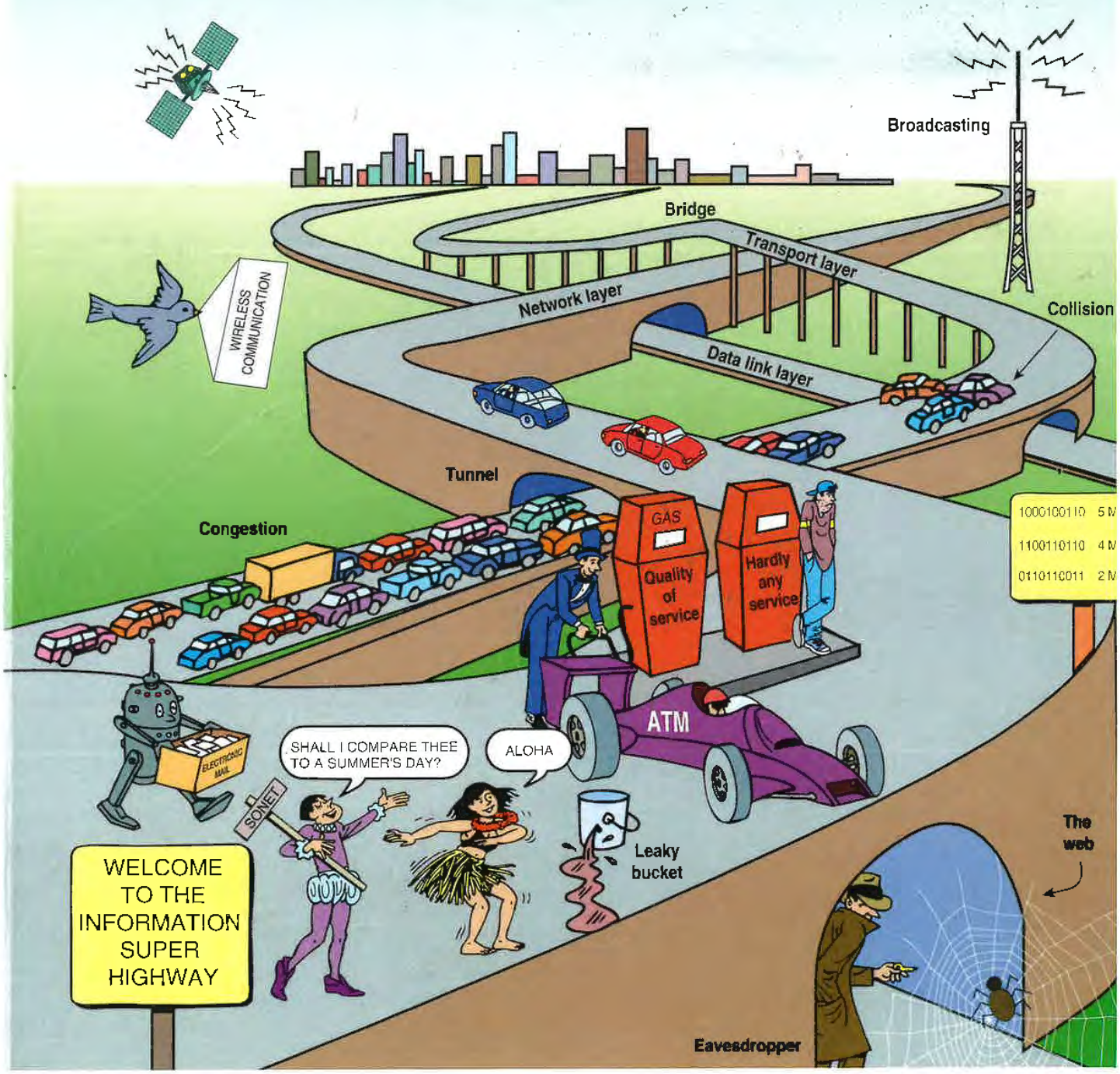
THIRD EDITION

COMPUTER NETWORKS

ANDREW S. TANENBAUM



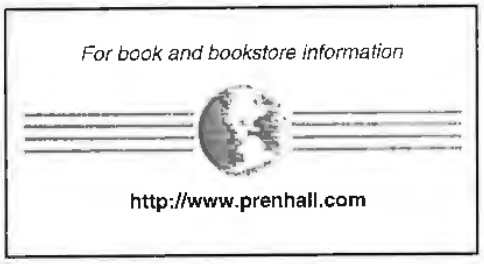
 GOOGLE EXHIBIT 1009

Computer Networks

Third Edition

Andrew S. Tanenbaum

Vrije Universiteit
Amsterdam, The Netherlands



Prentice Hall PTR
Upper Saddle River, New Jersey 07458

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 7 6

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*

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

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

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

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

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

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

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

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

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

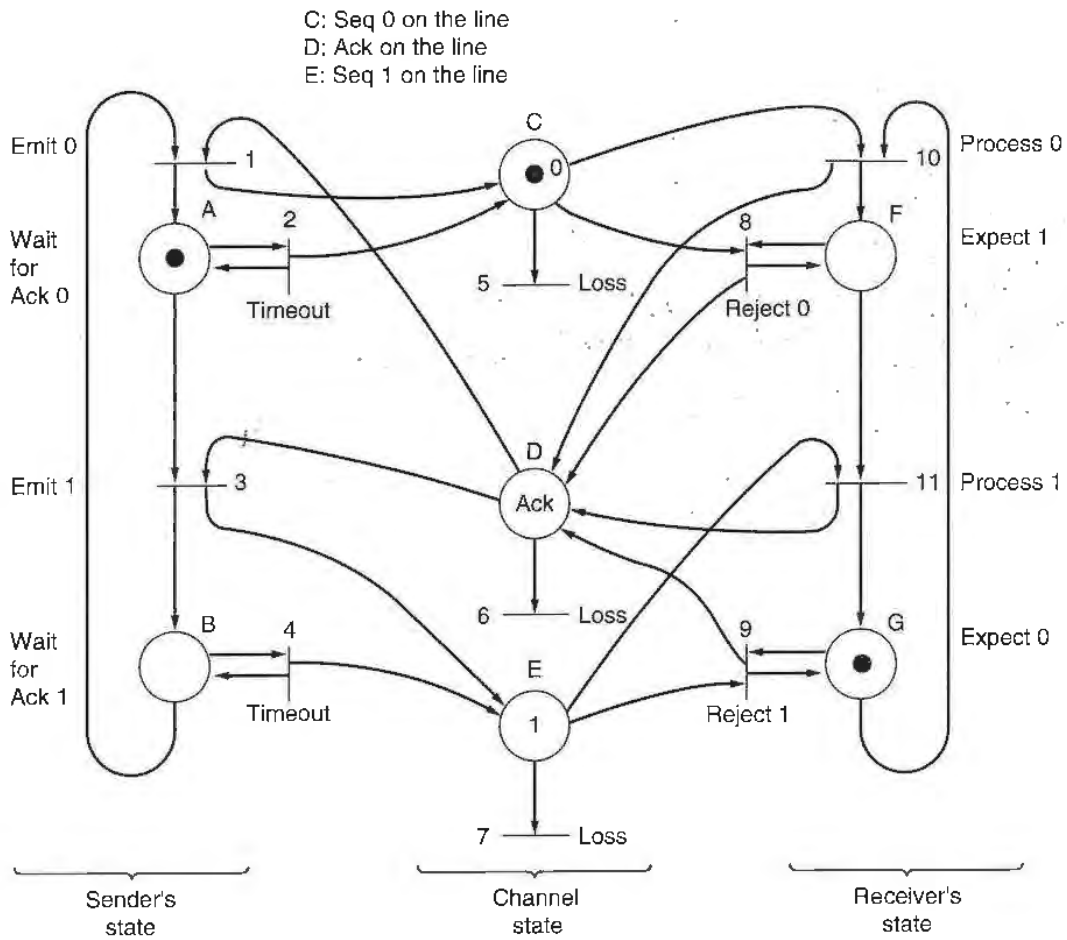


Fig. 3-23. A Petri net model for protocol 3.

