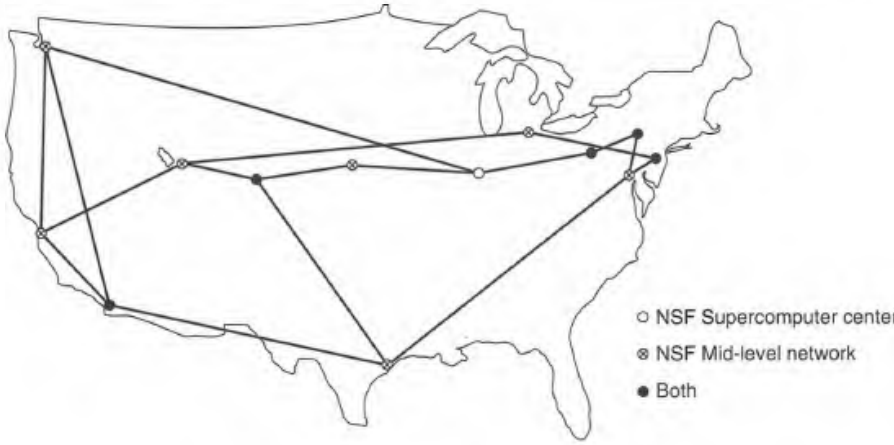


Fig. 1-26. The NSFNET backbone in 1988.

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 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 a 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**.

In December 1991, the U.S. Congress passed a bill authorizing **NREN**, the **National Research and Educational Network**, the research successor to NSFNET, only running at gigabits speeds. The goal was a national network

RED HAT 1057
RED HAT V. CAS

running at 3 Gbps before the millenium. This network is to act as a prototype for the much-discussed information superhighway.

By 1995, the NSFNET backbone was no longer needed to interconnect the NSF regional networks because numerous companies were running commercial IP networks. When ANSNET was sold to America Online in 1995, the NSF regional networks had to go out and buy commercial IP service to interconnect.

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, N.J. 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. In addition to the NSF NAPs, various government NAPs (e.g., FIX-E, FIX-W, MAE-East and MAE-West) and commercial NAPs (e.g., CIX) have also been created, so the concept of a single default backbone was replaced by a commercially-driven competitive infrastructure.

Other countries and regions are also building networks comparable to NSFNET. In Europe, for example, EuropaNET is an IP backbone for research organizations and EBONE is a more commercially oriented network. Both connect numerous cities in Europe with 2-Mbps lines. Upgrades to 34 Mbps are in progress. Each country in Europe has one or more national networks, which are roughly comparable to the NSF regional networks.

1.5.4. The Internet

The number of networks, machines, and users connected to the ARPANET grew rapidly after TCP/IP became the only official protocol on Jan. 1, 1983. When NSFNET and the ARPANET were interconnected, the growth became exponential. Many regional networks joined up, and connections were made to networks in Canada, Europe, and the Pacific.

Sometime in the mid-1980s, people began viewing the collection of networks as an internet, and later as the Internet, although there was no official dedication with some politician breaking a bottle of champagne over a fuzzball.

Growth continued exponentially, and by 1990 the Internet had grown to 3000 networks and 200,000 computers. In 1992, the one millionth host was attached. By 1995, there were multiple backbones, hundreds of mid-level (i.e., regional) networks, tens of thousands of LANs, millions of hosts, and tens of millions of users. The size doubles approximately every year (Paxson, 1994).

RED HAT 1057
RED HAT V. CAS

Much of the growth comes from connecting existing networks to the Internet. In the past these have included SPAN, NASA's space physics network, HEPNET, a high energy physics network, BITNET, IBM's mainframe network, EARN, a European academic network now widely used in Eastern Europe, and many others. Numerous transatlantic links are in use, running from 64 kbps to 2 Mbps.

The glue that holds the Internet together is the TCP/IP reference model and TCP/IP protocol stack. TCP/IP makes universal service possible and can be compared to the telephone system or the adoption of standard gauge by the railroads in the 19th Century.

What does it actually mean to be on the Internet? Our definition is that a machine is on the Internet if it runs the TCP/IP protocol stack, has an IP address, and has the ability to send IP packets to all the other machines on the Internet. The mere ability to send and receive electronic mail is not enough, since email is gatewayed to many networks outside the Internet. However, the issue is clouded somewhat by the fact that many personal computers have the ability to call up an Internet service provider using a modem, be assigned a temporary IP address, and send IP packets to other Internet hosts. It make sense to regard such machines as being on the Internet for as long as they are connected to the service provider's router.

With exponential growth, the old informal way of running the Internet no longer works. In January 1992, the **Internet Society** was set up, to promote the use of the Internet and perhaps eventually take over managing it.

Traditionally, the Internet had four main applications, as follows:

1. **Email**. The ability to compose, send, and receive electronic mail has been around since the early days of the ARPANET and is enormously popular. Many people get dozens of messages a day and consider it their primary way of interacting with the outside world, far outdistancing the telephone and snail mail. Email programs are available on virtually every kind of computer these days.
2. **News**. Newsgroups are specialized forums in which users with a common interest can exchange messages. Thousands of newsgroups exist, on technical and nontechnical topics, including computers, science, recreation, and politics. Each newsgroup has its own etiquette, style, and customs, and woe be to anyone violating them.
3. **Remote login**. Using the Telnet, Rlogin, or other programs, users anywhere on the Internet can log into any other machine on which they have an account.
4. **File transfer**. Using the FTP program, it is possible to copy files from one machine on the Internet to another. Vast numbers of articles, databases, and other information are available this way.

RED HAT 1057
RED HAT V. CAS

Up until the early 1990s, the Internet was largely populated by academic, government, and industrial researchers. One new application, the **WWW (World Wide Web)** changed all that and brought millions of new, nonacademic users to the net. This application, invented by CERN physicist Tim Berners-Lee, did not change any of the underlying facilities but made them easier to use. Together with the Mosaic viewer, written at the National Center for Supercomputer Applications, the WWW made it possible for a site to set up a number of pages of information containing text, pictures, sound, and even video, with embedded links to other pages. By clicking on a link, the user is suddenly transported to the page pointed to by that link. For example, many companies have a home page with entries pointing to other pages for product information, price lists, sales, technical support, communication with employees, stockholder information, and much more.

Numerous other kinds of pages have come into existence in a very short time, including maps, stock market tables, library card catalogs, recorded radio programs, and even a page pointing to the complete text of many books whose copyrights have expired (Mark Twain, Charles Dickens, etc.). Many people also have personal pages (home pages).

In the first year after Mosaic was released, the number of WWW servers grew from 100 to 7000. Enormous growth will undoubtedly continue for years to come, and will probably be the force driving the technology and use of the Internet into the next millenium.

Many books have been written about the Internet and its protocols. For more information, see (Black, 1995; Carl-Mitchell and Quarterman, 1993; Comer, 1995; and Santifaller, 1994).

1.5.5. Gigabit Testbeds

The Internet backbones operate at megabit speeds, so for people who want to push the technological envelope, the next step is gigabit networking. With each increase in network bandwidth, new applications become possible, and gigabit networks are no exception. In this section we will first say a few words about gigabit applications, mention two of them, and then list some example gigabit testbeds that have been built.

Gigabit networks provide better bandwidth than megabit networks, but not always much better delay. For example, sending a 1-kbit packet from New York to San Francisco at 1 Mbps takes 1 msec to pump the bits out and 20 msec for the transcontinental delay, for a total of 21 msec. A 1-Gbps network can reduce this to 20.001 msec. While the bits go out faster, the transcontinental delay remains the same, since the speed of light in optical fiber (or copper wire) is about 200,000 km/sec, independent of the data rate. Thus for wide area applications in which low delay is critical, going to higher speeds may not help much. Fortunately, for

RED HAT 1057
RED HAT V. CAS

some applications, bandwidth is what counts, and these are the applications for which gigabit networks will make a big difference.

One application is telemedicine. Many people think that a way to reduce medical costs is to reintroduce family doctors and family clinics on a large scale, so everyone has convenient access to first line medical care. When a serious medical problem occurs, the family doctor can order lab tests and medical imaging, such as X-rays, CAT scans, and MRI scans. The test results and images can then be sent electronically to a specialist who then makes the diagnosis.

Doctors are generally unwilling to make diagnoses from computer images unless the quality of the transmitted image is as good as the original image. This requirement means images will probably need 4K × 4K pixels, with 8 bits per pixel (black and white images) or 24 bits per pixel (color images). Since many tests require up to 100 images (e.g., different cross sections of the organ in question), a single series for one patient can generate 40 gigabits. Moving images (e.g., a beating heart) generate even more data. Compression can help some but doctors are leary of it because the most efficient algorithms reduce image quality. Furthermore, all the images must be stored for years but may need to be retrieved at a moment's notice in the event of a medical emergency. Hospitals do not want to become computer centers, so off-site storage combined with high-bandwidth electronic retrieval is essential.

Another gigabit application is the virtual meeting. Each meeting room contains a spherical camera and one or more people. The bit streams from each of the cameras are combined electronically to give the illusion that everyone is in the same room. Each person sees this image using virtual reality goggles. In this way meetings can happen without travel, but again, the data rates required are stupendous.

Starting in 1989, ARPA and NSF jointly agreed to finance a number of university-industry gigabit testbeds, later as part of the NREN project. In some of these, the data rate in each direction was 622 Mbps, so only by counting the data going in both directions do you get a gigabit. This kind of gigabit is sometimes called a "government gigabit." (Some cynics call it a gigabit after taxes.) Below we will briefly mention the first five projects. They have done their job and been shut down, but deserve some credit as pioneers, in the same way the ARPANET does.

1. **Aurora** was a testbed linking four sites in the Northeast: M.I.T., the University of Pennsylvania, IBM's T.J. Watson Lab, and Bellcore (Morristown, N.J.) at 622 Mbps using fiber optics provided by MCI, Bell Atlantic, and NYNEX. Aurora was largely designed to help debug Bellcore's Sunshine switch and IBM's (proprietary) plaNET switch using parallel networks. Research issues included switching technology, gigabit protocols, routing, network control, distributed virtual memory, and collaboration using videoconferencing. For more information, see (Clark et al., 1993).

RED HAT 1057
RED HAT V. CAS

2. **Blanca** was originally a research project called XUNET involving AT&T Bell Labs, Berkeley, and the University of Wisconsin. In 1990 it added some new sites (LBL, Cray Research, and the University of Illinois) and acquired NSF/ARPA funding. Some of it ran at 622 Mbps, but other parts ran at lower speeds. Blanca was the only nationwide testbed; the rest were regional. Consequently, much of the research was concerned with the effects of speed-of-light delay. The interest here was in protocols, especially network control protocols, host interfaces, and gigabit applications such as medical imaging, meteorological modeling, and radio astronomy. For more information, see (Catlett, 1992; and Fraser, 1993).

3. **CASA** was aimed at doing research on supercomputer applications, especially those in which part of the problem ran best on one kind of supercomputer (e.g., a Cray vector supercomputer) and part ran best on a different kind of supercomputer (e.g., a parallel supercomputer). The applications investigated included geology (analyzing Landsat images), climate modeling, and understanding chemical reactions. It operated in California and New Mexico and connected Los Alamos, Cal Tech, JPL, and the San Diego Supercomputer Center.

4. **Nectar** differed from the three testbeds given above in that it was an experimental gigabit MAN running from CMU to the Pittsburgh Supercomputer Center. The designers were interested in applications involving chemical process flowsheeting and operations research, as well as the tools for debugging them.

5. **VISTAnet** was a small gigabit testbed operated in Research Triangle Park, North Carolina, and connecting the University of North Carolina, North Carolina State University, and MCNC. The interest here was in a prototype for a public switched gigabit network with switches having hundreds of gigabit lines, meaning that the switches had to be capable of processing terabits/sec. The scientific research focused on using 3D images to plan radiation therapy for cancer patients, with the oncologist being able to vary the beam parameters and instantaneously see the radiation dosages being delivered to the tumor and surrounding tissue (Ransom, 1992).

1.6. EXAMPLE DATA COMMUNICATION SERVICES

Telephone companies and others have begun to offer networking services to any organization that wishes to subscribe. The subnet is owned by the network operator, providing communication service for the customers' hosts and terminals.

RED HAT 1057
RED HAT V. CAS

Such a system is called a **public network**. It is analogous to, and often a part of, the public telephone system. We already briefly looked at one new service, DQDB, in Fig. 1-4. In the following sections we will study four other example services, SMDS, X.25, frame relay, and broadband ISDN.

1.6.1. SMDS—Switched Multimegabit Data Service

The first service we will look at, **SMDS (Switched Multimegabit Data Service)**, was designed to connect together multiple LANs, typically at the branch offices and factories of a single company. It was designed by Bellcore in the 1980s and deployed in the early 1990s by regional and a few long distance carriers. The goal was to produce a high-speed data service and get it out into the world with a minimum of fuss. SMDS is the first broadband (i.e., high-speed) switched service offered to the public.

To see a situation in which SMDS would be useful, consider a company with four offices in four different cities, each with its own LAN. The company would like to connect all the LANs, so that packets can go from one LAN to another. One solution would be to lease six high-speed lines and fully connect the LANs as shown in Fig. 1-27(a). Such a solution is certainly possible, but expensive.

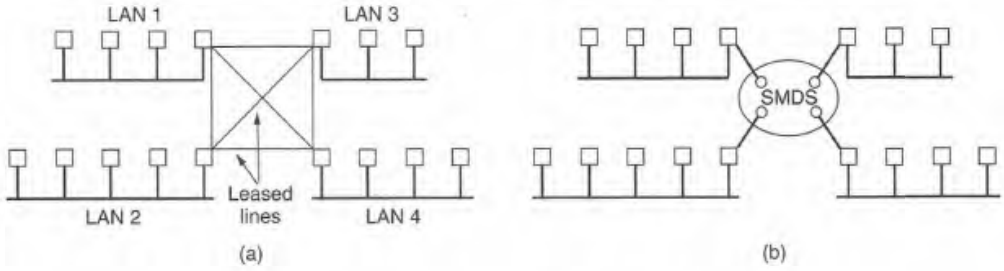


Fig. 1-27. (a) Four LANs interconnected with leased lines. (b) Interconnection using SMDS.

An alternative solution is to use SMDS, as shown in Fig. 1-27(b). The SMDS network acts like a high-speed LAN backbone, allowing packets from any LAN to flow to any other LAN. Between the LANs, in the customer's offices, and the SMDS network, in the telephone company's offices, is a (short) access line leased from the telephone company. Usually, this line is a MAN and uses DQDB, but other options may also be available.

Whereas most telephone company services are designed for continuous traffic, SMDS is designed to handle bursty traffic. In other words, once in a while a packet has to be carried from one LAN to another quickly, but much of the time there is no LAN to LAN traffic. The leased line solution of Fig. 1-27(a) has the problem of high monthly bills; once installed, the customer has to pay for the lines

RED HAT 1057
RED HAT V. CAS

whether or not they are used continuously. For intermittent traffic, leased lines are an expensive solution, and SMDS is priced to compete with them. With n LANs, a fully connected leased line network requires leasing $n(n-1)/2$ possibly long (i.e., expensive) lines, whereas SMDS only requires leasing n short access lines to the nearest SMDS router.

Since the goal of SMDS is to carry LAN to LAN traffic, it must be fast enough to do the job. The standard speed is 45 Mbps, although sometimes lower speed options are available. MANs can also operate at 45 Mbps, but they are not switched, that is, to connect four LANs using a MAN, the telephone company would have to run a single wire from LAN 1 to LAN 2 to LAN 3 to LAN 4, which is only possible if they are in the same city. With SMDS, each LAN connects to a telephone company switch which routes packets through the SMDS network as needed to reach the destination, possibly traversing multiple switches in the process.

The basic SMDS service is a simple connectionless packet delivery service. The packet format is shown in Fig. 1-28. It has three fields: the destination (where the packet is to go to), the source (who sent it), and a variable length payload field for up to 9188 bytes of user data. The machine on the sending LAN that is connected to the access line puts the packet on the access line, and SMDS makes a best effort attempt to deliver it to the correct destination. No guarantee is given.

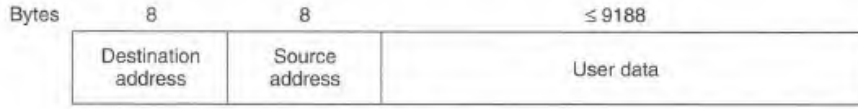


Fig. 1-28. The SMDS packet format.

The source and destination addresses consist of a 4-bit code followed by a telephone number of up to 15 decimal digits. Each digit is coded in a separate 4-bit field. The telephone numbers contain country code, area code, and subscriber number, so the service could eventually be offered internationally. It was thought that having decimal telephone numbers as network addresses would make the new offering seem familiar to nervous users.

When a packet arrives at the SMDS network, the first router checks to make sure that the source address corresponds to the incoming line, to prevent billing fraud. If the address is incorrect, the packet is simply discarded. If it is correct, the packet is sent along toward its destination.

A useful SMDS feature is broadcasting. The customer can specify a list of SMDS telephone numbers, and be assigned a special number for the whole list. Any packet sent to that number is delivered to all members on that list. The National Association of Securities Dealers uses this feature of MCI's SMDS service to broadcast new stock prices to all of its 5000 members.

RED HAT 1057
RED HAT V. CAS

An additional user feature is address screening, on both outgoing and incoming packets. With outgoing screening, the customer can give a list of telephone numbers and specify that no packets may be sent to any other addresses. With incoming screening, only packets from certain pre-arranged telephone numbers will be accepted. When both features are enabled, the user can effectively build a private network with no SMDS connections to the outside world. For companies with confidential data, this feature is highly valuable.

The payload can contain any byte sequence the user wishes, up to 9188 bytes. SMDS does not look at it. It can contain an Ethernet packet, an IBM token ring packet, an IP packet, or anything else. Whatever is present in the payload field is moved without modification from the source LAN to the destination LAN.

SMDS handles bursty traffic as follows. The router connected to each access line contains a counter that is incremented at a constant rate, say once every 10 μsec. When a packet arrives at the router, a check is made to see if the counter is greater than the packet length, in bytes. If it is, the packet is sent without delay and the counter is decremented by the packet length. If the packet length is greater than the counter, the packet is discarded.

In effect, with a tick every 10 μsec the user may send at an *average* rate of 100,000 bytes/sec, but the burst rate may be much higher. If, for example, the line has been idle for 10 msec, the counter will be 1000, and the user will be allowed to send a 1-kilobyte burst at the full 45 Mbps, so it will be transmitted in about 180 μsec. With a 100,000 byte/sec leased line, the same kilobyte would take 10 msec. Thus SMDS offers short delays for widely spaced independent data bursts, as long as the average rate remains below the agreed upon value. This mechanism provides fast response when needed but prevents users from using up more bandwidth than they have agreed to pay for.

1.6.2. X.25 Networks

Many older public networks, especially outside the United States, follow a standard called **X.25**. It was developed during the 1970s by CCITT to provide an interface between public packet-switched networks and their customers.

The physical layer protocol, called **X.21**, specifies the physical, electrical, and procedural interface between the host and the network. Very few public networks actually support this standard, because it requires digital, rather than analog signaling on the telephone lines. As an interim measure, an analog interface similar to the familiar RS-232 standard was defined.

The data link layer standard has a number of (slightly incompatible) variations. They all are designed to deal with transmission errors on the telephone line between the user's equipment (host or terminal) and the public network (router).

The network layer protocol deals with addressing, flow control, delivery confirmation, interrupts, and related issues. Basically, it allows the user to establish virtual circuits and then send packets of up to 128 bytes on them. These packets

RED HAT 1057
RED HAT V. CAS

are delivered reliably and in order. Most X.25 networks work at speeds up to 64 kbps, which makes them obsolete for many purposes. Nevertheless, they are still widespread, so readers should be aware of their existence.

X.25 is connection-oriented and supports both switched virtual circuits and permanent ones. A **switched virtual circuit** is created when one computer sends a packet to the network asking to make a call to a remote computer. Once established, packets can be sent over the connection, always arriving in order. X.25 provides flow control, to make sure a fast sender cannot swamp a slow or busy receiver.

A **permanent virtual circuit** is used the same way as a switched one, but it is set up in advance by agreement between the customer and the carrier. It is always present, and no call setup is required to use it. It is analogous to a leased line.

Because the world is still full of terminals that do not speak X.25, another set of standards was defined that describes how an ordinary (nonintelligent) terminal communicates with an X.25 public network. In effect, the user or network operator installs a "black box" to which these terminals can connect. The black box is called a **PAD (Packet Assembler Disassembler)**, and its function is described in a document known as **X.3**. A standard protocol has been defined between the terminal and the PAD, called **X.28**; another standard protocol exists between the PAD and the network, called **X.29**. Together, these three recommendations are often called **triple X**.

1.6.3. Frame Relay

Frame relay is a service for people who want an absolute bare-bones connection-oriented way to move bits from *A* to *B* at reasonable speed and low cost (Smith, 1993). Its existence is due to changes in technology over the past two decades. Twenty years ago, communication using telephone lines was slow, analog, and unreliable, and computers were slow and expensive. As a result, complex protocols were required to mask errors, and the users' computers were too expensive to have them do this work.

The situation has changed radically. Leased telephone lines are now fast, digital, and reliable, and computers are fast and inexpensive. This suggests the use of simple protocols, with most of the work being done by the users' computers, rather than by the network. It is this environment that frame relay addresses.

Frame relay can best be thought of as a virtual leased line. The customer leases a permanent virtual circuit between two points and can then send frames (i.e., packets) of up to 1600 bytes between them. It is also possible to lease permanent virtual circuits between a given site and multiple other sites, so each frame carries a 10-bit number telling which virtual circuit to use.

The difference between an actual leased line and a virtual leased line is that with an actual one, the user can send traffic all day long at the maximum speed. With a virtual one, data bursts may be sent at full speed, but the long-term average

RED HAT 1057
RED HAT V. CAS

usage must be below a predetermined level. In return, the carrier charges much less for a virtual line than a physical one.

In addition to competing with leased lines, frame relay also competes with X.25 permanent virtual circuits, except that it operates at higher speeds, usually 1.5 Mbps, and provides fewer features.

Frame relay provides a minimal service, primarily a way to determine the start and end of each frame, and detection of transmission errors. If a bad frame is received, the frame relay service simply discards it. It is up to the user to discover that a frame is missing and take the necessary action to recover. Unlike X.25, frame relay does not provide acknowledgements or normal flow control. It does have a bit in the header, however, which one end of a connection can set to indicate to the other end that problems exist. The use of this bit is up to the users.

1.6.4. Broadband ISDN and ATM

Even if the above services become popular, the telephone companies are still faced with a far more fundamental problem: multiple networks. POTS (Plain Old Telephone Service) and Telex use the old circuit-switched network. Each of the new data services such as SMDS and frame relay uses its own packet-switching network. DQDB is different from these, and the internal telephone company call management network (SSN 7) is yet another network. Maintaining all these separate networks is a major headache, and there is another network, cable television, that the telephone companies do not control and would like to.

The perceived solution is to invent a single new network for the future that will replace the entire telephone system and all the specialized networks with a single integrated network for all kinds of information transfer. This new network will have a huge data rate compared to all existing networks and services and will make it possible to offer a large variety of new services. This is not a small project, and it is certainly not going to happen overnight, but it is now under way.

The new wide area service is called **B-ISDN** (**Broadband Integrated Services Digital Network**). It will offer video on demand, live television from many sources, full motion multimedia electronic mail, CD-quality music, LAN interconnection, high-speed data transport for science and industry and many other services that have not yet even been thought of, all over the telephone line.

The underlying technology that makes B-ISDN possible is called **ATM** (**Asynchronous Transfer Mode**) because it is not synchronous (tied to a master clock), as most long distance telephone lines are. Note that the acronym ATM here has nothing to do with the Automated Teller Machines many banks provide (although an ATM machine can use an ATM network to talk to its bank).

A great deal of work has already been done on ATM and on the B-ISDN system that uses it, although there is more ahead. For more information on this subject, see (Fischer et al., 1994; Gasman, 1994; Goralski, 1995; Kim et al., 1994; Kyas, 1995; McDysan and Spohn, 1995; and Stallings, 1995a).

RED HAT 1057
RED HAT V. CAS

The basic idea behind ATM is to transmit all information in small, fixed-size packets called **cells**. The cells are 53 bytes long, of which 5 bytes are header and 48 bytes are payload, as shown in Fig. 1-29. ATM is both a technology (hidden from the users) and potentially a service (visible to the users). Sometimes the service is called **cell relay**, as an analogy to frame relay.

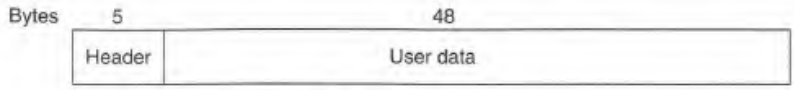


Fig. 1-29. An ATM cell.

The use of a cell-switching technology is a gigantic break with the 100-year old tradition of circuit switching (establishing a copper path) within the telephone system. There are a variety of reasons why cell switching was chosen, among them are the following. First, cell switching is highly flexible and can handle both constant rate traffic (audio, video) and variable rate traffic (data) easily. Second, at the very high speeds envisioned (gigabits per second are within reach), digital switching of cells is easier than using traditional multiplexing techniques, especially using fiber optics. Third, for television distribution, broadcasting is essential; cell switching can provide this and circuit switching cannot.

ATM networks are connection-oriented. Making a call requires first sending a message to set up the connection. After that, subsequent cells all follow the same path to the destination. Cell delivery is not guaranteed, but their order is. If cells 1 and 2 are sent in that order, then if both arrive, they will arrive in that order, never first 2 then 1.

ATM networks are organized like traditional WANs, with lines and switches (routers). The intended speeds for ATM networks are 155 Mbps and 622 Mbps, with the possibility of gigabit speeds later. The 155-Mbps speed was chosen because this is about what is needed to transmit high definition television. The exact choice of 155.52 Mbps was made for compatibility with AT&T's SONET transmission system. The 622 Mbps speed was chosen so four 155-Mbps channels could be sent over it. By now it should be clear why some of the gigabit testbeds operated at 622 Mbps: they used ATM.

When ATM was proposed, virtually all the discussion (i.e., the hype) was about video on demand to every home and replacing the telephone system, as described above. Since then, other developments have become important. Many organizations have run out of bandwidth on their campus or building-wide LANs and are being forced to go to some kind of switched system that has more bandwidth than does a single LAN. Also, in client-server computing, some applications need the ability to talk to certain servers at high speed. ATM is certainly a major candidate for both of these applications. Nevertheless, it is a bit of a letdown to go from a goal of trying to replace the entire low-speed analog telephone

RED HAT 1057
RED HAT V. CAS

system with a high-speed digital one to a goal of trying connect all the Ethernets on campus. LAN interconnection using ATM is discussed in (Kavak, 1995; Newman, 1994; and Truong et al., 1995).

It is also worth pointing out that different organizations involved in ATM have different (financial) interests. The long-distance telephone carriers and PTTs are mostly interested in using ATM to upgrade the telephone system and compete with the cable TV companies in electronic video distribution. The computer vendors see campus ATM LANs as the big moneymaker (for them). All these competing interests do not make the ongoing standardization process any easier, faster, or more coherent. Also, politics and power within the organization standardizing ATM (The ATM Forum) have considerable influence on where ATM is going.

The B-ISDN ATM Reference Model

Let us now turn back to the technology of ATM, especially as used in the (future) telephone system. Broadband ISDN using ATM has its own reference model, different from the OSI model and also different from the TCP/IP model. This model is shown in Fig. 1-30. It consists of three layers, the physical, ATM, and ATM adaptation layers, plus whatever the users want to put on top of that.

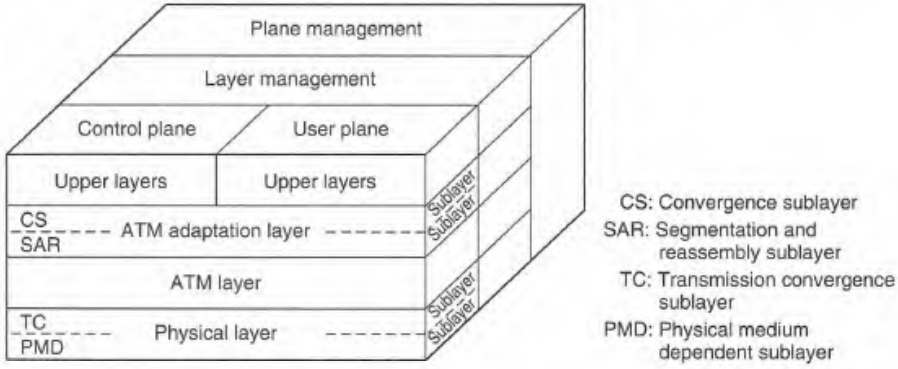


Fig. 1-30. The B-ISDN ATM reference model.

The physical layer deals with the physical medium: voltages, bit timing, and various other issues. ATM does not prescribe a particular set of rules, but instead says that ATM cells may be sent on a wire or fiber by themselves, but they may also be packaged inside the payload of other carrier systems. In other words, ATM has been designed to be independent of the transmission medium.

The **ATM layer** deals with cells and cell transport. It defines the layout of a cell and tells what the header fields mean. It also deals with establishment and release of virtual circuits. Congestion control is also located here.

RED HAT 1057
RED HAT V. CAS

Because most applications do not want to work directly with cells (although some may), a layer above the ATM layer has been defined that allows users to send packets larger than a cell. The ATM interface segments these packets, transmits the cells individually, and reassembles them at the other end. This layer is the **AAL (ATM Adaptation Layer)**.

Unlike the earlier two-dimensional reference models, the ATM model is defined as being three-dimensional, as shown in Fig. 1-30. The **user plane** deals with data transport, flow control, error correction, and other user functions. In contrast, the **control plane** is concerned with connection management. The layer and plane management functions relate to resource management and interlayer coordination.

The physical and AAL layers are each divided into two sublayers, one at the bottom that does the work and a convergence sublayer on top that provides the proper interface to the layer above it. The functions of the layers and sublayers are given in Fig. 1-31.

OSI layer	ATM layer	ATM sublayer	Functionality
3/4	AAL	CS	Providing the standard interface (convergence)
		SAR	Segmentation and reassembly
2/3	ATM		Flow control Cell header generation/extraction Virtual circuit/path management Cell multiplexing/demultiplexing
2	Physical	TC	Cell rate decoupling Header checksum generation and verification Cell generation Packing/unpacking cells from the enclosing envelope Frame generation
1		PMD	Bit timing Physical network access

Fig. 1-31. The ATM layers and sublayers, and their functions.

The **PMD (Physical Medium Dependent)** sublayer interfaces to the actual cable. It moves the bits on and off and handles the bit timing. For different carriers and cables, this layer will be different.

The other sublayer of the physical layer is the **TC (Transmission Convergence)** sublayer. When cells are transmitted, the TC layer sends them as a string of bits to the PMD layer. Doing this is easy. At the other end, the TC sublayer gets a pure incoming bit stream from the PMD sublayer. Its job is to convert this

RED HAT 1057
RED HAT V. CAS

bit stream into a cell stream for the ATM layer. It handles all the issues related to telling where cells begin and end in the bit stream. In the ATM model, this functionality is in the physical layer. In the OSI model and in pretty much all other networks, the job of framing, that is, turning a raw bit stream into a sequence of frames or cells, is the data link layer's task. For that reason we will discuss it in this book along with the data link layer, not with the physical layer.

As we mentioned earlier, the ATM layer manages cells, including their generation and transport. Most of the interesting aspects of ATM are located here. It is a mixture of the OSI data link and network layers, but it is not split into sublayers.

The AAL layer is split into a **SAR** (**Segmentation And Reassembly**) sublayer and a **CS** (**Convergence Sublayer**). The lower sublayer breaks packets up into cells on the transmission side and puts them back together again at the destination. The upper sublayer makes it possible to have ATM systems offer different kinds of services to different applications (e.g., file transfer and video on demand have different requirements concerning error handling, timing, etc.).

Perspective on ATM

To a considerable extent, ATM is a project invented by the telephone industry because after Ethernet was widely installed, the computer industry never rallied around any higher-speed network technology to make it standard. The telephone companies filled this vacuum with ATM, although in October 1991, many computer vendors joined with the telephone companies to set up the **ATM Forum**, an industry group that will guide the future of ATM.

Although ATM promises the ability to deliver information anywhere at speeds soon to exceed 1 Gbps, delivering on this promise will not be easy. ATM is basically high-speed packet-switching, a technology the telephone companies have little experience with. What they do have, is a massive investment in a different technology (circuit switching) that is in concept unchanged since the days of Alexander Graham Bell. Needless to say, this transition will not happen quickly, all the more so because it is a revolutionary change rather than an evolutionary one, and revolutions never go smoothly.

The economics of installing ATM worldwide also have to be considered. A substantial fraction of the existing telephone system will have to be replaced. Who will pay for this? How much will consumers be willing to pay to get a movie on demand electronically, when they can get one at the local video store for a couple of dollars? Finally, the question of where many of the advanced services are provided is crucial. If they are provided by the network, the telephone companies will profit from them. If they are provided by computers attached to the network, the manufacturers and operators of these devices make the profits. The users may not care, but the telephone companies and computer vendors certainly do, and this will surely affect their interest in making ATM happen.

RED HAT 1057
RED HAT V. CAS

1.6.5. Comparison of Services

The reader may be wondering why so many incompatible and overlapping services exist, including DQDB, SMDS, X.25, frame relay, ATM, and more. The underlying reason is the 1984 decision to break up AT&T and foster competition in the telecommunications industry. Different companies with different interests and technologies are now free to offer whatever services they think there is a demand for, and many of them are doing this with a vengeance.

To recap some of the services we have touched on in this chapter, DQDB is an unswitched MAN technology that allows 53-byte cells (of which 44 are payload) to be sent down long wires within a city. SMDS is a switched datagram technology for sending datagrams anywhere in a network at 45 Mbps. X.25 is an older connection-oriented networking technology for transmitting small variable-sized packets at 64 kbps. Frame relay is a service that provides virtual leased lines at speeds around 1.5 Mbps. Finally, ATM is designed to replace the entire circuit-switched telephone system with cell switching and be able to handle data and television as well. Some differences between these competitors are summarized in Fig. 1-32.

Issue	DQDB	SMDS	X.25	Frame Relay	ATM AAL
Connection oriented	Yes	No	Yes	Yes	Yes
Normal speed (Mbps)	45	45	.064	1.5	155
Switched	No	Yes	Yes	No	Yes
Fixed-size payload	Yes	No	No	No	No
Max payload	44	9188	128	1600	Variable
Permanent VCs	No	No	Yes	Yes	Yes
Multicasting	No	Yes	No	No	Yes

Fig. 1-32. Different networking services.

1.7. NETWORK STANDARDIZATION

Many network vendors and suppliers exist, each with their own ideas of how things should be done. Without coordination, there would be complete chaos, and users would be able to get nothing done. The only way out is to agree upon some network standards.

Not only do standards allow different computers to communicate, but they also increase the market for products adhering to the standard, which leads to

RED HAT 1057
RED HAT V. CAS

mass production, economies of scale in manufacturing, VLSI implementations, and other benefits that decrease price and further increase acceptance. In the following sections we will take a quick look at the important, but little-known, world of international standardization.

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. The IBM PC and its successors are de facto standards for small office computers because dozens of manufacturers have chosen to copy IBM's machines very closely. UNIX is the de facto standard for operating systems in university computer science departments.

De jure (Latin for "by law") standards, in contrast, are formal, legal standards adopted by some authorized standardization body. International standardization authorities are generally divided into two classes: those established by treaty among national governments, and voluntary, nontreaty organizations. In the area of computer network standards, there are several organizations of each type, which are discussed below.

1.7.1. 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 1500 separate, privately owned telephone companies. Before it was broken up in 1984, AT&T, at that time the world's largest corporation, completely dominated the scene. It provided telephone service to about 80 percent of America's telephones, spread throughout half of its geographical area, with all the other companies combined servicing the remaining (mostly rural) customers. Since the breakup, AT&T continues to provide long-distance service, although now in competition with other companies. The seven Regional Bell Operating Companies that were split off from AT&T and 1500 independents provide local and cellular telephone service. Some of these independents, such as GTE, are very large companies.

Companies in the United States that provide communication services to the public are called **common carriers**. Their offerings and prices are described by a document called a **tariff**, which must be approved by the Federal Communications Commission for the interstate and international traffic, and by the state public utilities commissions for intrastate traffic.

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 as well. Most of the world falls in 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.

RED HAT 1057
RED HAT V. CAS

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**). ITU's job was standardizing 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 as well. In 1947, ITU became an agency of the United Nations.

ITU has three main sectors:

1. Radiocommunications Sector (ITU-R).
2. Telecommunications Standardization Sector (ITU-T).
3. Development Sector (ITU-D).

ITU-R is concerned with allocating radio frequencies worldwide to the competing interest groups. We will be primarily concerned with ITU-T, which is concerned with telephone and data communication systems. From 1956 to 1993, ITU-T was known as **CCITT**, an acronym for its French name: Comité Consultatif International Télégraphique et Téléphonique. On March 1, 1993, CCITT was reorganized to make it less bureaucratic and renamed to reflect its new role. Both ITU-T and CCITT issued recommendations in the area of telephone and data communications. One still frequently runs into CCITT recommendations, such as CCITT X.25, although since 1993 recommendations bear the ITU-T label.

ITU-T has five classes of members:

1. Administrations (national PTTs).
2. Recognized private operators (e.g., AT&T, MCI, British Telecom).
3. Regional telecommunications organizations (e.g., the European ETSI).
4. Telecommunications vendors and scientific organizations.
5. Other interested organizations (e.g., banking and airline networks).

ITU-T has about 200 administrations, 100 private operators, and several hundred other members. Only administrations may vote, but all members may participate in ITU-T's work. Since the United States does not have a PTT, somebody else had to represent it in ITU-T. This task fell to the State Department, probably on the grounds that ITU-T had to do with foreign countries, the State Department's specialty.

ITU-T's task is to make technical recommendations about telephone, telegraph, and data communication interfaces. These often become internationally

RED HAT 1057
RED HAT V. CAS

recognized standards, for example, V.24 (also known as EIA RS-232 in the United States), which specifies the placement and meaning of the various pins on the connector used by most asynchronous terminals.

It should be noted that ITU-T recommendations are technically only suggestions that governments can adopt or ignore, as they wish. In practice, a country that wishes to adopt a different telephone standard than 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 Albania, but elsewhere it would be a real problem. The fiction of calling ITU-T standards "recommendations" was and is necessary to keep nationalist forces in many countries placated.

The real work of ITU-T is done in Study Groups, often as large as 400 people. 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 gets things done. Its current output runs to about 5000 pages of recommendations a year. The members chip in to cover ITU's costs. Big, rich countries are supposed to pay up to 30 contributory units a year; small, poor ones can get away with 1/16 of a contributory unit (a contributory unit is about 250,000 dollars). It is a testimony to ITU-T's value that pretty much everyone pays their fair share, even though contributions are completely voluntary.

As 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).

1.7.2. Who's Who in the International Standards World

International standards are produced by **ISO (International Standards Organization**†), a voluntary, nontreaty organization founded in 1946. Its members are the national standards organizations of the 89 member countries. These members include ANSI (U.S.), BSI (Great Britain), AFNOR (France), DIN (Germany), and 85 others.

ISO issues standards on a vast number of subjects, ranging from nuts and bolts (literally) to telephone pole coatings. Over 5000 standards have been issued, including the OSI standards. ISO has almost 200 Technical Committees, numbered in the order of their creation, each dealing with a specific subject. TC1 deals with the nuts and bolts (standardizing screw thread pitches). TC97 deals with computers and information processing. Each TC has subcommittees (SCs) divided into working groups (WGs).

The real work is done largely in the WGs by over 100,000 volunteers

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

RED HAT 1057
RED HAT V. CAS

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.

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.

The U.S. representative in ISO is **ANSI (American National Standards Institute)**, which despite its name, is a private, nongovernmental, nonprofit organization. Its members are manufacturers, common carriers, and other interested parties. ANSI standards are frequently adopted by ISO as international standards.

The procedure used by ISO for adopting standards is 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 an agency of the U.S. Dept. of Commerce. It was formerly known as 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 has 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 numerous conferences each year, IEEE has a standardization group that develops standards in the area of electrical engineering and computing. IEEE's 802 standard for local area networks is the key standard for LANs. It has subsequently been taken over by ISO as the basis for ISO 8802.

1.7.3. 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. The people who come to Internet standardization meetings wear either jeans or military uniforms.

ITU-T and ISO meetings are populated by corporate officials and government

RED HAT 1057
RED HAT V. CAS

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, definitely prefer anarchy as a matter of principle, but sometimes agreement is needed to make things work. Thus standards, however regrettable, are occasionally needed.

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 given a slighter broader mission, namely, to keep the researchers involved with the ARPANET and 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 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 on-line and can be fetched by anyone interested in them. They are numbered in chronological order of creation. Close to 2000 now exist.

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 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 together 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 completely

RED HAT 1057
RED HAT V. CAS

explained in an RFC and have sufficient interest in the community to warrant consideration. To advance to the **Draft Standard** stage, there must be a working implementation that has been thoroughly tested by at least two independent sites for 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. David Clark once made a now-famous remark about Internet standardization consisting of "rough consensus and running code."

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. Two networks are used as running examples throughout the text: the Internet and ATM networks. In a way, the two are complementary: ATM is mostly concerned with the lower layers, and the Internet is mostly concerned with upper layers. In the future, the Internet may run largely on an ATM backbone, so both of them may coexist. Other examples will be given where relevant.

The book is structured according to the hybrid model of Fig. 1-21. Starting with Chap. 2, we begin working our way up the protocol hierarchy beginning at the bottom. The second chapter provides some background in the field of data communication. It covers analog and digital transmission, multiplexing, switching, and the telephone system, past current, and future. This material is concerned with the physical layer, although we cover only the architectural rather than the hardware aspects. Several examples of the physical layer are also discussed, such as SONET and cellular radio.

Chap. 3 discusses the data link layer and its protocols by means of a number of increasingly complex examples. The analysis of these protocols is also covered. After that, some important real-world protocols are discussed, including HDLC (used in low- and medium-speed networks), SLIP, and PPP (used in the Internet), and ATM (used in B-ISDN).

Chap. 4 concerns the medium access sublayer, which is part of the data link layer. The basic question it deals with is how to determine who may use the network next when the network consists of a single shared channel, as in most LANs and some satellite networks. Many examples are given from the areas of LANs, fiber optic networks, and satellite networks. Bridges, which are used to connect LANs together, are also discussed here.

Chap. 5 deals with the network layer, especially routing, congestion control, and internetworking. It discusses both static and dynamic routing algorithms. Broadcast routing is also covered. The effect of poor routing, congestion, is

RED HAT 1057
RED HAT V. CAS

discussed in some detail. Connecting heterogeneous networks together to form internetworks leads to numerous problems that are discussed here. The network layers in the Internet and ATM networks are given extensive coverage.

Chap. 6 deals with the transport layer. Much of the emphasis is on connection-oriented protocols, since many applications need these. An example transport service and its implementation are discussed in detail. Both the Internet transport protocols (TCP and UDP) and the ATM transport protocols (AAL 1-5) are covered in detail.

The OSI session and presentation layers are not discussed in this book as they are not widely used for anything.

Chapter 7 deals with the application layer, its protocols and applications. Among the applications covered are security, naming, electronic mail, net news, network management, the World Wide Web, and multimedia.

Chap. 8 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 references cited in this book.

1.9. SUMMARY

Computer networks can be used for numerous services, both for companies and for individuals. For companies, networks of personal computers using shared servers often provide flexibility and a good price/performance ratio. For individuals, networks offer access to a variety of information and entertainment resources.

Roughly speaking, networks can be divided up into LANs, MANs, WANs, and internetworks, each with their own characteristics, technologies, speeds, and niches. LANs cover a building, MANs cover a city, and WANs cover a country or continent. LANs and MANs are unswitched (i.e., do not have routers); WANs are switched.

Network software consists of protocols, or rules by which processes can communicate. Protocols can be either connectionless or connection-oriented. Most networks support protocol hierarchies, with each layer providing services to the layers 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 the TCP/IP model. Both of these have network, transport, and application layers, but they differ on the other layers.

Well-known networks have included Novell's NetWare, the ARPANET (now defunct), NSFNET, the Internet, and various gigabit testbeds. Network services have included DQDB, SMDS, X.25, frame relay, and broadband ISDN. All of these are available commercially, from a variety of suppliers. The marketplace will determine which ones will survive and which ones will not.

RED HAT 1057
RED HAT V. CAS

PROBLEMS

1. 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.
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. A collection of five routers is 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 to find the one that best matches the expected load?
4. A group of $2^n - 1$ routers are interconnected in a centralized binary tree, with a router at each tree node. Router i communicates with router j by sending a message to the root of the tree. The root then sends the message back down to j. Derive an approximate expression for the mean number of hops per message for large n, assuming that all router pairs are equally likely.
5. A disadvantage of a broadcast subnet is the capacity wasted due to multiple hosts attempting 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 are wasted due to collisions?
6. What are the SAP addresses in FM radio broadcasting?
7. What is the principal difference between connectionless communication and connection-oriented communication?
8. 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.
9. What is the difference between a confirmed service and an unconfirmed service? For each of the following, tell whether it might be a confirmed service, an unconfirmed service, both, or neither.
   (a) Connection establishment.
   (b) Data transmission.
   (c) Connection release.
10. What does "negotiation" mean when discussing network protocols? Give an example of it.
11. What are two reasons for using layered protocols?
12. List two ways in which the OSI reference model and the TCP/IP reference model are the same. Now list two ways in which they differ.

RED HAT 1057
RED HAT V. CAS

13. The president of the Specialty Paint Corp. gets the idea to work together with a local beer brewer for the purpose of producing 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 other company 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. Is this an example of a multilayer protocol in the sense of the OSI model?

14. In most networks, the data link layer handles transmission errors by requesting damaged frames to 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 if acknowledgements are never lost?

15. Which of the OSI layers handles each of the following:
(a) Breaking the transmitted bit stream into frames.
(b) Determining which route through the subnet to use.

16. Do TPDUs encapsulate packets or the other way around? Discuss.

17. 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?

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

19. Does the Novell NetWare architecture look more like X.25 or like the Internet? Explain your answer.

20. 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 7 million in January 1996. Use these data to compute the expected number of Internet hosts in the year 2008.

21. Why was SMDS designed as a connectionless network and frame relay as a connection-oriented one?

22. Imagine that you have trained your St. Bernard, Bernie, to carry a box of three 8mm Exabyte 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 155-Mbps ATM line?

23. When transferring a file between two computers, (at least) 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.

24. Imagine that the SMDS packet of Fig. 1-28 were to be incorporated in OSI protocol hierarchy. In which layer would it appear?

25. Give an advantage and a disadvantage of frame relay over a leased telephone line.

RED HAT 1057
RED HAT V. CAS

26. Why does ATM use small, fixed-length cells?

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

28. When a system has a permanent part and a removable part, such as a diskette drive and the diskette, it is important that the system be standardized, so that different companies can make both the permanent and removable parts and have everything work 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.

RED HAT 1057
RED HAT V. CAS

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. For more information about it, see (Huitema, 1995; and Perlman, 1992).

To achieve its goals, the network layer must know about the topology of the communication subnet (i.e., the set of all routers) and choose appropriate paths through it. It must also take care to choose 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, it is up to the network layer to deal with these differences and solve the problems that result from them. In this chapter we will study all these issues and illustrate them with our two running examples, the Internet and ATM.

5.1. NETWORK LAYER DESIGN ISSUES

In the following sections we will provide 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 subnet.

RED HAT 1057
RED HAT V. CAS

5.1.1. Services Provided to the Transport Layer

The network layer provides services to the transport layer at the network layer/transport layer interface. This interface is often especially important for another reason: it frequently is the interface between the carrier and the customer, that is, the boundary of the subnet. The carrier often has control of the protocols and interfaces up to and including the network layer. Its job is to deliver packets given to it by its customers. For this reason, this interface must be especially well defined.

The network layer services have been designed with the following goals in mind.

1. The services should be independent of the subnet technology.
2. The transport layer should be shielded from the number, type, and topology of the subnets 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 the question of whether the network layer should provide connection-oriented service or connectionless service.

One camp (represented by the Internet community) argues that the subnet's job is moving bits around and nothing else. In their view (based on nearly 30 years of actual experience with a real, working computer network), the subnet is inherently unreliable, no matter how it is designed. Therefore, the hosts should accept the fact that it is unreliable and do error control (i.e., error detection and correction) and flow control themselves.

This viewpoint leads quickly 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 probably little to be gained by doing it twice. 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 subnet should provide a (reasonably) reliable, connection-oriented service. They claim 100 years of successful experience with the worldwide telephone system is a good guide. In this view, connections should have the following properties:

1. Before sending data, a network layer process on the sending side must set up a connection to its peer on the receiving side. This connection, which is given a special identifier, is then used until all the data have been sent, at which time it is explicitly released.

RED HAT 1057
RED HAT V. CAS

2. When a connection is set up, the two processes can enter into a negotiation about the parameters, quality, and cost of the service to be provided.
3. Communication is in both directions, and packets are delivered in sequence.
4. Flow control is provided automatically to prevent a fast sender from dumping packets into the pipe at a higher rate than the receiver can take them out, thus leading to overflow.

Other properties, such as guaranteed delivery, explicit confirmation of delivery, and high priority packets are optional. As we pointed out in Chap. 1, connectionless service is like the postal system, and connection-oriented service is like the telephone system.

The argument between connection-oriented and connectionless service really has to do with where to put the complexity. In the connection-oriented service, it is in the network layer (subnet); in the connectionless service, it is in the transport layer (hosts). Supporters of connectionless service say that user computing power has become cheap, so that there is no reason not to put the complexity in the hosts. Furthermore, they argue that the subnet is a major (inter)national investment that will last for decades, so it should not be cluttered up with features that may become obsolete quickly but will have to be calculated into the price structure for many years. Furthermore, some applications, such as digitized voice and real-time data collection may regard *speedy* delivery as much more important than *accurate* delivery.

On the other hand, supporters of connection-oriented service say that most users are not interested in running complex transport layer protocols in their machines. What they want is reliable, trouble-free service, and this service can be best provided with network layer connections. Furthermore, some services, such as real time audio and video are much easier to provide on top of a connection-oriented network layer than on top of a connectionless network layer.

Although it is rarely discussed in these terms, two separate issues are involved here. First, whether the network is connection-oriented (setup required) or connectionless (no setup required). Second, whether it is reliable (no lost, duplicated, or garbled packets) or unreliable (packets can be lost, duplicated, or garbled). In theory, all four combinations exist, but the dominant combinations are reliable connection-oriented and unreliable connectionless, so the other two tend to get lost in the noise.

These two camps are represented by our two running examples. The Internet has a connectionless network layer, and ATM networks have a connection-oriented network layer. An obvious question arises about how the Internet works when it runs over an ATM-based, carrier-provided subnet. The answer is that the source host first establishes an ATM network layer connection to the destination

RED HAT 1057
RED HAT V. CAS

host and then sends independent (IP) packets over it, as shown in Fig. 5-1. Although this approach works, it is inefficient because certain functionality is in both layers. For example, the ATM network layer guarantees that packets are always delivered in order, but the TCP code still contains the full mechanism for managing and reordering out-of-order packets. For more information about how to run IP over ATM, see RFC 1577 and (Armitage and Adams, 1995).

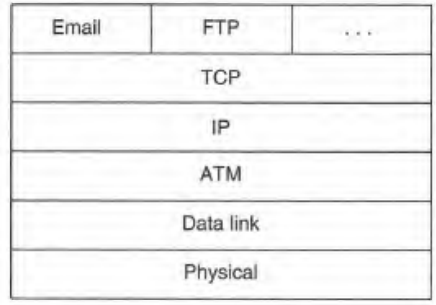


Fig. 5-1. Running TCP/IP over an ATM subnet.

5.1.2. Internal Organization of the Network Layer

Having looked at the two classes of service the network layer can provide to its users, it is time to see how it works inside. There are basically two different philosophies for organizing the subnet, one using connections and the other working connectionless. In the context of the *internal* operation of the subnet, a connection is usually called a **virtual circuit**, in analogy with the physical circuits set up by the telephone system. The independent packets of the connectionless organization are called **datagrams**, in analogy with telegrams.

Virtual circuits are generally used in subnets whose primary service is connection-oriented, so we will describe them in that context. The idea behind virtual circuits is to avoid having to choose a new route for every packet or cell sent. 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 remembered. 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.

In contrast, with a datagram subnet no routes are worked out in advance, even if the service is connection-oriented. Each packet sent is routed independently of its predecessors. Successive packets may follow different routes. While datagram subnets have to do more work, they are also generally more robust and adapt to failures and congestion more easily than virtual circuit subnets. We will discuss the pros and cons of the two approaches later.

RED HAT 1057
RED HAT V. CAS

If packets flowing over a given virtual circuit always take the same route through the subnet, each router must remember where to forward packets for each of the currently open virtual circuits passing through it. Every router must maintain a table with one entry per open virtual circuit passing through it. Each packet traveling through the subnet must contain a virtual circuit number field in its header, in addition to sequence numbers, checksums, and the like. When a packet arrives at a router, the router knows on which line it arrived and what the virtual circuit number is. Based on only this information, the packet must be forwarded on the correct output line.

When a network connection is set up, a virtual circuit number not already in use on that machine is chosen as the connection identifier. Since each machine chooses virtual circuit numbers independently, these numbers have only local significance. If they were globally significant over the whole network, it is likely that two virtual circuits bearing the same global virtual circuit number might pass through some intermediate router, leading to ambiguities.

Because virtual circuits can be initiated from either end, a problem occurs when call setups are propagating in both directions at once along a chain of routers. At some point they have arrived at adjacent routers. Each router must now pick a virtual circuit number to use for the (full-duplex) circuit it is trying to establish. If they have been programmed to choose the lowest number not already in use on the link, they will pick the same number, causing two unrelated virtual circuits over the same physical line to have the same number. When a data packet arrives later, the receiving router has no way of telling whether it is a forward packet on one circuit or a reverse packet on the other. If circuits are simplex, there is no ambiguity.

Note that every process must be required to indicate when it is through using a virtual circuit, so that the virtual circuit can be purged from the router tables to recover the space. In public networks, the motivation is the stick rather than the carrot: users are invariably charged for connect time as well as for data transported. In addition, some provision must be made for dealing with machines that terminate their virtual circuits by crashing rather than politely releasing them when done.

So much for the use of virtual circuits internal to the subnet. The other possibility is to use datagrams internally, in which case the routers do not have a table with one entry for each open virtual circuit. Instead, they have a table telling which outgoing line to use for each possible destination router. These tables are also needed when virtual circuits are used internally, to determine the route for a setup packet.

Each datagram must contain the full destination address. For a large network, these addresses can be quite long (e.g., a dozen bytes or more). When a packet comes in, the router looks up the outgoing line to use and sends the packet on its way. Also, the establishment and release of network or transport layer connections do not require any special work on the part of the routers.

RED HAT 1057
RED HAT V. CAS

5.1.3. Comparison of Virtual Circuit and Datagram Subnets

Both virtual circuits and datagrams have their supporters and their detractors. We will now attempt to summarize the arguments both ways. The major issues are listed in Fig. 5-2, although purists could probably find a counterexample for everything in the figure.

Issue	Datagram subnet	VC subnet
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Subnet does not hold state information	Each VC requires subnet table space
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow this route
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Congestion control	Difficult	Easy if enough buffers can be allocated in advance for each VC

Fig. 5-2. Comparison of datagram and virtual circuit subnets.

Inside the subnet, several trade-offs exist between virtual circuits and datagrams. One trade-off is between router memory space and bandwidth. Virtual circuits allow packets to contain circuit numbers instead of full destination addresses. If the packets tend to be fairly short, a full destination address in every packet may represent a significant amount of overhead, and hence wasted bandwidth. The price paid for using virtual circuits internally is the table space within the routers. Depending upon the relative cost of communication circuits versus router memory, one or the other may be cheaper.

Another trade-off is setup time versus address parsing time. Using virtual circuits requires a setup phase, which takes time and consumes resources. However, figuring out what to do with a data packet in a virtual circuit subnet is easy: the router just uses the circuit number to index into a table to find out where the packet goes. In a datagram subnet, a more complicated procedure is required to determine where the packet goes.

Virtual circuits have some advantages in avoiding congestion within the

RED HAT 1057
RED HAT V. CAS

subnet because resources 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 subnet, 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 switched virtual circuits inside the subnet makes little sense. On the other hand, permanent virtual circuits, which are set up manually and last for months or years, may be useful here.

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 up in the router at the time will suffer, and maybe not even all those, depending upon whether they have already been acknowledged or not. The loss of a communication line is fatal to virtual circuits using it but can be easily compensated for if datagrams are used. Datagrams also allow the routers to balance the traffic throughout the subnet, since routes can be changed halfway through a connection.

It is worth explicitly pointing out that the service offered (connection-oriented or connectionless) is a separate issue from the subnet structure (virtual circuit or datagram). In theory, all four combinations are possible. Obviously, a virtual circuit implementation of a connection-oriented service and a datagram implementation of a connectionless service are reasonable. Implementing connections using datagrams also makes sense when the subnet is trying to provide a highly robust service.

The fourth possibility, a connectionless service on top of a virtual circuit subnet, seems strange but certainly occurs. The obvious example is running IP over an ATM subnet. Here it is desired to run an existing connectionless protocol over a new connection-oriented network layer. As mentioned earlier, this is more of an ad hoc solution to a problem than a good design. In a new system designed to run over an ATM subnet, one would not normally put a connectionless protocol like IP over a connection-oriented network layer like ATM and then layer a connection-oriented transport protocol on top of the connectionless protocol. Examples of all four cases are shown in Fig. 5-3.

5.2. ROUTING ALGORITHMS

The main function of the network layer is routing packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to make the journey. The only notable exception is for broadcast

RED HAT 1057
RED HAT V. CAS

Upper layer	Type of subnet: Datagram	Type of subnet: Virtual circuit
Connectionless	UDP over IP	UDP over IP over ATM
Connection-oriented	TCP over IP	ATM AAL1 over ATM

Fig. 5-3. Examples of different combinations of service and subnet structure.

networks, but even here routing is an issue if the source and destination are not on the same network. 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 subnet 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 subnet uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the previously established route. The latter case is sometimes called **session routing**, because a route remains in force for an entire user session (e.g., a login session at a terminal or a file transfer).

Regardless of whether routes are chosen independently for each packet or only when new connections are established, there are certain properties that are desirable in a routing algorithm: correctness, simplicity, robustness, stability, fairness, and optimality. 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 systemwide failures. During that period there will be hardware and software failures of all kinds. Hosts, routers, and lines will go up and down 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 and the network to be rebooted every time some router crashes.

Stability is also an important goal for the routing algorithm. There exist routing algorithms that never converge to equilibrium, no matter how long they run. Fairness and optimality may sound obvious—surely no one would oppose them—but as it turns out, they are often contradictory goals. As a simple example of this conflict, look at Fig. 5-4. 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.

RED HAT 1057
RED HAT V. CAS

Unfortunately, X and X' may not see it that way. Evidently, some compromise between global efficiency and fairness to individual connections is needed.

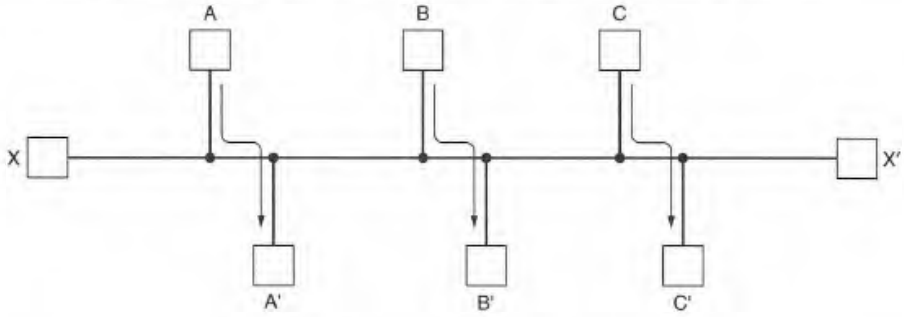


Fig. 5-4. Conflict between fairness and optimality.

Before we can even attempt to find trade-offs between fairness and optimality, we must decide what it is we seek to optimize. Minimizing mean packet delay is an obvious candidate, 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 number of hops a packet must make, because reducing the number of hops tends to improve the delay and also reduce the amount of bandwidth consumed, which tends to improve the throughput as well.

Routing algorithms can be grouped into two major classes: nonadaptive and adaptive. **Nonadaptive algorithms** do not base their routing decisions on measurements or estimates of the current traffic and topology. 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**.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive 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., every ΔT sec, when the load changes, or when the topology 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, both static and dynamic.

5.2.1. The Optimality Principle

Before getting 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**. It states that if router J is on the optimal path from router I to router K, then the optimal path

RED HAT 1057
RED HAT V. CAS

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. 5-5 where the distance metric is the number of hops. Note that a sink tree is not necessarily unique; other trees with the same path lengths may exist. The goal of all routing algorithms is to discover and use the sink trees for all routers.

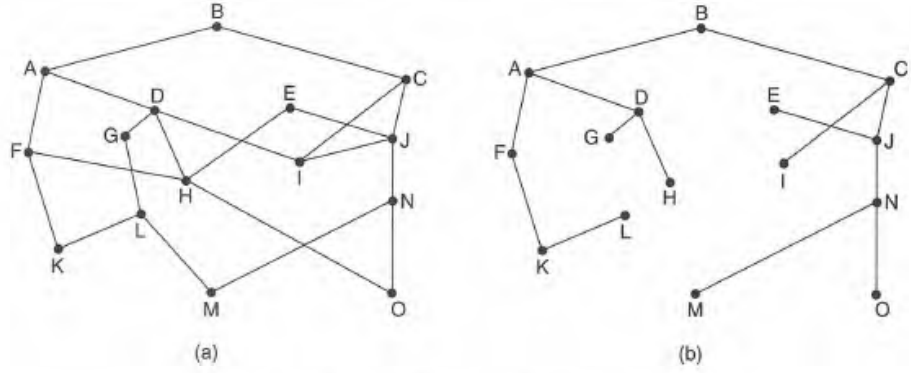


Fig. 5-5. (a) A subnet. (b) A sink tree for router B.

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.

In the next three sections, we will look at three different static routing algorithms. After that we will move on to adaptive ones.

5.2.2. Shortest Path Routing

Let us begin our study of routing algorithms with a technique that is widely used in many forms because it is simple and easy to understand. The idea is to build a graph of the subnet, with each node of the graph representing a router and

RED HAT 1057
RED HAT V. CAS

each arc of the graph representing a communication line (often called a 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. 5-6 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).

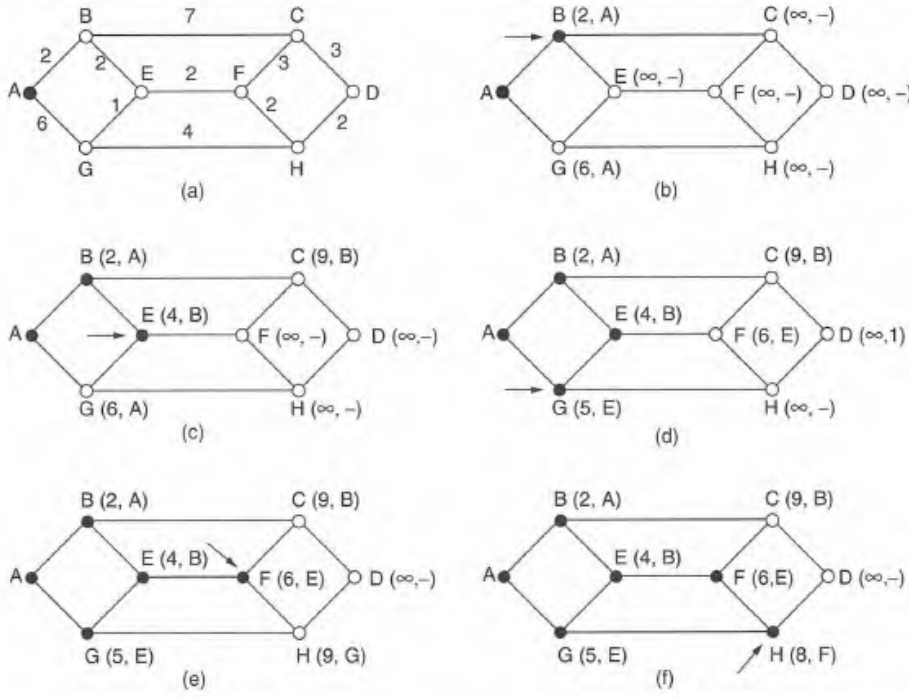


Fig. 5-6. The first five steps used in computing the shortest path from *A* to *D*. The arrows indicate the working node.

However, many other metrics are also possible besides hops and physical distance. For example, each arc could be labeled with the mean queueing and transmission delay for some standard test packet as determined by hourly test runs. With this graph labeling, the shortest path is the fastest path, rather than the path with the fewest arcs or kilometers.

In the most general case, the labels on the arcs could be computed as a function of the distance, bandwidth, average traffic, communication cost, mean queue length, measured delay, and other factors. By changing the weighting function,

RED HAT 1057
RED HAT V. CAS

the algorithm would then compute the "shortest" path measured according to any one of a number of criteria, or 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). Each node is labeled (in parentheses) with its distance from the source node along the best known path. 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. 5-6(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 we can reconstruct the final path later. 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. 5-6(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 5-6 shows the first five steps of the algorithm.

To see why the algorithm works, look at Fig. 5-6(c). At that point we have just made *E* permanent. Suppose that there were a shorter path than *ABE*, say *AXYZE*. 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.

Now consider the case where *Z* is still tentatively labeled. Either the label at *Z* is greater than or equal to that at *E*, in which case *AXYZE* cannot be a shorter path than *ABE*, or it is less than that of *E*, in which case *Z* and not *E* will become permanent first, allowing *E* to be probed from *Z*.

This algorithm is given in Fig. 5-7. The only difference between the program and the algorithm described above is that in Fig. 5-7, we compute the shortest path starting at the terminal node, *t*, rather than at the source node, *s*. Since the shortest path from *t* to *s* in an undirected graph is the same as the shortest path from *s* to *t*, it does not matter at which end we begin (unless there are several shortest paths,

RED HAT 1057
RED HAT V. CAS

in which case reversing the search might discover a different one). The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When copying the final path into the output variable, *path*, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

5.2.3. Flooding

Another static algorithm 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, which 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 subnet.

An alternative technique for damming the flood is to 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.

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 is a duplicate; if so, it is discarded. Furthermore, the full list below k is not needed, since k effectively summarizes it.

A variation of flooding that is slightly more practical is **selective flooding**. In this algorithm the routers do not send every incoming packet out on every line, only on those lines that are going approximately in the right direction. There is usually little point in sending a westbound packet on an eastbound line unless the topology is extremely peculiar.

Flooding is not practical in most applications, but it does have some uses. For example, in military applications, where large numbers of routers may be blown to bits at any instant, the tremendous robustness of flooding is highly desirable. In distributed database applications, it is sometimes necessary to update all the databases concurrently, in which case flooding can be useful. A third possible use of flooding is 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).

RED HAT 1057
RED HAT V. CAS

```
#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);
}
```

Fig. 5-7. Dijkstra's algorithm to compute the shortest path through a graph.

RED HAT 1057
RED HAT V. CAS

5.2.4. Flow-Based Routing

The algorithms studied so far take only the topology into account. They do not consider the load. If, for example, there is always a huge amount of traffic from *A* to *B*, in Fig. 5-6, then it may be better to route traffic from *A* to *C* via *AGEFC*, even though this path is much longer than *ABC*. In this section we will study a static algorithm that uses both topology and load for routing. It is called **flow-based routing**.

In some networks, the mean data flow between each pair of nodes is relatively stable and predictable. For example, in a corporate network for a retail store chain, each store might send orders, sales reports, inventory updates, and other well-defined types of messages to known sites in a predefined pattern, so that the total volume of traffic varies little from day to day. Under conditions in which the average traffic from i to j is known in advance and, to a reasonable approximation, constant in time, it is possible to analyze the flows mathematically to optimize the routing.

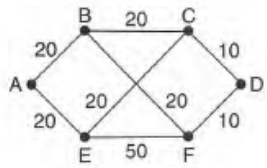
The basic idea behind the analysis is that for a given line, if the capacity and average flow are known, it is possible to compute the mean packet delay on that line from queueing theory. From the mean delays on all the lines, it is straightforward to calculate a flow-weighted average to get the mean packet delay for the whole subnet. The routing problem then reduces to finding the routing algorithm that produces the minimum average delay for the subnet.

To use this technique, certain information must be known in advance. First the subnet topology must be known. Second, the traffic matrix, F_{ij}, must be given. Third, the line capacity matrix, C_{ij}, specifying the capacity of each line in bps must be available. Finally, a (possibly tentative) routing algorithm must be chosen.

As an example of this method, consider the full-duplex subnet of Fig. 5-8(a). The weights on the arcs give the capacities, C_{ij}, in each direction measured in kbps. The matrix of Fig. 5-8(b) has an entry for each source-destination pair. The entry for source i to destination j shows the route to be used for i-j traffic, and also the number of packets/sec to be sent from source i to destination j. For example, 3 packets/sec go from *B* to *D*, and they use route *BFD* to get there. Notice that some routing algorithm has already been applied to derive the routes shown in the matrix.

Given this information, it is straightforward to calculate the total in line i, λ_i. For example, the *B*-*D* traffic contributes 3 packets/sec to the *BF* line and also 3 packets/sec to the *FD* line. Similarly, the *A*-*D* traffic contributes 1 packet/sec to each of three lines. The total traffic in each eastbound line is shown in the λ_i column of Fig. 5-9. In this example, all the traffic is symmetric, that is, the *XY* traffic is identical to the *YX* traffic, for all *X* and *Y*. In real networks this condition does not always hold. The figure also shows the mean number of packets/sec on each line, μC_i assuming a mean packet size of $1/\mu = 800$ bits.

RED HAT 1057
RED HAT V. CAS



(a)

Destination

Source	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA		8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB		3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC		3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD		5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	

(b)

Fig. 5-8. (a) A subnet with line capacities shown in kbps. (b) The traffic in packets/sec and the routing matrix.

The next-to-last column of Fig. 5-9 gives the mean delay for each line derived from the queueing theory formula

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

where $1/\mu$ is the mean packet size in bits, C is the capacity in bps, and λ is the mean flow in packets/sec. For example, with a capacity $\mu C = 25$ packets/sec and an actual flow $\lambda = 14$ packets/sec, the mean delay is 91 msec. Note that with $\lambda = 0$, the mean delay is still 40 msec, because the capacity is 25 packets/sec. In other words, the "delay" includes both queueing and service time.

To compute the mean delay time for the entire subnet, we take the weighted sum of each of the eight lines, with the weight being the fraction of the total traffic using that line. In this example, the mean turns out to be 86 msec.

To evaluate a different routing algorithm, we can repeat the entire process, only with different flows to get a new average delay. If we restrict ourselves to only single path routing algorithms, as we have done so far, there are only a finite number of ways to route packets from each source to each destination. It is always possible to write a program to simply try them all, one after another, and find out which one has the smallest mean delay. Since this calculation can be done off-line in advance, the fact that it may be time consuming is not necessarily a serious problem. This one is then the best routing algorithm. Bertsekas and Gallager (1992) discuss flow-based routing in detail.

RED HAT 1057
RED HAT V. CAS

i	Line	λ_i (pkts/sec)	C_i (kbps)	μC_i (pkts/sec)	T_i (msec)	Weight
1	AB	14	20	25	91	0.171
2	BC	12	20	25	77	0.146
3	CD	6	10	12.5	154	0.073
4	AE	11	20	25	71	0.134
5	EF	13	50	62.5	20	0.159
6	FD	8	10	12.5	222	0.098
7	BF	10	20	25	67	0.122
8	EC	8	20	25	59	0.098

Fig. 5-9. Analysis of the subnet of Fig. 5-8 using a mean packet size of 800 bits. The reverse traffic (*BA*, *CB*, etc.) is the same as the forward traffic.

5.2.5. Distance Vector Routing

Modern computer networks generally use dynamic routing algorithms rather than the static ones described above. 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 one we will study the latter one.

Distance vector routing algorithms operate by having each router maintain a table (i.e, a vector) giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors.

The distance vector routing algorithm is sometimes called by other names, including the distributed **Bellman-Ford** routing algorithm and the **Ford-Fulkerson** 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 and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in the subnet. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

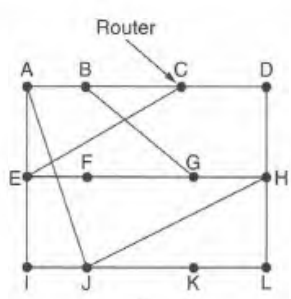
The router is assumed to know the "distance" to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure

RED HAT 1057
RED HAT V. CAS

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 via X. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table. Note that the old routing table is not used in the calculation.

This updating process is illustrated in Fig. 5-10. Part (a) shows a subnet. 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.



(a)

To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6	New routing table for J	

Vectors received from J's four neighbors

(b)

Fig. 5-10. (a) A subnet. (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 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.

RED HAT 1057
RED HAT V. CAS

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

Distance vector routing works in theory but 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 large. 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) subnet of Fig. 5-11, 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.

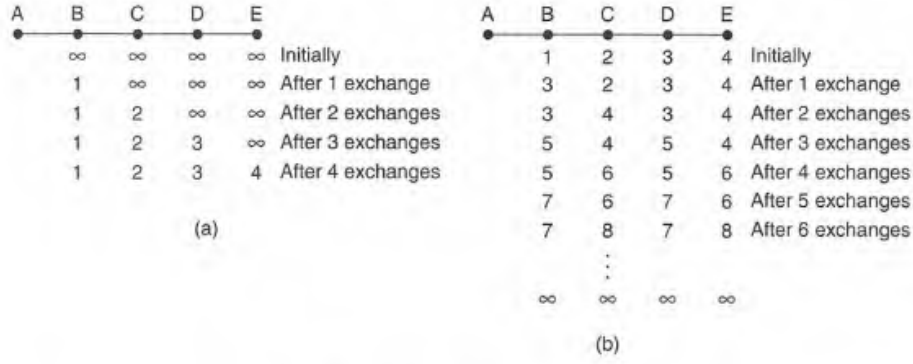


Fig. 5-11. The count-to-infinity problem.

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 neighbor has zero delay to A. B now makes an entry in its routing table 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. 5-11(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 subnet whose longest

RED HAT 1057
RED HAT V. CAS

path is of length N hops, within N exchanges everyone will know about newly revived lines and routers.

Now let us consider the situation of Fig. 5-11(b), in which all the lines and routers are initially up. Routers B, C, D, and E have distances to A of 1, 2, 3, and 4, respectively. Suddenly A goes down, or alternatively, the line 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 know that C's path runs through B itself. For all B knows, C might have ten outgoing lines all with independent paths to A of length 2. As a result, B now 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 the them at random and makes its new distance to A 4, as shown in the third row of Fig. 5-11(b). Subsequent exchanges produce the history shown in the rest of Fig. 5-11(b).

From this figure, its 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 the 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. If the metric is time delay, there is no well-defined upper bound, so a high value is needed to prevent a path with a long delay from being treated as down. Not entirely surprisingly, this problem is known as the **count-to-infinity** problem.

The Split Horizon Hack

Many ad hoc solutions to the count-to-infinity problem have been proposed in the literature, each one more complicated and less useful than the one before it. We will describe just one of them here and then tell why it, too, fails. The **split horizon** algorithm works the same way as distance vector routing, except that the distance to X is not reported on the line that packets for X are sent on (actually, it is reported as infinity). In the initial state of Fig. 5-11(b), for example, C tells D the truth about the distance to A, but C tells B that its distance to A is infinite. Similarly, D tells the truth to E but lies to C.

Now let us see what happens when A goes down. On the first exchange, B discovers that the direct line is gone, and C is reporting an infinite distance to A as well. Since neither of its neighbors can get to A, B sets its distance to infinity as well. On the next exchange, C hears that A is unreachable from both of its neighbors, so it marks A as unreachable too. Using split horizon, the bad news propagates one hop per exchange. This rate is much better than without split horizon.

The real bad news is that split horizon, although widely used, sometimes fails.

RED HAT 1057
RED HAT V. CAS

Consider, for example, the four-node subnet of Fig. 5-12. Initially, both *A* and *B* have a distance 2 to *D*, and *C* has a distance 1 there.

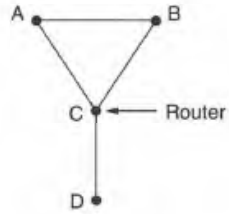


Fig. 5-12. An example where split horizon fails.

Now suppose that the *CD* line goes down. Using split horizon, both *A* and *B* tell *C* that they cannot get to *D*. Thus *C* immediately concludes that *D* is unreachable and reports this to both *A* and *B*. Unfortunately, *A* hears that *B* has a path of length 2 to *D*, so it assumes it can get to *D* via *B* in 3 hops. Similarly, *B* concludes it can get to *D* via *A* in 3 hops. On the next exchange, they each set their distance to *D* to 4. Both of them gradually count to infinity, precisely the behavior we were trying to avoid.

5.2.6. Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 kbps, so line bandwidth was not an issue, but after some lines had been upgraded to 230 kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem. Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed, namely, the algorithm often took too long to converge, even with tricks like split horizon. For these reasons, it was replaced by an entirely new algorithm now called **link state routing**. Variants of link state routing are now widely used.

The idea behind link state routing is simple and can be stated as five parts. Each router must

1. Discover its neighbors and learn their network addresses.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

RED HAT 1057
RED HAT V. CAS

In effect, the complete topology and all delays are experimentally measured and distributed to every router. Then Dijkstra's algorithm can be used 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 telling who it is. 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 or not all three mean the same *F*.

When two or more routers are connected by a LAN, the situation is slightly more complicated. Fig. 5-13(a) illustrates a 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.

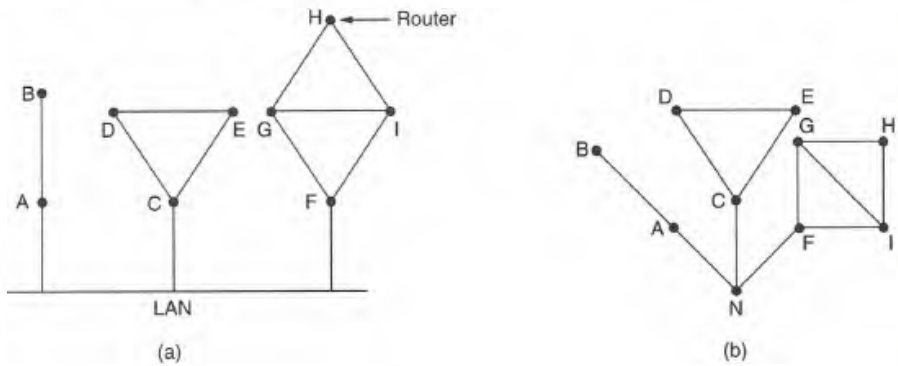


Fig. 5-13. (a) Nine routers and a LAN. (b) A graph model of (a).

One way to model the LAN is to consider it as a node itself, as shown in Fig. 5-13(b). Here we have introduced a new, artificial node, *N*, to which *A*, *C*, and *F* are connected. The fact that it is possible to go from *A* to *C* on the LAN is represented by the path *ANC* here.

Measuring Line Cost

The link state routing algorithm requires each router to know, or at least have a reasonable estimate, of the delay to each of its neighbors. The most direct way to determine this delay is to send a special ECHO packet over the line that the other

RED HAT 1057
RED HAT V. CAS

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. For even better results, the test can be conducted several times, and the average used.

An interesting issue is whether or not to take the load into account when measuring the delay. To factor the load in, the round-trip timer must be started when the ECHO packet is queued. To ignore the load, the timer should be started when the ECHO packet reaches the front of the queue.