3.6. EXAMPLE DATA LINK PROTOCOLS

In the following sections we will examine several widely-used data link protocols. The first one, HDLC, is common in X.25 and many other networks. After that, we will examine data link protocols used in the Internet and ATM networks, respectively. In subsequent chapters, we will also use the Internet and ATM as running examples as well.

3.6.1. HDLC—High-level Data Link Control

In this section we will examine a group of closely related protocols that are a bit old but are still heavily used in networks throughout the world. They are all derived from the data link protocol used in IBM's SNA, called **SDLC**

(**Synchronous Data Link Control**) protocol. After developing SDLC, IBM submitted it to ANSI and ISO for acceptance as U.S. and international standards, respectively. ANSI modified it to become **ADCCP** (**Advanced Data Communication Control Procedure**), and ISO modified it to become **HDLC** (**High-level Data Link Control**). CCITT then adopted and modified HDLC for its **LAP** (**Link Access Procedure**) as part of the X.25 network interface standard but later modified it again to **LAPB**, to make it more compatible with a later version of HDLC. The nice thing about standards is that you have so many to choose from. Furthermore, if you do not like any of them, you can just wait for next year's model.

All of these protocols are based on the same principles. All are bit-oriented, and all use bit stuffing for data transparency. They differ only in minor, but nevertheless irritating, ways. The discussion of bit-oriented protocols that follows is intended as a general introduction. For the specific details of any one protocol, please consult the appropriate definition.

All the bit-oriented protocols use the frame structure shown in Fig. 3-24. The *Address* field is primarily of importance on lines with multiple terminals, where it is used to identify one of the terminals. For point-to-point lines, it is sometimes used to distinguish commands from responses.

Bits	8	8	8	≥0	16	8
	01111110	Address	Control	Data	Checksum	01111110

Fig. 3-24. Frame format for bit-oriented protocols.

The *Control* field is used for sequence numbers, acknowledgements, and other purposes, as discussed below.

The *Data* field may contain arbitrary information. It may be arbitrarily long, although the efficiency of the checksum falls off with increasing frame length due to the greater probability of multiple burst errors.

The *Checksum* field is a minor variation on the well-known cyclic redundancy code, using CRC-CCITT as the generator polynomial. The variation is to allow lost flag bytes to be detected.

The frame is delimited with another flag sequence (01111110). On idle point-to-point lines, flag sequences are transmitted continuously. The minimum frame contains three fields and totals 32 bits, excluding the flags on either end.

There are three kinds of frames: **Information**, **Supervisory**, and **Unnumbered**. The contents of the *Control* field for these three kinds are shown in Fig. 3-25. The protocol uses a sliding window, with a 3-bit sequence number. Up to seven unacknowledged frames may be outstanding at any instant. The *Seq* field in Fig. 3-25(a) is the frame sequence number. The *Next* field is a piggybacked

acknowledgement. However, all the protocols adhere to the convention that instead of piggybacking the number of the last frame received correctly, they use the number of the first frame not received (i.e., the next frame expected). The choice of using the last frame received or the next frame expected is arbitrary; it does not matter which convention is used, provided that it is used consistently.

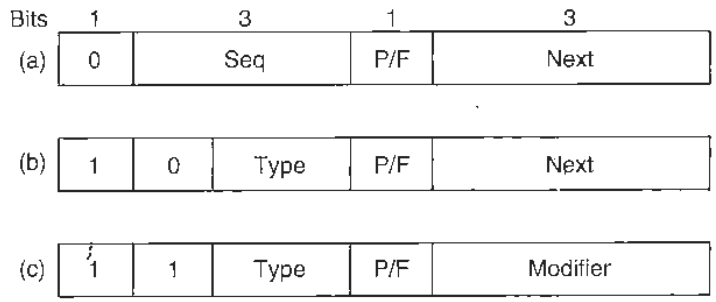


Fig. 3-25. Control field of (a) an information frame, (b) a supervisory frame, (c) an unnumbered frame.

The *P/F* bit stands for *Poll/Final*. It is used when a computer (or concentrator) is polling a group of terminals. When used as *P*, the computer is inviting the terminal to send data. All the frames sent by the terminal, except the final one, have the *P/F* bit set to *P*. The final one is set to *F*.

In some of the protocols, the *P/F* bit is used to force the other machine to send a Supervisory frame immediately rather than waiting for reverse traffic onto which to piggyback the window information. The bit also has some minor uses in connection with the Unnumbered frames.

The various kinds of Supervisory frames are distinguished by the *Type* field. Type 0 is an acknowledgement frame (officially called RECEIVE READY) used to indicate the next frame expected. This frame is used when there is no reverse traffic to use for piggybacking.

Type 1 is a negative acknowledgement frame (officially called REJECT). It is used to indicate that a transmission error has been detected. The *Next* field indicates the first frame in sequence not received correctly (i.e., the frame to be retransmitted). The sender is required to retransmit all outstanding frames starting at *Next*. This strategy is similar to our protocol 5 rather than our protocol 6.

Type 2 is RECEIVE NOT READY. It acknowledges all frames up to but not including *Next*, just as RECEIVE READY, but it tells the sender to stop sending. RECEIVE NOT READY is intended to signal certain temporary problems with the receiver, such as a shortage of buffers, and not as an alternative to the sliding window flow control. When the condition has been repaired, the receiver sends a RECEIVE READY, REJECT, or certain control frames.

Type 3 is the SELECTIVE REJECT. It calls for retransmission of only the frame specified. In this sense it is like our protocol 6 rather than 5 and is therefore most

useful when the sender's window size is half the sequence space size, or less. Thus if a receiver wishes to buffer out of sequence frames for potential future use, it can force the retransmission of any specific frame using Selective Reject. HDLC and ADCCP allow this frame type, but SDLC and LAPB do not allow it (i.e., there is no Selective Reject), and type 3 frames are undefined.

The third class of frame is the Unnumbered frame. It is sometimes used for control purposes but can also be used to carry data when unreliable connectionless service is called for. The various bit-oriented protocols differ considerably here, in contrast with the other two kinds, where they are nearly identical. Five bits are available to indicate the frame type, but not all 32 possibilities are used.

All the protocols provide a command, DISC (DISConnect), that allows a machine to announce that it is going down (e.g., for preventive maintenance). They also have a command that allows a machine that has just come back on-line to announce its presence and force all the sequence numbers back to zero. This command is called SNRM (Set Normal Response Mode). Unfortunately, "Normal Response Mode" is anything but normal. It is an unbalanced (i.e., asymmetric) mode in which one end of the line is the master and the other the slave. SNRM dates from a time when data communication meant a dumb terminal talking to a computer, which clearly is asymmetric. To make the protocol more suitable when the two partners are equals, HDLC and LAPB have an additional command, SABM (Set Asynchronous Balanced Mode), which resets the line and declares both parties to be equals. They also have commands SABME and SNRME, which are the same as SABM and SNRM, respectively, except that they enable an extended frame format that uses 7-bit sequence numbers instead of 3-bit sequence numbers.

A third command provided by all the protocols is FRMR (FRaMe Reject), used to indicate that a frame with a correct checksum but impossible semantics arrived. Examples of impossible semantics are a type 3 Supervisory frame in LAPB, a frame shorter than 32 bits, an illegal control frame, and an acknowledgement of a frame that was outside the window, etc. FRMR frames contain a 24-bit data field telling what was wrong with the frame. The data include the control field of the bad frame, the window parameters, and a collection of bits used to signal specific errors.

Control frames may be lost or damaged, just like data frames, so they must be acknowledged too. A special control frame is provided for this purpose, called UA (Unnumbered Acknowledgement). Since only one control frame may be outstanding, there is never any ambiguity about which control frame is being acknowledged.

The remaining control frames deal with initialization, polling, and status reporting. There is also a control frame that may contain arbitrary information, UI (Unnumbered Information). These data are not passed to the network layer but are for the receiving data link layer itself.

Despite its widespread use, HDLC is far from perfect. A discussion of a variety of problems associated with it can be found in (Fiorini et al., 1995).