Arguments can be made both ways. Including traffic-induced delays in the measurements means that when a router has a choice between two lines with the same bandwidth, one of which is heavily loaded all the time and one of which is not, it will regard the route over the unloaded line as a shorter path. This choice will result in better performance.

Unfortunately, there is also an argument against including the load in the delay calculation. Consider the subnet of Fig. 5-14, which is divided up into two parts, East and West, connected by two lines, *CF* and *EI*. Suppose that most of the traffic between East and West is using line *CF*, and as a result, this line is heavily loaded with long delays. Including queueing delay in 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*, overloading this line. 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. If load is ignored and only bandwidth is considered, this problem does not occur. Alternatively, the load can be spread over both lines, but this solution does not fully utilize the best path.

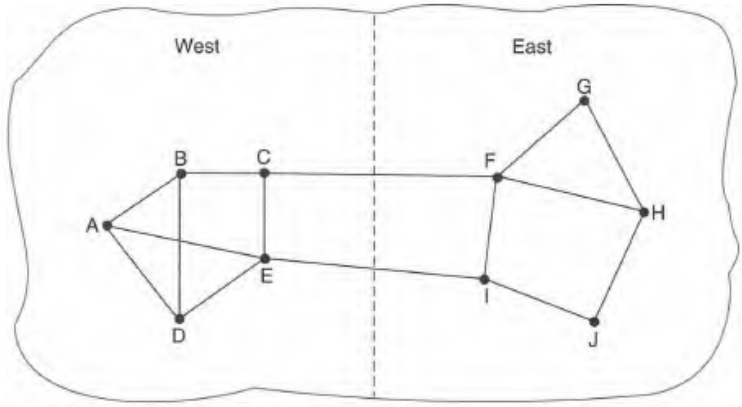
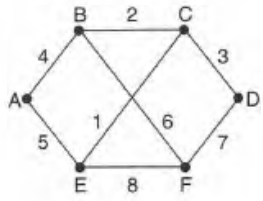


Fig. 5-14. A subnet in which the East and West parts are connected by two lines.

RED HAT 1057
RED HAT V. CAS

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. For each neighbor, the delay to that neighbor is given. An example subnet is given in Fig. 5-15(a) with delays shown in the lines. The corresponding link state packets for all six routers are shown in Fig. 5-15(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)

Fig. 5-15. (a) A subnet. (b) The link state packets for this subnet.

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 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 reliably. As the packets are distributed and installed, the routers getting the first ones will change their routes. Consequently, the different routers may be using different versions of the topology, which can lead to inconsistencies, loops, unreachable machines, and other problems.

First we will describe the basic distribution algorithm. Later we will give some refinements. The fundamental idea is to use flooding to distribute the link state packets. 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.

RED HAT 1057
RED HAT V. CAS

If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete.

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 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 is 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 minutes, 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 first. If another link state packet from the same source comes in before it 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 router-router lines, all link state packets are acknowledged. When a line goes idle, the holding area is scanned in round robin order to select a packet or acknowledgement to send.

The data structure used by router *B* for the subnet shown in Fig. 5-15(a) is depicted in Fig. 5-16. 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 lines (to *A*, *C*, and *F*, respectively). The send flags mean that the packet must be sent on the indicated line. The acknowledgement flags mean that it must be acknowledged there.

In Fig. 5-16, the link state packet from *A* arrived 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 arrived twice, once via *EAB* and once via *EFB*. Consequently, it has to be sent only to *C*, but 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

RED HAT 1057
RED HAT V. CAS

Source	Seq.	Age	Send flags A	Send flags C	Send flags F	ACK flags A	ACK flags C	ACK flags 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	

Fig. 5-16. The packet buffer for router *B* in Fig. 5-15.

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 subnet graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The two values can be averaged or used separately.

Now Dijkstra's algorithm can be run locally to construct the shortest path to all possible destinations. The results of this algorithm can be installed in the routing tables, and normal operation resumed.

For a subnet with n routers, each of which has k neighbors, the memory required to store the input data is proportional to kn. For large subnets, this can be a problem. Also, the computation time can be an issue. Nevertheless, in many practical situations, link state routing works well.

However, problems with the hardware or software can wreak havoc with this algorithm (also with other ones). For example, if a router claims to have a line it does not have, or forgets a line it does have, the subnet graph will be incorrect. If a router fails to forward packets, or corrupts them while forwarding them, trouble will arise. Finally, if it runs out of memory or does the routing calculation wrong, bad things will happen. As the subnet 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 solutions in detail.

Link state routing is widely used in actual networks, so a few words about some example protocols using it are in order. The OSPF protocol, which is

RED HAT 1057
RED HAT V. CAS

increasingly being used in the Internet, uses a link state algorithm. We will describe OSPF in Sec. 5.5.5.

Another important link state protocol is **IS-IS** (**Intermediate System-Intermediate System**), which was designed for DECnet and later adopted by ISO for use with its connectionless network layer protocol, CLNP. Since then it has been modified to handle other protocols as well, most notably, IP. IS-IS is used in numerous Internet backbones (including the old NSFNET backbone), and in some digital cellular systems such as CDPD. Novell NetWare uses a minor variant of IS-IS (NLSP) for routing IPX packets.

Basically IS-IS distributes a picture of the router topology, from which the shortest paths are computed. Each router announces, in its link state information, which network layer addresses it can reach directly. These addresses can be IP, IPX, AppleTalk, or any other addresses. IS-IS can even support multiple network layer protocols at the same time.

Many of the innovations designed for IS-IS were adopted by OSPF (OSPF was designed several years after IS-IS). These 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 is encoded in such a way that it is easy and natural to simultaneously carry information about multiple network layer protocols, a feature OSPF does not have. This advantage is especially valuable in large multiprotocol environments.

5.2.7. 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.

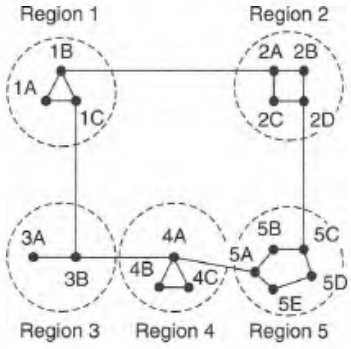
When hierarchical routing is used, the routers are divided into what we will call **regions**, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions. When different networks are connected together, it is natural to regard each one as a separate region in order 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

RED HAT 1057
RED HAT V. CAS

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 to other domestic routers, but would send 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 5-17 gives a quantitative example of routing in a two-level hierarchy with five regions. The full routing table for router 1*A* has 17 entries, as shown in Fig. 5-17(b). When routing is done hierarchically, as in Fig. 5-17(c), there are entries for all the local routers as before, but all other regions have been condensed into a single router, so all traffic for region 2 goes via the 1*B*–2*A* line, but the rest of the remote traffic goes via the 1*C*–3*B* 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.



(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)

Fig. 5-17. Hierarchical routing.

Unfortunately, these gains in space are not free. There is a penalty to be paid, and this penalty is in the form of increased path length. For example, the best route from 1*A* to 5*C* 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 subnet with 720

RED HAT 1057
RED HAT V. CAS

routers. If there is no hierarchy, each router needs 720 routing table entries. If the subnet 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 eight 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) have discovered that the optimal number of levels for an N router subnet 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.

5.2.8. Routing for Mobile Hosts

Millions of people have portable computers nowadays, and they generally want to read their email and access their normal file systems wherever in the world they may be. These mobile hosts introduce a new complication: to route a packet to a mobile host, the network first has to find it. The subject of incorporating mobile hosts into a network is very young, but in this section we will sketch some of the issues here and give a possible solution.

The model of the world that network designers typically use is shown in Fig. 5-18. Here we have a WAN consisting of routers and hosts. Connected to the WAN are LANs, MANs, and wireless cells of the type we studied in Chap. 2.

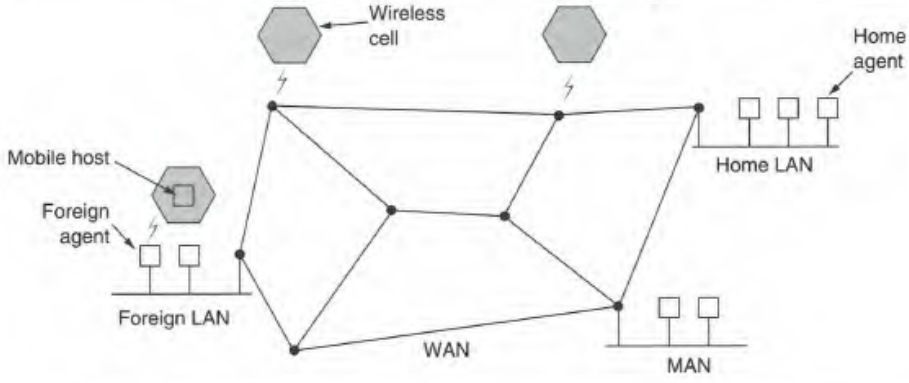


Fig. 5-18. A WAN to which LANs, MANs, and wireless cells are attached.

Users who never move are said to be stationary. They are connected to the network by copper wires or fiber optics. In contrast, we can distinguish two other kinds of users. Migratory users are basically stationary users who move from one fixed site to another from time to time but use the network only when they are

RED HAT 1057
RED HAT V. CAS

physically connected to it. Roaming users actually compute on the run and want to maintain their connections as they move around. We will use the term **mobile users** to mean either of the latter two categories, that is, all users who are away from home.

All users are assumed to have a permanent **home location** that never changes. Users also have a permanent home address that can be used to determine their home locations, 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 users is to make it possible to send packets to mobile users using their home addresses, and have the packets efficiently reach them wherever they may be. The trick, of course, is to find them.

In the model of Fig. 5-18, the world is divided up (geographically) into small units. Let us call them areas, where an area is typically a LAN or wireless cell. Each area has one or more **foreign agents**, which keep track of all mobile users visiting the area. In addition, each area has a **home agent**, which keeps track of users whose home is in the area, but who are currently visiting another area.

When a new user enters an area, either by connecting to it (e.g., plugging into the LAN), or just wandering into the cell, his computer must register itself with the foreign agent there. The registration procedure typically works like this:

1. Periodically, each foreign agent broadcasts a packet announcing its existence and address. A newly arrived mobile host may wait for one of these messages, but if none arrives quickly enough, the mobile host can broadcast a packet saying: "Are there any foreign agents around?"
2. The mobile host registers with the foreign agent, giving its home address, current data link layer address, and some security information.
3. The foreign agent contacts the mobile host's home agent and says: "One of your hosts is over here." The message from the foreign agent to the home agent contains the foreign agent's network address. It also includes the security information, to convince the home agent that the mobile host is really there.
4. The home agent examines the security information, which contains a timestamp, to prove that it was generated within the past few seconds. If it is happy, it tells the foreign agent to proceed.
5. When the foreign agent gets the acknowledgement from the home agent, it makes an entry in its tables and informs the mobile host that it is now registered.

Ideally, when a user leaves an area, that, too, should be announced to allow deregistration, but many users abruptly turn off their computers when done.

RED HAT 1057
RED HAT V. CAS

When a packet is sent to a mobile user, it is routed to the user's home LAN because that is what the address says should be done, as illustrated in step 1 of Fig. 5-19. Packets sent to the mobile user on its home LAN are intercepted by the home agent. The home agent then looks up the mobile user's new (temporary) location and finds the address of the foreign agent handling the mobile user. The home agent then does two things. First, it encapsulates the packet in the payload field of an outer packet and sends the latter to the foreign agent (step 2 in Fig. 5-19). This mechanism is called tunneling; we will look at it in more detail later. After getting the encapsulated packet, the foreign agent removes the original packet from the payload field and sends it to the mobile user as a data link frame.

Second, the home agent tells the sender to henceforth send packets to the mobile host by encapsulating them in the payload of packets explicitly addressed to the foreign agent, instead of just sending them to the mobile user's home address (step 3). Subsequent packets can now be routed directly to the user via the foreign agent (step 4), bypassing the home location entirely.

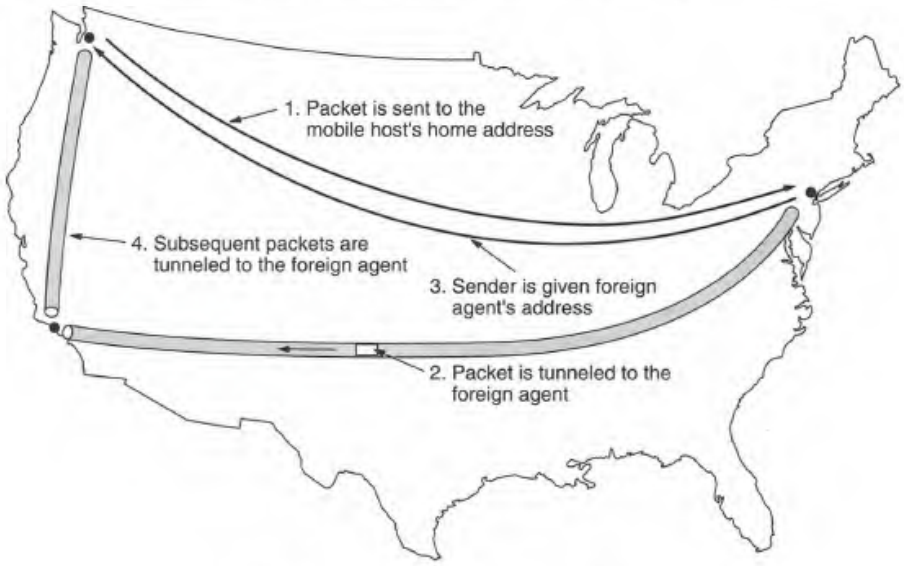


Fig. 5-19. Packet routing for mobile users.

The various schemes that have been proposed differ in several ways. First, there is the issue of how much of this protocol is carried out by the routers and how much by the hosts, and in the latter case, by which layer in the hosts. Second, in a few schemes, routers along the way record mapped addresses so they can intercept and redirect traffic even before it gets to the home location. Third, in some schemes each visitor is given a unique temporary address; in others, the temporary address refers to an agent that handles traffic for all visitors.

RED HAT 1057
RED HAT V. CAS

Fourth, the schemes differ in how they actually manage to arrange for packets that are addressed to one destination to be delivered to a different one. One choice is changing the destination address and just retransmitting the modified packet. Alternatively, the whole packet, home address and all, can be encapsulated inside the payload of another packet sent to the temporary address. Finally, the schemes differ in their security aspects. In general, when a host or router gets a message of the form "Starting right now, please send all of Cayla's mail to me," it might have a couple of questions about whom it was talking to and whether or not this is a good idea. Several mobile host protocols are discussed and compared in (Ioannidis and Maguire, 1993; Myles and Skellern, 1993; Perkins, 1993; Teraoka et al., 1993; and Wada et al., 1993).

5.2.9. Broadcast Routing

For 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 broadcasting 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 subnet is for the source to simply send a distinct packet to each destination. Not only is the method wasteful of bandwidth, but it also requires the source to have a complete list of all destinations. In practice this may be the only possibility, but it is the least desirable of the methods.

Flooding is another obvious candidate. Although flooding is ill-suited for ordinary point-to-point communication, for broadcasting it might rate serious consideration, especially if none of the methods described below are applicable. The problem with flooding as a broadcast technique is the same problem it has as a point-to-point routing algorithm: it generates too many packets and consumes too much bandwidth.

A third algorithm is **multidestination routing**. If this method is used, 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 and can be treated as a normal packet. Multidestination routing is like separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.

A fourth broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast, or any other convenient spanning tree for that matter. A

RED HAT 1057
RED HAT V. CAS

spanning tree is a subset of the subnet that includes all the routers but contains no loops. 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. The only problem is that each router must have knowledge of some spanning tree for it to be applicable. Sometimes this information is available (e.g., with link state routing) but sometimes it is not (e.g., with distance vector routing).

Our last broadcast algorithm is an attempt to approximate the behavior of the previous one, even when the routers do not know anything at all about spanning trees. The idea is remarkably simple once it has been pointed out. When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the line that is normally used for sending packets *to* 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 lines except the one it arrived on. If, however, the broadcast packet arrived on a line other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

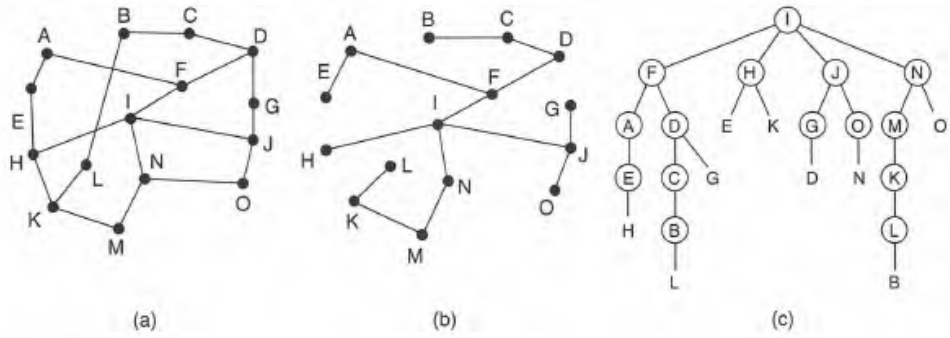


Fig. 5-20. Reverse path forwarding. (a) A subnet. (b) A spanning tree. (c) The tree built by reverse path forwarding.

An example of the algorithm, called **reverse path forwarding**, is shown in Fig. 5-20. Part (a) shows a subnet, part (b) shows a sink tree for router *I* of that subnet, 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

RED HAT 1057
RED HAT V. CAS

(at *C*, *E*, and *K*); the others are duplicates. After five hops and 23 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 both reasonably efficient and easy to implement. It does not require routers to know about spanning trees, nor does it have the overhead of a destination list or bit map in each broadcast packet as does multidestination addressing. Nor does it require any special mechanism to stop the process, as flooding does (either a hop counter in each packet and a priori knowledge of the subnet diameter, or a list of packets already seen per source).

5.2.10. Multicast Routing

For some applications, widely-separated processes work together in groups, for example, a group of processes implementing a distributed database system. It frequently is necessary for one process to send a message to all the other members of the group. If the group is small, it can just send each other member a point-to-point message. If the group is large, this strategy is expensive. Sometimes broadcasting can be used, but using broadcasting to inform 1000 machines on a million-node network is inefficient because 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 its routing algorithm is called **multicast routing**. In this section we will describe one way of doing multicast routing. For additional information, see (Deering and Cheriton, 1990; Deering et al., 1994; and Rajagopalan, 1992).

To do multicasting, group management is required. Some way is needed to create and destroy groups, and for processes to join and leave groups. How these tasks are accomplished is not of concern to the routing algorithm. What is of concern is that when a process joins a group, it informs its host of this fact. It is important that routers know which of their hosts belong to which groups. Either hosts must inform their routers about changes in group membership, or routers must query their hosts periodically. Either way, routers learn about which of their hosts are in which groups. Routers tell their neighbors, so the information propagates through the subnet.

To do multicast routing, each router computes a spanning tree covering all other routers in the subnet. For example, in Fig. 5-21(a) we have a subnet with two groups, 1 and 2. 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. 5-21(b).

When a process sends a multicast packet to a group, the first router examines its spanning tree and prunes it, removing all lines that do not lead to hosts that are

RED HAT 1057
RED HAT V. CAS

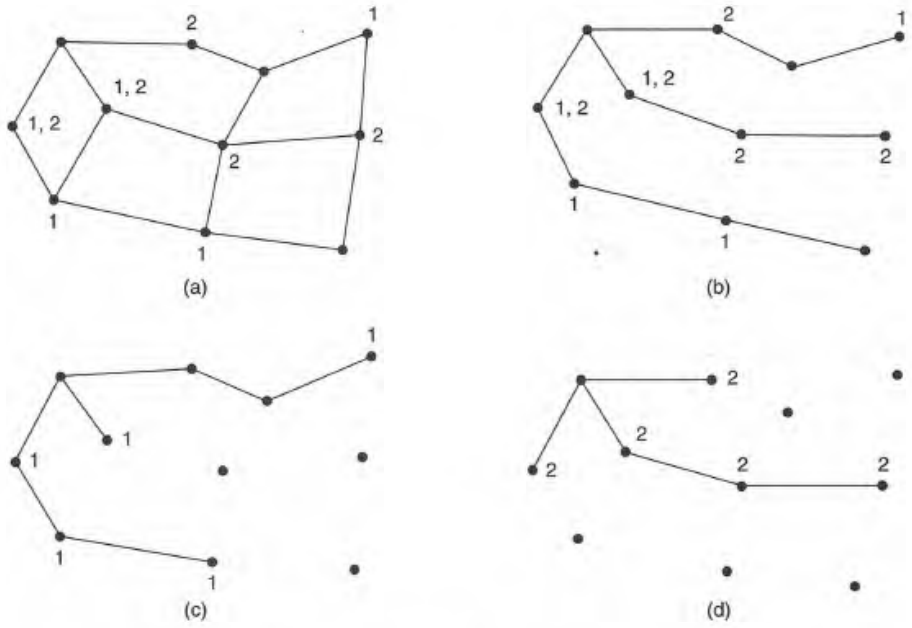


Fig. 5-21. (a) A subnet. (b) A spanning tree for the leftmost router. (c) A multicast tree for group 1. (d) A multicast tree for group 2.

members of the group. In our example, Fig. 5-21(c) shows the pruned spanning tree for group 1. Similarly, Fig. 5-21(d) shows the pruned spanning tree for group 2. Multicast packets are forwarded only along the appropriate spanning tree.

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 subnet topology, including which hosts belong to which groups. Then the spanning tree can be pruned by starting at the end of each path and working toward the root, removing all routers that do not belong to the group in question.

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 sender not to send it any more multicasts for that group. When a router with no group members among its own hosts has received such messages on all its lines, it, too, can respond with a PRUNE message. In this way, the subnet is recursively pruned.

One potential disadvantage of this algorithm is that it scales poorly to large networks. Suppose that a network has n groups, each with an average of m

RED HAT 1057
RED HAT V. CAS

members. For each group, m pruned spanning trees must be stored, for a total of mn trees. When many large groups exist, considerable storage is needed to store all the trees.

An alternative design uses **core-base trees** (Ballardie et al., 1993). Here, a single spanning tree per group is computed, with the root (the core) near the middle of the group. To send a multicast message, a host sends it to the core, which then does the multicast along the spanning tree. Although this tree will not be optimal for all sources, the reduction in storage costs from m trees to one tree per group is a major saving.

5.3. CONGESTION CONTROL ALGORITHMS

When too many packets are present in (a part of) the subnet, performance degrades. This situation is called **congestion**. Figure 5-22 depicts the symptom. When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with transmission errors), and the number delivered is proportional to the number sent. However, as traffic increases too far, the routers are no longer able to cope, and they begin losing packets. This tends to make matters worse. At very high traffic, performance collapses completely, and almost no packets are delivered.

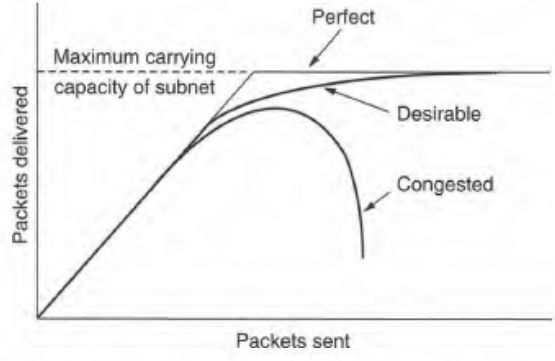


Fig. 5-22. When too much traffic is offered, congestion sets in and performance degrades sharply.

Congestion can be brought about by several factors. 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) discovered that if routers have an infinite amount of memory,

RED HAT 1057
RED HAT V. CAS

congestion gets worse, not better, because by the time packets get to the front of the queue, they have already timed out (repeatedly), and duplicates have been sent. All these packets will be dutifully forwarded to the next router, increasing the load all the way to the destination.

Slow processors can also cause congestion. If the routers' CPUs are slow at performing the bookkeeping tasks required of them (queueing buffers, updating tables, etc.), queues can build up, even though there is excess line capacity. Similarly, low-bandwidth lines can also cause congestion. Upgrading the lines but not changing the processors, or vice versa, often helps a little, but frequently just shifts the bottleneck. Also, upgrading part, but not all, of the system, often just moves the bottleneck somewhere else. The real problem is frequently a mismatch between parts of the system. This problem will persist until all the components are in balance.

Congestion tends to feed upon itself and become worse. If a router has no free buffers, it must ignore newly arriving packets. When a packet is discarded, the sending router (a neighbor) may time out and retransmit it, perhaps ultimately many times. Since it cannot discard the packet until it has been acknowledged, congestion at the receiver's end forces the sender to refrain from releasing a buffer it would have normally freed. In this manner, congestion backs up, like cars approaching a toll booth.

It is worth explicitly pointing out the difference between congestion control and flow control, as the relationship is subtle. Congestion control has to do with making sure the subnet is able to carry the offered traffic. It is a global issue, involving the behavior of all the hosts, all the routers, the store-and-forwarding processing within the routers, and all the other factors that tend to diminish the carrying capacity of the subnet.

Flow control, in contrast, relates to the point-to-point traffic between a given sender and a given receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver can absorb it. Flow control nearly always involves some direct feedback from the receiver to the sender to tell the sender how things are doing at the other end.

To see the difference between these two concepts, consider a fiber optic network with a capacity of 1000 gigabits/sec on which a supercomputer is trying to transfer a file to a personal computer at 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 store-and-forward 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 simply that the total offered traffic exceeds what the network can handle.

The reason congestion control and flow control are often confused is that some congestion control algorithms operate by sending messages back to the

RED HAT 1057
RED HAT V. CAS

various sources telling them to slow down when the network gets into trouble. 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 later.

We will start our study of congestion control by looking at a general model for dealing with it. Then we will look at broad approaches to preventing it in the first place. After that, we will look at various dynamic algorithms for coping with it once it has set in.

5.3.1. General Principles of Congestion Control

Many problems in complex systems, such as computer networks, can be viewed from a control theory point of view. This approach leads to dividing all solutions into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made.

Tools for doing open-loop control include deciding when to accept new traffic, deciding when to discard packets and which ones, and making scheduling decisions at various points in the network. All of these have in common the fact that they make decisions without regard to the current state of the network.

In contrast, closed loop solutions are based on the concept of a feedback loop. This approach has three parts when applied to congestion control:

1. Monitor the system to detect when and where congestion occurs.
2. Pass this information to places where action can be taken.
3. Adjust system operation to correct the problem.

Various metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay, and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion.

The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it. The obvious way is for the router detecting the congestion to send a packet to the traffic source or sources, announcing the problem. Of course, these extra packets increase the load at precisely the moment that more load is not needed, namely, when the subnet is congested.

However, other possibilities also exist. For example, a bit or field can be reserved in every packet for routers to fill in whenever congestion gets above some threshold level. When a router detects this congested state, it fills in the field in all outgoing packets, to warn the neighbors.

RED HAT 1057
RED HAT V. CAS

Still another approach is to have hosts or routers send probe packets out periodically to explicitly ask about congestion. This information can then be used to route traffic around problem areas. Some radio stations have helicopters flying around their cities to report on road congestion in the hope that their listeners will route their packets (cars) around hot spots.

In all feedback schemes, the hope is that knowledge of congestion will cause the hosts to take appropriate action to reduce the congestion. 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 real use. To work well, some kind of averaging is needed, but getting the time constant right is a nontrivial matter.

Many congestion control algorithms are known. To provide a way to organize them in a sensible way, Yang and Reddy (1995) have developed a taxonomy for congestion control algorithms. They begin by dividing all algorithms into open loop or closed loop, as described above. They further divide the open loop algorithms into ones that act at the source versus ones that act at the destination. The closed loop algorithms are also divided into two subcategories: explicit feedback versus implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithms, the source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. Two solutions come to mind: increase the resources or decrease the load. For example, the subnet may start using dial-up telephone lines to temporarily increase the bandwidth between certain points. In systems like SMDS (see Chap. 1), it may ask the carrier for additional bandwidth for a while. On satellite systems, increasing transmission power often gives higher bandwidth. Splitting traffic over multiple routes instead of always using the best one may also effectively increase the bandwidth. Finally, spare routers that are normally used only as backups (to make the system fault tolerant) can be put on-line to give more capacity when serious congestion appears.

However, sometimes it is not possible to increase the capacity, or it has already been increased to the limit. The only way then to beat back the congestion is to decrease the load. Several ways exist to reduce the load, including denying service to some users, degrading service to some or all users, and having users schedule their demands in a more predictable way.

Some of these methods, which we will study shortly, can best be applied to virtual circuits. For subnets that use virtual circuits internally, these methods can be used at the network layer. For datagram subnets, they can nevertheless sometimes be used on transport layer connections. In this chapter, we will focus on

RED HAT 1057
RED HAT V. CAS

their use in the network layer. In the next one, we will see what can be done at the transport layer to manage congestion.

5.3.2. Congestion Prevention Policies

Let us begin our study of methods to control congestion by looking at open loop systems. These systems are designed to minimize congestion in the first place, rather than letting it happen and reacting after the fact. They try to achieve their goal by using appropriate policies at various levels. In Fig. 5-23 we see different data link, network, and transport policies that can affect congestion (Jain, 1990).

Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

Fig. 5-23. Policies that affect congestion.

Let us start at the data link layer and work our way upward. The retransmission policy deals with how fast a sender times out and what it transmits upon timeout. A jumpy sender that times out quickly and retransmits all outstanding packets using go back n will put a heavier load on the system than a leisurely sender that uses selective repeat. Closely related to this is caching policy. If receivers routinely discard all out-of-order packets, these packets will have to be transmitted again later, creating extra load.

Acknowledgement policy also affects congestion. If each packet is acknowledged immediately, the acknowledgement packets generate extra traffic. However, if acknowledgements are saved up to piggyback onto reverse traffic, extra timeouts and retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.

RED HAT 1057
RED HAT V. CAS

At the network layer, the choice between virtual circuits and datagrams affects congestion, since many congestion control algorithms work only with virtual circuit subnets. Packet queueing and service policy relates to whether routers have one queue per input line, one queue per output line, or both. It also relates to the order packets are processed (e.g., round robin, or priority based). Discard policy is the rule telling which packet is dropped when there is no space. A good policy can help alleviate congestion and a bad one can make it worse.

The routing algorithm can help avoid congestion by spreading the traffic over all the lines, whereas a bad one can send too much traffic over already congested lines. Finally, packet lifetime management deals with how long a packet may live before being discarded. If it is too long, lost packets may clog up the works for a long time, but if it is too short, packets may sometimes time out before reaching their destination, thus inducing retransmissions.

In the transport layer, the same issues occur as in the data link layer, but in addition, determining the timeout interval is harder because the transit time across the network is less predictable than the transit time over a wire between two routers. If it is too short, extra packets will be sent unnecessarily. If it is too long, congestion will be reduced, but the response time will suffer whenever a packet is lost.

5.3.3. Traffic Shaping

One of the main causes of congestion is that traffic is often bursty. If hosts could be made to transmit at a uniform rate, congestion would be less common. Another open loop method to help manage congestion is forcing the packets to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called **traffic shaping**.

Traffic shaping is about regulating the average *rate* (and burstiness) of data transmission. In contrast, the sliding window protocols we studied earlier limit the amount of data in transit at once, not the rate at which it is sent. When a virtual circuit is set up, the user and the subnet (i.e., the customer and the carrier) agree on a certain traffic pattern (i.e., shape) for that circuit. As long as the customer fulfills her part of the bargain and only sends packets according to the agreed upon contract, the carrier promises to deliver them all in a timely fashion. Traffic shaping reduces congestion and thus helps the carrier live up to its promise. Such agreements are not so important for file transfers but are of great importance for real-time data, such as audio and video connections, which do not tolerate congestion well.

In effect, with traffic shaping the customer says to the carrier: "My transmission pattern will look like this. Can you handle it?" If the carrier agrees, the issue arises of how the carrier can tell if the customer is following the agreement, and what to do if the customer is not. Monitoring a traffic flow is called **traffic policing**. Agreeing to a traffic shape and policing it afterward are easier with virtual

RED HAT 1057
RED HAT V. CAS

circuit subnets than with datagram subnets. However, even with datagram subnets, the same ideas can be applied to transport layer connections.

The Leaky Bucket Algorithm

Imagine a bucket with a small hole in the bottom, as illustrated in Fig. 5-24(a). No matter at what rate water enters the bucket, the outflow is at a constant rate, ρ, when there is any water in the bucket, and zero when the bucket is empty. Also, once the bucket is full, any additional water entering it spills over the sides and is lost (i.e., does not appear in the output stream under the hole).

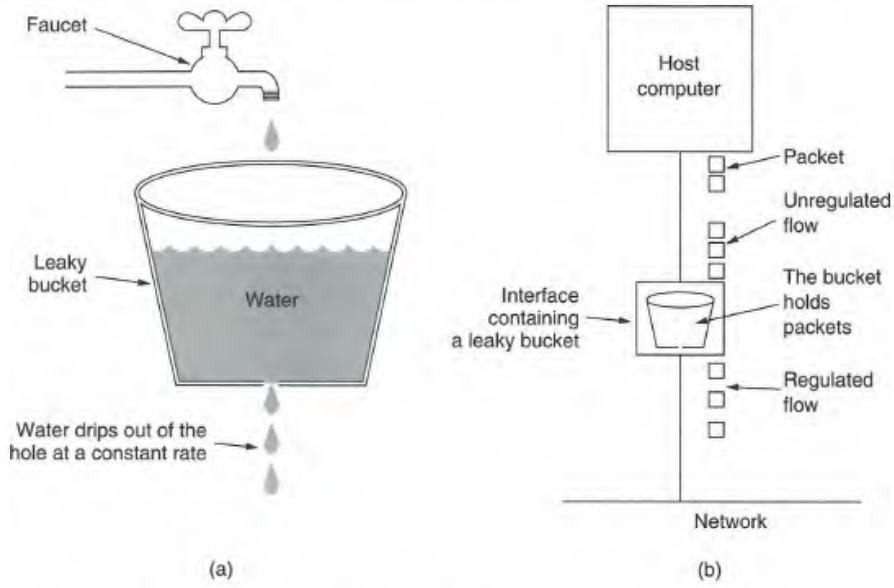


Fig. 5-24. (a) A leaky bucket with water. (b) A leaky bucket with packets.

The same idea can be applied to packets, as shown in Fig. 5-24(b). Conceptually, each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more processes within the host try to send a packet when the maximum number are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. It was first proposed by Turner (1986) and is called the **leaky bucket algorithm**. In fact, it is nothing other than a single-server queueing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. Again, this can be enforced by the interface card or by the operating system. This

RED HAT 1057
RED HAT V. CAS

mechanism turns an uneven flow of packets from the user processes inside the host into an even flow of packets onto the network, smoothing out bursts and greatly reducing the chances of congestion.

When the packets are all the same size (e.g., ATM cells), this algorithm can be used as described. However, when variable-sized packets are being used, it is often better to allow a fixed number of bytes per tick, rather than just one packet. Thus if the rule is 1024 bytes per tick, a single 1024-byte packet can be admitted on a tick, two 512-byte packets, four 256-byte packets, and so on. If the residual byte count is too low, the next packet must wait until the next tick.

Implementing the original leaky bucket algorithm is easy. The leaky bucket consists of a finite queue. When a packet arrives, if there is room on the queue it is appended to the queue; otherwise, it is discarded. At every clock tick, one packet is transmitted (unless the queue is empty).

The byte-counting leaky bucket is implemented almost the same way. At each tick, a counter is initialized to n. If the first packet on the queue has fewer bytes than the current value of the counter, it is transmitted, and the counter is decremented by that number of bytes. Additional packets may also be sent, as long as the counter is high enough. When the counter drops below the length of the next packet on the queue, transmission stops until the next tick, at which time the residual byte count is overwritten and lost.

As an example of a leaky bucket, imagine that a computer can produce data at 25 million bytes/sec (200 Mbps) and that the network also runs at this speed. However, the routers can handle this data rate only for short intervals. For long intervals, they work best at rates not exceeding 2 million bytes/sec. Now suppose data comes in 1-million-byte bursts, one 40-msec burst every second. To reduce the average rate to 2 MB/sec, we could use a leaky bucket with $\rho = 2$ MB/sec and a capacity, C, of 1 MB. This means that bursts of up to 1 MB can be handled without data loss, and that such bursts are spread out over 500 msec, no matter how fast they come in.

In Fig. 5-25(a) we see the input to the leaky bucket running at 25 MB/sec for 40 msec. In Fig. 5-25(b) we see the output draining out at a uniform rate of 2 MB/sec for 500 msec.

The Token Bucket Algorithm

The leaky bucket algorithm enforces a rigid output pattern at the average rate, no matter how bursty the traffic is. For many applications, it is better to allow the output to speed up somewhat when large bursts arrive, so a more flexible algorithm is needed, preferably one that never loses data. One such algorithm is the **token bucket algorithm**. In this algorithm, the leaky bucket holds tokens, generated by a clock at the rate of one token every ΔT sec. In Fig. 5-26(a) we see a bucket holding three tokens, with five packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. In Fig. 5-26(b)

RED HAT 1057
RED HAT V. CAS

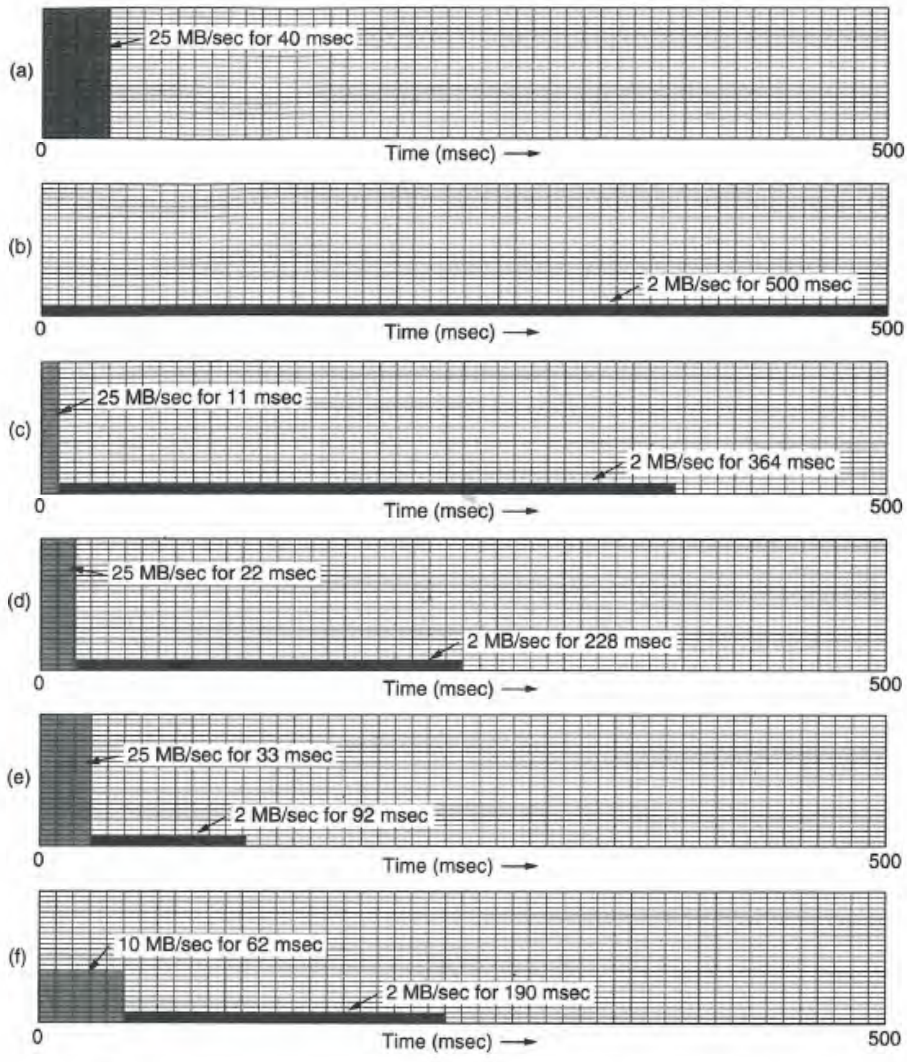


Fig. 5-25. (a) Input to a leaky bucket. (b) Output from a leaky bucket. (c) - (e) Output from a token bucket with capacities of 250KB, 500KB, and 750KB. (f) Output from a 500KB token bucket feeding a 10 MB/sec leaky bucket.

we see that three of the five packets have gotten through, but the other two are stuck waiting for two more tokens to be generated.

The token bucket algorithm provides a different kind of traffic shaping than the leaky bucket algorithm. The leaky bucket algorithm does not allow idle hosts

RED HAT 1057
RED HAT V. CAS

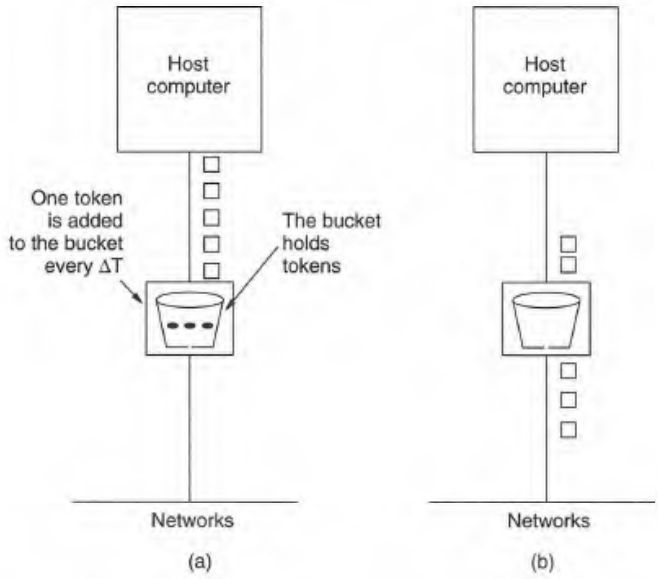


Fig. 5-26. The token bucket algorithm. (a) Before. (b) After.

to save up permission to send large bursts later. The token bucket algorithm does allow saving, up to the maximum size of the bucket, n. This property means that bursts of up to n packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden bursts of input.

Another difference between the two algorithms is that the token bucket algorithm throws away tokens when the bucket fills up but never discards packets. In contrast, the leaky bucket algorithm discards packets when the bucket fills up.

Here too, a minor variant is possible, in which each token represents the right to send not one packet, but k bytes. A packet can only be transmitted if enough tokens are available to cover its length in bytes. Fractional tokens are kept for future use.

The leaky bucket and token bucket algorithms can also be used to smooth traffic between routers, as well as being used to regulate host output as in our examples. However, one clear difference is that a token bucket regulating a host can make the host stop sending when the rules say it must. Telling a router to stop sending while its input keeps pouring in may result in lost data.

The implementation of the basic token bucket algorithm is just a variable that counts tokens. The counter is incremented by one every ΔT and decremented by one whenever a packet is sent. When the counter hits zero, no packets may be sent. In the byte-count variant, the counter is increment by k bytes every ΔT and decremented by the length of each packet sent.

RED HAT 1057
RED HAT V. CAS

Essentially what the token bucket does is allow bursts, but up to a regulated maximum length. Look at Fig. 5-25(c) for example. Here we have a token bucket with a capacity of 250 KB. Tokens arrive at a rate allowing output at 2 MB/sec. Assuming the token bucket is full when the 1-MB burst arrives, the bucket can drain at the full 25 MB/sec for about 11 msec. Then it has to cut back to 2 MB/sec until the entire input burst has been sent.

Calculating the length of the maximum rate burst is slightly tricky. It is not just 1 MB divided by 25 MB/sec because while the burst is being output, more tokens arrive. If we call the burst length S sec, the token bucket capacity C bytes, the token arrival rate ρ bytes/sec, and the maximum output rate M bytes/sec, we see that an output burst contains a maximum of $C + \rho S$ bytes. We also know that the number of bytes in a maximum-speed burst of length S seconds is MS. Hence we have

$$C + \rho S = MS$$

We can solve this equation to get $S = C/(M - \rho)$. For our parameters of $C = 250$ KB, $M = 25$ MB/sec, and $\rho = 2$ MB/sec, we get a burst time of about 11 msec. Figure 5-25(d) and Fig. 5-25(e) show the token bucket for capacities of 500-KB and 750 KB, respectively.

A potential problem with the token bucket algorithm is that it allows large bursts again, even though the maximum burst interval can be regulated by careful selection of ρ and M. Frequently it is desirable to reduce the peak rate, but without going back to the low value of the original leaky bucket.

One way to get smoother traffic is to put a leaky bucket after the token bucket. The rate of the leaky bucket should be higher than the token bucket's ρ but lower than the maximum rate of the network. Figure 5-25(f) shows the output for a 500 KB token bucket followed by a 10-MB/sec leaky bucket.

Policing all these schemes can be a bit tricky. Essentially, the network has to simulate the algorithm and make sure that no more packets or bytes are being sent than are permitted. Excess packets are then discarded or downgraded, as discussed later.

5.3.4. Flow Specifications

Traffic shaping is most effective when the sender, receiver, and subnet all agree to it. To get agreement, it is necessary to specify the traffic pattern in a precise way. Such an agreement is called a **flow specification**. It consists of a data structure that describes both the pattern of the injected traffic and the quality of service desired by the applications. A flow specification can apply either to the packets sent on a virtual circuit, or to a sequence of datagrams sent between a source and a destination (or even to multiple destinations).

In this section we will describe an example flow specification designed by Partridge (1992). It is shown in Fig. 5-27. The idea is that before a connection is

RED HAT 1057
RED HAT V. CAS

established or before a sequence of datagrams are sent, the source gives the flow specification to the subnet for approval. The subnet can either accept it, reject it, or come back with a counterproposal ("I cannot give you 100 msec average delay; can you live with 150 msec?"). Once the sender and subnet have struck a deal, the sender can ask the receiver if it, too, agrees.

Characteristics of the Input	Service Desired
Maximum packet size (bytes)	Loss sensitivity (bytes)
Token bucket rate (bytes/sec)	Loss interval (μsec)
Token bucket size (bytes)	Burst loss sensitivity (packets)
Maximum transmission rate (bytes/sec)	Minimum delay noticed (μsec)
	Maximum delay variation (μsec)
	Quality of guarantee

Fig. 5-27. An example flow specification.

Let us now examine the parameters of our example flow specification starting with the traffic specification. The *Maximum packet size* tells how big packets may be. The next two parameters implicitly assume that traffic will be shaped by the token bucket algorithm working in bytes. They tell how many bytes are put into the token bucket per second, and how big the bucket is. If the rate is r bytes/sec and the bucket size is b bytes, then in any arbitrary time interval Δt, the maximum number of bytes that may be sent is $b + r\Delta t$. Here the first term represents the maximum possible contents of the bucket at the start of the interval and the second one represents the new credits that come in during the interval. The *Maximum transmission rate* is the top rate the host is capable of producing under any conditions and implicitly specifies the shortest time interval in which the token bucket could be emptied.

The second column specifies what the application wants from the subnet. The first and second parameters represent the numerator and denominator of a fraction giving the maximum acceptable loss rate (e.g., 1 byte per hour). Alternatively, they can indicate that the flow is insensitive to packet loss. The *Burst loss sensitivity* tells how many consecutive lost packets can be tolerated.

The next two service parameters deal with delay. The *Minimum delay noticed* says how long a packet can be delayed without the application noticing. For a file transfer, it might be a second, but for an audio stream 3 msec might be the limit. The *Maximum delay variation* tries to quantify the fact that some applications are not sensitive to the actual delay but are highly sensitive to the **jitter**, that is, the amount of variation in the end-to-end packet transit time. It is two times the number of microseconds a packet's delay may vary from the average. Thus a value of 2000 means that a packet may be up to 1 msec early or late, but no more.

RED HAT 1057
RED HAT V. CAS

Finally, the *Quality of guarantee* indicates whether or not the application really means it. On the one hand, the loss and delay characteristics might be ideal goals, but no harm is done if they are not met. On the other hand, they might be so important that if they cannot be met, the application simply terminates. Intermediate positions are also possible.

Although we have looked at the flow specification as a request from the application to the subnet, it can also be a return value telling what the subnet can do. Thus it can potentially be used for an extended negotiation about the service level.

A problem inherent with any flow specification is that the application may not know what it really wants. For example, an application program running in New York might be quite happy with a delay of 200 msec to Sydney, but most unhappy with the same 200-msec delay to Boston. Here the "minimum service" is clearly a function of what is thought to be possible.

5.3.5. Congestion Control in Virtual Circuit Subnets

The congestion control methods described above are basically open loop: they try to prevent congestion from occurring in the first place, rather than dealing with it after the fact. In this section we will describe some approaches to dynamically controlling congestion in virtual circuit subnets. In the next two, we will look at techniques that can be used in any subnet.

One technique that is widely used to keep congestion that has already started from getting worse is **admission control**. The idea is simple: once congestion has been signaled, no more virtual circuits are set up until the problem has gone away. Thus, attempts to set up new transport layer connections fail. Letting more people in just makes matters worse. While this approach is crude, it is simple and easy to carry out. In the telephone system, when a switch gets overloaded, it also practices admission control, by not giving dial tones.

An alternative approach is to allow new virtual circuits but carefully route all new virtual circuits around problem areas. For example, consider the subnet of Fig. 5-28(a), in which two routers are congested, as indicated.

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 subnet as shown in Fig. 5-28(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.

Another strategy relating to virtual circuits is to negotiate an agreement between the host and subnet when a virtual circuit is set up. This agreement normally specifies the volume and shape of the traffic, quality of service required, and other parameters. To keep its part of the agreement, the subnet will typically reserve resources along the path when the circuit is set up. These resources can include table and buffer space in the routers and bandwidth on the lines. In this

RED HAT 1057
RED HAT V. CAS

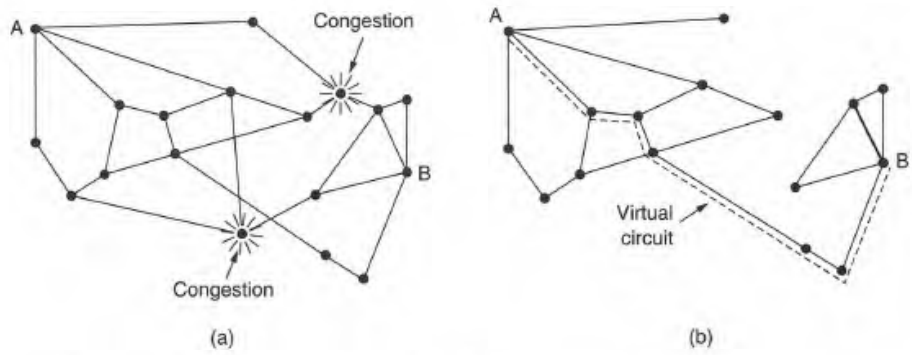


Fig. 5-28. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion and a virtual circuit from *A* to *B*.

way, congestion is unlikely to occur on the new virtual circuits because all the necessary resources are guaranteed to be available.

This kind of reservation can be done all the time as standard operating procedure, or only when the subnet is congested. A disadvantage of doing it all the time is that it tends to waste resources. If six virtual circuits that might use 1 Mbps all pass through the same physical 6-Mbps line, the line has to be marked as full, even though it may rarely happen that all six virtual circuits are transmitting at the same time. Consequently, the price of the congestion control is unused bandwidth.

5.3.6. Choke Packets

Let us now turn to an approach that can be used in both virtual circuit and datagram subnets. Each router can easily monitor the utilization of its output lines and other resources. For example, it can associate with each line a real variable, u, whose value, between 0.0 and 1.0, reflects the recent utilization of that line. To maintain a good estimate of u, a sample of the instantaneous line utilization, f (either 0 or 1), can be made periodically and u updated according to

$$u_{\text{new}} = au_{\text{old}} + (1 - a)f$$

where the constant a determines how fast the router forgets recent history.

Whenever u moves above the threshold, the output line enters a "warning" state. Each newly arriving packet is checked to see if its output line is in warning state. If so, the router sends a **choke packet** back to the source host, giving it the destination found in the packet. The original packet is tagged (a header bit is turned on) so that it will not generate any more choke packets further along the path and is then forwarded in the usual way.

RED HAT 1057
RED HAT V. CAS

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by X percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval. After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again. The feedback implicit in this protocol can help prevent congestion yet not throttle any flow unless trouble occurs.

Hosts can reduce traffic by adjusting their policy parameters, for example, window size or leaky bucket output rate. Typically, the first choke packet causes the data rate to be reduced to 0.50 of its previous rate, the next one causes a reduction to 0.25, and so on. Increases are done in smaller increments to prevent congestion from reoccurring quickly.

Several variations on this congestion control algorithm have been proposed. For one, the routers can maintain several thresholds. Depending on which threshold has been crossed, the choke packet can contain a mild warning, a stern warning, or an ultimatum.

Another variation is to use queue lengths or buffer utilization instead of line utilization as the trigger signal. The same exponential weighting can be used with this metric as with u, of course.

Weighted Fair Queueing

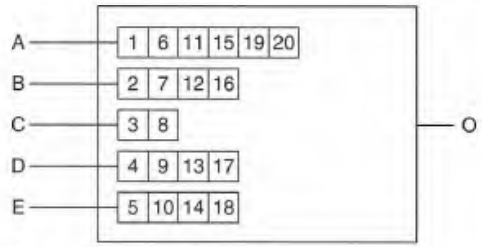
A problem with using choke packets is that the action to be taken by the source hosts is voluntary. Suppose that a router is being swamped by packets from four sources, and it sends choke packets to all of them. One of them cuts back, as it is supposed to, but the other three just keep blasting away. The result is that the honest host gets an even smaller share of the bandwidth than it had before.

To get around this problem, and thus make compliance more attractive, Nagle (1987) proposed a **fair queueing** algorithm. The essence of the algorithm is that routers have multiple queues for each output line, one for each source. When a line becomes idle, the router scans the queues round robin, taking the first packet on the next queue. In this way, with n hosts competing for a given output line, each host gets to send one out of every n packets. Sending more packets will not improve this fraction. Some ATM switches use this algorithm.

Although a start, the algorithm has a problem: 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 simulate a byte-by-byte round robin, instead of a packet-by-packet round

RED HAT 1057
RED HAT V. CAS

robin. In effect, it scans the queues repeatedly, byte-for-byte, until it finds the tick on which each packet will be finished. The packets are then sorted in order of their finishing and sent in that order. The algorithm is illustrated in Fig. 5-29.



Packet	Finishing time
C	8
B	16
D	17
E	18
A	20

(b)

Fig. 5-29. (a) A router with five packets queued for line *O*. (b) Finishing times for the five packets.

In Fig. 5-29(a) we see packets of length 2 to 6 bytes. At (virtual) clock tick 1, the first byte of the packet on line *A* is sent. Then goes the first byte of the packet on line *B*, and so on. The first packet to finish is *C*, after eight ticks. The sorted order is given in Fig. 5-29(b). In the absence of new arrivals, the packets will be sent in the order listed, from *C* to *A*.

One problem with this algorithm is that it gives all hosts the same priority. In many situations, it is desirable to give the file and other servers more bandwidth than clients, so they can be given two or more bytes per tick. This modified algorithm is called **weighted fair queueing** and is widely used. Sometimes the weight is equal to the number of virtual circuits or flows coming out of a machine, so each process gets equal bandwidth. An efficient implementation of the algorithm is discussed in (Shreedhar and Varghese, 1995).

Hop-by-Hop Choke Packets

At high speeds and over long distances, sending a choke packet to the source hosts does not work well because the reaction is so slow. Consider, for example, a host in San Francisco (router *A* in Fig. 5-30) that is sending traffic to a host in New York (router *D* in Fig. 5-30) at 155 Mbps. If the New York host begins to run out of buffers, it will take about 30 msec for a choke packet to get back to San Francisco to tell it to slow down. The choke packet propagation is shown as the second, third, and fourth steps in Fig. 5-30(a). In those 30 msec, another 4.6 megabits (e.g., over 10,000 ATM cells) will have been sent. Even if the host in San Francisco completely shuts down immediately, the 4.6 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5-30(a) will the New York router notice a slower flow.

RED HAT 1057
RED HAT V. CAS

An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-30(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 flow, 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 more detail and simulation results are given in (Mishra and Kanakia, 1992).

5.3.7. 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.

A router drowning in packets can just pick packets at random to drop, but usually it can do better than that. Which packet to discard may depend on the applications running. For file transfer, an old packet is worth more than a new one because dropping packet 6 and keeping packets 7 through 10 will cause a gap at the receiver that may force packets 6 through 10 to be retransmitted (if the receiver routinely discards out-of-order packets). In a 12-packet file, dropping 6 may require 7 through 12 to be retransmitted, whereas dropping 10 may require only 10 through 12 to be retransmitted. In contrast, for multimedia, a new packet is more important than an old one. The former policy (old is better than new) is often called **wine** and the latter (new is better than old) is often called **milk**.

A step above this in intelligence requires cooperation from the senders. For many applications, some packets are more important than others. For example, certain algorithms for compressing video periodically transmit an entire frame and then send subsequent 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. As another example, consider transmitting a document containing ASCII text and pictures. Losing a line of pixels in some image is far less damaging than losing a line of readable text.

To implement an intelligent discard policy, applications must mark their packets in priority classes to indicate how important they are. If they do this, when

RED HAT 1057
RED HAT V. CAS

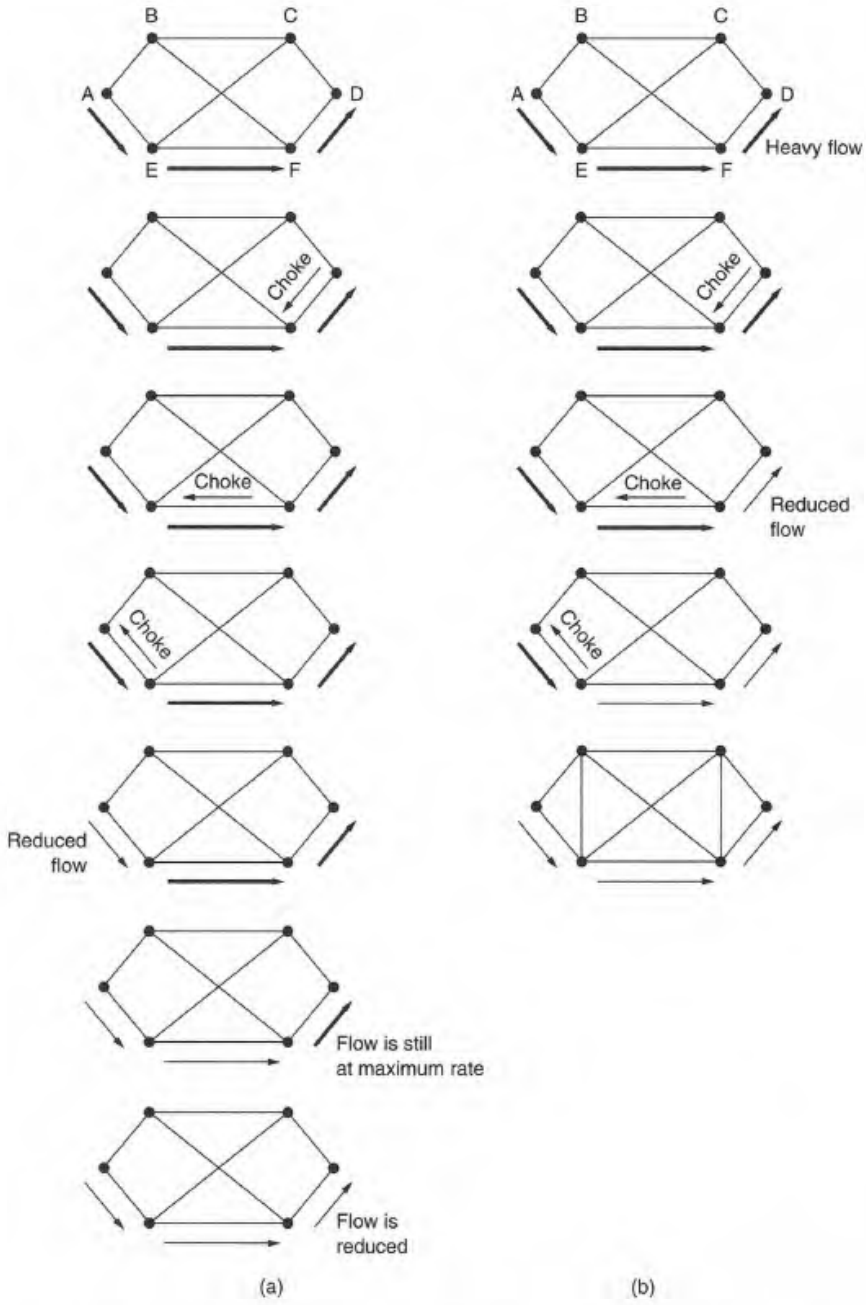


Fig. 5-30. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

RED HAT 1057
RED HAT V. CAS

packets have to be discarded, routers can first drop packets from the lowest class, then the next lowest class, and so on. Of course, unless there is some significant incentive to mark packets as anything other than VERY IMPORTANT—NEVER, EVER DISCARD, nobody will do it.

The incentive might be in the form of money, with the low-priority packets being cheaper to send than the high-priority ones. Alternatively, priority classes could be coupled with traffic shaping. For example, there might be a rule saying that when the token bucket algorithm is being used and a packet arrives at a moment when no token is available, it may still be sent, provided that it is marked as the lowest possible priority, and thus subject to discard the instant trouble appears. Under conditions of light load, users might be happy to operate in this way, but as the load increases and packets actually begin to be discarded, they might cut back and only send packets when tokens are available.

Another option is to allow hosts to exceed the limits specified in the agreement negotiated when the virtual circuit was set up (e.g., use a higher bandwidth than allowed), but subject to the condition that all excess traffic be marked 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.

Marking packets by class requires one or more header bits in which to put the priority. ATM cells have 1 bit reserved in the header for this purpose, so every ATM cell is labeled either as low priority or high priority. ATM switches indeed use this bit when making discard decisions.

In some networks, packets are grouped together into larger units that are used for retransmission purposes. For example, in ATM networks, what we have been calling "packets" are fixed-length cells. These cells are just fragments of "messages." When a cell is dropped, ultimately the entire "message" will be retransmitted, not just the missing cell. Under these conditions, a router that drops a cell might as well drop all the rest of the cells in that message, since transmitting them costs bandwidth and wins nothing—even if they get through they will still be retransmitted later.

Simulation results show that when a router senses trouble on the horizon, it is better off starting to discard packets early, rather than wait until it becomes completely clogged up (Floyd and Jacobson, 1993; Romanow and Floyd, 1994). Doing so may prevent the congestion from getting a foothold.

5.3.8. Jitter Control

For applications such as audio and video transmission, it does not matter much if the packets take 20 msec or 30 msec to be delivered, as long as the transit time is constant. Having some packets taking 20 msec and others taking 30 msec will give an uneven quality to the sound or image. Thus the agreement might be that 99 percent of the packets be delivered with a delay in the range of 24.5 msec

RED HAT 1057
RED HAT V. CAS

to 25.5 msec. The mean value chosen must be feasible, of course. In other words, an average amount of congestion must be calculated in.

The jitter can be bounded by computing the expected transit time for each hop along the path. When a packet arrives at a router, the router checks to see how much the packet is behind or ahead of its schedule. This information is stored in the packet and updated at each hop. If the packet is ahead of schedule, it is held just long enough to get it back on schedule. If it is behind schedule, the router tries to get it out the door quickly. In fact, the algorithm for determining which of several packets competing for an output line should go next can always choose the packet furthest behind in its schedule. In this way, packets that are ahead of schedule get slowed down and packets that are behind schedule get speeded up, in both cases reducing the amount of jitter.

5.3.9. Congestion Control for Multicasting

All of the congestion control algorithms discussed so far deal with messages from a single source to a single destination. In this section we will describe a way of managing multicast flows from multiple sources to multiple destinations. For example, imagine several closed-circuit television stations transmitting audio and video streams to a group of receivers, each of whom can view one or more stations at once and are free to switch from station to station at will. An application of this technology might be a video conference, in which each participant could focus on the current speaker or on the boss' expression, as desired.

In many multicast applications, groups can change membership dynamically, for example, as people enter a video conference or get bored and switch to a soap opera. Under these conditions, the approach of having the senders reserve bandwidth in advance does not work well, as it would require each sender to track all entries and exits of its audience and regenerate the spanning tree at each change. For a system designed to transmit cable television, with millions of subscribers, it would not work at all.

RSVP—Resource reSerVation Protocol

One interesting solution that can handle this environment is the **RSVP** protocol (Zhang et al., 1993). It 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

RED HAT 1057
RED HAT V. CAS

algorithm is not part of RSVP. The only difference with 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. 5-31(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. 5-31(b) and Fig. 5-31(c), respectively.

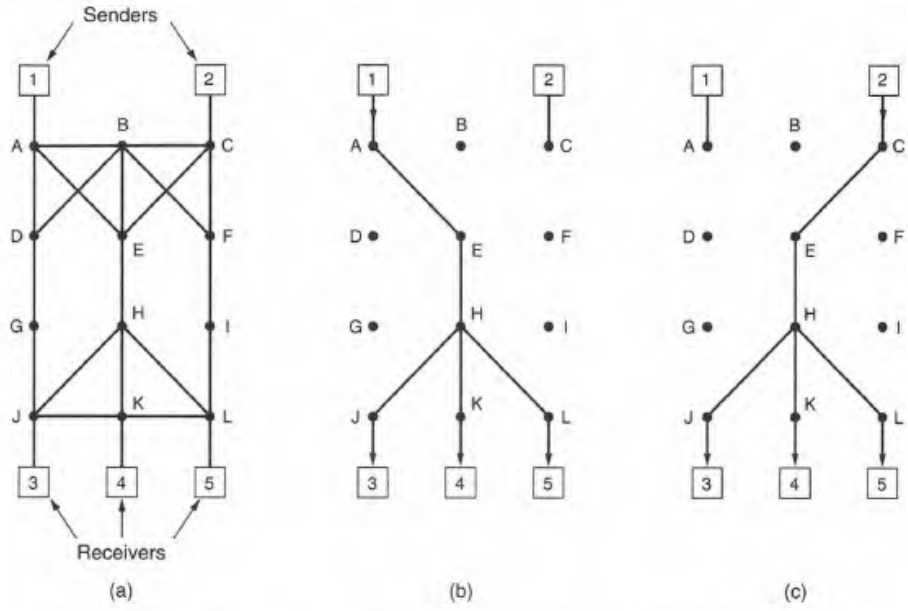


Fig. 5-31. (a) A network. (b) The multicast spanning tree for host 1. (c) The multicast spanning tree for host 2.

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. 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. 5-32(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

RED HAT 1057
RED HAT V. CAS

programs at once. A second path is reserved, as illustrated in Fig. 5-32(b). Note that two separate channels are needed from host 3 to router *E* because two independent streams are being transmitted.

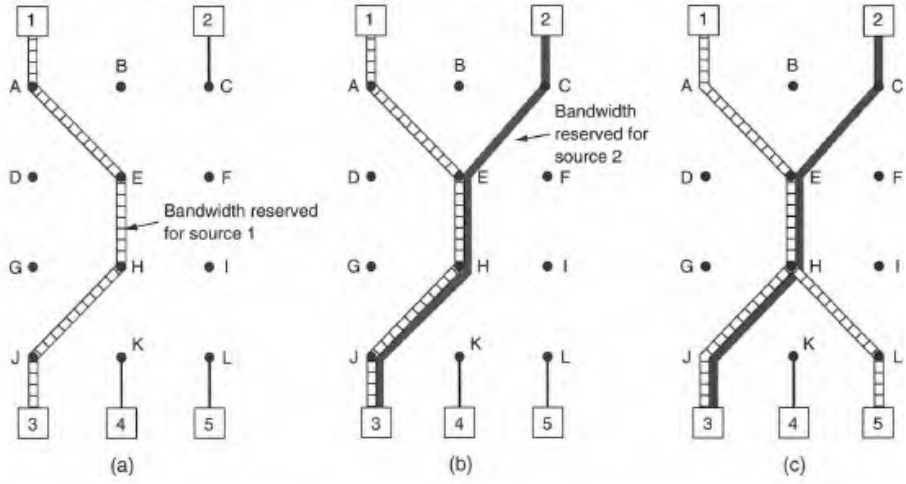


Fig. 5-32. (a) Host 3 requests a channel to host 1. (b) Host 3 then requests a second channel, to host 2. (c) Host 5 requests a channel to host 1.

Finally, in Fig. 5-32(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., 3 has a black-and-white television set, so it does not want the color 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, for example, host 3 may switch between them at will without changing its reservation: the routers do not care what program the receiver is watching.

RED HAT 1057
RED HAT V. CAS

5.4. INTERNETWORKING

Up 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 LANs, MANs, and WANs. Numerous protocols are in widespread use in every layer. In the following sections we will take a careful look at the issues that arise when two or more networks are together to form an **internet**.

Considerable controversy exists about the question of whether today's abundance of network types is a temporary condition that will go away as soon as everyone realizes how wonderful [fill in your favorite network] is, or whether it is an inevitable, but permanent feature of the world that is here to stay. Having different networks invariably means having different protocols.

We believe that a variety of different networks (and thus protocols) will always be around, for the following reasons. First of all, the installed base of different networks is large and growing. Nearly all UNIX shops run TCP/IP. Many large businesses still have mainframes running SNA. DEC is still developing DECnet. Personal computer LANs often use Novell NCP/IPX or AppleTalk. ATM systems are starting to be widespread. Finally, specialized protocols are often used on satellite, cellular, and infrared networks. This trend will continue for years due to the large number of existing networks and because not all vendors perceive it in their interest for their customers to be able to easily migrate to another vendor's system.

Second, as computers and networks get cheaper, the place where decisions get made moves downward. Many companies have a policy to the effect that purchases costing over a million dollars have to be approved by top management, purchases costing over 100,000 dollars have to be approved by middle management, but purchases under 100,000 dollars can be made by department heads without any higher approval. This can easily lead to the accounting department installing an Ethernet, the engineering department installing a token bus, and the personnel department installing a token ring.

Third, different networks (e.g., ATM and wireless) have radically different technology, so it should not be surprising that as new hardware developments occur, new software will be created to fit the new hardware. For example, the average home now is like the average office ten years ago: it is full of computers that do not talk to one another. In the future, it may be commonplace for the telephone, the television set, and other appliances all to be networked together, so they can be controlled remotely. This new technology will undoubtedly bring new protocols.

As an example of how different networks interact, consider the following example. At most universities, the computer science and electrical engineering departments have their own LANs, often different. In addition, the university computer center often has mainframes and supercomputers, the former for faculty

RED HAT 1057
RED HAT V. CAS

members in the humanities who do not wish to get into the computer maintenance business, and the latter for physicists who want to crunch numbers. As a consequence of these various networks and facilities, the following scenarios are easy to imagine:

1. LAN-LAN: A computer scientist downloading a file to engineering.
2. LAN-WAN: A computer scientist sending mail to a distant physicist.
3. WAN-WAN: Two poets exchanging sonnets.
4. LAN-WAN-LAN: Engineers at different universities communicating.

Figure 5-33 illustrates these four types of connections as dotted lines. In each case, it is necessary to insert a "black box" at the junction between two networks, to handle the necessary conversions as packets move from one network to the other.

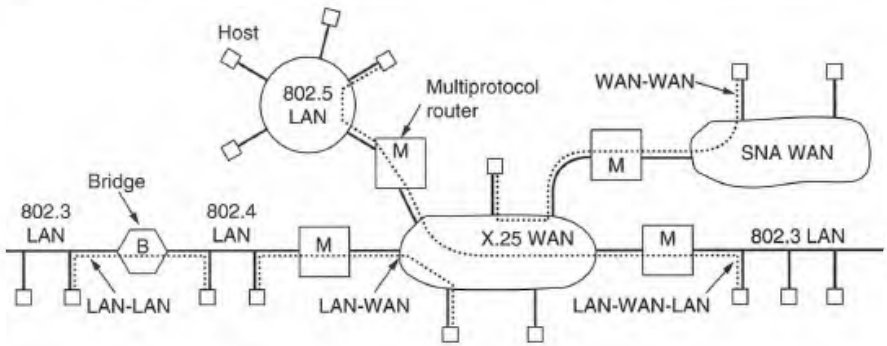


Fig. 5-33. Network interconnection.

The name used for the black box connecting two networks depends on the layer that does the work. Some common names are given below (although there is not much agreement on terminology in this area).

Layer 1: Repeaters copy individual bits between cable segments.

Layer 2: Bridges store and forward data link frames between LANs.

Layer 3: Multiprotocol routers forward packets between dissimilar networks.

Layer 4: Transport gateways connect byte streams in the transport layer.

Above 4: Application gateways allow interworking above layer 4.

For convenience, we will sometimes use the term "gateway" to mean any device that connects two or more dissimilar networks.

RED HAT 1057
RED HAT V. CAS

Repeaters are low-level devices that just amplify or regenerate weak signals. They are needed to provide current to drive long cables. In 802.3, for example, the timing properties of the MAC protocol (the value of τ chosen) allow cables up to 2.5 km, but the transceiver chips can only provide enough power to drive 500 meters. The solution is to use repeaters to extend the cable length where that is desired.

Unlike repeaters, which copy the bits as they arrive, **bridges** are store-and-forward devices. A bridge accepts an entire frame and passes it up to the data link layer where the checksum is verified. Then the frame is sent down to the physical layer for forwarding on a different network. Bridges can make minor changes to the frame before forwarding it, such as adding or deleting some fields from the frame header. Since they are data link layer devices, they do not deal with headers at layer 3 and above and cannot make changes or decisions that depend on them.

Multiprotocol routers are conceptually similar to bridges, except that they are found in the network layer. They just take incoming packets from one line and forward them on another, just as all routers do, but the lines may belong to different networks and use different protocols (e.g., IP, IPX, and the OSI connectionless packet protocol, CLNP). Like all routers, multiprotocol routers operate at the level of the network layer.

Transport gateways make a connection between two networks at the transport layer. We will discuss this possibility later when we come to concatenated virtual circuits.

Finally, **application gateways** connect two parts of an application in the application layer. For example, to send mail from an Internet machine using the Internet mail format to an ISO MOTIS mailbox, one could send the message to a mail gateway. The mail gateway would unpack the message, convert it to MOTIS format, and then forward it on the second network using the network and transport protocols used there.

When a gateway is between two WANs run by different organizations, possibly in different countries, the joint operation of one workstation-class machine can lead to a lot of finger pointing. To eliminate these problems, a slightly different approach can be taken. The gateway is effectively ripped apart in the middle and the two parts are connected with a wire. Each of the halves is called a **half-gateway** and each one is owned and operated by one of the network operators. The whole problem of gatewaying then reduces to agreeing to a common protocol to use on the wire, one that is neutral and does not favor either party. Figure 5-34 shows both full and half-gateways. Either kind can be used in any layer (e.g., half-bridges also exist).

That all said, the situation is murkier in practice than it is in theory. Many devices on the market combine bridge and router functionality. The key property of a pure bridge is that it examines data link layer frame headers and does not inspect or modify the network layer packets inside the frames. A bridge cannot

RED HAT 1057
RED HAT V. CAS

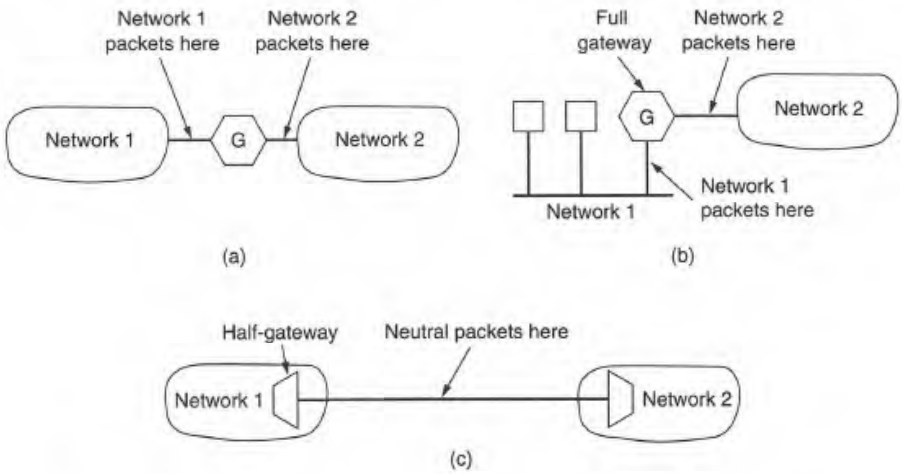


Fig. 5-34. (a) A full gateway between two WANs. (b) A full gateway between a LAN and a WAN. (c) Two half-gateways.

tell and does not care whether the frame it is forwarding from an 802.x LAN to an 802.y contains an IP, IPX, or CLNP packet in the payload field.

A router, in contrast, knows very well whether it is an IP router, an IPX router, a CLNP router, or all three combined. It examines these headers and makes decisions based on the addresses found there. On the other hand, when a pure router hands off a packet to the data link layer, it does not know or care whether it will be carried in an Ethernet frame or a token ring frame. That is the data link layer's responsibility.

The confusion in the industry comes from two sources. First, functionally, bridges and routers are not all that different. They each accept incoming PDUs (Protocol Data Units), examine some header fields, and make decisions about where to send the PDUs based on header information and internal tables.

Second, many commercial products are sold under the wrong label or combine the functionality of both bridges and routers. For example, source routing bridges are not really bridges at all, since they involve a protocol layer above the data link layer to do their job. For an illuminating discussion of bridges versus routers, see Chap. 12 of (Perlman, 1992).

5.4.1. How Networks Differ

Networks can differ in many ways. In Fig. 5-35 we list some of the differences that can occur in the network layer. It is papering over these differences that makes internetworking more difficult than operating within a single network.

RED HAT 1057
RED HAT V. CAS

Item	Some Possibilities
Service offered	Connection-oriented versus connectionless
Protocols	IP, IPX, CLNP, AppleTalk, DECnet, etc.
Addressing	Flat (802) versus hierarchical (IP)
Multicasting	Present or absent (also broadcasting)
Packet size	Every network has its own maximum
Quality of service	May be present or absent; many different kinds
Error handling	Reliable, ordered, and unordered delivery
Flow control	Sliding window, rate control, other, or none
Congestion control	Leaky bucket, choke packets, etc.
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, by packet, by byte, or not at all

Fig. 5-35. Some of the many ways networks can differ.

When packets sent by a source on one network must transit one or more foreign networks before reaching the destination network (which also may be different from the source network), many problems can occur at the interfaces between networks. To start with, when packets from a connection-oriented network must transit a connectionless one, they may be reordered, something the sender does not expect and the receiver is not prepared to deal with. Protocol conversions will often be needed, which can be difficult if the required functionality cannot be expressed. Address conversions will also be needed, which may require some kind of directory system. Passing multicast packets through a network that does not support multicasting requires generating separate packets for each destination.

The differing maximum packet sizes used by different networks is a major headache. How do you pass an 8000-byte packet through a network whose maximum size is 1500 bytes? Differing qualities of service is an issue when a packet that has real-time delivery constraints passes through a network that does not offer any real-time guarantees.

Error, flow, and congestion control frequently differ among different networks. If the source and destination both expect all packets to be delivered in sequence without error, yet an intermediate network just discards packets whenever it smells congestion on the horizon, or packets can wander around aimlessly for a while and then suddenly emerge and be delivered, many applications will break. Different security mechanisms, parameter settings, and accounting rules, and even national privacy laws also can cause problems.

RED HAT 1057
RED HAT V. CAS

5.4.2. Concatenated Virtual Circuits

Two styles of internetworking are common: a connection-oriented concatenation of virtual circuit subnets, and a datagram internet style. We will now examine these in turn. In the concatenated virtual circuit model, shown in Fig. 5-36, a connection to a host in a distant network is set up in a way similar to the way connections are normally established. The subnet sees that the destination is remote and builds a virtual circuit to the router nearest the destination network. Then it constructs a virtual circuit from that router to an external "gateway" (multiprotocol router). This gateway records the existence of the virtual circuit in its tables and proceeds to build another virtual circuit to a router in the next subnet. This process continues until the destination host has been reached.