3.6.2. The Data Link Layer in the Internet

The Internet consists of individual machines (hosts and routers), and the communication infrastructure that connects them. Within a single building, LANs are widely used for interconnection, but most of the wide area infrastructure is built up from point-to-point leased lines. In Chap. 4, we will look at LANs; here we will examine the data link protocols used on point-to-point lines in the Internet.

In practice, point-to-point communication is primarily used in two situations. First, thousands of organizations have one or more LANs, each with some number of hosts (personal computers, user workstations, servers, and so on) along with a router (or a bridge, which is functionally similar). Often, the routers are interconnected by a backbone LAN. Typically, all connections to the outside world go through one or two routers that have point-to-point leased lines to distant routers. It is these routers and their leased lines that make up the communication subnets on which the Internet is built.

The second situation where point-to-point lines play a major role in the Internet is the millions of individuals who have home connections to the Internet using modems and dial-up telephone lines. Usually, what happens is that the user's home PC calls up an **Internet provider**, which includes commercial companies like America Online, CompuServe, and the Microsoft Network, but also many universities and companies that provide home Internet connectivity to their students and employees. Sometimes the home PC just functions as a character-oriented terminal logged into the Internet service provider's timesharing system. In this mode, the user can type commands and run programs, but the graphical Internet services, such as the World Wide Web, are not available. This way of working is called having a **shell account**.

Alternatively, the home PC can call an Internet service provider's router and then act like a full-blown Internet host. This method of operation is no different than having a leased line between the PC and the router, except that the connection is terminated when the user ends the session. With this approach, all Internet services, including the graphical ones, become available. A home PC calling an Internet service provider is illustrated in Fig. 3-26.

For both the router-router leased line connection and the dial-up host-router connection, some point-to-point data link protocol is required on the line for framing, error control, and the other data link layer functions we have studied in this chapter. Two such protocols are widely used in the Internet, SLIP and PPP. We will now examine each of these in turn.

SLIP—Serial Line IP

SLIP is the older of the two protocols. It was devised by Rick Adams in 1984 to connect Sun workstations to the Internet over a dial-up line using a modem. The protocol, which is described in RFC 1055, is very simple. The workstation

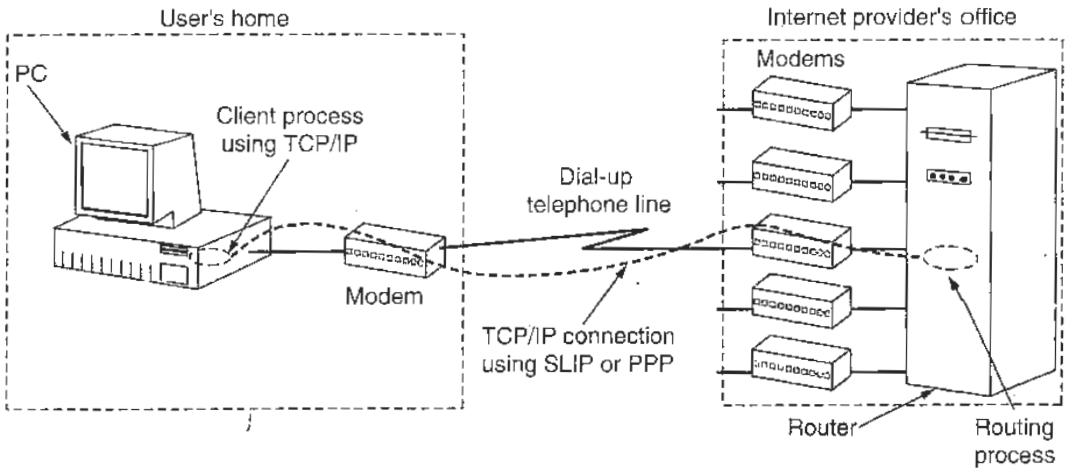


Fig. 3-26. A home personal computer acting as an Internet host.

just sends raw IP packets over the line, with a special flag byte (0xC0) at the end for framing. If the flag byte occurs inside the IP packet, a form of character stuffing is used, and the two byte sequence (0xDB, 0xDC) is sent in its place. If 0xDB occurs inside the IP packet, it, too, is stuffed. Some SLIP implementations attach a flag byte to both the front and back of each IP packet sent.

More recent versions of SLIP do some TCP and IP header compression. What they do is take advantage of the fact that consecutive packets often have many header fields in common. These are compressed by omitting those fields that are the same as the corresponding fields in the previous IP packet. Furthermore, the fields that do differ are not sent in their entirety, but as increments to the previous value. These optimizations are described in RFC 1144.

Although it is still widely used, SLIP has some serious problems. First, it does not do any error detection or correction, so it is up to higher layers to detect and recover from lost, damaged, or merged frames.

Second, SLIP supports only IP. With the growth of the Internet to encompass networks that do not use IP as their native language (e.g., Novell LANs), this restriction is becoming increasingly serious.

Third, each side must know the other's IP address in advance; neither address can be dynamically assigned during setup. Given the current shortage of IP addresses, this limitation is a major issue as it is impossible to give each home Internet user a unique IP address.

Fourth, SLIP does not provide any form of authenticatiou, so neither party knows whom it is really talking to. With leased lines, this is not an issue, but with dial-up lines it is.

Fifth, SLIP is uot an approved Internet Standard, so many different (and incompatible) versions exist. This situation does not make interworking easier.

PPP—Point-to-Point Protocol

To improve the situation, the IETF set up a group to devise a data link protocol for point-to-point lines that solved all these problems and that could become an official Internet Standard. This work culminated in **PPP (Point-to-Point Protocol)**, which is defined in RFC 1661 and further elaborated on in several other RFCs (e.g., RFCs 1662 and 1663). PPP handles error detection, supports multiple protocols, allows IP addresses to be negotiated at connection time, permits authentication, and has many other improvements over SLIP. While many Internet service providers still support both SLIP and PPP, the future clearly lies with PPP, not only for dial-up lines, but also for leased router-router lines.

PPP provides three things:

1. A framing method that unambiguously delineates the end of one frame and the start of the next one. The frame format also handles error detection.
2. A link control protocol for bringing lines up, testing them, negotiating options, and bringing them down again gracefully when they are no longer needed. This protocol is called **LCP (Link Control Protocol)**.
3. A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used. The method chosen is to have a different **NCP (Network Control Protocol)** for each network layer supported.

To see how these pieces fit together, let us consider the typical scenario of a home user calling up an Internet service provider to make a home PC a temporary Internet host. The PC first calls the provider's router via a modem. After the router's modem has answered the phone and established a physical connection, the PC sends the router a series of LCP packets in the payload field of one or more PPP frames. These packets, and their responses, select the PPP parameters to be used.

Once these have been agreed upou, a series of NCP packets are sent to configure the network layer. Typically, the PC wants to run a TCP/IP protocol stack, so it needs an IP address. There are not enough IP addresses to go around, so normally each Internet provider gets a block of them and then dynamically assigns one to each newly attached PC for the duration of its login session. If a provider owns n IP addresses, it can have up to n machines logged in simultaneously, but its total customer base may be many times that. The NCP for IP is used to do the IP address assignment.

At this point, the PC is now an Internet host and can send and receive IP packets, just as hardwired hosts can. When the user is finished, NCP is used to tear down the network layer connection and free up the IP address. Then LCP is used

to shut down the data link layer connection. Finally, the computer tells the modem to hang up the phone, releasing the physical layer connection.

The PPP frame format was chosen to closely resemble the HDLC frame format, since there was no reason to reinvent the wheel. The major difference between PPP and HDLC is that the former is character oriented rather than bit oriented. In particular, PPP, like, SLIP, uses character stuffing on dial-up modem lines, so all frames are an integral number of bytes. It is not possible to send a frame consisting of 30.25 bytes, as it is with HDLC. Not only can PPP frames be sent over dial-up telephone lines, but they can also be sent over SONET or true bit-oriented HDLC lines (e.g., for router-router connections). The PPP frame format is shown in Fig. 3-27.

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

Fig. 3-27. The PPP full frame format for unnumbered mode operation.