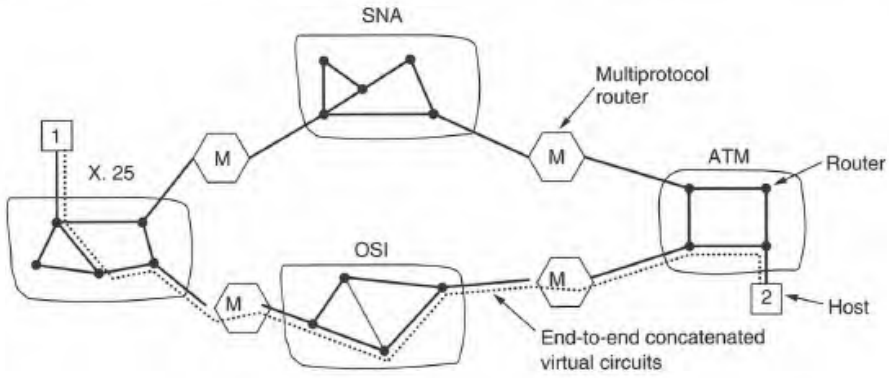


Fig. 5-36. Internetworking using concatenated virtual circuits.

Once data packets begin flowing along the path, each gateway relays incoming packets, converting between packet formats and virtual circuit numbers as needed. Clearly, all data packets must traverse the same sequence of gateways, and thus arrive in order.

The essential feature of this approach is that a sequence of virtual circuits is set up from the source through one or more gateways to the destination. Each gateway maintains tables telling which virtual circuits pass through it, where they are to be routed, and what the new virtual circuit number is.

Although Fig. 5-36 shows the connection made with a full gateway, it could equally well be done with half-gateways.

This scheme works best when all the networks have roughly the same properties. For example, if all of them guarantee reliable delivery of network layer packets, then barring a crash somewhere along the route, the flow from source to destination will also be reliable. Similarly, if none of them guarantee reliable delivery, then the concatenation of the virtual circuits is not reliable either. On

RED HAT 1057
RED HAT V. CAS

the other hand, if the source machine is on a network that does guarantee reliable delivery, but one of the intermediate networks can lose packets, the concatenation has fundamentally changed the nature of the service.

Concatenated virtual circuits are also common in the transport layer. In particular, it is possible to build a bit pipe using, say, OSI, which terminates in a gateway, and have a TCP connection go from the gateway to the next gateway. In this manner, an end-to-end virtual circuit can be built spanning different networks and protocols.

5.4.3. Connectionless Internetworking

The alternative internetwork model is the datagram model, shown in Fig. 5-37. In this model, the only service the network layer offers to the transport layer is the ability to inject datagrams into the subnet and hope for the best. There is no notion of a virtual circuit at all in the network layer, let alone a concatenation of them. This model does not require all packets belonging to one connection to traverse the same sequence of gateways. In Fig. 5-37 datagrams from host 1 to host 2 are shown taking different routes through the internetwork. A routing decision is made separately for each packet, possibly depending on the traffic at the moment the packet is sent. This strategy can use multiple routes and thus achieve a higher bandwidth than the concatenated virtual circuit model. On the other hand, there is no guarantee that the packets arrive at the destination in order, assuming that they arrive at all.

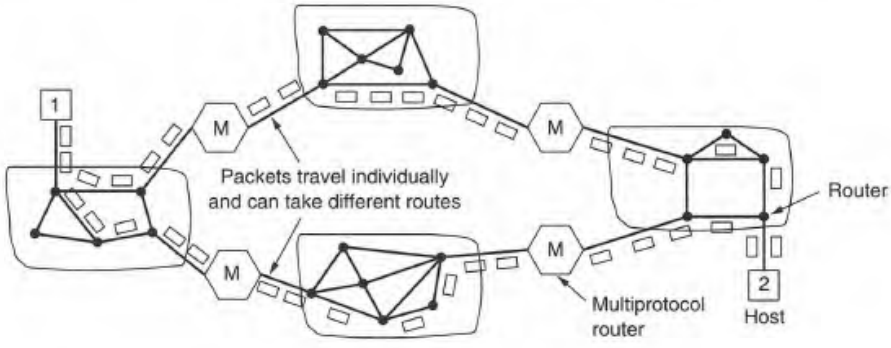


Fig. 5-37. A connectionless internet.

The model of Fig. 5-37 is not quite as simple as it looks. For one thing, if each network has its own network layer protocol, it is not possible for a packet from one network to transit another one. One could imagine the multiprotocol routers actually trying to translate from one format to another, but unless the two

RED HAT 1057
RED HAT V. CAS

formats are close relatives with the same information fields, such conversions will always be incomplete and often doomed to failure. For this reason, conversion is rarely attempted.

A second, and more serious problem, is addressing. Imagine a simple case: a host on the Internet is trying to send an IP packet to a host on an adjoining OSI network. The OSI datagram protocol, CLNP, was based on IP and is close enough to it that a conversion might well work. The trouble is that IP packets all carry the 32-bit Internet address of the destination host in a header field. OSI hosts do not have 32-bit Internet addresses. They use decimal addresses similar to telephone numbers.

To make it possible for the multiprotocol router to convert between formats, someone would have to assign a 32-bit Internet address to each OSI host. Taken to the limit, this approach would mean assigning an Internet address to every machine in the world that an Internet host might want to talk to. It would also mean assigning an OSI address to every machine in the world that an OSI host might want to talk to. The same problem occurs with every other address space (SNA, AppleTalk, etc.). The problems here are insurmountable. In addition, someone would have to maintain a database mapping everything to everything.

Another idea is to design a universal "internet" packet and have all routers recognize it. This approach is, in fact, what IP is—a packet designed to be carried through many networks. The only problem is that IPX, CLNP, and other "universal" packets exist too, making all of them less than universal. Getting everybody to agree to a single format is just not possible.

Let us now briefly recap the two ways internetworking can be attacked. The concatenated virtual circuit model has essentially the same advantages as using virtual circuits within a single subnet: buffers can be reserved in advance, sequencing can be guaranteed, short headers can be used, and the troubles caused by delayed duplicate packets can be avoided.

It also has the same disadvantages: table space required in the routers for each open connection, no alternate routing to avoid congested areas, and vulnerability to router failures along the path. It also has the disadvantage of being difficult, if not impossible, to implement if one of the networks involved is an unreliable datagram network.

The properties of the datagram approach to internetworking are the same as those of datagram subnets: more potential for congestion, but also more potential for adapting to it, robustness in the face of router failures, and longer headers needed. Various adaptive routing algorithms are possible in an internet, just as they are within a single datagram network.

A major advantage of the datagram approach to internetworking is that it can be used over subnets that do not use virtual circuits inside. Many LANs, mobile networks (e.g., aircraft and naval fleets), and even some WANs fall into this category. When an internet includes one of these, serious problems occur if the internetworking strategy is based on virtual circuits.

RED HAT 1057
RED HAT V. CAS

5.4.4. Tunneling

Handling the general case of making two different networks interwork is exceedingly difficult. However, there is a common special case that is manageable. 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 a TCP/IP based Ethernet in Paris, a TCP/IP based Ethernet in London, and a PTT WAN in between, as shown in Fig. 5-38.

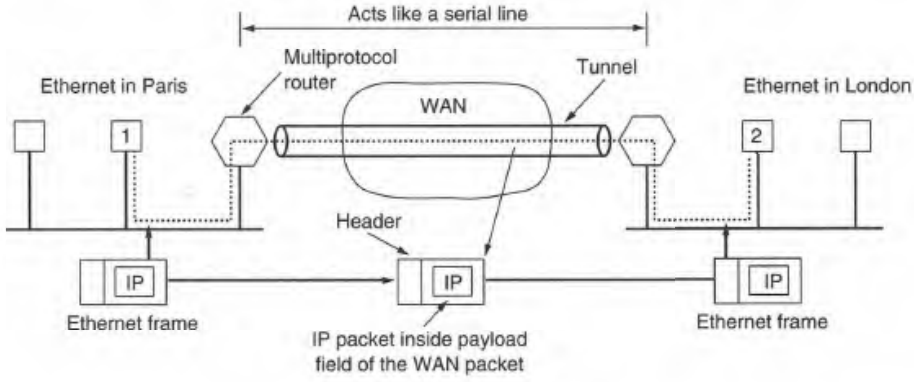


Fig. 5-38. Tunneling a packet from Paris to London.

The solution to this problem is a technique called **tunneling**. To send an IP packet to host 2, host 1 constructs the packet containing the IP address of host 2, inserts it into an Ethernet frame addressed to the Paris multiprotocol router, and puts it on the Ethernet. When the multiprotocol router gets the frame, it removes the IP packet, inserts it in the payload field of the WAN network layer packet, and addresses the latter to the WAN address of the London multiprotocol router. When it gets there, the London router removes the IP packet and sends it to host 2 inside an Ethernet frame.

The WAN can be seen as a big tunnel extending from one multiprotocol router to the other. The IP 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 the WAN at all. Neither do the hosts on either Ethernet. Only the multiprotocol router has to understand IP and WAN packets. In effect, the entire distance from the middle of one multiprotocol router to the middle of the other acts like a serial line.

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 into 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. 5-39.

RED HAT 1057
RED HAT V. CAS

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.

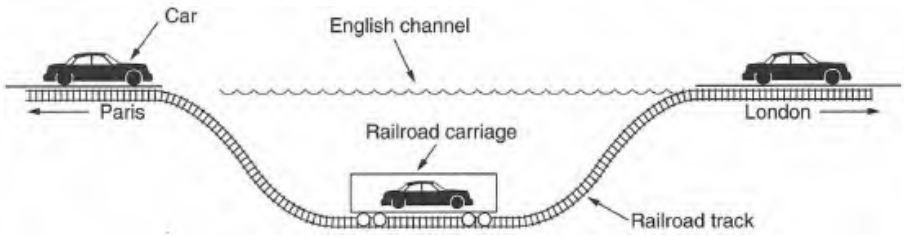


Fig. 5-39. Tunneling a car from France to England.

5.4.5. Internetwork Routing

Routing through an internetwork is similar to routing within a single subnet, but with some added complications. Consider, for example, the internetwork of Fig. 5-40(a) in which five networks are connected by six multiprotocol routers. Making a graph model of this situation is complicated by the fact that every multiprotocol router can directly access (i.e., send packets to) every other router connected to any network to which it is connected. For example, *B* in Fig. 5-40(a) can directly access *A* and *C* via network 2 and also *D* via network 3. This leads to the graph of Fig. 5-40(b).

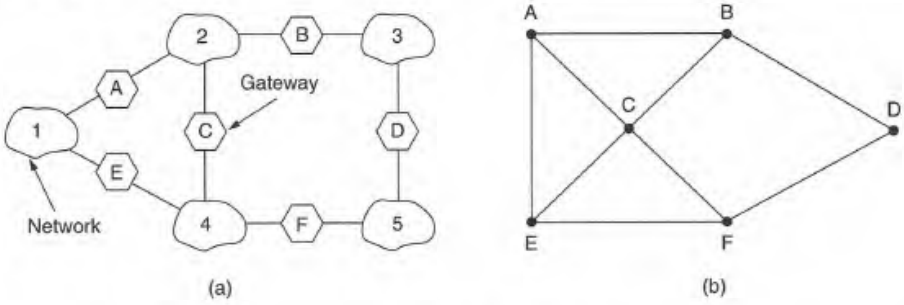


Fig. 5-40. (a) An internetwork. (b) A graph of the internetwork.

Once the graph has been constructed, known routing algorithms, such as the distance vector and link state algorithms, can be applied to the set of multiprotocol routers. This gives a two-level routing algorithm: within each network an **interior gateway protocol** is used, but between the networks, an **exterior gateway protocol** is used ("gateway" is an older term for "router"). In fact, since

RED HAT 1057
RED HAT V. CAS

each network is independent, they may all use different algorithms. Because each network in an internetwork is independent of all the others, it is often referred to as an **Autonomous System (AS)**.

A typical internet packet starts out on its LAN addressed to the local multiprotocol router (in the MAC layer header). After it gets there, the network layer code decides which multiprotocol router to forward the packet to, using its own routing tables. If that router can be reached using the packet's native network protocol, it is forwarded there directly. Otherwise it is tunneled there, encapsulated in the protocol required by the intervening network. This process is repeated until the packet reaches the destination network.

One of the differences between internetwork routing and intranetwork routing is that internetwork routing often requires crossing international boundaries. Various laws suddenly come into play, such as Sweden's strict privacy laws about exporting personal data about Swedish citizens from Sweden. Another example is the Canadian law saying that data traffic originating in Canada and ending in Canada may not leave the country. This law means that traffic from Windsor, Ontario to Vancouver may not be routed via nearby Detroit.

Another difference between interior and exterior routing is the cost. Within a single network, a single charging algorithm normally applies. However, different networks may be under different managements, and one route may be less expensive than another. Similarly, the quality of service offered by different networks may be different, and this may be a reason to choose one route over another.

In a large internetwork, choosing the best route may be a time-consuming operation. Estrin et al. (1992) have proposed dealing with this problem by precomputing routes for popular (source, destination) pairs and storing them in a database to be consulted at route selection time.

5.4.6. Fragmentation

Each network imposes some maximum size on its packets. These limits have various causes, among them:

1. Hardware (e.g., the width of a TDM transmission slot).
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 maximum packet size they wish. Maximum payloads range from 48 bytes

RED HAT 1057
RED HAT V. CAS

(ATM cells) to 65,515 bytes (IP packets), although the payload size in higher layers is often larger.

An obvious problem appears when a large packet wants to travel through a network whose maximum packet size is too small. One solution is to make sure the problem does not occur in the first place. In other words, the internet should use a routing algorithm that avoids sending packets through networks that cannot handle them. However, this solution is no solution at all. What happens if the original source packet is too large to be handled by the destination network? The routing algorithm can hardly bypass the destination.

Basically, the only solution to the problem is to allow gateways to break packets up into **fragments**, sending each fragment as a separate internet 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.

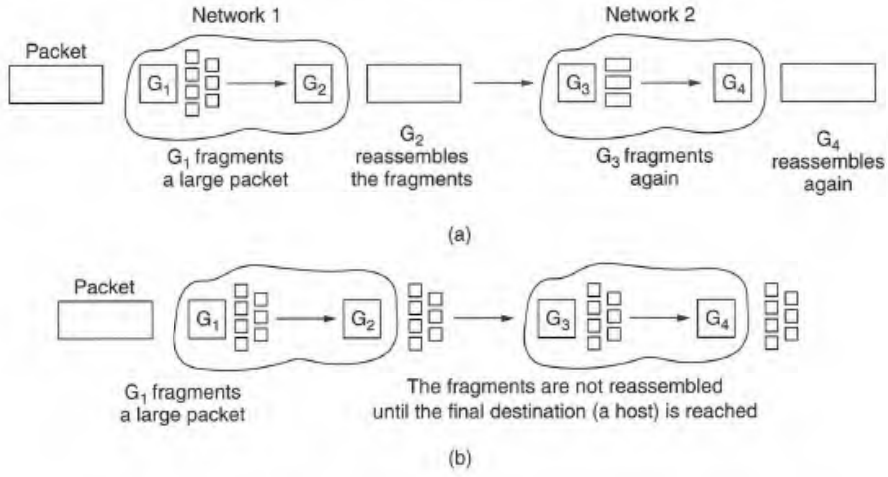


Fig. 5-41. (a) Transparent fragmentation. (b) Nontransparent fragmentation.

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. 5-41(a). When an oversized packet arrives at a gateway, the gateway breaks it up into fragments. Each fragment is addressed to the same exit gateway, where the pieces are recombined. In this way passage through the small-packet network has been made transparent. Subsequent networks are not even aware that fragmentation has occurred. ATM networks, for example, have special hardware to

RED HAT 1057
RED HAT V. CAS

provide transparent fragmentation of packets into cells and then reassembly of cells into packets. In the ATM world, fragmentation is called segmentation; the concept is the same, but some of the details are different.

Transparent fragmentation is simple but has some problems. For one thing, the exit gateway must know when it has received all the pieces, so that either a count field or an "end of packet" bit must be included in each packet. For another thing, all packets must exit via the same gateway. By not allowing some fragments to follow one route to the ultimate destination, and other fragments a disjoint route, some performance may be lost. A last problem is the overhead required to repeatedly reassemble and then refragment a large packet passing through a series of small-packet networks.

The other fragmentation strategy is to refrain from recombining fragments at any intermediate gateways. Once a packet has been fragmented, each fragment is treated as though it were an original packet. All fragments are passed through the exit gateway (or gateways), as shown in Fig. 5-41(b). Recombination occurs only at the destination host.

Nontransparent fragmentation also has some problems. For example, it requires *every* host to be able to do reassembly. Yet another problem is that when a large packet is fragmented the total overhead increases, because each fragment must have a header. Whereas in the first method this overhead disappears as soon as the small-packet network is exited, in this method the overhead remains for the rest of the journey. An advantage of this method, however, is that multiple exit gateways can now be used and higher performance can be achieved. Of course, if the concatenated virtual circuit model is being used, this advantage is of no use.

When a packet is fragmented, the fragments must be numbered in such a way that the original data stream can be reconstructed. One way of numbering the fragments is to use a tree. If packet 0 must be split up, the pieces are called 0.0, 0.1, 0.2, etc. If these fragments themselves must be fragmented later on, the pieces are numbered 0.0.0, 0.0.1, 0.0.2, . . . , 0.1.0, 0.1.1, 0.1.2, etc. If enough fields have been reserved in the header for the worst case and no duplicates are generated anywhere, this scheme is sufficient to ensure that all the pieces can be correctly reassembled at the destination, no matter what order they arrive in.

However, if even one network loses or discards packets, there is a need for end-to-end retransmissions, with unfortunate effects for the numbering system. Suppose that a 1024-bit packet is initially fragmented into four equal-sized fragments, 0.0, 0.1, 0.2, and 0.3. Fragment 0.1 is lost, but the other parts arrive at the destination. Eventually, the source times out and retransmits the original packet again. Only this time the route taken passes through a network with a 512-bit limit, so two fragments are generated. When the new fragment 0.1 arrives at the destination, the receiver will think that all four pieces are now accounted for and reconstruct the packet incorrectly.

A completely different (and better) numbering system is for the internetwork protocol to define an elementary fragment size small enough that the elementary

RED HAT 1057
RED HAT V. CAS

fragment can pass through every network. When a packet is fragmented, all the pieces are equal to the elementary fragment size except the last one, which may be shorter. An internet packet may contain several fragments, for efficiency reasons. The internet header must provide the original packet number, and the number of the (first) elementary fragment contained in the packet. As usual, there must also be a bit indicating that the last elementary fragment contained within the internet packet is the last one of the original packet.

This approach requires two sequence fields in the internet header: the original packet number, and the fragment number. There is clearly a trade-off between the size of the elementary fragment and the number of bits in the fragment number. Because the elementary fragment size is presumed to be acceptable to every network, subsequent fragmentation of an internet packet containing several fragments causes no problem. The ultimate limit here is to have the elementary fragment be a single bit or byte, with the fragment number then being the bit or byte offset within the original packet, as shown in Fig. 5-42.

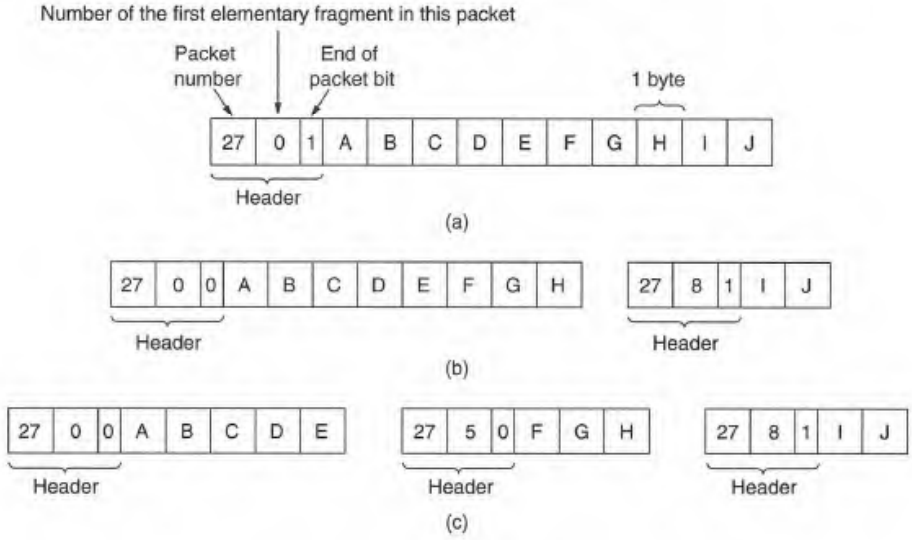


Fig. 5-42. Fragmentation when the elementary data size is 1 byte. (a) Original packet, containing 10 data bytes. (b) Fragments after passing through a network with maximum packet size of 8 bytes. (c) Fragments after passing through a size 5 gateway.

Some internet protocols take this method even further and consider the entire transmission on a virtual circuit to be one giant packet, so that each fragment contains the absolute byte number of the first byte within the fragment. Some other issues relating to fragmentation are discussed in (Kent and Mogul, 1987).

RED HAT 1057
RED HAT V. CAS

5.4.7. Firewalls

The ability to connect any computer, anywhere, to any other computer, anywhere, is a mixed blessing. For individuals at home, wandering around the Internet is lots of fun. For corporate security managers, it is a nightmare. Most companies have large amounts of confidential information on-line—trade secrets, product development plans, marketing strategies, financial analyses, etc. Disclosure of this information to a competitor could have dire consequences.

In addition to the danger of information leaking out, there is also a danger of information leaking in. In particular, viruses, worms, and other digital pests (Kaufman et al., 1995) can breach security, destroy valuable data, and waste large amounts of administrators' time trying to clean up the mess they leave. Often they are imported by careless employees who want to play some nifty new game.

Consequently, mechanisms are needed to keep "good" bits in and "bad" bits out. One method is to use encryption. This approach protects data in transit between secure sites. We will study it in Chap. 7. However, encryption does nothing to keep digital pests and hackers out. To accomplish this goal, we need to look at firewalls (Chapman and Zwicky, 1995; and Cheswick and Bellovin, 1994).

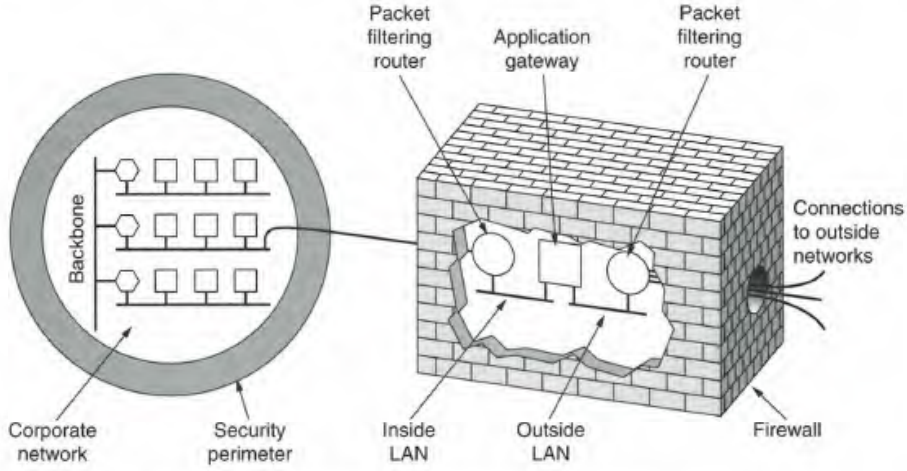


Fig. 5-43. A firewall consisting of two packet filters and an application gateway.

Firewalls are just a modern adaptation of that old medieval security standby: digging a deep moat around your castle. This design forced everyone entering or leaving the castle to pass over a single drawbridge, where they could be inspected by the I/O police. With networks, the same trick is possible: a company can have many LANs connected in arbitrary ways, but all traffic to or from the company is forced through an electronic drawbridge (firewall), as shown in Fig. 5-43.

RED HAT 1057
RED HAT V. CAS

The firewall in this configuration has two components: two routers that do packet filtering and an application gateway. Simpler configurations also exist, but the advantage of this design is that every packet must transit two filters and an application gateway to go in or out. No other route exists. Readers who think that one security checkpoint is enough clearly have not made an international flight on a scheduled airline recently.

Each **packet filter** is a standard router equipped with some extra functionality. The extra functionality allows every incoming or outgoing packet to be inspected. Packets meeting some criterion are forwarded normally. Those that fail the test are dropped.

In Fig. 5-43, most likely the packet filter on the inside LAN checks outgoing packets and the one on the outside LAN checks incoming packets. Packets crossing the first hurdle go to the application gateway for further examination. The point of putting the two packet filters on different LANs is to ensure that no packet gets in or out without having to pass through the application gateway: there is no path around it.

Packet filters are typically driven by tables configured by the system administrator. These tables list sources and destinations that are acceptable, sources and destinations that are blocked, and default rules about what to do with packets coming from or going to other machines.

In the common case of a UNIX setting, a source or destination consists of an IP address and a port. Ports indicate which service is desired. For example, port 23 is for Telnet, port 79 is for Finger, and port 119 is for USENET news. A company could block incoming packets for all IP addresses combined with one of these ports. In this way, no one outside the company could log in via Telnet, or look up people using the Finger daemon. Furthermore, the company would be spared from having employees spend all day reading USENET news.

Blocking outgoing packets is trickier because although most sites stick to the standard port naming conventions, they are not forced to do so. Furthermore, for some important services, such as FTP (File Transfer Protocol), port numbers are assigned dynamically. In addition, although blocking TCP connections is difficult, blocking UDP packets is even harder because so little is known a priori about what they will do. Many packet filters simply ban UDP traffic altogether.

The second half of the firewall mechanism is the **application gateway**. Rather than just looking at raw packets, the gateway operates at the application level. A mail gateway, for example, can be set up to examine each message going in or coming out. For each one it makes a decision to transmit or discard it based on header fields, message size, or even the content (e.g., at a military installation, the presence of words like "nuclear" or "bomb" might cause some special action to be taken).

Installations are free to set up one or more application gateways for specific applications, but it is not uncommon for suspicious organizations to permit email in and out, and perhaps use of the World Wide Web, but ban everything else as

RED HAT 1057
RED HAT V. CAS

too dicey. Combined with encryption and packet filtering, this arrangement offers a limited amount of security at the cost of some inconvenience.

One final note concerns wireless communication and firewalls. It is easy to design a system that is logically completely secure, but which, in practice, leaks like a sieve. This situation can occur if some of the machines are wireless and use radio communication, which passes right over the firewall in both directions.

5.5. THE NETWORK LAYER IN THE INTERNET

At the network layer, the Internet can be viewed as a collection of subnetworks or **Autonomous Systems** (**ASes**) that are connected together. There is no real structure, but several major backbones exist. These are constructed from high-bandwidth lines and fast routers. Attached to the backbones are regional (midlevel) networks, and attached to these regional networks are the LANs at many universities, companies, and Internet service providers. A sketch of this quasihierarchical organization is given in Fig. 5-44.

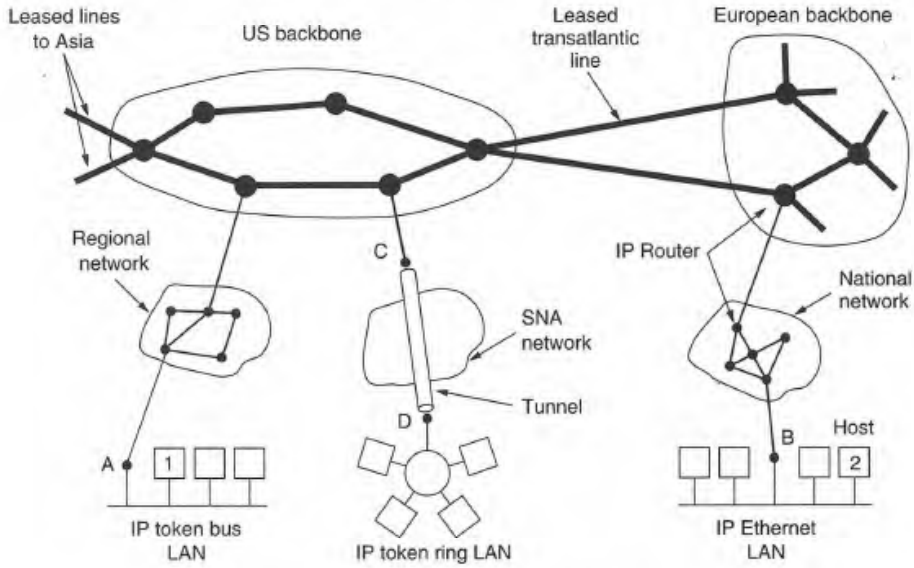


Fig. 5-44. The Internet is an interconnected collection of many networks.

The glue that holds the Internet together is the network layer protocol, **IP** (**Internet Protocol**). Unlike most older network layer protocols, it 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-efforts way to transport datagrams

RED HAT 1057
RED HAT V. CAS

from source to destination, without regard to whether or not these machines are on the same network, or whether or not there are other networks in between them.

Communication in the Internet works as follows. The transport layer takes data streams and breaks them up into datagrams. In theory, datagrams can be up to 64 Kbytes each, but in practice they are usually around 1500 bytes. Each datagram is transmitted through the Internet, possibly being fragmented into smaller units as it goes. 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, which inserts it into the receiving process' input stream.

5.5.1. The IP Protocol

An appropriate place to start our study of the network layer in the Internet is the format of the IP datagrams themselves. An IP datagram consists of a header part and a text part. The header has a 20-byte fixed part and a variable length optional part. The header format is shown in Fig. 5-45. It is transmitted in big endian order: from left to right, with the high-order bit of the *Version* field going first. (The SPARC is big endian; the Pentium is little endian.) On little endian machines, software conversion is required on both transmission and reception.

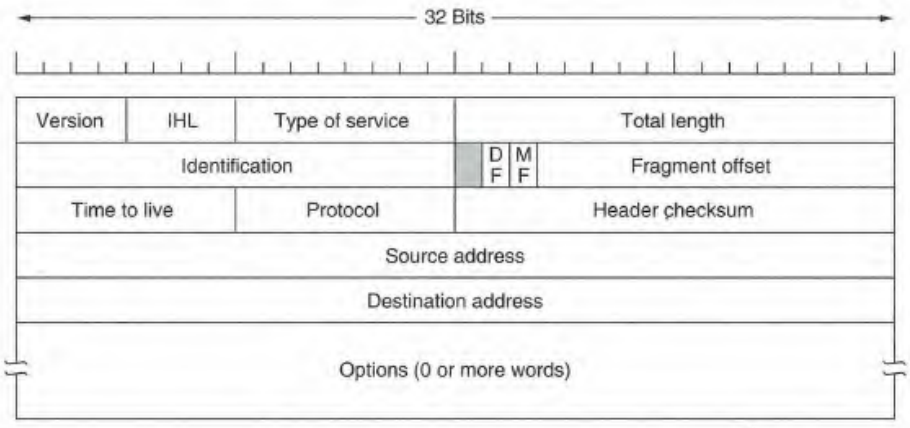


Fig. 5-45. The IP (Internet Protocol) header.

The *Version* field keeps track of which version of the protocol the datagram belongs to. By including the version in each datagram, it becomes possible to have the transition between versions take months, or even years, with some machines running the old version and others running the new one.

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

RED HAT 1057
RED HAT V. CAS

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 the option useless.

The *Type of service* field allows the host to tell the subnet what kind of service it wants. 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 field itself contains (from left to right), a three-bit *Precedence* field, three flags, *D*, *T*, and *R*, and 2 unused bits. The *Precedence* field is a priority, from 0 (normal) to 7 (network control packet). The three flag bits allow the host to specify what it cares most about from the set {Delay, Throughput, Reliability}. In theory, these fields allow routers to make choices between, for example, a satellite link with high throughput and high delay or a leased line with low throughput and low delay. In practice, current routers ignore the *Type of Service* field altogether.

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 gigabit networks larger datagrams may be needed.

The *Identification* field is needed to allow the destination host to determine which datagram a newly arrived fragment belongs to. All the fragments of a datagram contain the same *Identification* value.

Next comes an unused bit and then two 1-bit fields. *DF* stands for Don't Fragment. It is an order to the routers not to fragment the datagram because the destination is incapable of putting the pieces back together again. For example, when a computer boots, its ROM might ask for a memory image to be sent to it as a single datagram. By marking the datagram with the *DF* bit, the sender knows it will arrive in one piece, even if this means that the datagram must avoid a small-packet network on the best path and take a suboptimal route. All machines are required to accept fragments of 576 bytes or less.

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 datagram 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, giving a maximum datagram length of 65,536 bytes, one more than the *Total length* field.

The *Time to live* field is a counter used to limit packet lifetimes. It is 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 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 datagrams for wandering around forever, something that otherwise might happen if the routing tables ever become corrupted.

RED HAT 1057
RED HAT V. CAS

When the network layer has assembled a complete datagram, it needs to know what to do with it. The *Protocol* field tells it which transport process to give it to. TCP is one possibility, but so are UDP and some others. The numbering of protocols is global across the entire Internet and is defined in RFC 1700.

The *Header checksum* verifies the header only. Such a checksum is useful for detecting errors generated by bad memory words inside a router. The algorithm is to add up all the 16-bit halfwords 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. This algorithm is more robust than using a normal add. Note that the *Header checksum* 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 network number and host number. 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 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 four bytes. Currently five options are defined, as listed in Fig. 5-46, but not all routers support all of them.

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

Fig. 5-46. IP options.

The *Security* option tells how secret the information is. In theory, a military router might use this field to specify not to route 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 exact route. It is most useful for system managers to send emergency packets when the routing tables are corrupted, or for making timing measurements.

The *Loose source routing* option requires the packet to traverse the list of routers specified, and in the order specified, but it is allowed to pass through other

RED HAT 1057
RED HAT V. CAS

routers on the way. Normally, this option would only provide 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 the routers along the path to append their IP address to the option field. This allows system managers to track down bugs in the routing algorithms ("Why are packets from Houston to Dallas all visiting Tokyo first?") When the ARPANET was first set up, no packet ever passed through more than nine routers, so 40 bytes of option was ample. 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 for debugging routing algorithms.

5.5.2. IP Addresses

Every host and router on the Internet has an IP address, which encodes its network number and host number. The combination is unique: no two machines have the same IP address. All IP addresses are 32 bits long and are used in the *Source address* and *Destination address* fields of IP packets. The formats used for IP address are shown in Fig. 5-47. Those machines connected to multiple networks have a different IP address on each network.

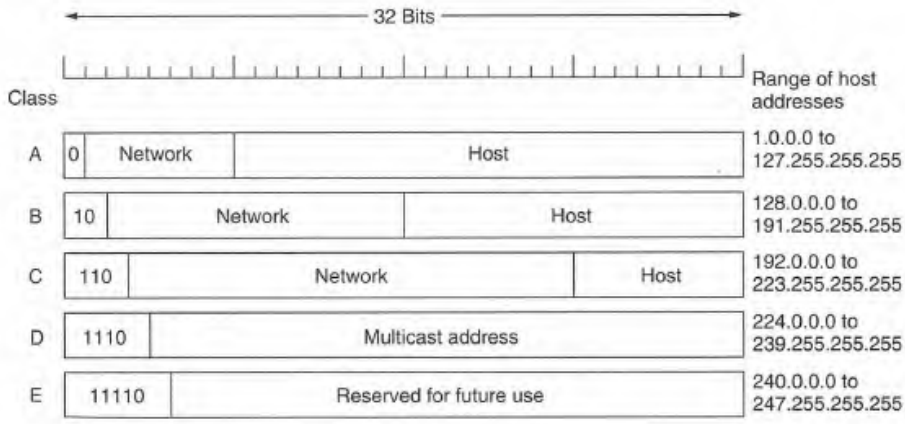


Fig. 5-47. IP address formats.

The class A, B, C, and D formats allow for up to 126 networks with 16 million hosts each, 16,382 networks with up to 64K hosts, 2 million networks, (e.g.,

RED HAT 1057
RED HAT V. CAS

LANs), with up to 254 hosts each, and multicast, in which a datagram is directed to multiple hosts. Addresses beginning with 11110 are reserved for future use. Tens of thousands of networks are now connected to the Internet, and the number doubles every year. Network numbers are assigned by the **NIC** (**Network Information Center**) to avoid conflicts.

Network addresses, which are 32-bit numbers, are usually written in **dotted decimal notation**. In this format, each of the 4 bytes is written in decimal, from 0 to 255. For example, the hexadecimal address C0290614 is written as 192.41.6.20. The lowest IP address is 0.0.0.0 and the highest is 255.255.255.255.

The values 0 and −1 have special meanings, as shown in Fig. 5-48. The value 0 means this network or this host. The value of −1 is used as a broadcast address to mean all hosts on the indicated network.

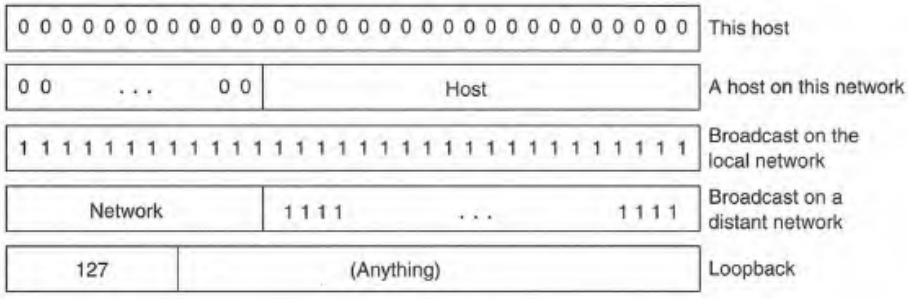


Fig. 5-48. Special IP addresses.

The IP address 0.0.0.0 is used by hosts when they are being booted but is not used afterward. IP addresses with 0 as 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 its class to know how many 0s to include). The address consisting of all 1s 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. 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 local network without the sender knowing its number. This feature is also used for debugging network software.

5.5.3. Subnets

As we have seen, all the hosts in a network must have the same network number. This property of IP addressing can cause problems as networks grow. For example, consider a company that starts out with one class C LAN on the

RED HAT 1057
RED HAT V. CAS

Internet. As time goes on, it might acquire more than 254 machines, and thus need a second class C address. Alternatively, it might acquire a second LAN of a different type and want a separate IP address for it (the LANs could be bridged to form a single IP network, but bridges have their own problems). Eventually, it might end up with many LANs, each with its own router and each with its own class C network number.

As the number of distinct local networks grows, managing them can become a serious headache. Every time a new network is installed the system administrator has to contact NIC to get a new network number. Then this number must be announced worldwide. Furthermore, moving a machine from one LAN to another requires it to change its IP address, which in turn may mean modifying its configuration files and also announcing the new IP address to the world. If some other machine is given the newly-released IP address, that machine will get email and other data intended for the original machine until the address has propagated all over the world.

The solution to these problems is to allow a network to be split into several parts for internal use but still act like a single network to the outside world. In the Internet literature, these parts are called **subnets**. As we mentioned in Chap. 1, this usage conflicts with "subnet" to mean the set of all routers and communication lines in a network. Hopefully it will be clear from the context which meaning is intended. In this section, the new definition will be the one used. If our growing company started up with a class B address instead of a class C address, it could start out just numbering the hosts from 1 to 254. When the second LAN arrived, it could decide, for example, to split the 16-bit host number into a 6-bit subnet number and a 10-bit host number, as shown in Fig. 5-49. This split allows 62 LANs (0 and −1 are reserved), each with up to 1022 hosts.

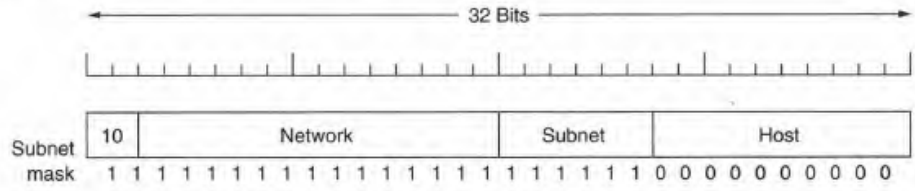


Fig. 5-49. One of the ways to subnet a class B network.

Outside the network, the subnetting is not visible, so allocating a new subnet does not require contacting NIC or changing any external databases. In this example, the first subnet might use IP addresses starting at 130.50.4.1, the second subnet might start at 130.50.8.1, and so on.

To see how subnets work, it is necessary to explain how IP packets are processed at a router. Each router has a table listing some number of (network, 0) IP addresses and some number of (this-network, host) IP addresses. The first kind

RED HAT 1057
RED HAT V. CAS

tells how to get to distant networks. The second kind tells how to get to local hosts. Associated with each table is the network interface to use to reach the destination, and certain other information.

When an IP packet arrives, its destination address is looked up in the routing table. If the packet is for a distant network, it is forwarded to the next router on the interface given in the table. If it is a local host (e.g., on the router's LAN), it is sent directly to the destination. If the network is not present, the packet is forwarded to a default router with more extensive tables. This algorithm means that each router only has to keep track of other networks and local hosts, not (network, host) pairs, greatly reducing the size of the routing table.

When subnetting is introduced, the routing tables are changed, adding entries of the form (this-network, subnet, 0) and (this-network, this-subnet, host). Thus a router on subnet *k* knows how to get to all the other subnets and also how to get to all the hosts on subnet *k*. It does not have to know the details about hosts on other subnets. In fact, all that needs to be changed is to have each router do a Boolean AND with the network's **subnet mask** (see Fig. 5-49) to get rid of the host number and look up the resulting address in its tables (after determining which network class it is). For example, a packet addressed to 130.50.15.6 and arriving at a router on subnet 5 is ANDed with the subnet mask of Fig. 5-49 to give the address 130.50.12.0. This address is looked up in the routing tables to find out how to get to hosts on subnet 3. The router on subnet 5 is thus spared the work of keeping track of the data link addresses of hosts other than those on subnet 5. Subnetting thus reduces router table space by creating a three-level hierarchy.

5.5.4. Internet Control Protocols

In addition to IP, which is used for data transfer, the Internet has several control protocols used in the network layer, including ICMP, ARP, RARP, and BOOTP. In this section we will look at each of these in turn.

The Internet Control Message Protocol

The operation of the Internet is monitored closely by the routers. When something unexpected occurs, the event is reported by the **ICMP** (**Internet Control Message Protocol**), which is also used to test the Internet. About a dozen types of ICMP messages are defined. The most important ones are listed in Fig. 5-50. Each ICMP message type is encapsulated in an IP packet.

The DESTINATION UNREACHABLE message is used when the subnet or a router cannot locate the destination, or a packet with the *DF* bit cannot be delivered because a "small-packet" network stands in the way.

The TIME EXCEEDED message is sent when a packet is dropped due to its counter reaching zero. This event is a symptom that packets are looping, that there is enormous congestion, or that the timer values are being set too low.

RED HAT 1057
RED HAT V. CAS

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 request	Ask a machine if it is alive
Echo reply	Yes, I am alive
Timestamp request	Same as Echo request, but with timestamp
Timestamp reply	Same as Echo reply, but with timestamp

Fig. 5-50. The principal ICMP message types.

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 formerly 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 any more because when congestion occurs, these packets tend to add more fuel to the fire. Congestion control in the Internet is now done largely in the transport layer and will be studied in detail in Chap. 6.

The REDIRECT message is used when a router notices that a packet seems to be routed wrong. It is used by the router to tell the sending host about the probable error.

The ECHO REQUEST and ECHO REPLY messages are used to see if a given destination is reachable and alive. Upon receiving the ECHO message, the destination is expected to send an ECHO REPLY message back. 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 is used to measure network performance.

In addition to these messages, there are four others that deal with Internet addressing, to allow hosts to discover their network numbers and to handle the case of multiple LANs sharing a single IP address. ICMP is defined in RFC 792.

The Address Resolution Protocol

Although every machine on the Internet has one (or more) IP addresses, these cannot actually be used for sending packets because the data link layer hardware does not understand Internet addresses. Nowadays, most hosts are attached to a

RED HAT 1057
RED HAT V. CAS

LAN by an interface board that only understands LAN addresses. For example, every Ethernet board ever manufactured comes equipped with a 48-bit Ethernet address. Manufacturers of Ethernet boards request a block of addresses from a central authority to ensure that no two boards have the same address (to avoid conflicts should the two boards ever appear on the same LAN). The boards 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. 5-51, in which a small university with several class C networks is illustrated. Here we have two Ethernets, one in the Computer Science department, with IP address 192.31.65.0 and one in Electrical Engineering, with IP address 192.31.63.0. These are connected by a campus FDDI ring with IP address 192.31.60.0. Each machine on an Ethernet has a unique Ethernet address, labeled *E1* through *E6*, and each machine on the FDDI ring has an FDDI address, labeled *F1* through *F3*.

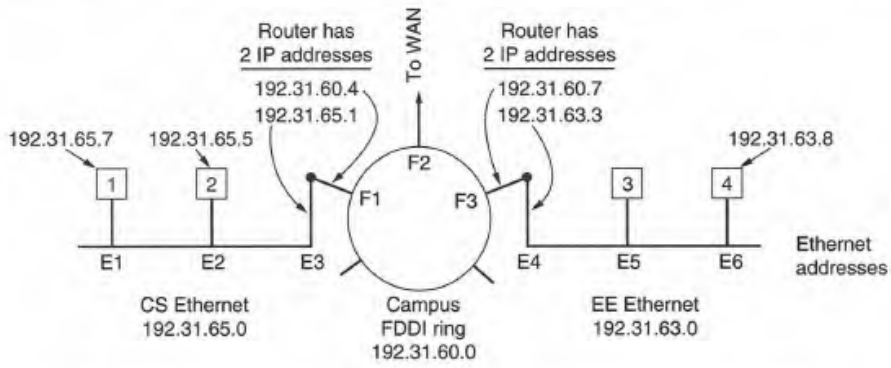


Fig. 5-51. Three interconnected class C networks: two Ethernets and an FDDI ring.

Let us start out by seeing how a user on host 1 sends a packet to a user on host 2. Let us assume the sender knows the name of the intended receiver, possibly something like *mary@eagle.cs.uni.edu*. The first step is to find the IP address for host 2, known as *eagle.cs.uni.edu*. This lookup is performed by the Domain Name System, which we will study in Chap. 7. For the moment, we will just assume that DNS returns the IP address for host 2 (192.31.65.5).

The upper layer software on host 1 now builds a packet with 192.31.65.5 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 its own network, but it needs a way to find the destination's Ethernet address. One solution is to have a configuration file somewhere in the system that maps IP addresses onto

RED HAT 1057
RED HAT V. CAS

Ethernet addresses. This solution is certainly possible, but for organizations with thousands of machines, keeping 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 192.31.65.5?" The broadcast will arrive at every machine on Ethernet 192.31.65.0, 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 192.31.65.5 is on the host with Ethernet address *E2*. The protocol for asking this question and getting the reply is called **ARP** (**Address Resolution Protocol**). Almost every machine on the Internet runs it. It 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 192.31.65.5) in the payload field, and dumps it onto the Ethernet. The Ethernet board 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 more efficient. 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 ARP broadcast arrives at host 2, the pair (192.31.65.7, E1) is entered into host 2's ARP cache for future use. In fact, all machines on the Ethernet can enter this mapping into their ARP caches.

Yet another optimization is to have every machine broadcast its mapping when it boots. 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 any entry in everyone's ARP cache. If a response does arrive, two machines have been assigned the same IP address. The new one should inform the system manager and not boot.

To allow mappings to change, for example, when an Ethernet board breaks and is replaced with a new one (and thus a new Ethernet address), entries in the ARP cache should time out after a few minutes.

Now let us look at Fig. 5-51 again, only this time host 1 wants to send a packet to host 4 (192.31.63.8). Using ARP will fail because host 4 will not see the broadcast (routers do not forward Ethernet-level broadcasts). There are two solutions. First, the CS router could be configured to respond to ARP requests for network 192.31.63.0 (and possibly other local networks). In this case, host 1 will make an ARP cache entry of (192.31.63.8, E3) and happily send all traffic for host

RED HAT 1057
RED HAT V. CAS

4 to the local router. This solution is called **proxy ARP**. The second solution is to have host 1 immediately see that the destination is on a remote network and just send all such traffic to a default Ethernet address that handles all remote traffic, in this case *E3*. This solution does not require having the CS router know which remote networks it is serving.

Either way, what happens is that host 1 packs the IP packet into the payload field of an Ethernet frame addressed to *E3*. When the CS router gets the Ethernet frame, it removes the IP packet from the payload field and looks up the IP address in its routing tables. It discovers that packets for network 192.31.63.0 are supposed to go to router 192.31.60.7. If it does not already know the FDDI address of 192.31.60.7, it broadcasts an ARP packet onto the ring and learns that its ring address is *F3*. It then inserts the packet into the payload field of an FDDI frame addressed to *F3* and puts it on the ring.

At the EE router, the FDDI driver removes the packet from the payload field and gives it to the IP software, which sees that it needs to send the packet to 192.31.63.8. If this IP address is not in its ARP cache, it broadcasts an ARP request on the EE Ethernet and learns that the destination address is *E6* so it builds an Ethernet frame addressed to *E6*, puts the packet in the payload field, and sends it over the Ethernet. When the Ethernet frame arrives at host 4, the packet is extracted from the frame and passed to the IP software for processing.

Going from host 1 to a distant network over a WAN works essentially the same way, except that this time the CS router's tables tell it to use the WAN router whose FDDI address is *F2*.

The Reverse Address Resolution Protocol

ARP solves the problem of finding out which Ethernet address corresponds to a given IP address. Sometimes the reverse problem has to solved: Given an Ethernet address, what is the corresponding IP address? In particular, this problem occurs when booting a diskless workstation. Such a machine will normally get the binary image of its operating system from a remote file server. But how does it learn its IP address?

The solution is to use the **RARP (Reverse Address Resolution Protocol)** (defined in RFC 903). This protocol allows a newly-booted workstation to broadcast its Ethernet address and say: "My 48-bit Ethernet address is 14.04.05.18.01.25. Does anyone out there know my IP address?" The RARP server sees this request, looks up the Ethernet address in its configuration files, and sends back the corresponding IP address.

Using RARP is better than embedding an IP address in the memory image because it allows the same image to be used on all machines. If the IP address were buried inside the image, each workstation would need its own image.

A disadvantage of RARP is that it uses a destination address of all 1s (limited broadcasting) to reach the RARP server. However, such broadcasts are not

RED HAT 1057
RED HAT V. CAS

forwarded by routers, so a RARP server is needed on each network. To get around this problem, an alternative bootstrap protocol called **BOOTP** has been invented (see RFCs 951, 1048, and 1084). Unlike RARP, it uses UDP messages, which are forwarded over routers. It also provides a diskless workstation with additional information, including the IP address of the file server holding the memory image, the IP address of the default router, and the subnet mask to use. BOOTP is described in RFC 951.

5.5.5. The Interior Gateway Routing Protocol: OSPF

As we mentioned earlier, the Internet is made up of a large number of autonomous systems. Each AS is operated by a different organization and can use its own routing algorithm inside. For example, the internal networks of companies *X*, *Y*, and *Z* would usually be seen as three ASes if all three were on the Internet. All three may use different routing algorithms internally. Nevertheless, having standards, even for internal routing, simplifies the implementation at the boundaries between ASes and allows reuse of code. In this section we will study routing within an AS. In the next one, we will look at routing between ASes. A routing algorithm within an AS is called an **interior gateway protocol**; an algorithm for routing between ASes is called an **exterior gateway protocol**.

The original Internet interior gateway protocol was a distance vector protocol (RIP) based on the Bellman-Ford algorithm. It worked well in small systems, but less well as ASes got larger. It also suffered from the count-to-infinity problem and generally slow convergence, so it was replaced in May 1979 by a link state protocol. In 1988, the Internet Engineering Task Force began work on a successor. That successor, called **OSPF** (**Open Shortest Path First**) became a standard in 1990. Many router vendors are now supporting it, and it will become the main interior gateway protocol in the near future. Below we will give a sketch of how OSPF works. For the complete story, see RFC 1247.

Given the long experience with other routing protocols, the group designing the new protocol 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. The IP protocol has a *Type of Service* field, but no existing routing protocol used it.

Fifth, and related to the above, the new protocol had to do load balancing, splitting the load over multiple lines. Most previous protocols sent all packets

RED HAT 1057
RED HAT V. CAS

over the best route. The second-best route was not used at all. In many cases, splitting the load over multiple lines gives better performance.

Sixth, support for hierarchical systems was needed. By 1988, the Internet had grown so large that no router could be expected to know the entire topology. The new routing protocol 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 three kinds of connections and networks:

1. Point-to-point lines between exactly two routers.
2. Multiaccess networks with broadcasting (e.g., most LANs).
3. Multiaccess networks without broadcasting (e.g., most packet-switched WANs).

A **multiaccess** network is one that can have multiple routers on it, each of which can directly communicate with all the others. All LANs and WANs have this property. Figure 5-52(a) shows an AS containing all three kinds of networks. Note that hosts do not generally play a role in OSPF.

OSPF works by abstracting the collection of actual networks, routers, and lines into a directed graph in which each arc is assigned a cost (distance, delay, etc.). It then computes the shortest path based on the weights on the arcs. A serial connection between two routers is represented by a pair of arcs, one in each direction. Their weights may be different. A multiaccess network is represented by a node for the network itself plus a node for each router. The arcs from the network node to the routers have weight 0 and are omitted from the graph.

Figure 5-52(b) shows the graph representation of the network of Fig. 5-52(a). What OSPF fundamentally does is represent the actual network as a graph like this and then compute the shortest path from every router to every other router.

Many of the ASes in the Internet are themselves large and nontrivial to manage. OSPF allows them to be divided up 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. An area is a generalization of a subnet. Outside an area, its topology and details are not visible.

Every AS has a **backbone** area, called area 0. 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 an arc and has a cost. Each router that is connected to two or more areas is part of the backbone. As with other areas, the topology of the backbone is not visible outside the backbone.

Within an area, each router 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

RED HAT 1057
RED HAT V. CAS

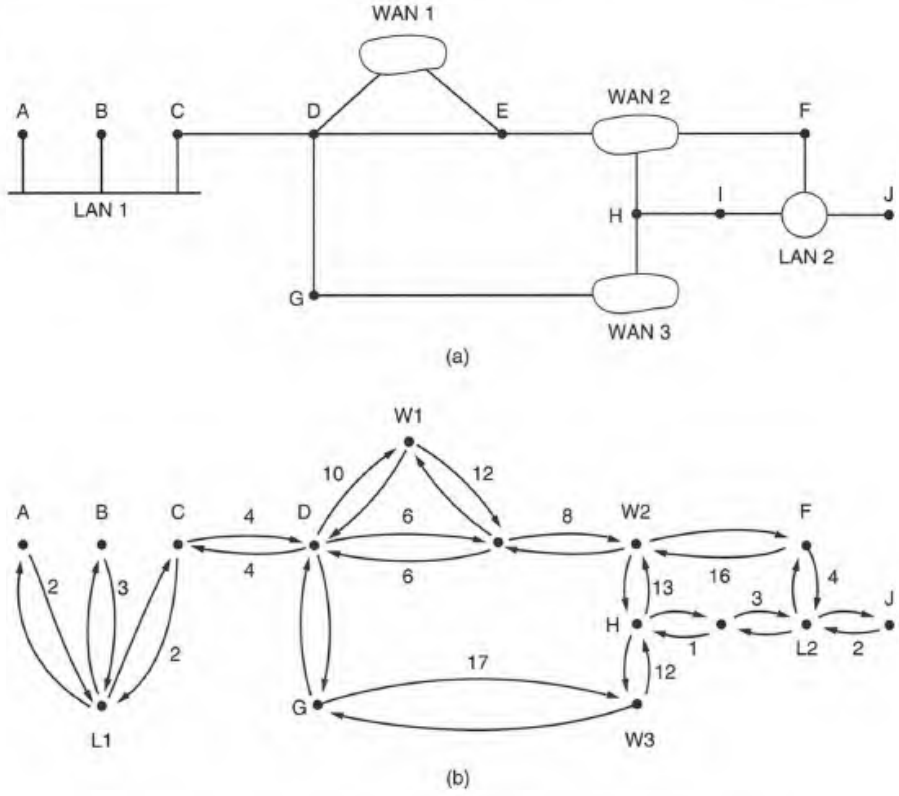


Fig. 5-52. (a) An autonomous system. (b) A graph representation of (a).

every other router in the area, including the router that is connected to the backbone, of which there must be at least one. A router that connects to two areas needs the databases for both areas and must run the shortest path algorithm for each one separately.

The way OSPF handles type of service routing is to have multiple graphs, one labeled with the costs when delay is the metric, one labeled with the costs when throughput is the metric, and one labeled with the costs when reliability is the metric. Although this triples the computation needed, it allows separate routes for optimizing delay, throughput, and reliability.

During normal operation, three kinds of routes may be needed: intra-area, interarea, and interAS. Intra-area routes are the easiest, since the source router already knows the shortest path to the destination router. Interarea routing always proceeds in three steps: go from the source to the backbone; go across the backbone to the destination area; go to the destination. This algorithm forces a star

RED HAT 1057
RED HAT V. CAS

configuration on OSPF with the backbone being the hub and the other areas being spokes. 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. Figure 5-53 shows part of the Internet with ASes and areas.

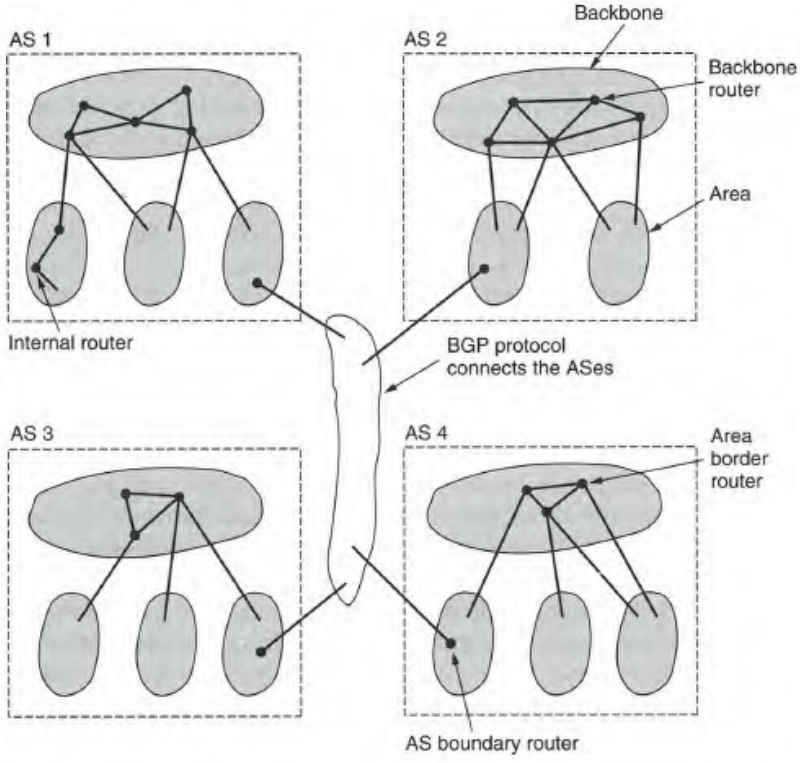


Fig. 5-53. The relation between ASes, backbones, and areas in OSPF.

OSPF distinguishes four classes of routers:

1. Internal routers are wholly within one area.
2. Area border routers connect two or more areas.
3. Backbone routers are on the backbone.
4. AS boundary routers talk to routers in other ASes.

These classes are allowed to overlap. For example, all the border routers are automatically part of the backbone. In addition, a router that is in the backbone

RED HAT 1057
RED HAT V. CAS

but not part of any other area is also an internal router. Examples of all four classes of routers are illustrated in Fig. 5-53.

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. On WANs, it needs some configuration information to know who to contact. From the responses, each router learns who its neighbors are.

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, and exchanges information with them. 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.

During normal operation, each router periodically floods LINK STATE UPDATE messages to each of its adjacent routers. This message gives its state and provides 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 line 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 line is brought up.

Either partner can request link state information from the other one using LINK STATE REQUEST messages. The net 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 as raw IP packets. The five kinds of messages are summarized in Fig. 5-54.

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

Fig. 5-54. 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 neighbors and costs. This

RED HAT 1057
RED HAT V. CAS

information allows each router to construct the graph for its area(s) and compute the shortest path. The backbone area does this 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, a router about to send an interarea packet can select the best exit router to the backbone.

5.5.6. The Exterior Gateway Routing Protocol: BGP

Within a single AS, the recommended routing protocol on the Internet is OSPF (although it is certainly not the only one in use). Between ASes, a different protocol, **BGP (Border Gateway Protocol)**, is used. A different protocol is needed between ASes because the goals of an interior gateway protocol and an exterior gateway protocol are not the same. All an interior gateway 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.

Exterior gateway protocol routers have to worry about politics a great deal. 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 was 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 a carrier 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 routing constraints are

1. No transit traffic through certain ASes.
2. Never put Iraq on a route starting at the Pentagon.
3. Do not use the United States to get from British Columbia to Ontario.
4. Only transit Albania if there is no alternative to the destination.
5. Traffic starting or ending at IBM® should not transit Microsoft®.

Policies are manually configured into each BGP router. They are not part of the protocol itself.

From the point of view of a BGP router, the world consists of other BGP routers and the lines connecting them. Two BGP routers are considered connected if they share a common network. Given BGP's special interest in transit

RED HAT 1057
RED HAT V. CAS

traffic, networks are grouped into one of three categories. The first category is the **stub networks**, which have only one connection to the BGP graph. These cannot be used for transit traffic because there is no one on the other side. Then come the **multiconnected networks**. These could be used for transit traffic, except that they refuse. Finally, there are the **transit networks**, such as backbones, which are willing to handle third-party packets, possibly with some restrictions.

Pairs of BGP routers communicate with each other by establishing TCP connections. Operating this way provides reliable communication and hides all the details of the network being passed through.

BGP is fundamentally a distance vector protocol, but quite different from most others such as RIP. Instead of maintaining just the cost to each destination, each BGP router keeps track of the exact path used. Similarly, instead of periodically giving each neighbor its estimated cost to each possible destination, each BGP router tells its neighbors the exact path it is using.

As an example, consider the BGP routers shown in Fig. 5-55(a). In particular, consider *F*'s routing table. Suppose that it uses the path *FGCD* to get to *D*. When the neighbors give it routing information, they provide their complete paths, as shown in Fig. 5-55(b) (for simplicity, only destination *D* is shown here).

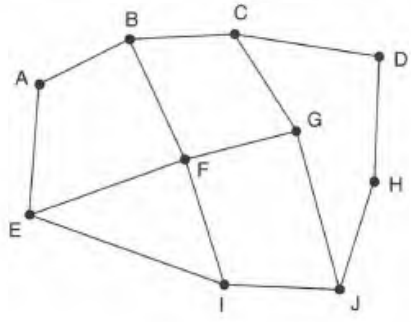


Fig. 5-55. (a) A set of BGP routers. (b) Information sent to *F*.

After all the paths come in from the neighbors, *F* examines them to see which is the best. It quickly discards the paths from *I* and *E*, since these paths pass through *F* itself. The choice is then between using *B* and *G*. Every BGP router contains a module that examines routes to a given destination and scores them, returning a number for the "distance" to that destination for each route. Any route violating a policy constraint automatically gets a score of infinity. The router then adopts the route with the shortest distance. The scoring function is not part of the BGP protocol and can be any function the system managers want.

BGP easily solves the count-to-infinity problem that plagues other distance vector routing algorithms. For example, suppose *G* crashes or the line *FG* goes

RED HAT 1057
RED HAT V. CAS

down. *F* then receives routes from its three remaining neighbors. These routes are *BCD*, *IFGCD*, and *EFGCD*. It can immediately see that the two latter routes are pointless, since they pass through *F* itself, so it chooses *FBCD* as its new route. Other distance vector algorithms often make the wrong choice because they cannot tell which of their neighbors have independent routes to the destination, and which do not. The current definition of BGP is in RFC 1654. Additional useful information can be found in RFC 1268.

5.5.7. 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 updating replicated, distributed databases, transmitting stock quotes to multiple brokers, and handling digital conference (i.e., multiparty) telephone calls.

IP supports multicasting, using class D 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-efforts 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.

Two kinds of group addresses are supported: permanent addresses and temporary ones. A permanent group is always there and does not have to be set up. Each permanent group has a permanent group address. Some examples of permanent group 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
224.0.0.6 All designated OSPF routers on a LAN

Temporary groups must be created before they can be used. A process can ask its host to join a specific group. It can also ask its host to leave the group. When the last process on a host leaves a group, that group is no longer present on the host. Each host keeps track of which groups its processes currently belong to.

Multicasting is implemented by special multicast routers, which may or may not be colocated with the standard routers. About once a minute, each multicast router sends a hardware (i.e., data link layer) multicast to the hosts on its LAN (address 224.0.0.1) asking them to report back on the groups their processes currently belong to. 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**), which is vaguely analogous to ICMP. It has only two kinds of packets: query and response, each with a simple fixed format

RED HAT 1057
RED HAT V. CAS

containing some control information in the first word of the payload field and a class D address in the second word. It is described in RFC 1112.

Multicast routing is done using spanning trees. Each multicast router exchanges information with its neighbors using a modified distance vector protocol in order for each one to construct a spanning tree per group covering all group members. Various optimizations are used to prune the tree to eliminate routers and networks not interested in particular groups. The protocol makes heavy use of tunneling to avoid bothering nodes not in a spanning tree.

5.5.8. Mobile IP

Many users of the Internet have portable computers and want to stay connected to the Internet when they visit a distant Internet site and even on the road in between. Unfortunately, the IP addressing system makes working far from home easier said than done. In this section we will examine the problem and the solution. A more detailed description is given in (Johnson, 1995).

The real villain is the addressing scheme itself. Every IP address contains three fields: the class, the network number, and the host number. For example, consider the machine with IP address 160.80.40.20. The 160.80 gives the class (B) and network number (8272); the 40.20 is the host number (10260). Routers all over the world have routing tables telling which line 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). The owner will no longer get email, and so on. Giving the machine a new IP address corresponding to its new location is unattractive because large numbers of people, programs, and databases would have to be informed of the change.

Another approach is to have the routers use complete IP addresses for routing, instead of just the class and network. However, this strategy would require each router to have millions of table entries, at astronomical cost to the Internet.

When people began demanding the ability to have mobile hosts, the 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. 5.2.8. To review it briefly, every site that wants to allow its users to roam has to create a home agent. Every

RED HAT 1057
RED HAT V. CAS

site that wants to allow visitors has to create a foreign agent. When a mobile host shows up at a foreign site, it contacts the foreign host there and registers. The foreign host then contacts the user's home agent and gives it a **care-of address**, normally the foreign agent's own IP address.

When a packet arrives at the user's home LAN, it comes in at some router attached to the LAN. The router then tries to locate the host in the usual way, by broadcasting an ARP packet asking, for example: "What is the Ethernet address of 160.80.40.20?" The home agent responds to this query by giving its own Ethernet address. The router then sends packets for 160.80.40.20 to the home agent. It, in turn, tunnels them to the care-of address by encapsulating them in the payload field of an IP packet addressed to the foreign agent. The foreign agent then decapsulates and delivers them to the data link address of the mobile host. In addition, the home agent gives the care-of address to the sender, so future packets can be tunnelled directly to the foreign agent. This solution meets all the requirements stated above.

One small detail is probably worth mentioning. At the time the mobile host moves, the router probably has its (soon-to-be-invalid) Ethernet address cached. To replace that Ethernet address with the home agent's, a trick called **gratuitous ARP** is used. This is a special, unsolicited message to the router that causes it to replace a specific cache entry, in this case, that of the mobile host about to leave. When the mobile host returns later, the same trick is used to update the router's cache again.

Nothing in the design prevents a mobile host from being its own foreign agent, but that approach only works if the mobile host (in its capacity as foreign agent) is logically connected to the Internet at its current site. Also, it must be able to acquire a (temporary) care-of IP address to use. That IP address must belong to the LAN to which it is currently attached.

The IETF solution for mobile hosts solves a number of other problems not mentioned so far. For example, how are agents located? The solution is for each agent to periodically broadcast its address and the type of services it is willing to provide (e.g., home, foreign, or both). When a mobile host arrives somewhere, it can just listen for these broadcasts, called **advertisements**. Alternatively, it can broadcast a packet announcing its arrival and hope that the local foreign agent responds to it.

Another problem that had to be solved is what to do about impolite mobile hosts that leave without saying goodbye. The solution is to make registration valid only for a fixed time interval. If it is not refreshed periodically, it times out, so the foreign host can clear its tables.

Yet another issue is security. When a home agent gets a message asking it to please forward all of Nora's packets to some IP address, it had better not comply unless it is convinced that Nora 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. 7.

RED HAT 1057
RED HAT V. CAS

A final point addressed by the Working Group relates to levels of mobility. Imagine an airplane with an on-board Ethernet used by the navigation and avionics computers. On this Ethernet is a standard router that talks to the wired Internet on the ground over a radio link. One fine day, some clever marketing executive gets the idea to install Ethernet connectors in all the arm rests so passengers with mobile computers can also plug in.

Now we have two levels of mobility: the aircraft's own computers, which are stationary with respect to the Ethernet, and the passengers' computers, which are mobile with respect to it. In addition, the on-board router is mobile with respect to routers on the ground. Being mobile with respect to a system that is itself mobile can be handled using recursive tunneling.

5.5.9. CIDR—Classless InterDomain Routing

IP has been in heavy use for over a decade. It has worked extremely well, as demonstrated by the exponential growth of the Internet. Unfortunately, IP is rapidly becoming a victim of its own popularity: it is running out of addresses. This looming disaster has 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. A more complete description is given in (Huitema, 1996).

Back in 1987, a few visionaries predicted that some day the Internet might grow to 100,000 networks. Most experts pooh-poohed this as being decades in the future, if ever. The 100,000th network was connected in 1996. The problem, simply stated, is that the Internet is rapidly running out of IP addresses. In principle, over 2 billion addresses exist, but the practice of organizing the address space by classes (see Fig. 5-47), 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*).

In reality, 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 eight for the host number, allowing 1022 hosts per network. Had this been the case, most organizations would have probably settled for a class C network, and there would have been half a million of them (versus only 16,384 class B networks).

However, then another problem would have emerged more quickly: the routing table explosion. From the point of view of the routers, the IP address space is

RED HAT 1057
RED HAT V. CAS

a two-level hierarchy, with network numbers and host numbers. Routers do not have to know about all the hosts, but they do have to know about all the networks. If half a million class C networks were in use, every router in the entire Internet would need a table with half a million entries, one per network, telling which line to use to get to that network, as well as other information.

The actual physical storage of half a million entry tables is probably doable, although expensive for critical routers that keep the tables in static RAM on I/O boards. A more serious problem is that the complexity of various algorithms relating to management of the tables grows faster than linear. Worse yet, much of the existing router software and firmware was designed at a time when the Internet had 1000 connected networks and 10,000 networks seemed decades away. Design choices made then often are far from optimal now.

In addition, various routing algorithms require each router to transmit its tables periodically. The larger the tables, the more likely some parts will get lost underway, leading to incomplete data at the other end and possibly routing instabilities.

The routing table problem could have been solved by going to a deeper hierarchy. For example, having each IP address contain a country, state, 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 (Liechtenstein would have as many bits as the United States).

In short, most solutions solve one problem but create a new one. One solution that is now being implemented and which will give the Internet a bit of extra breathing room is **CIDR** (**Classless InterDomain Routing**). The basic idea behind CIDR, which is described in RFC 1519, is to allocate the remaining class C networks, of which there are almost two million, in variable-sized blocks. If a site needs, say, 2000 addresses, it is given a block of 2048 addresses (eight contiguous class C networks), and not a full class B address. Similarly, a site needing 8000 addresses gets 8192 addresses (32 contiguous class C networks).

In addition to using blocks of contiguous class C networks as units, the allocation rules for the class C addresses were also changed in RFC 1519. The world was partitioned into four zones, and each one given a portion of the class C address space. The allocation was as follows:

Addresses 194.0.0.0 to 195.255.255.255 are for Europe
Addresses 198.0.0.0 to 199.255.255.255 are for North America
Addresses 200.0.0.0 to 201.255.255.255 are for Central and South America
Addresses 202.0.0.0 to 203.255.255.255 are for Asia and the Pacific

In this way, each region was given about 32 million addresses to allocate, with another 320 million class C addresses from 204.0.0.0 through 223.255.255.255

RED HAT 1057
RED HAT V. CAS

held in reserve for the future. The advantage of this allocation is that now any router outside of Europe that gets a packet addressed to 194.xx.yy.zz or 195.xx.yy.zz can just send it to its standard European gateway. In effect 32 million addresses have now been compressed into one routing table entry. Similarly for the other regions.

Of course, once a 194.xx.yy.zz packet gets to Europe, more detailed routing tables are needed. One possibility is to have 131,072 entries for networks 194.0.0.xx through 195.255.255.xx, but this is precisely this routing table explosion that we are trying to avoid. Instead, each routing table entry is extended by giving it a 32-bit mask. When a packet comes in, its destination address is first extracted. Then (conceptually) the routing table is scanned entry by entry, masking the destination address and comparing it to the table entry looking for a match.

To make this comparison process clearer, let us consider an example. 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. Next, Oxford University asks for 4096 addresses. Since a block of 4096 addresses must lie on a 4096-byte boundary, they cannot be given addresses starting at 194.24.8.0. Instead they get 194.24.16.0 through 194.24.31.255 along with mask 255.255.240.0. Now 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.

The routing tables all over Europe are now updated with three entries, each one containing a base address and a mask. These entries (in binary) are:

Address	Mask
11000010 00011000 00000000 00000000	11111111 11111111 11111000 00000000
11000010 00011000 00010000 00000000	11111111 11111111 11110000 00000000
11000010 00011000 00001000 00000000	11111111 11111111 11111100 00000000

Now consider what happens when a packet comes in addressed to 194.24.17.4, which in binary is

11000010 00011000 00010001 00000100

First it is Boolean ANDed with the Cambridge mask to get

11000010 00011000 00010000 00000000

This value does not match the Cambridge base address, so the original address is next ANDed with the Oxford mask to get

11000010 00011000 00010000 00000000

This value does match the Oxford base, so the packet is sent to the Oxford router. In practice, the router entries are not tried sequentially; indexing tricks are used to speed up the search. Also, it is possible for two entries to match, in which case the one whose mask has the most 1 bits wins. Finally, the same idea can be applied to all addresses, not just the new class C addresses, so with CIDR, the old

RED HAT 1057
RED HAT V. CAS

class A, B, and C networks are no longer used for routing. This is why CIDR is called classless routing. CIDR is described in more detail in (Ford et al., 1993; and Huitema, 1995).

5.5.10. IPv6

While CIDR may buy a few more years' time, everyone realizes that the days of IP in its current form (IPv4) are numbered. In addition to these technical problems, there is another issue looming in the background. Up until recently, the Internet has been used largely by universities, high-tech industry, and the government (especially the Dept. of Defense). With the explosion of interest in the Internet starting in the mid 1990s, it is likely that in the next millenium, it will be used by a much larger group of people, especially people with different requirements. For one thing, millions of people with wireless portables may 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 long before every television set in the world is an Internet node, producing a billion machines being used for 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 which 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 to

1. Support billions of hosts, even with inefficient address space 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) than current IP.
5. Pay more attention to 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.

To find a protocol that met all these requirements, IETF issued a call for proposals and discussion in RFC 1550. Twenty-one responses were received, not all of them full proposals. 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 with a completely different protocol.

RED HAT 1057
RED HAT V. CAS

One proposal was to run TCP over CLNP, which, with its 160-bit addresses would have provided enough address space forever and would 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** (IPv5 was already in use for an experimental real-time stream protocol).

IPv6 meets the 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 all the other Internet protocols, including TCP, UDP, ICMP, IGMP, OSPF, BGP, and DNS, sometimes with small modifications being required (mostly to deal with longer addresses). The main features of IPv6 are discussed below. More information about it can be found in RFC 1883 through RFC 1887.

First and foremost, IPv6 has longer addresses than IPv4. They are 16 bytes long, which solves the problem that IPv6 was set out to solve: provide 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 7 fields (versus 13 in IPv4). This change allows routers to process packets faster and thus improve throughout. We will discuss the header shortly, too.

The third major improvement was better support for options. This change was essential with the new header because fields that previously were required are now optional. 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.

Finally, more attention has been paid to type of service than in the past. IPv4 actually has an 8-bit field devoted to this matter, but with the expected growth in multimedia traffic in the future, much more is needed.

RED HAT 1057
RED HAT V. CAS

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 will probably take 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, so many implementations are likely to try to avoid it by using some field in the data link header to distinguish IPv4 packets from IPv6 packets. In this way, packets can be passed to the correct network layer handler directly. However, having the data link layer be aware of network packet types completely violates the design principle that each layer should not be aware of the meaning of the bits given to it from the layer above. The discussions between the "Do it right" and "Make it fast" camps will no doubt be lengthy and vigorous.

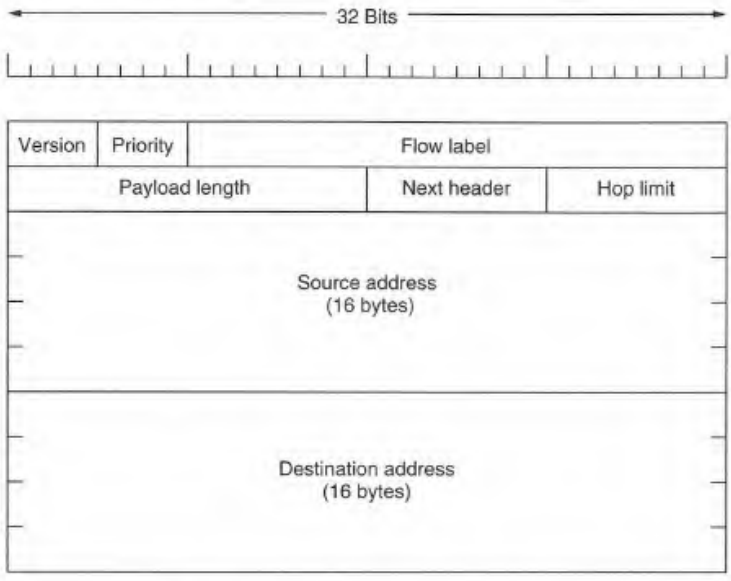


Fig. 5-56. The IPv6 fixed header (required).

The *Priority* field is used to distinguish between packets whose sources can be flow controlled and those that cannot. Values 0 through 7 are for transmissions that are capable of slowing down in the event of congestion. Values 8 through 15 are for real-time traffic whose sending rate is constant, even if all the packets are being lost. Audio and video fall into the latter category. This distinction allows routers to deal with packets better in the event of congestion. Within each group, lower-numbered packets are less important than higher-numbered ones. The IPv6 standard suggests, for example, to use 1 for news, 4 for FTP, and 6 for Telnet

RED HAT 1057
RED HAT V. CAS

connections, since delaying a news packet for a few seconds is not noticeable, but delaying a Telnet packet certainly is.

The *Flow label* field is still experimental but will be used to allow a source and destination to set up a pseudoconnection with particular properties and requirements. For example, a stream of packets from one process on a certain source host to a certain process on a certain 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 subnet and the guarantees of a virtual circuit subnet.

Each flow is designated by the source address, destination address, and flow number, so many flows may be active at the same time between a given pair of IP addresses. Also, in this way, even if two flows coming from different hosts but with the same flow number 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 numbers will be chosen randomly, rather than assigned sequentially starting at 1, to make it easy for routers 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.