All PPP frames begin with the standard HDLC flag byte (01111110), which is character stuffed if it occurs within the payload field. Next comes the *Address* field, which is always set to the binary value 11111111 to indicate that all stations are to accept the frame. Using this value avoids the issue of having to assign data link addresses.

The *Address* field is followed by the *Control* field, the default value of which is 00000011. This value indicates an unnumbered frame. In other words, PPP does not provide reliable transmission using sequence numbers and acknowledgements as the default. In noisy environments, such as wireless networks, reliable transmission using numbered mode can be used. The exact details are defined in RFC 1663.

Since the *Address* and *Control* fields are always constant in the default configuration, LCP provides the necessary mechanism for the two parties to negotiate an option to just omit them altogether and save 2 bytes per frame.

The fourth PPP field is the *Protocol* field. Its job is to tell what kind of packet is in the *Payload* field. Codes are defined for LCP, NCP, IP, IPX, AppleTalk, and other protocols. Protocols starting with a 0 bit are network layer protocols such as IP, IPX, OSI CLNP, XNS. Those starting with a 1 bit are used to negotiate other protocols. These include LCP and a different NCP for each network layer protocol supported. The default size of the *Protocol* field is 2 bytes, but it can be negotiated down to 1 byte using LCP.

The *Payload* field is variable length, up to some negotiated maximum. If the length is not negotiated using LCP during line setup, a default length of 1500 bytes is used. Padding may follow the payload if need be.

After the *Payload* field comes the *Checksum* field, which is normally 2 bytes, but a 4-byte checksum can be negotiated.

In summary, PPP is a multiprotocol framing mechanism suitable for use over modems, HDLC bit-serial lines, SONET, and other physical layers. It supports error detection, option negotiation, header compression, and optionally, reliable transmission using HDLC framing.

Let us now turn from the PPP frame format to the way lines are brought up and down. The (simplified) diagram of Fig. 3-28 shows the phases that a line goes through when it is brought up, used, and taken down again. This sequence applies both to modem connections and to router-router connections.

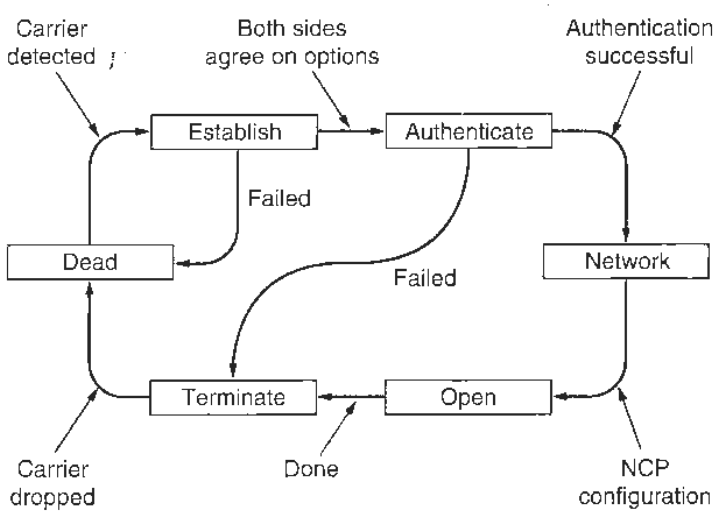


Fig. 3-28. A simplified phase diagram for bringing a line up and down.

When the line is *DEAD*, no physical layer carrier is present and no physical layer connection exists. After physical connection is established, the line moves to *ESTABLISH*. At that point LCP option negotiation begins, which, if successful, leads to *AUTHENTICATE*. Now the two parties can check on each other's identities, if desired. When the *NETWORK* phase is entered, the appropriate NCP protocol is invoked to configure the network layer. If the configuration is successful, *OPEN* is reached and data transport can take place. When data transport is finished, the line moves into the *TERMINATE* phase, and from there, back to *DEAD* when the carrier is dropped.

LCP is used to negotiate data link protocol options during the *ESTABLISH* phase. The LCP protocol is not actually concerned with the options themselves, but with the mechanism for negotiation. It provides a way for the initiating process to make a proposal and for the responding process to accept or reject it, in whole or in part. It also provides a way for the two processes to test the line

quality, to see if they consider it good enough to set up a connection. Finally, the LCP protocol also allows lines to be taken down when they are no longer needed.

Eleven types of LCP packets are defined in RFC 1661. These are listed in Fig. 3-29. The four *Configure-* types allow the initiator (I) to propose option values and the responder (R) to accept or reject them. In the latter case, the responder can make an alternative proposal or announce that it is not willing to negotiate certain options at all. The options being negotiated and their proposed values are part of the LCP packets.

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

Fig. 3-29. The LCP packet types.

The *Terminate-* codes are used to shut a line down when it is no longer needed. The *Code-reject* and *Protocol-reject* codes are used by the responder to indicate that it got something that it does not understand. This situation could mean that an undetected transmission error has occurred, but more likely it means that the initiator and responder are running different versions of the LCP protocol. The *Echo-* types are used to test the line quality. Finally, *Discard-request* is used for debugging. If either eud is having trouble getting bits onto the wire, the programmer can use this type for testing. If it manages to get through, the receiver just throws it away, rather than taking some other action, which might confuse the person doing the testing.

The options that can be negotiated include setting the maximum payload size for data frames, enabling authentication aud choosing a protocol to use, enabling line quality monitoring during normal operation, and selecting various header compression options.

There is little to say about the NCP protocols in a general way. Each one is specific to some network layer protocol and allows configuration requests to be

made that are specific to that protocol. For IP, for example, dynamic address assignment is the most important possibility.

3.6.3. The Data Link Layer in ATM

It is now time to begin our journey up through the ATM protocol layers of Fig. 1-30. The ATM physical layer covers roughly the OSI physical and data link layers, with the physical medium dependent sublayer being functionally like the OSI physical layer and the transmission convergence (TC) sublayer having data link functionality. There are no physical layer characteristics specific to ATM. Instead, ATM cells are carried by SONET, FDDI, and other transmission systems. Therefore we will concentrate here on the data link functionality of the TC sublayer, but we will discuss some aspects of the interface with the lower sublayer later on.

When an application program produces a message to be sent, that message works its way down the ATM protocol stack, having headers and trailers added and undergoing segmentation into cells. Eventually, the cells reach the TC sublayer for transmission. Let us see what happens to them on the way out the door.

Cell Transmission

The first step is header checksumming. Each cell contains a 5-byte header consisting of 4 bytes of virtual circuit and control information followed by a 1-byte checksum. Although the contents of the header are not relevant to the TC sublayer, curious readers wishing a sneak preview should turn to Fig. 5-62. The checksum only covers the first four header bytes, not the payload field. It consists of the remainder after the 32 header bits have been divided by the polynomial $x^8 + x^2 + x + 1$. To this the constant 01010101 is added, to provide robustness in the face of headers containing mostly 0 bits.

The decision to checksum only the header was made to reduce the probability of cells being delivered incorrectly due to a header error, but to avoid paying the price of checksumming the much larger payload field. It is up to higher layers to perform this function, if they so desire. For many real-time applications, such as voice and video, losing a few bits once in a while is acceptable (although for some compression schemes, all frames are equal but some frames are more equal). Because it covers only the header, the 8-bit checksum field is called the **HEC** (**Header Error Control**).

A factor that played a major role in this checksumming scheme is the fact that ATM was designed for use over fiber, and fiber is highly reliable. Furthermore, a major study of the U.S. telephone network has shown that during normal operation 99.64 percent of all errors on fiber optic lines are single-bit errors (AT&T and Bellcore, 1989). The HEC scheme corrects all single-bit errors and detects many

multibit errors as well. If we assume that the probability of a single-bit error is 10^{-8}, then the probability of a cell containing a detectable multibit header error is about 10^{-13}. The probability of a cell slipping through with an undetected header error is about 10^{-20}, which means that at OC-3 speed, one bad cell header will get through every 90,000 years. Although this may sound like a long time, once the earth has, say, 1 billion ATM telephones, each used 10 percent of the time, over 1000 bad cell headers per year will go undetected.