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, follows 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. Another faction wanted variable-sized addresses. After much discussion, it was decided that fixed-length 16-byte addresses were the best compromise.

The IPv6 address space is divided up as shown in Fig. 5-57. Addresses beginning with 80 zeros are reserved for IPv4 addresses. Two variants are supported, distinguished by the next 16 bits. These variants relate to how IPv6 packets will be tunneled over the existing IPv4 infrastructure.

RED HAT 1057
RED HAT V. CAS

Prefix (binary)	Usage	Fraction
0000 0000	Reserved (including IPv4)	1/256
0000 0001	Unassigned	1/256
0000 001	OSI NSAP addresses	1/128
0000 010	Novell NetWare IPX addresses	1/128
0000 011	Unassigned	1/128
0000 1	Unassigned	1/32
0001	Unassigned	1/16
001	Unassigned	1/8
010	Provider-based addresses	1/8
011	Unassigned	1/8
100	Geographic-based addresses	1/8
101	Unassigned	1/8
110	Unassigned	1/8
1110	Unassigned	1/16
1111 0	Unassigned	1/32
1111 10	Unassigned	1/64
1111 110	Unassigned	1/128
1111 1110 0	Unassigned	1/512
1111 1110 10	Link local use addresses	1/1024
1111 1110 11	Site local use addresses	1/1024
1111 1111	Multicast	1/256

Fig. 5-57. IPv6 addresses

The use of separate prefixes for provider-based and geographic-based addresses is a compromise between two different visions of the future of the Internet. Provider-based addresses make sense if you think that in the future there will be some number of companies providing Internet service to customers, analogous to AT&T, MCI, Sprint, British Telecom, and so on providing telephone service now. Each of these companies will be given some fraction of the address space. The first 5 bits following the 010 prefix are used to indicate which registry to look the provider up in. Currently three registries are operating, for North America, Europe, and Asia. Up to 29 new registries can be added later.

Each registry is free to divide up the remaining 15 bytes as it sees fit. It is expected that many of them will use a 3-byte provider number, giving about 16

RED HAT 1057
RED HAT V. CAS

million providers, in order to allow large companies to act as their own provider. Another possibility is to use 1 byte to indicate national providers and let them do further allocation. In this manner, additional levels of hierarchy can be introduced as needed.

The geographic model is the same as the current Internet, in which providers do not play a large role. In this way, IPv6 can handle both kinds of addresses.

The link and site local addresses have only a local significance. They can be reused at each organization without conflict. They cannot be propagated outside organizational boundaries, making them well suited to organizations that currently use firewalls to wall themselves off from the rest of the Internet.

Multicast addresses have a 4-bit flag field and a 4-bit scope field following the prefix, then a 112-bit group identifier. One of the flag bits distinguishes permanent from transient groups. The scope field allows a multicast to be limited to the current link, site, organization, or planet. These four scopes are spread out over the 16 values to allow new scopes to be added later. For example, the planetary scope is 14, so code 15 is available to allow future expansion of the Internet to other planets, solar systems, and galaxies.

In addition to supporting the standard unicast (point-to-point) and multicast addresses, IPv6 also supports a new kind of addressing: anycast. **Anycasting** is like multicasting in that the destination is a group of addresses, but instead of trying to deliver the packet to all of them, it tries to deliver it to just one, usually the nearest one. For example, when contacting a group of cooperating file servers, a client can use anycast to reach the nearest one, without having to know which one that is. Anycasting uses regular unicast addresses. It is up to the routing system to choose the lucky host that gets the packet.

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 zeros 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

RED HAT 1057
RED HAT V. CAS

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 1715, 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 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 also worth noting that only 28 percent of the address space has been allocated so far. The other 72 percent is available for future purposes not yet thought of.

It is instructive to compare the IPv4 header (Fig. 5-45) 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 and routers must support packets of 576 bytes. This rule makes fragmentation less likely to occur in the first place. In addition, when a host sends an IPv6 packet that is too large, instead of fragmenting it, the router that is unable to forward it sends back an error message. 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.

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 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—has been met by this design.

Extension Headers

Nevertheless, some of the missing fields are occasionally still needed so IPv6 has introduced the concept of an (optional) **extension header**. These headers can be supplied to provide extra information, but encoded in an efficient way. Six

RED HAT 1057
RED HAT V. CAS

kinds of extension headers are defined at present, as listed in Fig. 5-58. 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
Routing	Full or partial route to follow
Fragmentation	Management of datagram fragments
Authentication	Verification of the sender's identity
Encrypted security payload	Information about the encrypted contents
Destination options	Additional information for the destination

Fig. 5-58. IPv6 extension headers.

Some of the headers have a fixed format; others contain a variable number of variable-length fields. 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 the same as the previous one, except 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 64K. The format of this header is shown in Fig. 5-59.

Next header	0	194	0
Jumbo payload length			

Fig. 5-59. The hop-by-hop extension header for large datagrams (jumbograms).

As with all extension headers, this one starts out 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. The next 2 bytes indicate that this option defines the datagram size (code 194) as a 4-byte number. The last 4 bytes give the size of the datagram. Sizes less than 65,536 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

RED HAT 1057
RED HAT V. CAS

jumbograms. The use of jumbograms is important for supercomputer applications that must transfer gigabytes of data efficiently across the Internet.

The routing header lists one or more routers that must be visited on the way to the destination. Both strict routing (the full path is supplied) and loose routing (only selected routers are supplied) are available, but they are combined. The format of the routing header is shown in Fig. 5-60.

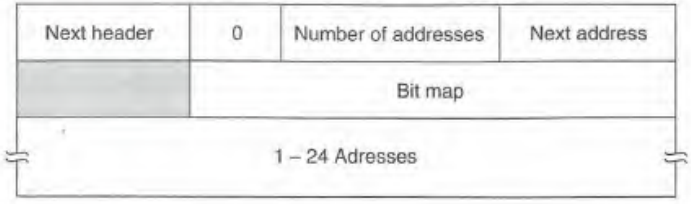


Fig. 5-60. The extension header for routing.

The first 4 bytes of the routing extension header contain four 1-byte integers: the next header type, the routing type (currently 0), the number of addresses present in this header (1 to 24), and the index of the next address to visit. The latter field starts at 0 and is incremented as each address is visited.

Then comes a reserved byte followed by a bit map with bits for each of the 24 potential IPv6 addresses following it. These bits tell whether each address must be visited directly after the one before it (strict source routing), or whether other routers may come in between (loose source routing).

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. Although this change is a major philosophical break with the past, 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 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. With IPv4, no such guarantee is present. 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 cryptographic techniques to accomplish their missions. We will give brief descriptions below, but readers not already familiar with modern cryptography may not understand the full description now. They should come back after having read Chap. 7 (in particular, Sec. 7.1), which treats cryptographic protocols.

When a sender and receiver wish to communicate securely, they must first agree on one or more secret keys that only they know. How they do this is outside

RED HAT 1057
RED HAT V. CAS

the scope of IPv6. Each of these keys is assigned a unique 32-bit key number. The key numbers are global, so that if Alice is using key 4 to talk to Bob, she cannot also have a key 4 to talk to Carol. Associated with each key number are other parameters, such as key lifetime, and so on.

To send an authenticated message, the sender first constructs a packet consisting of all the IP headers and the payload and then replaces the fields that change underway (e.g., *Hop limit*) with zeros. The packet is then padded out with zeros to a multiple of 16 bytes. Similarly, the secret key to be used is also padded out with zeros to a multiple of 16 bytes. Now a cryptographic checksum is computed on the concatenation of the padded secret key, the padded packet, and the padded secret key again. Users may define their own cryptographic checksum algorithms, but cryptographically unsophisticated users should use the default algorithm, MD5.

Now we come to the role of the authentication header. Basically, it contains three parts. The first part consists of 4 bytes holding the next header number, the length of the authentication header, and 16 zero bits. Then comes the 32-bit key number. Finally, the MD5 (or other) checksum is included.

The receiver then uses the key number to find the secret key. The padded version of it is then prepended and appended to the padded payload, the variable header fields are zeroed out, and the checksum computed. If it agrees with the checksum included in the authentication header, the receiver can be sure that the packet came from the sender with whom the secret key is shared and also be sure that the packet was not tampered with underway. The properties of MD5 make it computationally infeasible for an intruder to forge the sender's identity or modify the packet in a way that escapes detection.

It is important to note that the payload of an authenticated packet is sent unencrypted. Any router along the way can read what it says. For many applications, secrecy is not really important, just authentication. For example, if a user instructs his bank to pay his telephone bill, there is probably no real need for secrecy, but there is a very real need for the bank to be absolutely sure it knows who sent the packet containing the payment order.

For packets that must be sent secretly, the encrypted security payload extension header is used. It starts out with a 32-bit key number, followed by the encrypted payload. The encryption algorithm is up to the sender and receiver, but DES in cipher block chaining mode is the default. When DES-CBC is used, the payload field starts out with the initialization vector (a multiple of 4 bytes), then the payload, then padding out to multiple of 8 bytes. If both encryption and authentication are desired, both headers are needed.

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.

RED HAT 1057
RED HAT V. CAS

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. We will summarize a few of these below. For all the gory details, see (Huitema, 1996).

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 a long time and 65,535 hops is 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 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 at the destination with the same IPv6 address, or does it have to use a scheme with home agents and foreign

RED HAT 1057
RED HAT V. CAS

agents? Mobile hosts also introduce asymmetries into the routing system. It may well be the case that a small mobile computer can easily hear the powerful signal put out by a large stationary router, but the stationary router cannot hear the feeble signal put out by the mobile host. Consequently, 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 needed. The war was about where 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 have 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 stringent export laws concerning cryptography. Some, notably France and Iraq, also greatly restrict its use domestically, so that people cannot have secrets from the police. 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 potential solution is for all vendors to move their cryptography shops to a country that does not regulate cryptography, such as Finland or Switzerland. Strong cryptographic software could be designed and manufactured there and then shipped legally to all countries except France and Iraq. The problem with this approach is that designing part of the router software in one country and part in another can lead to integration problems.

The final controversy concerning security relates to the choice of the default algorithms that all implementations must support. While MD5 was thought to be relatively secure, recent advances in cryptography may weaken it. No serious cryptographer believes that DES is secure against attacks by major governments, but it is probably good enough to foil even the most precocious 12-year-olds for the time being. The compromise was thus to mandate security in IPv6, use a state-of-the-art checksum algorithm for good authentication and a weakish algorithm for secrecy but give users the option of replacing these algorithms with their own.

One point on which there was no controversy is that no one expects the IPv4 Internet to be turned off on a Sunday morning and come back up as an IPv6

RED HAT 1057
RED HAT V. CAS

Internet Monday morning. Instead, isolated "islands" of IPv6 will be converted, initially communicating via tunnels. As the IPv6 islands grow, they will merge into bigger islands. Eventually, all the islands will merge, and the Internet will be fully converted. Given the massive investment in IPv4 routers currently deployed, the conversion process will probably take a decade. For this reason, an enormous amount of effort has gone into making sure that this transition will be as painless as possible.

5.6. THE NETWORK LAYER IN ATM NETWORKS

The layers of the ATM model (see Fig. 1-30) do not map onto the OSI layers especially well, which leads to ambiguities. The OSI data link layer deals with framing and transfer protocols between two machines on the same physical wire (or fiber). Data link layer protocols are single-hop protocols. They do not deal with end-to-end connections because switching and routing do not occur in the data link layer. About this there is no doubt.

The lowest layer that goes from source to destination, and thus involves routing and switching (i.e., is multihop), is the network layer. The ATM layer deals with moving cells from source to destination and definitely involves routing algorithms and protocols within the ATM switches. It also deals with global addressing. Thus functionally, the ATM layer performs the work expected of the network layer. The ATM layer is not guaranteed to be 100 percent reliable, but that is not required for a network layer protocol.

Also, the ATM layer resembles layer 3 of X.25, which everyone agrees is a network layer protocol. Depending on bit settings, the X.25 network layer protocol may or may not be reliable, but most implementations treat it as unreliable. Because the ATM layer has the functionality expected of the network layer, does not have the functionality expected of the data link layer, and is quite similar to existing network layer protocols, we will discuss the ATM layer in this chapter.

Confusion arises because many people in the ATM community regard the ATM layer as a data link layer, or when doing LAN emulation, even a physical layer. Many people in the Internet community also regard it as a data link layer because they want to put IP on top of it, and making the ATM layer a data link layer fits well with this idea. (Although following through with this line of reasoning, to the Internet community, *all* networks operate at the data link layer, no matter what their physical characteristics are.)

The only problem is that the ATM layer does not have the characteristics of a data link layer protocol: a single-hop protocol used by machines at the opposite ends of a wire, such as protocols 1 through 6 in Chap. 3. It has the characteristics of a network layer protocol: end-to-end virtual circuits, switching, and routing.

The author is reminded of an old riddle:

Q: How many legs would a mule have if you called the tail a leg?
A: Four. *Calling* the tail a leg does not *make* it a leg.

RED HAT 1057
RED HAT V. CAS

Suffice it to say, the reader is warned of controversy here, combined with a major dose of raw emotion.

The ATM layer is connection oriented, both in terms of the service it offers and the way it operates internally. The basic element of the ATM layer is the virtual circuit (officially called a **virtual channel**). A virtual circuit is normally a connection from one source to one destination, although multicast connections are also permitted. Virtual circuits are unidirectional, but a pair of circuits can be created at the same time. Both parts of the pair are addressed by the same identifier, so effectively a virtual circuit is full duplex. However, the channel capacity and other properties may be different in the two directions and may even be zero for one of them.

The ATM layer is unusual for a connection-oriented protocol in that it does not provide any acknowledgements. The reason for this design is that ATM was designed for use on fiber optic networks, which are highly reliable. It was thought adequate to leave error control to higher layers. After all, sending acknowledgements in the data link or network layer is really only an optimization. It is always sufficient for the transport layer to send a message and then send the entire message again if it is not acknowledged on time.

Furthermore, ATM networks are often used for real-time traffic, such as audio and video. For this kind of traffic, retransmitting an occasional bad cell is worse than just ignoring it.

Despite its lack of acknowledgements, the ATM layer does provide one hard guarantee: cells sent along a virtual circuit will never arrive out of order. The ATM subnet is permitted to discard cells if congestion occurs but under no conditions may it reorder the cells sent on a single virtual circuit. No ordering guarantees are given about cells sent on *different* virtual circuits, however. For example, if a host sends a cell on virtual circuit 10 and later sends a cell on virtual circuit 20 to the same destination, the second cell may arrive first. If the two cells had been sent on the same virtual circuit, the first one sent would always be the first one to arrive.

The ATM layer supports a two-level connection hierarchy that is visible to the transport layer. Along any transmission path from a given source to a given destination, a group of virtual circuits can be grouped together into what is called a **virtual path**, as depicted in Fig. 5-61. Conceptually, a virtual path is like a bundle of twisted copper pairs: when it is rerouted, all the pairs (virtual circuits) are rerouted together. We will consider the implications of this two-level design in detail later.

5.6.1. Cell Formats

In the ATM layer, two interfaces are distinguished: the **UNI** (**User-Network Interface**) and the **NNI** (**Network-Network Interface**). The former defines the boundary between a host and an ATM network (in many cases, between the

RED HAT 1057
RED HAT V. CAS

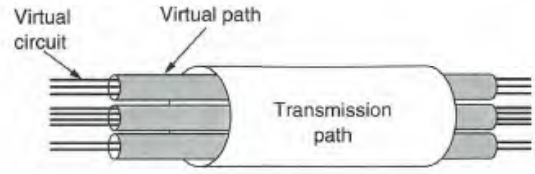


Fig. 5-61. A transmission path can hold multiple virtual paths, each of which can hold multiple virtual circuits.

customer and the carrier). The latter applies to the line between two ATM switches (the ATM term for routers).

In both cases the cells consist of a 5-byte header followed by a 48-byte payload, but the two headers are slightly different. The headers, as defined by the ATM Forum, are illustrated in Fig. 5-62. Cells are transmitted leftmost byte first and leftmost bit within a byte first.

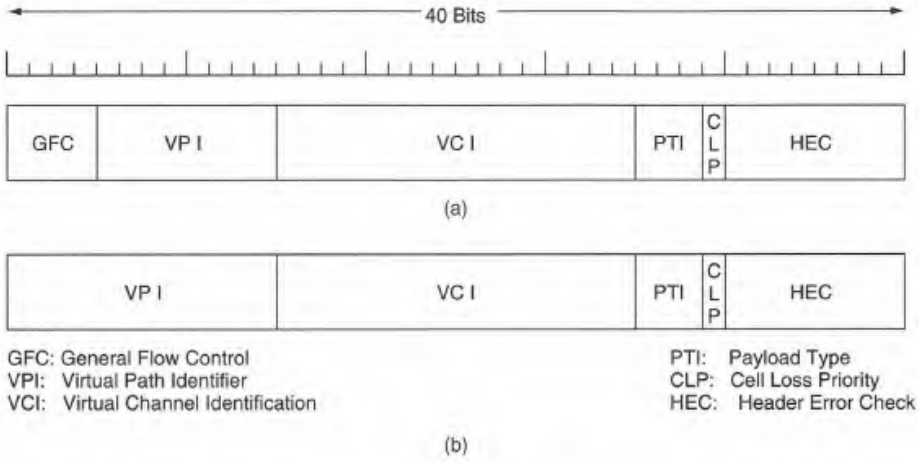


Fig. 5-62. (a) The ATM layer header at the UNI. (b) The ATM layer header at the NNI.

The *GFC* field is present only in cells between a host and the network. It is overwritten by the first switch it reaches, so it does not have end-to-end significance and is not delivered to the destination. It was originally conceived as perhaps having some utility for flow control or priority between hosts and the networks, but no values are defined for it and the network ignores it. The best way to think of it is as a bug in the standard.

The *VPI* field is a small integer selecting a particular virtual path (see Fig. 5-61). Similarly, the *VCI* field selects a particular virtual circuit within the chosen virtual path. Since the *VPI* field has 8 bits (at the UNI) and the *VCI* field has 16

RED HAT 1057
RED HAT V. CAS

bits, theoretically, a host could have up to 256 VP bundles, each containing up to 65,536 virtual circuits. Actually, slightly fewer of each are available because some *VCI*s are reserved for control functions, such as setting up virtual circuits.

The *PTI* field defines the type of payload the cell contains in accordance with the values given in Fig. 5-63. Here the cell types are user supplied, but the congestion information is network supplied. In other words, a cell sent with *PTI* 000 might arrive with 010 to warn the destination of problems underway.

Payload type	Meaning
000	User data cell, no congestion, cell type 0
001	User data cell, no congestion, cell type 1
010	User data cell, congestion experienced, cell type 0
011	User data cell, congestion experienced, cell type 1
100	Maintenance information between adjacent switches
101	Maintenance information between source and destination switches
110	Resource Management cell (used for ABR congestion control)
111	Reserved for future function

Fig. 5-63. Values of the *PTI* field.

The *CLP* bit can be set by a host to differentiate between high-priority traffic and low-priority traffic. If congestion occurs and cells must be discarded, switches first attempt to discard cells with *CLP* set to 1 before throwing out any set to 0.

Finally, the *HEC* field is a checksum over the header. It does not check the payload. A Hamming code on a 40-bit number only requires 5 bits, so with 8 bits a more sophisticated code can be used. The one chosen can correct all single-bit errors and can detect about 90 percent of all multibit errors. Various studies have shown that the vast majority of errors on optical links are single-bit errors.

Following the header comes 48 bytes of payload. Not all 48 bytes are available to the user, however, since some of the AAL protocols put their headers and trailers inside the payload.

The NNI format is the same as the UNI format, except that the *GFC* field is not present and those 4 bits are used to make the *VPI* field 12 bits instead of 8.

5.6.2. Connection Setup

ATM supports both permanent virtual circuits and switched virtual circuits. The former are always present and can be used at will, like leased lines. The latter have to be established each time they are used, like making phone calls. In this section we will describe how switched virtual circuits are established.

RED HAT 1057
RED HAT V. CAS

Technically, connection setup is not part of the ATM layer but is handled by the control plane (see Fig. 1-30) using a highly-complex ITU protocol called **Q.2931** (Stiller, 1995). Nevertheless, the logical place to handle setting up a network layer connection is in the network layer, and similar network layer protocols do connection setup here, so we will discuss it here.

Several ways are provided for setting up a connection. The normal way is to first acquire a virtual circuit for signaling and use it. To establish such a circuit, cells containing a request are sent on virtual path 0, virtual circuit 5. If successful, a new virtual circuit is opened on which connection setup requests and replies can be sent and received.

The reason for this two-step setup procedure is that this way the bandwidth reserved for virtual circuit 5 (which is barely used at all) can be kept extremely low. Also, an alternative way is provided to set up virtual circuits. Some carriers may allow users to have permanent virtual paths between predefined destinations, or allow them to set these up dynamically. Once a host has a virtual path to some other host, it can allocate virtual circuits on it itself, without the switches being involved.

Virtual circuit establishment uses the six message types listed in Fig. 5-64. Each message occupies one or more cells and contains the message type, length, and parameters. The messages can be sent by a host to the network or can be sent by the network (usually in response to a message from another host) to a host. Various other status and error reporting messages also exist but are not shown here.

Message	Meaning when sent by host	Meaning when sent by network
SETUP	Please establish a circuit	Incoming call
CALL PROCEEDING	I saw the incoming call	Your call request will be attempted
CONNECT	I accept the incoming call	Your call request was accepted
CONNECT ACK	Thanks for accepting	Thanks for making the call
RELEASE	Please terminate the call	The other side has had enough
RELEASE COMPLETE	Ack for RELEASE	Ack for RELEASE

Fig. 5-64. Messages used for connection establishment and release.

The normal procedure for establishing a call is for a host to send a SETUP message on a special virtual circuit. The network then responds with CALL PROCEEDING to acknowledge receipt of the request. As the SETUP message propagates toward the destination, it is acknowledged at each hop by CALL PROCEEDING.

When the SETUP message finally arrives, the destination host can respond with CONNECT to accept the call. The network then sends a CONNECT ACK message to indicate that it has received the CONNECT message. As the CONNECT message

RED HAT 1057
RED HAT V. CAS

propagates back toward the originator, each switch receiving it acknowledges it with a CONNECT ACK message. This sequence of events is shown in Fig. 5-65(a).

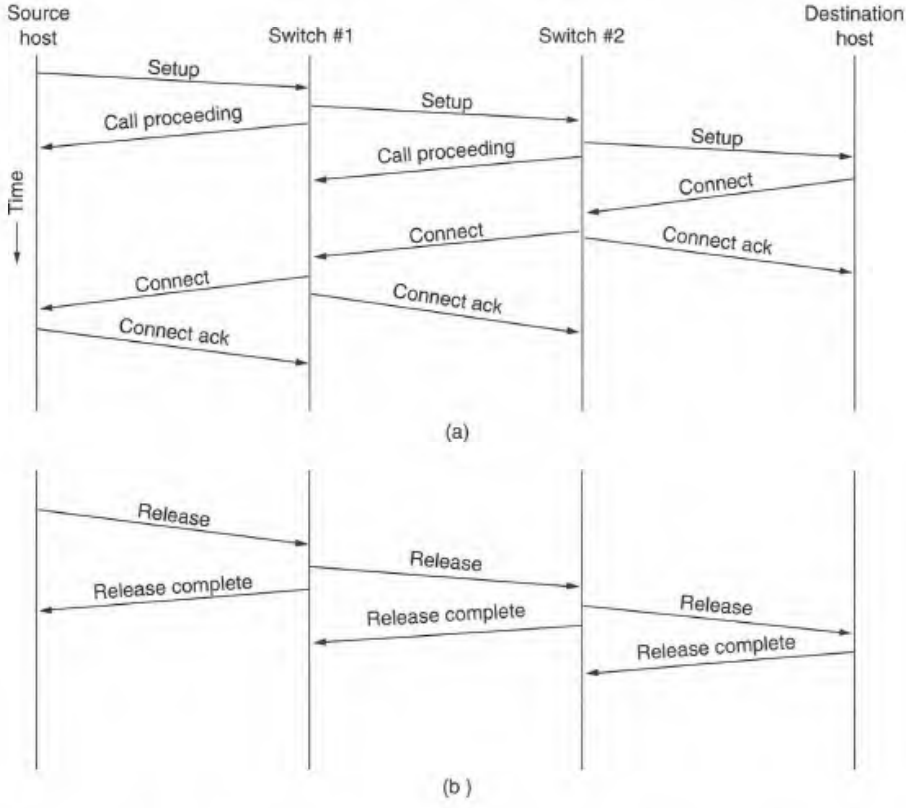


Fig. 5-65. (a) Connection setup in an ATM network. (b) Connection release.

The sequence for terminating a virtual circuit is simple. The host wishing to hang up just sends a RELEASE message, which propagates to the other end and causes the circuit to be released. Each hop along the way, the message is acknowledged, as shown in Fig. 5-65(b).

ATM networks allow multicast channels to be set up. A multicast channel has one sender and more than one receiver. These are constructed by setting up a connection to one of the destinations in the usual way. Then the ADD PARTY message is sent to attach a second destination to the virtual circuit returned by the previous call. Additional ADD PARTY messages can be sent afterward to increase the size of the multicast group.

In order to set up a connection to a destination, it is necessary to specify which destination, by including its address in the SETUP message. ATM addresses

RED HAT 1057
RED HAT V. CAS

come in three forms. The first is 20 bytes long and is based on OSI addresses. The first byte indicates which of three formats the address is in. In the first format, bytes 2 and 3 specify a country, and byte 4 gives the format of the rest of the address, which contains a 3-byte authority, a 2-byte domain, a 2-byte area, and a 6-byte address, plus some other items. In the second format, bytes 2 and 3 designate an international organization instead of a country. The rest of the address is the same as in format 1. Alternatively, a older form of addressing (CCITT E.164) using 15-digit decimal ISDN telephone numbers is also permitted.

5.6.3. Routing and Switching

When a virtual circuit is set up, the SETUP message wends its way through the network from source to destination. The routing algorithm determines the path taken by this message, and thus by the virtual circuit. The ATM standard does not specify any particular routing algorithm, so the carrier is free to choose among the algorithms discussed earlier in this chapter, or to use a different one.

Experience with previous connection-oriented networks, such as X.25, have shown that a considerable amount of computing power in the switches can be expended determining how to convert the virtual circuit information in each cell to the choice of output line. The ATM designers wanted to avoid this fate, so the ATM layer has been designed to make efficient routing possible. In particular, the idea was to route on the *VPI* field, but not the *VCI* field, except at the final hop in each direction, when cells are sent between a switch and a host. Between two switches, only the virtual path was to be used.

Using only the *VPI*s between interior switches has several advantages. To start with, once a virtual path has been established from a source to a destination, any additional virtual circuits along that path can just follow the existing path. No new routing decisions have to be made. It is as though a bundle of twisted pairs has already been pulled from the source to the destination. Setting up a new connection merely requires allocating one of the unused pairs.

Second, routing of individual cells is easier when all virtual circuits for a given path are always in the same bundle. The routing decision only involves looking at a 12-bit number, not a 12-bit number and a 16-bit number. We will describe how cell switching is done below, but even without going into the details, it should be clear that indexing into a table of 2^{12} entries is feasible whereas indexing into a table of 2^{28} entries is not.

Third, basing all routing on virtual paths makes it easier to switch a whole group of virtual circuits. Consider, for example, the hypothetical U.S. ATM backbone illustrated in Fig. 5-66. Normally, virtual circuits from NY to SF pass through Omaha and Denver. However, suppose a disturbance occurs on the Omaha-Denver line. By rerouting the Omaha-Denver virtual path to LA and then SF, all the virtual circuits (potentially up to 65,535 of them) can be switched in one operation instead of potentially thousands of operations.

RED HAT 1057
RED HAT V. CAS

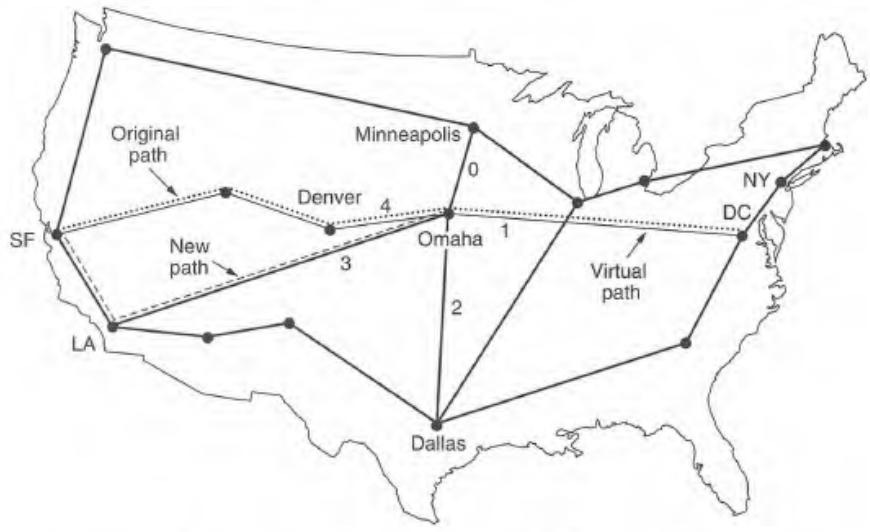


Fig. 5-66. Rerouting a virtual path reroutes all of its virtual circuits.

Finally, virtual paths make it easier for carriers to offer closed user groups (private networks) to corporate customers. A company can set up a network of permanent virtual paths among its various offices, and then allocate virtual circuits within these paths on demand. No calls can come into the private network from outside and no calls can leave the private network, except via special gateways. Many companies like this kind of security.

Whether switches will actually use the *VPI* field for routing as planned, or will, in fact, use the combination of the *VPI* and *VCI* fields (thus negating all the advantages just discussed) remains to be seen. Initial evidence from the field is not encouraging.

Let us now see how cells could be routed within an interior switch (one that is attached only to other switches and not to hosts). To make matters concrete, let us consider the Omaha switch of Fig. 5-66. For each of its five incoming lines, it has a table, *vpi_table*, indexed by incoming *VPI* that tells which of the five outgoing lines to use and what VPI to put in outgoing cells. Let us assume the five lines are numbered from 0 to 4, clockwise starting at Minneapolis. For each outgoing line, the switch maintains a bit map telling which *VPI*s are currently in use on that line.

When the switch is booted, all the entries in all the *vpi_table* structures are marked as not in use. Similarly, all the bit maps are marked to indicate that all *VPI*s are available (except the reserved ones). Now suppose calls come as shown in Fig. 5-67.

As each virtual path (and virtual circuit) is set up, entries are made in the tables. We will assume the virtual circuits are full duplex, so that each one set up

RED HAT 1057
RED HAT V. CAS

Source	Incoming line	Incoming VPI	Destination	Outgoing line	Outgoing VPI	Path:
NY	1	1	SF	4	1	New
NY	1	2	Denver	4	2	New
LA	3	1	Minneapolis	0	1	New
DC	1	3	LA	3	2	New
NY	1	1	SF	4	1	Old
SF	4	3	DC	1	4	New
DC	1	5	SF	4	4	New
NY	1	2	Denver	4	2	Old
SF	4	5	Minneapolis	0	2	New
NY	1	1	SF	4	1	Old

Fig. 5-67. Some routes through the Omaha switch of Fig. 5-66.

results in two entries, one for the forward traffic from the source and one for the reverse traffic from the destination.

The tables corresponding to the routes of Fig. 5-67 are shown in Fig. 5-68. For example, the first call generates the (4, 1) entry for *VPI* 1 in the DC table because it refers to cells coming in on line 1 with *VPI* 1 and going to SF. However, an entry is also made in the Denver table for *VPI* 1 showing that cells coming in from Denver with *VPI* 1 should go out on line 1 with *VPI* 1. These are cells traveling the other way (from SF to NY) on this virtual path. Note that in some cases two or three virtual circuits are sharing a common path. No new table entries are needed for additional virtual circuits connecting a source and destination that already have a path assigned.

Now we can explain how cells are processed inside a switch. Suppose that a cell arrives on line 1 (DC) with *VPI* 3. The switch hardware or software uses the 3 as an index into the table for line 1 and sees that the cell should go out on line 3 (LA) with *VPI* 2. It overwrites the *VPI* field with a 2 and puts the outgoing line number, 3, somewhere in the cell, for example, in the *HEC* field, since that has to be recomputed later anyway.

Now the question is how to get the cell from its current input buffer to line 3. However, this issue (routing within a switch) was discussed in detail in Chap. 2, and we saw how it was done in knockout and Batcher-banyan switches.

At this point it is straightforward to see how an entire bundle of virtual circuits can be rerouted, as is done in Fig. 5-66. By changing the entry for *VPI* 1 in the DC table from (4, 1) to (3, 3), cells from NY headed for SF will be diverted to LA. Of course, the LA switch has to be informed of this event, so the switch has

RED HAT 1057
RED HAT V. CAS

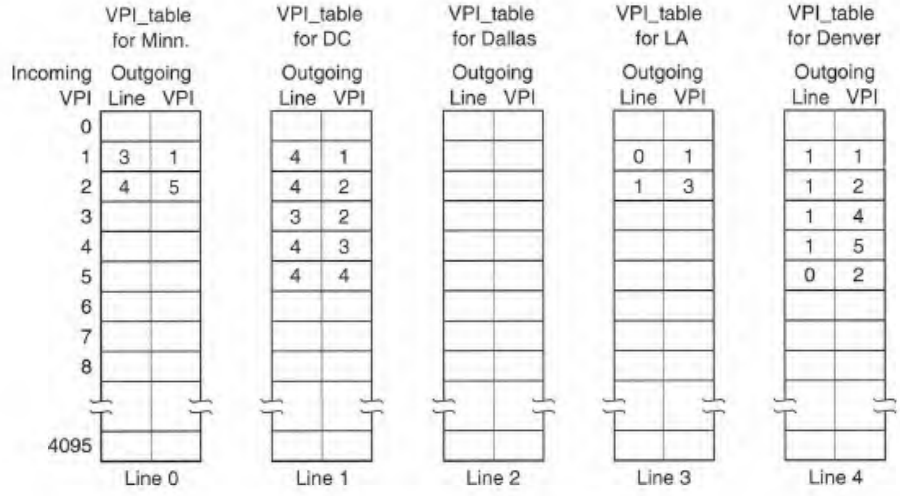


Fig. 5-68. The table entries for the routes of Fig. 5-67.

to generate and send a SETUP message to LA to establish the new path with *VPI* 3. Once this path has been set up, all the virtual circuits from NY to SF are now rerouted via LA, even if there are thousands of them. If virtual paths did not exist, each virtual circuit would have its own table entry and would have to be rerouted separately.

It is worth pointing out explicitly that the discussion above is about ATM in WANs. In a LAN, matters are much simpler. For example, a single virtual path can be used for all virtual circuits.

5.6.4. Service Categories

After a fair amount of trial and error, by version 4.0 of the ATM specification, it was becoming clear what kinds of traffic ATM networks were carrying and what kind of services their customers wanted. Consequently, the standard was modified to explicitly list the service categories commonly used, in order to allow equipment vendors to optimize their adaptor boards and switches for some or all of these categories. The service categories chosen as being important are listed in Fig. 5-69.

The **CBR (Constant Bit Rate)** class is intended to emulate a copper wire or optical fiber (only at much greater expense). Bits are put on one end and they come off the other end. No error checking, flow control, or other processing is done. Nevertheless, this class is essential to making a smooth transition between

RED HAT 1057
RED HAT V. CAS

Class	Description	Example
CBR	Constant bit rate	T1 circuit
RT-VBR	Variable bit rate: real time	Real-time videoconferencing
NRT-VBR	Variable bit rate: non-real time	Multimedia email
ABR	Available bit rate	Browsing the Web
UBR	Unspecified bit rate	Background file transfer

Fig. 5-69. The ATM service categories.

the current telephone system and future B-ISDN systems, since voice-grade PCM channels, T1 circuits, and most of the rest of the telephone system use constant-rate, synchronous bit transmission. With the CBR class, all of this traffic can be carried directly by an ATM system. CBR is also suited to all other interactive (i.e., real-time) audio and video streams.

The next class, **VBR (Variable Bit Rate)**, is divided into two subclasses, for real time and non-real time, respectively. RT-VBR is intended for services that have variable bit rates combined with stringent real-time requirements, such as interactive compressed video (e.g., videoconferencing). Due to the way MPEG and other compression schemes work, with a complete base frame followed by a series of differences between the current frame and the base frame, the transmission rate varies strongly in time (Pancha and El Zarki, 1994). Despite this variation, it is essential that the ATM network not introduce any jitter in the cell arrival pattern, as this will cause the display to appear jerky. In other words, both the average cell delay and the variation in cell delay must be tightly controlled. On the other hand, an occasional lost bit or cell here is tolerable and is best just ignored.

The other VBR subclass is for traffic where timely delivery is important but a certain amount of jitter can be tolerated by the application. For example, multimedia email is typically spooled to the receiver's local disk before being displayed, so any variation in cell delivery times will be eliminated before the email is viewed.

The **ABR (Available Bit Rate)** service category is designed for bursty traffic whose bandwidth range is known roughly. A typical example might be for use in a company that currently connects its offices by a collection of leased lines. Typically, the company has a choice of putting in enough capacity to handle the peak load, which means that some lines are idle part of the day, or putting in just enough capacity for the minimum load, which leads to congestion during the busiest part of the day.

Using ABR service avoids having to make a long term commitment to a fixed bandwidth. With ABR it is possible to say, for example, that the capacity between two points must always be 5 Mbps, but might have peaks up to 10 Mbps.

RED HAT 1057
RED HAT V. CAS

The system will then guarantee 5 Mbps all the time, and do its best to provide 10 Mbps when needed, but with no promises.

ABR is the only service category in which the network provides rate feedback to the sender, asking it to slow down when congestion occurs. Assuming that the sender complies with such requests, cell loss for ABR traffic is expected to be low. Traveling ABR is a little like flying standby: if there are seats left over (excess capacity), standby passengers are transported without delay. If there is insufficient capacity, they have to wait (unless some of the minimum bandwidth is available).

Finally, we come to **UBR (Unspecified Bit Rate)**, which makes no promises and gives no feedback about congestion. This category is well suited to sending IP packets, since IP also makes no promises about delivery. All UBR cells are accepted, and if there is capacity left over, they will also be delivered. If congestion occurs, UBR cells will be discarded, with no feedback to the sender and no expectation that the sender slows down.

To continue our standby analogy, with UBR, all standby passengers get to board, but if halfway to the destination the pilot sees that fuel is running low, standby passengers are unceremoniously pushed through the emergency exit. To make UBR attractive, carriers are likely to make it cheaper than the other classes. For applications that have no delivery constraints and want to do their own error control and flow control anyway, UBR is a perfectly reasonably choice. File transfer, email, and USENET news are all potential candidates for UBR service because none of these applications have real-time characteristics.

The properties of the various service categories are summarized in Fig. 5-70.

Service characteristic	CBR	RT-VBR	NRT-VBR	ABR	UBR
Bandwidth guarantee	Yes	Yes	Yes	Optional	No
Suitable for real-time traffic	Yes	Yes	No	No	No
Suitable for bursty traffic	No	No	Yes	Yes	Yes
Feedback about congestion	No	No	No	Yes	No

Fig. 5-70. Characteristics of the ATM service categories.

5.6.5. Quality of Service

Quality of service is an important issue for ATM networks, in part because they are used for real-time traffic, such as audio and video. When a virtual circuit is established, both the transport layer (typically a process in the host machine, the "customer") and the ATM network layer (e.g., a network operator, the "carrier") must agree on a contract defining the service. In the case of a public network, this contract may have legal implications. For example, if the carrier agrees not to

RED HAT 1057
RED HAT V. CAS

lose more than one cell per billion and it loses two cells per billion, the customer's legal staff may get all excited and start running around yelling "breach of contract."

The contract between the customer and the network has three parts:

1. The traffic to be offered.
2. The service agreed upon.
3. The compliance requirements.

It is worth noting that the contract may be different for each direction. For a video-on-demand application, the required bandwidth from the user's remote control to the video server might be 1200 bps. In the other direction it might be 5 Mbps. It should be noted that if the customer and the carrier cannot agree on terms, or the carrier is unable to provide the service desired, the virtual circuit will not be set up.