For applications that need reliable transmission in the data link layer, Shacham and McKenney (1990) have developed a scheme in which a sequence of consecutive cells are EXCLUSIVE ORed together. The result, an entire cell, is appended to the sequence. If one cell is lost or badly garbled, it can be reconstructed from the available information.

Once the HEC has been generated and inserted into the cell header, the cell is ready for transmission. Transmission media come in two categories: asynchronous and synchronous. When an asynchronous medium is used, a cell can be sent whenever it is ready to go. No timing restrictions exist.

With a synchronous medium, cells must be transmitted according to a predefined timing pattern. If no data cell is available when needed, the TC sublayer must invent one. These are called **idle cells**.

Another kind of nondata cell is the **OAM** (**Operation And Maintenance**) cell. OAM cells are also used by the ATM switches for exchanging control and other information necessary for keeping the system running. OAM cells also have some other special functions. For example, the 155.52-Mbps OC-3 speed matches the gross data rate of SONET, but an STM-1 frame has a total of 10 columns of overhead out of 270, so the SONET payload is only $260/270 \times 155.52$ Mbps or 149.76 Mbps. To keep from swamping SONET, an ATM source using SONET would normally put out an OAM cell as every 27th cell, to slow the data rate down to 26/27 of 155.52 Mbps and thus match SONET exactly. The job of matching the ATM output rate to the rate of the underlying transmission system is an important task of the TC sublayer.

On the receiver's side, idle cells are processed in the TC sublayer, but OAM cells are given to the ATM layer. OAM cells are distinguished from data cells by having the first three header bytes be all zeros, something not allowed for data cells. The fourth byte describes the nature of the OAM cell.

Another important task of the TC sublayer is generating the framing information for the underlying transmission system, if any. For example, an ATM video camera might just produce a sequence of cells on the wire, but it might also produce SONET frames with the ATM cells embedded inside the SONET payload. In the latter case, the TC sublayer would generate the SONET framing and pack the ATM cells inside, not entirely a trivial business since a SONET payload does not hold an integral number of 53-byte cells.

Although the telephone companies clearly intend to use SONET as the underlying transmission system for ATM, mappings from ATM onto the payload fields

of other systems have also been defined, and new ones are being worked on. In particular, mappings onto T1, T3, and FDDI also exist.

Cell Reception

On output, the job of the TC sublayer is to take a sequence of cells, add a HEC to each one, convert the result to a bit stream, and match the bit stream to the speed of the underlying physical transmission system by inserting OAM cells as filler. On input, the TC sublayer does exactly the reverse. It takes an incoming bit stream, locates the cell boundaries, verifies the headers (discarding cells with invalid headers), processes the OAM cells, and passes the data cells up to the ATM layer.

The hardest part is locating the cell boundaries in the incoming bit stream. At the bit level, a cell is just a sequence of $53 \times 8 = 424$ bits. No 01111110 flag bytes are present to mark the start and end of a cell, as they are in HDLC. In fact, there are no markers at all. How can cell boundaries be recognized under these circumstances?

In some cases, the underlying physical layer provides help. With SONET, for example, cells can be aligned with the synchronous payload envelope, so the SPE pointer in the SONET header points to the start of the first full cell. However, sometimes the physical layer provides no assistance in framing. What then?

The trick is to use the HEC. As the bits come in, the TC sublayer maintains a 40-bit shift register, with bits entering on the left and exiting on the right. The TC sublayer then inspects the 40 bits to see if it is potentially a valid cell header. If it is, the rightmost 8 bits will be valid HEC over the leftmost 32 bits. If this condition does not hold, the buffer does not hold a valid cell, in which case all the bits in the buffer are shifted right one bit, causing one bit to fall off the end, and a new input bit is inserted at the left end. This process is repeated until a valid HEC is located. At that point, the cell boundary is known because the shift register contains a valid header.

The trouble with this heuristic is that the HEC is only 8 bits wide. For any given shift register, even one containing random bits, the probability of finding a valid HEC is 1/256, a moderately large value. Used by itself, this procedure would incorrectly detect cell headers far too often.

To improve the accuracy of the recognition algorithm, the finite state machine of Fig. 3-30 is used. Three states are used: *HUNT*, *PRESYNCH*, and *SYNCH*. In the *HUNT* state, the TC sublayer is shifting bits into the shift registers one at a time looking for a valid HEC. As soon as one is found, the finite state machine switches to *PRESYNCH* state, meaning that it has tentatively located a cell boundary. It now shifts in the next 424 bits (53 bytes) without examining them. If its guess about the cell boundary was correct, the shift register should now contain another valid cell header, so it once again runs the HEC algorithm. If the HEC is

incorrect, the TC goes back to the *HUNT* state and continues to search bit-by-bit for a header whose HEC is correct.

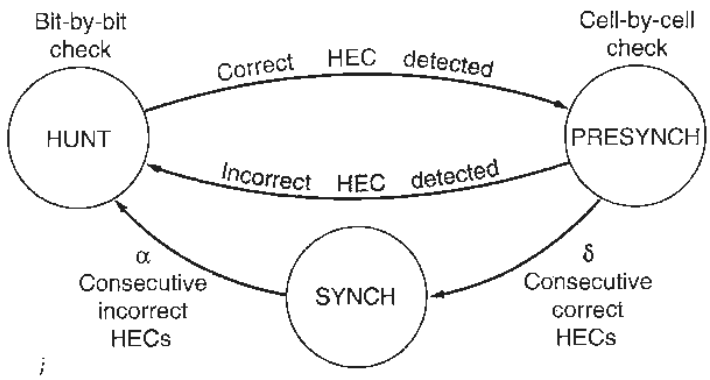


Fig. 3-30. The cell delineation heuristic.

On the other hand, if the second HEC is also correct, the TC may be onto something, so it shifts in another 424 bits and tries again. It continues inspecting headers in this fashion until it has found δ correct headers in a row, at which time it assumes that it is synchronized and moves into the *SYNCH* state to start normal operation. Note that the probability of getting into *SYNCH* state by accident with a purely random bit stream is $2^{-8\delta}$, which can be made arbitrarily small by choosing a large enough δ. The price paid for a large δ, however, is a longer time to synchronize.

In addition to resynchronizing after losing synchronization (or at startup), the TC sublayer needs a heuristic to determine when it has lost synchronization, for example after a bit has been inserted or deleted from the bit stream. It would be unwise to give up if just one HEC was incorrect, since most errors are bit inversions, not insertions or deletions. The wisest course here is just to discard the cell with the bad header and hope the next one is good. However, if α HECs in a row are bad, the TC sublayer has to conclude that it has lost synchronization and must retnrn to the *HUNT* state.

Although unlikely, it is conceivable that a malicious user could try to spoof the TC sublayer by inserting a data pattern into the payload field of many consecutive cells that imitates the HEC algorithm. Then, if synchronization were ever lost, it might be regained in the wrong place. To make this trick much harder, the payload bits are scrambled on transmission and descrambled on reception.

Before leaving the TC sublayer, one comment is in order. The mechanism chosen for cell delineation requires the TC sublayer to understand and use the header of the ATM layer above it. Having one layer make use of the header of a higher layer is in complete violation of the basic rules of protocol engineering. The idea of having layered protocols is to make each layer be independent of the

ones above it. It should be possible, for example, to change the header format of the ATM layer without affecting the TC sublayer. However due to the way cell delineation is accomplished, making such a change is not possible.

3.7. SUMMARY

The task of the data link layer is to convert the raw bit stream offered by the physical layer into a stream of frames for use by the network layer. Various framing methods are used, including character count, character stuffing, and bit stuffing. Data link protocols can provide error control to retransmit damaged or lost frames. To prevent a fast sender from overrunning a slow receiver, the data link protocol can also provide flow control. The sliding window mechanism is widely used to integrate error control and flow control in a convenient way.

Sliding window protocols can be categorized by the size of the sender's window and the size of the receiver's window. When both are equal to 1, the protocol is stop-and-wait. When the sender's window is greater than 1, for example to prevent the sender from blocking on a circuit with a long propagation delay, the receiver can be programmed either to discard all frames other than the next one in sequence (protocol 5) or buffer out of order frames until they are needed (protocol 6).

Protocols can be modeled using various techniques to help demonstrate their correctness (or lack thereof). Finite state machine models and Petri net models are commonly used for this purpose.

Many networks use one of the bit-oriented protocols—SDLC, HDLC, ADCCP, or LAPB—at the data link level. All of these protocols use flag bytes to delimit frames, and bit stuffing to prevent flag bytes from occurring in the data. All of them also use a sliding window for flow control. The Internet uses SLIP and PPP as data link protocols. ATM systems have their own simple protocol, which does a bare minimum of error checking and no flow control.

PROBLEMS

1. An upper layer message is split into 10 frames, each of which has an 80 percent chance of arriving undamaged. If no error control is done by the data link protocol, how many times must the message be sent on the average to get the entire thing through?

2. The following data fragment occurs in the middle of a data stream for which the character-stuffing algorithm described in the text is used: DLE, STX, A, DLE, B, DLE, ETX. What is the output after stuffing?

3. If the bit string 011110111110111111 0 is bit stuffed, what is the output string?

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 hytes. 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.

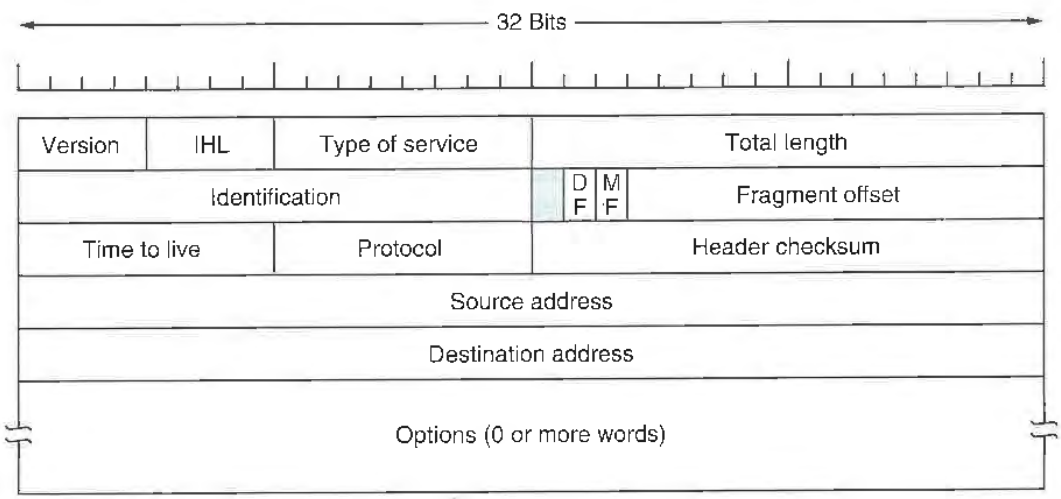


Fig. 5-45. The IP (Internet Protocol) header.

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

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.

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

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.

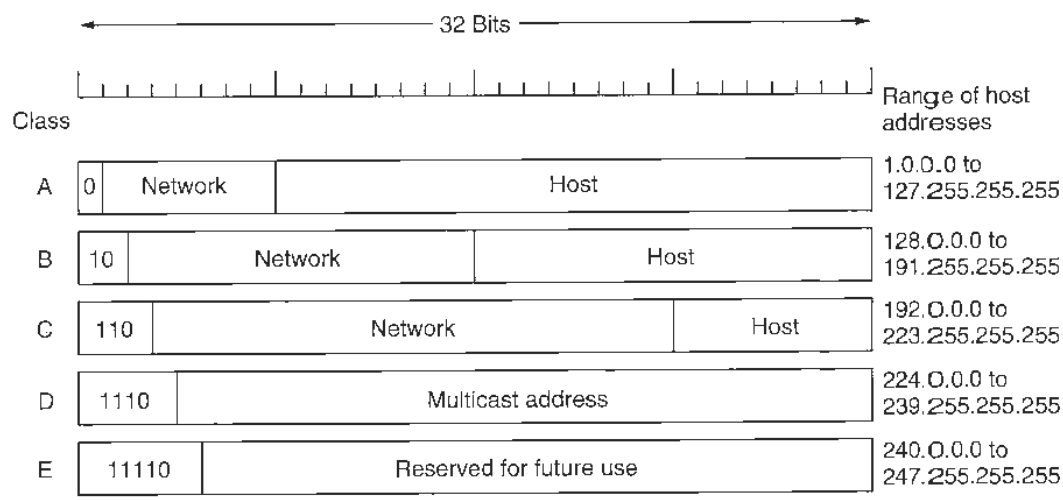


Fig. 5-47. IP address formats.

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.,

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.

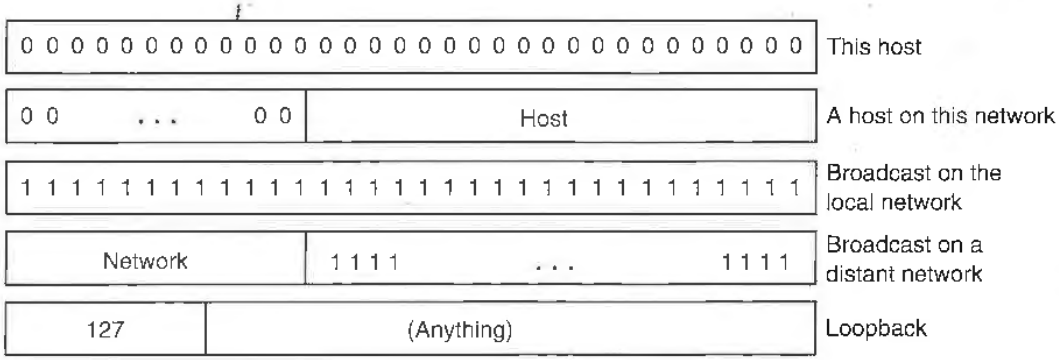


Fig. 5-48. Special IP addresses.

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

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 onr 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.

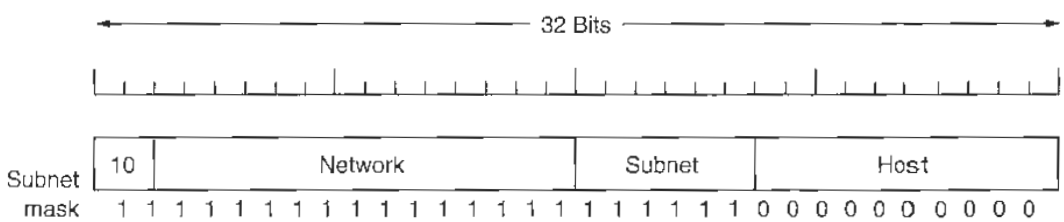


Fig. 5-49. One of the ways to subnet a class B network.

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

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 nsed 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 wben 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.

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 hy 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.

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.

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 instrnctions 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 donbt be lengthy and vigorous.

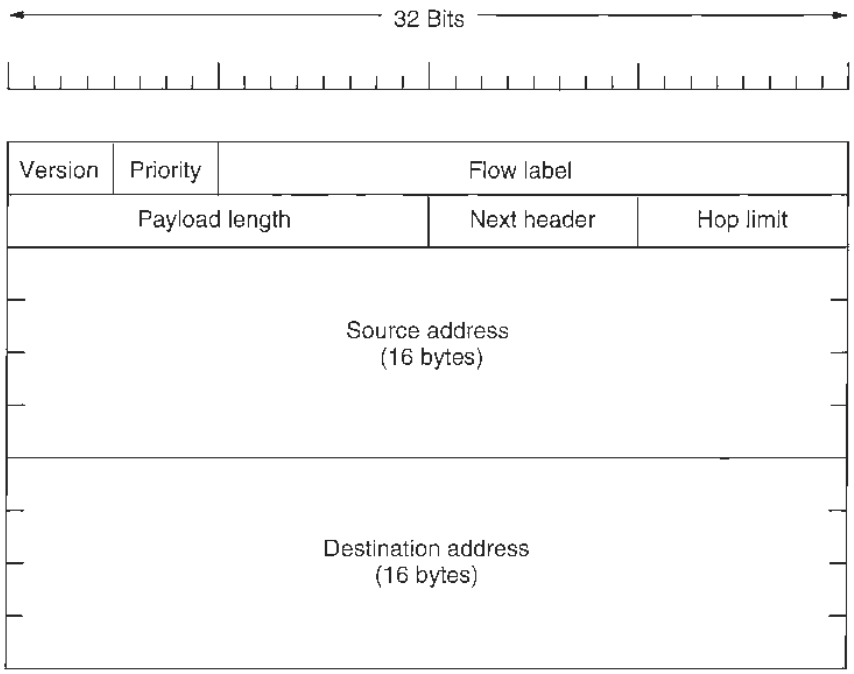


Fig. 5-56. The IPv6 fixed header (required).