The first part of the contract is the **traffic descriptor**. It characterizes the load to be offered. The second part of the contract specifies the quality of service desired by the customer and accepted by the carrier. Both the load and the service must be formulated in terms of measurable quantities, so compliance can be objectively determined. Merely saying "moderate load" or "good service" will not do.

To make it possible to have concrete traffic contracts, the ATM standard defines a number of **QoS** (**Quality of Service**) parameters whose values the customer and carrier can negotiate. For each quality of service parameter, the worst case performance for each parameter is specified, and the carrier is required to meet or exceed it. In some cases, the parameter is a minimum; in others it is a maximum. Again here, the quality of service is specified separately for each direction. Some of the more important ones are listed in Fig. 5-71, but not all of them are applicable to all service categories.

The first three parameters specify how fast the user wants to send. **PCR** (**Peak Cell Rate**) is the maximum rate at which the sender is planning to send cells. This parameter may be lower than what the bandwidth of the line permits. If the sender is planning to push out a cell every 4 μsec, its *PCR* is 250,000 cells/sec, even though the actual cell transmission time may be 2.7 μsec.

SCR (**Sustained Cell Rate**) is the expected or required cell rate averaged over a long time interval. For CBR traffic, *SCR* will be equal to *PCR*, but for all the other service categories, it will be substantially lower. The *PCR/SCR* ratio is one measure of the burstiness of the traffic.

MCR (**Minimum Cell Rate**) is the minimum number of cells/sec that the customer considers acceptable. If the carrier is unable to guarantee to provide this much bandwidth it must reject the connection. When ABR service is requested, then the actual bandwidth used must lie between *MCR* and *PCR*, but it may vary

RED HAT 1057
RED HAT V. CAS

Parameter	Acronym	Meaning
Peak cell rate	PCR	Maximum rate at which cells will be sent
Sustained cell rate	SCR	The long-term average cell rate
Minimum cell rate	MCR	The minimum acceptable cell rate
Cell delay variation tolerance	CDVT	The maximum acceptable cell jitter
Cell loss ratio	CLR	Fraction of cells lost or delivered too late
Cell transfer delay	CTD	How long delivery takes (mean and maximum)
Cell delay variation	CDV	The variance in cell delivery times
Cell error rate	CER	Fraction of cells delivered without error
Severely-errored cell block ratio	SECBR	Fraction of blocks garbled
Cell misinsertion rate	CMR	Fraction of cells delivered to wrong destination

Fig. 5-71. Some of the quality of service parameters.

dynamically during the lifetime of the connection. If the customer and carrier agree to setting MCR to 0, then ABR service becomes similar to UBR service.

CVDT (Cell Variation Delay Tolerance) tells how much variation will be present in cell transmission times. It is specified independently for *PCR* and *SCR*. For a perfect source operating at *PCR*, every cell will appear *exactly* $1/PCR$ after the previous one. No cell will ever be early and no cell will ever be late, not even by a picosecond. For a real source operating at *PCR*, some variation will occur in cell transmission times. The question is: How much variation is acceptable? Can a cell be 1 nsec early? How about 30 seconds? *CDVT* controls the amount of variability acceptable using a leaky bucket algorithm to be described shortly.

The next three parameters describe characteristics of the network and are measured at the receiver. All three are negotiable. **CLR (Cell Loss Ratio)** is straightforward. It measures the fraction of the transmitted cells that are not delivered at all or are delivered so late as to be useless (e.g., for real-time traffic). **CTD (Cell Transfer Delay)** is the average transit time from source to destination. **CDV (Cell Delay Variation)** measures how uniformly the cells are delivered.

The model for *CTD* and *CDV* is shown in Fig. 5-72. Here we see the probability of a cell taking time t to arrive, as a function of t. For a given source, destination, and route through the intermediate switches, some minimum delay always exists due to propagation and switching time. However, not all cells make it in the minimum time; the probability density function usually has a long tail. By choosing a value of *CTD*, the customer and the carrier are, in effect, agreeing, on how late a cell can be delivered and still count as a correctly delivered cell. Normally, *CDV* will be chosen so that, α, the fraction of cells that are rejected for

RED HAT 1057
RED HAT V. CAS

being too late will be on the order of 10^{-10} or less. *CDV* measures the spread in arrival times. For real-time traffic, this parameter is often more important than *CDT*.

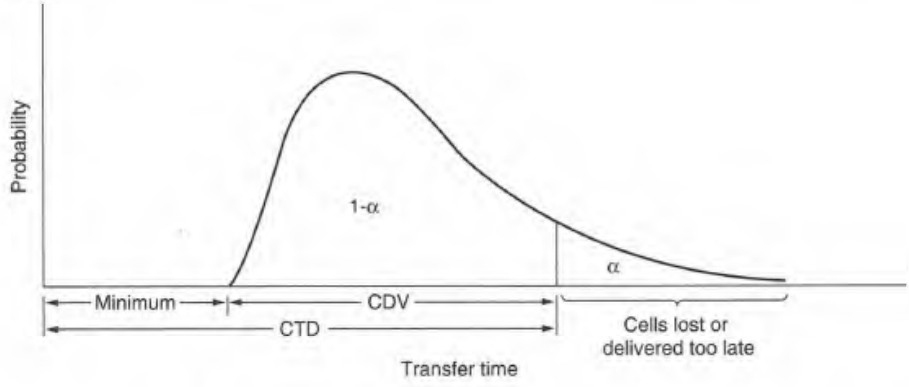


Fig. 5-72. The probability density function for cell arrival times.

The last three QoS parameters specify characteristics of the network. They are generally not negotiable. **CER (Cell Error Ratio)** is the fraction of cells that are delivered with one or more bits wrong. **SECBR (Severely-Errored Cell Block Ratio)** is the fraction of *N*-cell blocks of which *M* or more cells contain an error. Finally, **CMR (Cell Misinsertion Rate)** is the number of cells/sec that are delivered to the wrong destination on account of an undetected error in the header.

The third part of the traffic contract tells what constitutes obeying the rules. If the customer sends one cell too early, does this void the contract? If the carrier fails to meet one of its quality targets for a period of 1 msec, can the customer sue? Effectively, this part of the contract is negotiated between the parties and says how strictly the first two parts will be enforced.

The ATM and Internet quality of service models differ somewhat, which impacts their respective implementations. The ATM model is based strictly on connections, whereas the Internet model uses datagrams plus flows (e.g., RSVP). A comparison of these two models is given in (Crowcroft et al., 1995).

5.6.6. Traffic Shaping and Policing

The mechanism for using and enforcing the quality of service parameters is based (in part) on a specific algorithm, the **Generic Cell Rate Algorithm (GCRA)**, and is illustrated in Fig. 5-73. It works by checking every cell to see if it conforms to the parameters for its virtual circuit.

GCRA has two parameters. These specify the maximum allowed arrival rate (*PCR*) and the amount of variation herein that is tolerable (*CDVT*). The

RED HAT 1057
RED HAT V. CAS

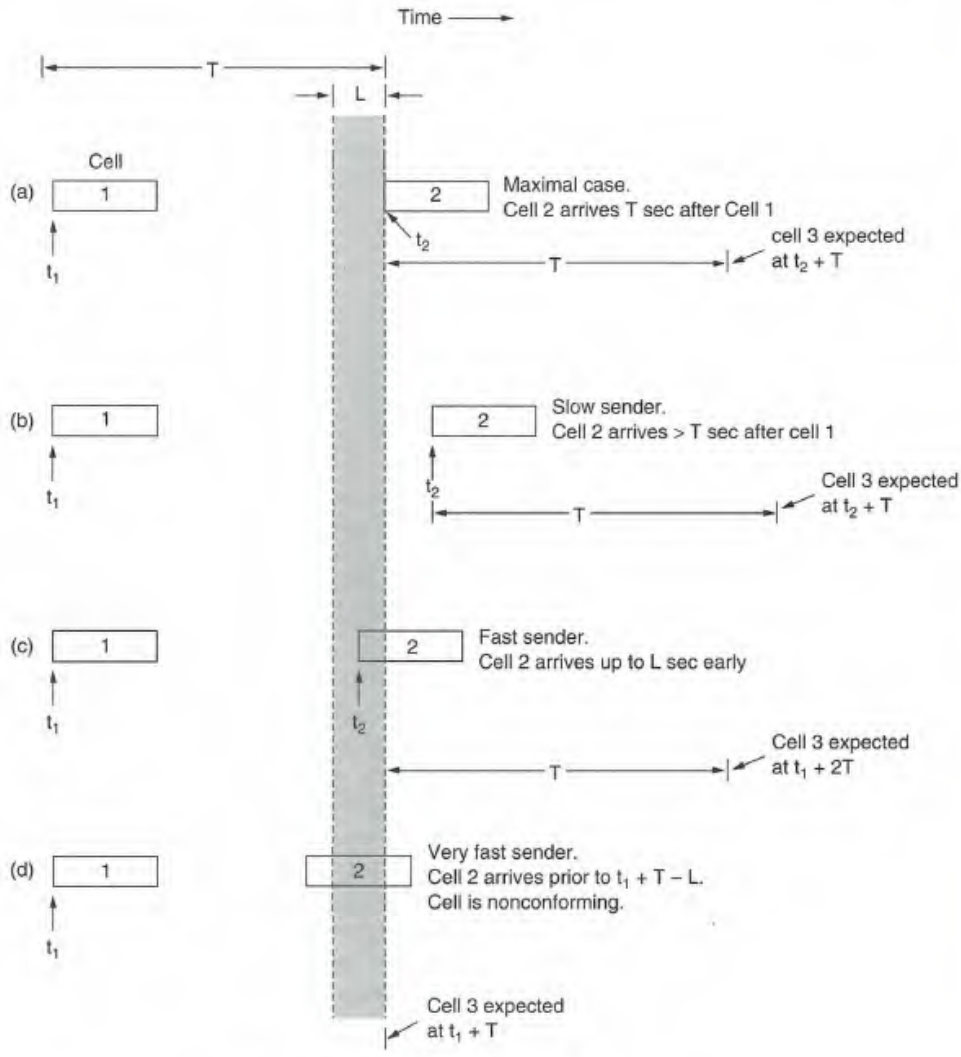


Fig. 5-73. The generic cell rate algorithm.

reciprocal of *PCR*, $T = 1/PCR$, is the minimum cell interarrival time, as shown in Fig. 5-73(a). If the customer agrees not to send more than 100,000 cells/sec, then $T = 10$ μsec. In the maximal case, one cell arrives promptly every 10 μsec. To avoid tiny numbers, we will work in microseconds, but since all the parameters are real numbers, the unit of time does not matter.

RED HAT 1057
RED HAT V. CAS

A sender is always permitted to space consecutive cells more widely than T, as shown in Fig. 5-73(b). Any cell arriving more than T μsec after the previous one is conforming.

The problem arises with senders that tend to jump the gun, as in Fig. 5-73(c) and (d). If a cell arrives a little early (at or later than $t_1 + T - L$), it is conforming, but the next cell is still expected at $t_1 + 2T$, (not at $t_2 + T$), to prevent the sender from transmitting every cell L μsec early, and thus increasing the peak cell rate.

If a cell arrives more than L μsec early, it is declared as nonconforming. The treatment of nonconforming cells is up to the carrier. Some carriers may simply discard them; others may keep them, but set the *CLP* bit, to mark them as low priority to allow switches to drop nonconforming cells first in the event of congestion. The use of the *CLP* bit may also be different for the different service categories of Fig. 5-69.

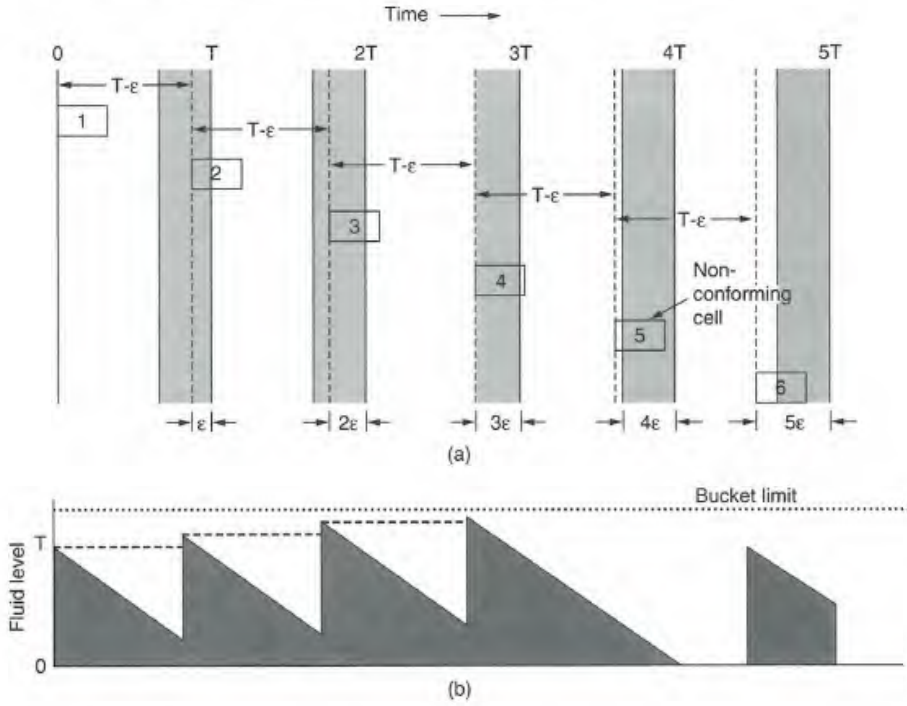


Fig. 5-74. (a) A sender trying to cheat. (b) The same cell arrival pattern, but now viewed in terms of a leaky bucket.

Now let us consider what happens if a sender tries to cheat a little bit, as shown in Fig. 5-74(a). Instead of waiting until time T to send cell 2, the sender

RED HAT 1057
RED HAT V. CAS

transmits it a wee bit early, at $T - \varepsilon$, where, say, $\varepsilon = 0.3L$. This cell is accepted without problems.

Now the sender transmits cell 3, again at $T - \varepsilon$ after the previous cell, that is, at $T - 2\varepsilon$. Again it is accepted. However, every successive cell inches closer and closer to the fatal $T - L$ boundary. In this case, cell 5 arrives at $T - 4\varepsilon$ ($T - 1.2L$) which is too early, so cell 5 is declared nonconforming and is discarded by the network interface.

When viewed in these terms, the GCRA algorithm is called a **virtual scheduling algorithm**. However, viewed differently, it is equivalent to a leaky bucket algorithm, as depicted in Fig. 5-74(b). Imagine that each conforming cell that arrives pours T units of fluid into a leaky bucket. The bucket leaks fluid at a rate of 1 unit/μsec, so that after T μsec it is all gone. If cells arrive precisely every T μsec, each arriving cell will find the bucket (just) emptied, and will refill it with T units of fluid. Thus the fluid level is raised to T when a cell arrives and is reduced linearly until it gets to zero. This situation is illustrated in Fig. 5-74(b) between 0 and T.

Since fluid drains out linearly in time, at a time t after a cell arrives, the amount of its fluid left is $T - t$. At the time cell 2 arrives, at $T - \varepsilon$, there are still ε units of fluid in the bucket. The addition of the new cell raises this value to $T + \varepsilon$. Similarly, at the time cell 3 arrives, 2ε units are left in the bucket so the new cell raises the fluid level to $T + 2\varepsilon$. When cell 4 arrives, it is raised to $T + 3\varepsilon$.

If this goes on indefinitely, some cell is going to raise the level to above the bucket capacity and thus be rejected. To see which one it is, let us now compute what the bucket capacity is. We want the leaky bucket algorithm to give the same result as Fig. 5-74(a), so we want overflow to occur when a cell arrives L μsec early. If the fluid left requires L μsec to drain out, the amount of fluid must be L, since the drain rate is 1 unit/μsec. Thus we want the bucket capacity to be $L + T$ so that any cell arriving more than L μsec early will be rejected due to bucket overflow. In Fig. 5-74(b), when cell 5 arrives, the addition of T units to the 4ε units of fluid already present raises the bucket level to $T + 4\varepsilon$. Since we are using $\varepsilon = 0.3L$ in this example, the bucket would be raised to $T + 1.2L$ by the addition of cell 5, so the cell is rejected, no new fluid is added, and the bucket eventually empties.

For a given T, if we set L very small, the capacity of the bucket will be hardly more than T, so all cells will have to be sent with a very uniform spacing. However, if we now raise L to a value much greater than T, the bucket can hold multiple cells because $T + L \gg T$. This means that the sender can transmit a burst of cells back-to-back at the peak rate and have them still accepted.

We can easily compute the number of conforming cells, N, that can be transmitted back-to-back at the peak cell rate ($PCR = 1/T$). During a burst of N cells, the total amount of fluid added to the bucket is NT because each cell adds T. However, during the maximum burst, fluid drains out of the bucket at a rate of 1 unit per time interval. Let us call the cell transmission time δ time units. Note

RED HAT 1057
RED HAT V. CAS

that $\delta \leq T$ because it is entirely possible for a sender on a 155.52 Mbps line to agree to send no more than 100,000 cells/sec, in which case $\delta = 2.73$ μsec and $T = 10$ μsec. During the burst of N cells, the amount of drainage is $(N - 1)\delta$ because drainage does not start until the first cell has been entirely transmitted.

From these observations, we see that the net increase in fluid in the bucket during the maximum burst is $NT - (N - 1)\delta$. The bucket capacity is $T + L$. Equating these two quantities we get

$$NT - (N - 1)\delta = T + L$$

Solving this equation for N we get

$$N = 1 + \frac{L}{T - \delta}$$

However, if this number is not an integer, it must be rounded downward to an integer to prevent bucket overflow. For example, with $PCR = 100{,}000$ cells/sec, $\delta = 2.73$ μsec, and $L = 50$ μsec, seven cells may be sent back-to-back at the 155.52 Mbps rate without filling the bucket. An eighth cell would be nonconforming.

The GCRA is normally specified by giving the parameters T and L. T is just the reciprocal of *PCR*; L is *CDVT*. The GCRA is also used to make sure the mean cell rate does not exceed *SCR* for any substantial period.

In this example we assumed that cells arrive uniformly. In reality, they do not. Nevertheless, the leaky bucket algorithm can also be used here, too. At every cell arrival, a check is made to see if there is room in the bucket for an additional T units of fluid. If there is, the cell is conforming; otherwise it is not.

In addition to providing a rule about which cells are conforming and which ones are not, the GCRA also shapes the traffic to remove some of the burstiness. The smaller *CDVT* is, the greater the smoothing effect, but the greater the chance that cells will be discarded as nonconforming. Some implementations combine the GCRA leaky bucket with a token bucket, to provide additional smoothing.

5.6.7. Congestion Control

Even with traffic shaping, ATM networks do not automatically meet the performance requirements set forth in the traffic contract. For example, congestion at intermediate switches is always a potential problem, especially when over 350,000 cells/sec are pouring in on each line, and a switch can have 100 lines. Consequently, a great deal of thought has gone into the subject of performance and congestion in ATM networks. In this section, we will discuss some of the approaches used. For additional information, see (Eckberg, 1992; Eckberg et al., 1991; Hong and Suda, 1991; Jain, 1995; and Newman, 1994).

ATM networks must deal with both long-term congestion, caused by more traffic coming in than the system can handle, and short-term congestion, caused

RED HAT 1057
RED HAT V. CAS

by burstiness in the traffic. As a result, several different strategies are used together. The most important of these fall into three categories:

1. Admission control.
2. Resource reservation.
3. Rate-based congestion control.

We will now discuss each of these strategies in turn.

Admission Control

In low-speed networks, it is usually adequate to wait for congestion to occur and then react to it by telling the source of the packets to slow down. In high-speed networks, this approach often works poorly, because in the interval between sending the notification and notification arriving at the source, thousands of additional packets may arrive.

Furthermore, many ATM networks have real-time traffic sources that produce data at an intrinsic rate. Telling such a source to slow down may not work (imagine a new digital telephone with a red light on it; when congestion is signaled, the red light comes on and the speaker is required to talk 25 percent slower).

Consequently, ATM networks emphasize preventing congestion from occurring in the first place. However, for CBR, VBR, and UBR traffic, no dynamic congestion control is present at all, so here an ounce of prevention is worth a pound (actually, more like a metric ton) of cure. A major tool for preventing congestion is admission control. When a host wants a new virtual circuit, it must describe the traffic to be offered and the service expected. The network can then check to see if it is possible to handle this connection without adversely affecting existing connections. Multiple potential routes may have to be examined to find one which can do the job. If no route can be located, the call is rejected.

Denying admission should be done fairly. Is it fair that one couch potato zapping through dozens of television programs can wipe out 100 busy beavers trying to read their email? If no controls are applied, a small number of high-bandwidth users can severely affect many low-bandwidth users. To prevent this, users should be divided into classes based on usage. The probability of service denial should be roughly the same for all classes (possibly by giving each class its own resource pool).

Resource Reservation

Closely related to admission control is the technique of reserving resources in advance, usually at call setup time. Since the traffic descriptor gives the peak cell rate, the network has the possibility of reserving enough bandwidth along the path

RED HAT 1057
RED HAT V. CAS

to handle that rate. Bandwidth can be reserved by having the SETUP message earmark bandwidth along each line it traverses, making sure, of course, that the total bandwidth earmarked along a line is less than the capacity of that line. If the SETUP message hits a line that is full, it must backtrack and look for an alternative path.

The traffic descriptor can contain not only the peak bandwidth, but also the average bandwidth. If a host wants, for example, a peak bandwidth of 100,000 cells/sec, but an average bandwidth of only 20,000 cells/sec, in principle, five such circuits could be multiplexed onto the same physical trunk. The trouble is that all five connections could be idle for half an hour, then start blasting away at the peak rate, causing massive cell loss. Since VBR traffic can be statistically multiplexed, problems can occur with this service category. Possible solutions are being studied.

Rate-Based Congestion Control

With CBR and VBR traffic, it is generally not possible for the sender to slow down, even in the event of congestion, due to the inherent real-time or semi-real-time nature of the information source. With UBR, nobody cares; if there are too many cells, the extra ones are just dropped.

However, with ABR traffic, it is possible and reasonable for the network to signal one or more senders and ask them to slow down temporarily until the network can recover. It is in the interest of a sender to comply, since the network can always punish it by throwing out its (excess) cells.

How congestion should be detected, signaled, and controlled for ABR traffic was a hot topic during the development of the ATM standard, with vigorous arguments for various proposed solutions. Let us now briefly look at some of the solutions that were quickly rejected before examining the winner.

In one proposal, whenever a sender wished to send a burst of data, it first had to send a special cell reserving the necessary bandwidth. After the acknowledgement came back, the burst could begin. The advantage here is that congestion never occurs because the required bandwidth is always there when it is needed. The ATM Forum rejected this solution due to the potentially long delay before a host may begin to send.

A second proposal had switches sending back choke cells whenever congestion began to occur. Upon receipt of such a cell, a sender was expected to cut back to half its current cell transmission rate. Various schemes were proposed for getting the rate back up again later when the congestion cleared up. This scheme was rejected because choke cells might get lost in the congestion, and because the scheme seemed unfair to small users. For example, consider a switch getting 100-Mbps streams from each of five users, and one 100-kbps stream from another user. Many committee members felt it was inappropriate to tell the 100 kbps user to give up 50 kbps because he was causing too much congestion.

RED HAT 1057
RED HAT V. CAS

A third proposal used the fact that packet boundaries are marked by a bit in the last cell. The idea here was to discard cells to relieve the congestion but to do this highly selectively. The switch was to scan the incoming cell stream for the end of a packet and then throw out all the cells in the next packet. Of course, this one packet would be transmitted later, but dropping all k cells in one packet ultimately leads to one packet retransmission, which is far better than dropping k random cells, which might lead to k packet retransmissions. This scheme was rejected on fairness grounds because the next end-of-packet mark seen might not belong to the sender overloading the switch. Also the scheme did not need to be standardized. Any switch vendor is free to pick which cells to discard when congestion occurs.

After much discussion, the battle focused on two contenders, a credit-based solution (Kung and Morris, 1995) and rate-based solution (Bonomi and Fendick, 1995). The credit-based solution was essentially a dynamic sliding window protocol. It required each switch to maintain, per virtual circuit, a credit—effectively the number of buffers reserved for that circuit. As long as each transmitted cell had a buffer waiting for it, congestion could never arise.

The argument against it came from the switch vendors. They did not want to do all the accounting to keep track of the credits and did not want to reserve so many buffers in advance. The amount of overhead and waste required was thought to be too much, so ultimately, the rate-based congestion control scheme was adopted. It works like this.

The basic model is that after every k data cells, each sender transmits a special **RM** (**Resource Management**) cell. This cell travels along the same path as the data cells, but is treated specially by the switches along the way. When it gets to the destination, it is examined, updated, and sent back to the sender. The full path for RM cells is shown in Fig. 5-75.

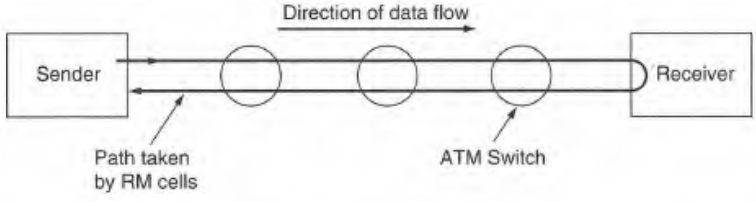


Fig. 5-75. The path taken by RM cells.

In addition, two other congestion control mechanisms are provided. First, overloaded switches can spontaneously generate RM cells and ship them back to the sender. Second, overloaded switches can set the middle *PTI* bit on data cells traveling from the sender to the receiver. Neither of these methods are fully reliable however, since these cells may be lost in the congestion without anyone noticing. In contrast, a lost RM cell will be noticed by the sender when it fails to

RED HAT 1057
RED HAT V. CAS

return within the expected time interval. As an aside, the *CLP* bit is not used for ABR congestion control.

ABR congestion control is based on the idea that each sender has a current rate, **ACR (Actual Cell Rate)** that falls between *MCR* and *PCR*. When congestion occurs, *ACR* is reduced (but not below *MCR*). When congestion is absent, *ACR* is increased (but not above *PCR*). Each RM cell sent contains the rate at which the sender would currently like to transmit, possibly *PCR*, possibly lower. This value is called **ER (Explicit Rate)**. As the RM cell passes through various switches on the way to the receiver, those that are congested may reduce *ER*. No switch may increase it. Reduction can occur either in the forward direction or in the reverse direction. When the sender gets the RM cell back, it can then see what the maximum acceptable rate is according to all the switches along the path. It can then adjust *ACR*, if need be, to bring it into line with what the slowest switch can handle.

The congestion mechanism using the middle *PTI* bit is integrated into the RM cells by having the receiver include this bit (taken from the last data cell) in each RM cell sent back. The bit cannot be taken from the RM cell itself because all RM cells have this bit set all the time, as shown in Fig. 5-63.

The ATM layer is quite complicated. In this chapter, we have highlighted only a portion of the issues. For additional information, see (De Prycker, 1993; McDysan and Spohn, 1995; Minoli and Vitella, 1994; and La Porta et al., 1994). However, the reader should be warned that all these references discuss the ATM 3 standard, not the ATM 4 standard, which was not finalized until 1996.

5.6.8. ATM LANs

As it becomes increasingly obvious that ITU's original goal of replacing the public switched telephone network by an ATM network is going to take a very long time, attention is shifting to the use of ATM technology to connect existing LANs together. In this approach, an ATM network can function either as a LAN, connecting individual hosts, or as a bridge, connecting multiple LANs. Although both concepts are interesting, they raise some challenging issues that we will discuss below. Additional information about ATM LANs can be found in (Chao et al., 1994; Newman, 1994; Truong et al., 1995).

The major problem that must be solved is how to provide connectionless LAN service over a connection-oriented ATM network. One possible solution is to introduce a connectionless server into the network. Every host initially sets up a connection to this server, and sends all packets to it for forwarding. While simple, this solution does not use the full bandwidth of the ATM network, and the connectionless server can easily become a bottleneck.

An alternative approach, proposed by the ATM Forum, is shown in Fig. 5-76. Here every host has a (potential) ATM virtual circuit to every other host. These virtual circuits can be established and released dynamically as needed, or they can

RED HAT 1057
RED HAT V. CAS

be permanent virtual circuits. To send a frame, the source host first encapsulates the packet in the payload field of an ATM AAL message and sends it to the destination, the same way frames are shipped over Ethernets, token rings, and other LANs.

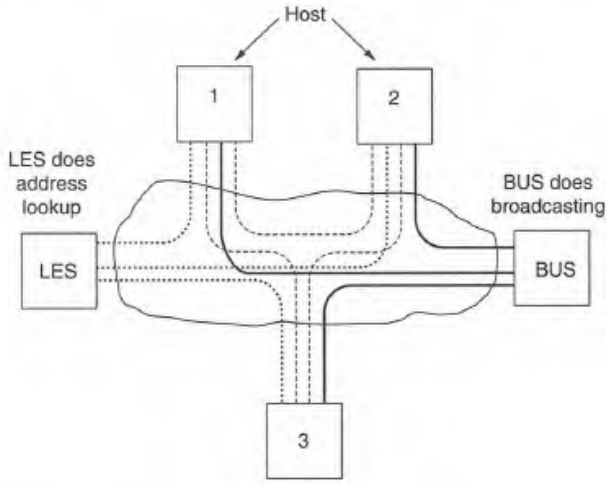


Fig. 5-76. ATM LAN emulation.

The main problem introduced by this scheme is how to tell which IP (or other network layer address) belongs to which virtual circuit. In an 802 LAN, this problem is solved by the ARP protocol, in which a host can broadcast a request such as: "Who has IP address 192.31.20.47?" The host using that address then sends back a point-to-point response, which is cached for later use.

With an ATM LAN, this solution does not work because ATM LANs do not support broadcasting. This problem is solved by introducing a new server, the **LES (LAN Emulation Server)**. To look up a network layer address (e.g., an IP address), a host sends a packet (e.g., an ARP request) to the LES, which then looks up the corresponding ATM address and returns it to the machine requesting it. This address can then be used to send encapsulated packets to the destination.

However, this solution only solves the host location problem. Some programs use broadcasting or multicasting as an essential part of the application. For these applications, the **BUS (Broadcast/Unknown Server)** is introduced. It has connections to all hosts and can simulate broadcasting by sending a packet to all of them, one at a time. Hosts can also speed up delivery of a packet to an unknown host by sending the packet to the BUS for broadcasting and then (in parallel) looking up the address (for future use) using the LES.

A model similar to this one has been adopted by the IETF as the official Internet way to use an ATM network for transporting IP packets. In this model the

RED HAT 1057
RED HAT V. CAS

LES server is called the **ATMARP** server, but the functionality is essentially the same. Broadcasting and multicasting are not supported in the IETF proposal. The model is described in RFC 1483 and RFC 1577. Another good source of information is (Comer, 1995).

In the IETF method, a set of ATM hosts can be grouped together to form a **logical IP subnet**. Each LIS has its own ATMARP server. In effect, a LIS acts like a virtual LAN. Hosts on the same LIS may exchange IP packets directly, but hosts on different ones are required to go through a router. The reason for having LISes is that every host on a LIS must (potentially) have an open virtual circuit to every other host on its LIS. By restricting the number of hosts per LIS, the number of open virtual circuits can be reduced to a manageable number.

Another use of ATM networks is to use them as bridges to connect existing LANs. In this configuration, only one machine on each LAN needs an ATM connection. Like all transparent bridges, the ATM bridge must listen promiscuously to all LANs to which it is attached, forwarding frames where needed. Since bridges use only MAC addresses (not IP addresses), ATM bridges must build a spanning tree, just as 802 bridges.

In short, while ATM LAN emulation is an interesting idea, there are serious questions about its performance and price, and there is certainly heavy competition from existing LANs and bridges, which are well established and highly optimized. Whether ATM LANs and bridges ever replace 802 LANs and bridges remains to be seen.

5.7. SUMMARY

The network layer provides services to the transport layer. It can be based on either virtual circuits or datagrams. In both cases, its main job is routing packets from the source to the destination. In virtual circuit subnets, a routing decision is made when the virtual circuit is set up. In datagram subnets, it is made on every packet.

Many routing algorithms are used in computer networks. Static algorithms include shortest path routing, flooding, and flow-based routing. Dynamic algorithms include distance vector routing and link state routing. Most actual networks use one of these. Other important routing topics are hierarchical routing, routing for mobile hosts, broadcast routing, and multicast routing.

Subnets can become congested, increasing the delay and lowering the throughput for packets. Network designers attempt to avoid congestion by proper design. Techniques include traffic shaping, flow specifications, and bandwidth reservation. If congestion does occur, it must be dealt with. Choke packets can be sent back, load can be shed, and other methods applied.

Networks differ in various ways, so when multiple networks are connected together problems can occur. Sometimes the problems can be finessed by

RED HAT 1057
RED HAT V. CAS

tunneling a packet through a hostile network, but if the source and destination networks are different, this approach fails. When different networks have different maximum packet sizes, fragmentation may be called for.

The Internet has a rich variety of protocols related to the network layer. These include the data transport protocol, IP, but also the control protocols ICMP, ARP, and RARP, and the routing protocols OSPF and BGP. The Internet is rapidly running out of IP addresses, so a new version of IP, IPv6, has been developed.

Unlike the datagram-based Internet, ATM networks use virtual circuits inside. These must be set up before data can be transferred and torn down after transmission is completed. Quality of service and congestion control are major issues with ATM networks.

PROBLEMS

1. Give two example applications for which connection-oriented service is appropriate. Now give two examples for which connectionless service is best.
2. Are there any circumstances when a virtual circuit service will (or at least should) deliver packets out of order? Explain.
3. Datagram subnets route each packet as a separate unit, independent of all others. Virtual circuit subnets do not have to do this, since each data packet follows a predetermined route. Does this observation mean that virtual circuit subnets do not need the capability to route isolated packets from an arbitrary source to an arbitrary destination? Explain your answer.
4. Give three examples of protocol parameters that might be negotiated when a connection is set up.
5. Consider the following design problem concerning implementation of virtual circuit service. If virtual circuits are used internal to the subnet, each data packet must have a 3-byte header, and each router must tie up 8 bytes of storage for circuit identification. If datagrams are used internally, 15-byte headers are needed, but no router table space is required. Transmission capacity costs 1 cent per 10^6 bytes, per hop. Router memory can be purchased for 1 cent per byte and is depreciated over two years (business hours only). The statistically average session runs for 1000 sec, in which time 200 packets are transmitted. The mean packet requires four hops. Which implementation is cheaper, and by how much?
6. 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?

RED HAT 1057
RED HAT V. CAS

7. 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.

8. Consider the subnet of Fig. 5-15(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 measured delays 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 expected delay.

9. If delays are recorded as 8-bit numbers in a 50-router network, and delay 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.

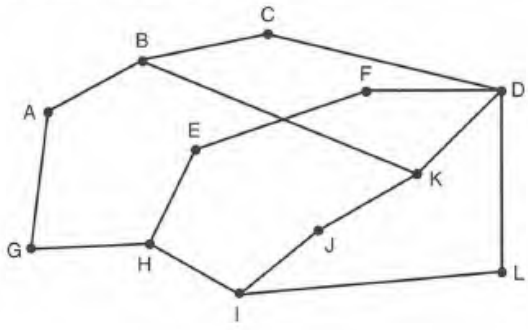
10. In Fig. 5-16 the Boolean OR of the two sets of ACF bits are 111 in every row. Is this just an accident here, or does it hold for all subnets under all circumstances?

11. 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?

12. 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. For an IP network on an 802.3 LAN, how does the home agent accomplish this interception?

13. Looking at the subnet of Fig. 5-5, how many packets are generated by a broadcast from *B*, using
    (a) reverse path forwarding?
    (b) the sink tree?

14. Compute a multicast spanning tree for router *C* in the subnet below for a group with members at routers *A*, *B*, *C*, *D*, *E*, *F*, *I*, and *K*.



15. As a possible congestion control mechanism in a subnet 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

RED HAT 1057
RED HAT V. CAS

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 subnet 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. Give an argument why the leaky bucket algorithm should allow just one packet per tick, independent of how large the packet is.

18. The byte-counting variant of the leaky bucket algorithm is used in a particular system. The rule is that one 1024-byte packet, two 512-byte packets, etc. may be sent on each tick. Give a serious restriction of this system that was not mentioned in the text.

19. An ATM network uses a token bucket scheme for traffic shaping. A new token is put into the bucket every 5 μsec. What is the maximum sustainable net data rate (i.e., excluding header bits)?

20. 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?

21. Figure 5-27 shows four input characteristics for a proposed flow specification. Imagine that the maximum packet size is 1000 bytes, the token bucket rate is 10 million bytes/sec, the token bucket size is 1 million bytes, and the maximum transmission rate is 50 million bytes/sec. How long can a burst at maximum speed last?

22. A device accepts frames from the Ethernet to which it is attached. It removes the packet inside each frame, adds framing information around it, and transmits it over a leased telephone line (its only connection to the outside world) to an identical device at the other end. This device removes the framing, inserts the packet into a token ring frame, and transmits it to a local host over a token ring LAN. What would you call the device?

23. Is fragmentation needed in concatenated virtual circuit internets, or only in datagram systems?

24. Tunneling through a concatenated virtual circuit subnet is straightforward: the multiprotocol router at one end just sets up a virtual circuit to the other end and passes packets through it. Can tunneling also be used in datagram subnets? If so, how?

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.

RED HAT 1057
RED HAT V. CAS

26. Suppose that instead of using 16 bits for the network part of a class B address, 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 class B network on the Internet has a subnet mask of 255.255.240.0. What is the maximum number of hosts per subnet?

29. 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?

30. ARP and RARP both map addresses from one space to another. In this respect, they are similar. However, their implementations are fundamentally different. In what major way do they differ?

31. Describe a way to do reassembly of IP fragments at the destination.

32. Most IP datagram reassembly algorithms have a timer to avoid having a lost fragment tie up reassembly buffers forever. Suppose a datagram is fragmented into four fragments. The first three fragments arrrive, 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?

33. Most IP routing protocols use number of hops as the metric to be minimized when doing routing computations. For ATM networks, number of hops is not terribly important. Why not? *Hint*: Take a look at Chap. 2. to see how ATM switches work. Do they use store-and-forward?

34. In both IP and ATM, the checksum covers only the header and not the data. Why do you suppose this design was chosen?

35. 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?

36. IPv6 uses 16-byte addresses. If a block of 1 million addresses is allocated every picosecond, how long will the addresses last?

37. The *Protocol* field used in the IPv4 header is not present in the fixed IPv6 header. Why not?

38. When the IPv6 protocol is introduced, does the ARP protocol have to be changed? If so, are the changes conceptual or technical?

39. In Chap. 1, we classified interactions between the network and the hosts using four classes of primitives: *request*, *indication*, *response*, and *confirm*. Classify the SETUP and CONNECT messages of Fig. 5-65 into these categories.

40. A new virtual circuit is being set up in an ATM network. Between the source and destination hosts lie three ATM switches. How many messages (including acknowledgements) will be sent to establish the circuit?

RED HAT 1057
RED HAT V. CAS

41. The logic used to construct the table of Fig. 5-67 is simple: the lowest unused *VPI* is always assigned to a connection. If a new virtual circuit is requested between NY and Denver, which *VPI* will be assigned to it?

42. In Fig. 5-73(c), if a cell arrives early, the next one is still due at $t_1 + 2T$. Suppose that the rule were different, namely that the next cell was expected at $t_2 + T$, and the sender made maximum use of this rule. What maximum peak cell rate could then be achieved? For $T = 10$ μsec and $L = 2$ μsec, give the original and new peak cell rates, respectively.

43. What is the maximum burst length on an 155.52 Mbps ATM ABR connection whose *PCR* value is 200,000 and whose L value is 25 μsec?

44. 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 delay and bandwidth used.

45. 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 timeouts 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 timeout interval, parametrized by error rate.

RED HAT 1057
RED HAT V. CAS