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

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, distinguisbed by the next 16 bits. These variants relate to how IPv6 packets will be tunneled over the existing IPv4 infrastructure.

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

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 nse 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

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

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

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) aud loose routing (only selected routers arc supplied) are available, but they are combiued. The format of the routing header is shown in Fig. 5-60.

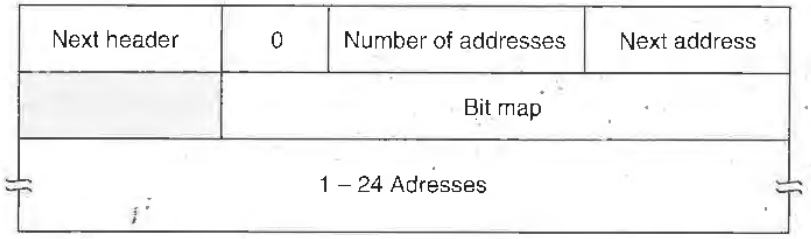


Fig. 5-60. The extension header for routing.

The first 4 bytes of the routing exteusion header coutaiu four 1-byte integers: the next header type, the routiug type (currently 0), the number of addresses present in this header (1 to 24), aud 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, aud a bit telling whether more fragments will follow. Iu IPv6, unlike iu 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 informatiou 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

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.

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 argumeuts 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 waudering around for a long time and 65,535 hops is far too long. Fiually, 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 abont 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 antichecksnm camp won this one, and IPv6 does not have a checksum.

Mobile hosts were also a point of contention. If a portable compnter 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

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, hut 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 hattle 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 uot 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 conntry 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 tbe IPv4 Internet to be turned off on a Sunday morning and come back up as an IPv6

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 he as painless as possihle.

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. (Altbough 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.

(Schneier, 1996). Chapter 8 of his email book does too (Schneier, 1995). Privacy and computers are also discussed in (Adam, 1995). These references are highly recommended for readers who wish to pursue their study of this subject.

7.2. DNS—Domain Name System

Programs rarely refer to hosts, mailboxes, and other resources by their binary network addresses. Instead of binary numbers, they use ASCII strings, such as *tana@art.ucsb.edu*. Nevertheless, the network itself only understands binary addresses, so some mechanism is required to convert the ASCII strings to network addresses. In the following sections we will study how this mapping is accomplished in the Internet.

Way back in the ARPANET, there was simply a file, *hosts.txt*, that listed all the hosts and their IP addresses. Every night, all the hosts would fetch it from the site at which it was maintained. For a network of a few hundred large timesharing machines, this approach worked reasonably well.

However, when thousands of workstations were connected to the net, everyone realized that this approach could not continue to work forever. For one thing, the size of the file would become too large. However, even more important, host name conflicts would occur constantly unless names were centrally managed, something unthinkable in a huge international network. To solve these problems, **DNS** (the **Domain Name System**) was invented.

The essence of DNS is the invention of a hierarchical, domain-based naming scheme and a distributed database system for implementing this naming scheme. It is primarily used for mapping host names and email destinations to IP addresses but can also be used for other purposes. DNS is defined in RFCs 1034 and 1035.

Very briefly, the way DNS is used is as follows. To map a name onto an IP address, an application program calls a library procedure called the **resolver**, passing it the name as a parameter. The resolver sends a UDP packet to a local DNS server, which then looks up the name and returns the IP address to the resolver, which then returns it to the caller. Armed with the IP address, the program can then establish a TCP connection with the destination, or send it UDP packets.

7.2.1. The DNS Name Space

Managing a large and constantly changing set of names is a nontrivial problem. In the postal system, name management is done by requiring letters to specify (implicitly or explicitly) the country, state or province, city, and street address of the addressee. By using this kind of hierarchical addressing, there is no confusion between the Marvin Anderson on Main St. in White Plains, N.Y. and the Marvin Anderson on Main St. in Austin, Texas. DNS works the same way.

Conceptually, the Internet is divided into several hundred top-level **domains**, where each domain covers many hosts. Each domain is partitioned into subdomains, and these are further partitioned, and so on. All these domains can be represented by a tree, as shown in Fig. 7-25. The leaves of the tree represent domains that have no subdomains (but do contain machines, of course) A leaf domain may contain a single host, or it may represent a company and contains thousands of hosts.

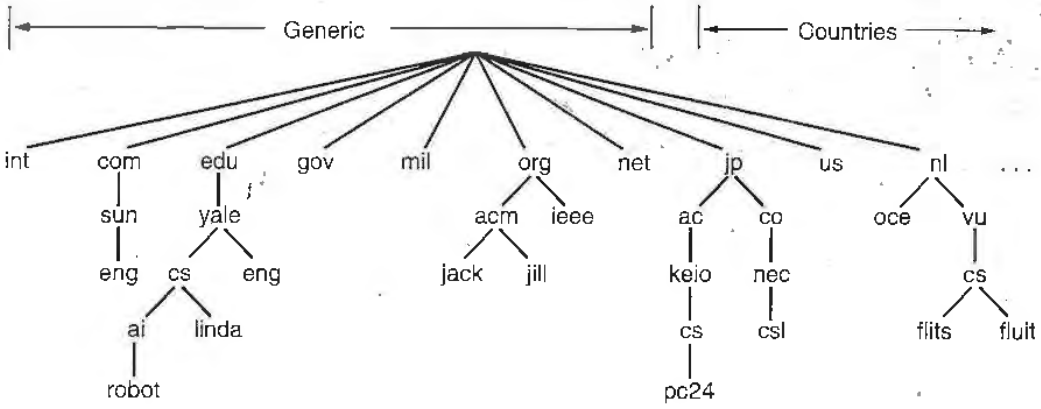


Fig. 7-25. A portion of the Internet domain name space.

The top-level domains come in two flavors: generic and countries. The generic domains are *com (commercial), edu* (educational institutions), *gov* (the U.S. federal government), *int* (certain international organizations), *mil* (the U.S. armed forces), *net* (network providers), and *org* (nonprofit organizations). The country domains include one entry for every country, as defined in ISO 3166.

Each domain is named by the path upward from it to the (unnamed) root. The components are separated by periods (pronounced "dot"). Thus Sun Microsystems engineering department might be *eng.sun.com.*, rather than a UNIX-style name such as */com/sun/eng*. Notice that this hierarchical naming means that *eng.sun.com.* does not conflict with a potential use of *eng* in *eng.yale.edu.*, which might be used by the Yale English department.

Domain names can be either absolute or relative. An absolute domain name ends with a period (e.g., *eng.sun.com.*), whereas a relative one does not. Relative names have to be interpreted in some context to uniquely determine their true meaning. In both cases, a named domain refers to a specific node in the tree and all the nodes under it.

Domain names are case insensitive, so *edu* and *EDU* mean the same thing. Component names can be up to 63 characters long, and full path names must not exceed 255 characters.

In principle, domains can be inserted into the tree in two different ways. For example, *cs.yale.edu* could equally well be listed under the *us* country domain as

cs.yale.ct.us. In practice, however, nearly all organizations in the United States are under a generic domain, and nearly all outside the United States are under the domain of their country. There is no rule against registering under two top-level domains, but doing so might be confusing, so few organizations do it.

Each domain controls how it allocates the domains under it. For example, Japan has domains *ac.jp* and *co.jp* that mirror *edu* and *com*. The Netherlands does not make this distinction and puts all organizations directly under *nl*. Thus all three of the following are university computer science departments:

1. *cs.yale.edu* (Yale University, in the United States)
2. *cs.vu.nl* (Vrije Universiteit, in The Netherlands)
3. *cs.keio.ac.jp* (Keio University, in Japan)

To create a new domain, permission is required of the domain in which it will be included. For example, if a VLSI group is started at Yale and wants to be known as *vlsi.cs.yale.edu*, it needs permission from whomever manages *cs.yale.edu*. Similarly, if a new university is chartered, say, the University of Northern South Dakota, it must ask the manager of the *edu* domain to assign it *unsd.edu*. In this way, name conflicts are avoided and each domain can keep track of all its subdomains. Once a new domain has been created and registered, it can create subdomains, such as *cs.unsd.edu*, without getting permission from anybody higher up the tree.

Naming follows organizational boundaries, not physical networks. For example, if the computer science and electrical engineering departments are located in the same building and share the same LAN, they can nevertheless have distinct domains. Similarly, even if computer science is split over Babbage Hall and Turing Hall, all the hosts in both buildings will normally belong to the same domain.

7.2.2. Resource Records

Every domain, whether it is a single host or a top-level domain, can have a set of **resource records** associated with it. For a single host, the most common resource record is just its IP address, but many other kinds of resource records also exist. When a resolver gives a domain name to DNS, what it gets back are the resource records associated with that name. Thus the real function of DNS is to map domain names onto resource records.

A resource record is a five-tuple. Although they are encoded in binary for efficiency, in most expositions resource records are presented as ASCII text, one line per resource record. The format we will use is as follows:

```
Domain_name   Time_to_live   Class   Type   Value
```

The *Domain_name* tells the domain to which this record applies. Normally, many records exist for each domain and each copy of the database holds information

about multiple domains. This field is thus the primary search key used to satisfy queries. The order of the records in the database is not significant. When a query is made about a domain, all the matching records of the class requested are returned.

The *Time_to_live* field gives an indication of how stable the record is. Information that is highly stable is assigned a large value, such as 86400 (the number of seconds in 1 day). Information that is highly volatile is assigned a small value, such as 60 (1 minute). We will come back to this point later when we have discussed caching.

The third field of every resource record is the *Class*. For Internet information, it is always *IN*. For nonInternet information, other codes can be used, but in practice, these are rarely seen.

The *Type* field tells what kind of record this is. The most important types are listed in Fig. 7-26.

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept email
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

Fig. 7-26. The principal DNS resource record types.

An *SOA* record provides the name of the primary source of information about the name server's zone (described below), the email address of its administrator, a unique serial number, and various flags and timeouts.

The most important record type is the *A* (Address) record. It holds a 32-bit IP address for some host. Every Internet host must have at least one IP address, so other machines can communicate with it. Some hosts have two or more network connections, in which case they will have one type *A* resource record per network connection (and thus per IP address).

The next most important record type is the *MX* record. It specifies the name of the host prepared to accept email for the specified domain. A common use of this record is to allow a machine that is not on the Internet to receive email from Internet sites. Delivery is accomplished by having the non-Internet site make an

arrangement with some Internet site to accept email for it and forward it using whatever protocol the two of them agree on.

For example, suppose that Cathy is a computer science graduate student at UCLA. After she gets her degree in AI, she sets up a company, Electrobrain Corporation, to commercialize her ideas. She cannot afford an Internet connection yet, so she makes an arrangement with UCLA to allow her to have her email sent there. A few times a day she will call up and collect it.

Next, she registers her company with the *com* domain and is assigned the domain *electrobrain.com*. She might then ask the administrator of the *com* domain to add an *MX* record to the *com* database as follows:

```
electrobrain.com   86400   IN   MX   1   mailserver.cs.ucla.edu
```

In this way, mail will be forwarded to UCLA where she can pick it up by logging in. Alternatively, UCLA could call her and transfer the email by any protocol they mutually agree on.

The *NS* records specify name servers. For example, every DNS database normally has an *NS* record for each of the top-level domains, so email can be sent to distant parts of the naming tree. We will come back to this point later.

CNAME records allow aliases to be created. For example, a person familiar with Internet naming in general wanting to send a message to someone whose login name is *paul* in the computer science department at M.I.T. might guess that *paul@cs.mit.edu* will work. Actually this address will not work, because the domain for M.I.T.'s computer science department is *lcs.mit.edu*. However, as a service to people who do not know this, M.I.T. could create a *CNAME* entry to point people and programs in the right direction. An entry like this one might do the job:

```
cs.mit.edu   86400   IN   CNAME   lcs.mit.edu
```

Like *CNAME*, *PTR* points to another name. However, unlike *CNAME*, which is really just a macro definition, *PTR* is a regular DNS datatype whose interpretation depends on the context in which it is found. In practice, it is nearly always used to associate a name with an IP address to allow lookups of the IP address and return the name of the corresponding machine.

HINFO records allow people to find out what kind of machine and operating system a domain corresponds to. Finally, *TXT* records allow domains to identify themselves in arbitrary ways. Both of these record types are for user convenience. Neither is required, so programs cannot count on getting them (and probably cannot deal with them if they do get them).

Finally, we have the *Value* field. This field can be a nnmber, a domain name, or an ASCII string. The semantics depend on the record type. A short description of the *Value* fields for each of the principal records types is given in Fig. 7-26.

As an example of the kind of information one might find in the DNS database of a domain, see Fig. 7-27. This figure depicts part of a (semihypothetical)

database for the *cs.vu.nl* domain shown in Fig. 7-25. The database contains seven types of resource records.

```
; Authoritative data for cs.vu.nl
cs.vu.nl.        86400  IN  SOA     star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.        86400  IN  TXT     "Faculteit Wiskunde en Informatica."
cs.vu.nl.        86400  IN  TXT     "Vrije Universiteit Amsterdam."
cs.vu.nl.        86400  IN  MX      1 zephyr.cs.vu.nl.
cs.vu.nl.        86400  IN  MX      2 top.cs.vu.nl.

flits.cs.vu.nl.  86400  IN  HINFO   Sun Unix
flits.cs.vu.nl.  86400  IN  A       130.37.16.112
flits.cs.vu.nl.  86400  IN  A       192.31.231.165
flits.cs.vu.nl.  86400  IN  MX      1 flits.cs.vu.nl.
flits.cs.vu.nl.  86400  IN  MX      2 zephyr.cs.vu.nl.
flits.cs.vu.nl.  86400  IN  MX      3 top.cs.vu.nl.
www.cs.vu.nl.    86400  IN  CNAME   star.cs.vu.nl
ftp.cs.vu.nl.    86400  IN  CNAME   zephyr.cs.vu.nl

rowboat                 IN  A       130.37.56.201
                        IN  MX      1 rowboat
                        IN  MX      2 zephyr
                        IN  HINFO   Sun Unix

little-sister           IN  A       130.37.62.23
                        IN  HINFO   Mac MacOS

laserjet                IN  A       192.31.231.216
                        IN  HINFO   "HP Laserjet IIISi" Proprietary
```

Fig. 7-27. A portion of a possible DNS database for *cs.vu.nl*

The first noncomment line of Fig. 7-27 gives some basic information about the domain, which will not concern us further. The next two lines give textual information about where the domain is located. Then come two entries giving the first and second places to try to deliver email sent to *person@cs.vu.nl*. The *zephyr* (a specific machine) should be tried first. If that fails, the *top* should be tried next.

After the blank line, added for readability, come lines telling that the *flits* is a Sun workstation running UNIX and giving both of its IP addresses. Then three choices are given for handling email sent to *flits.cs.vu.nl*. First choice is naturally the *flits* itself, but if it is down, the *zephyr* and *top* are the second and third choices. Next comes an alias, *www.cs.vu.nl*, so that this address can be used without designating a specific machine. Creating this alias allows *cs.vu.nl* to change its World Wide Web server without invalidating the address people use to get to it. A similar argument holds for *ftp.cs.vu.nl*.

The next four lines contain a typical entry for a workstation, in this case, *rowboat.cs.vu.nl*. The information provided contains the IP address, the primary and secondary mail drops, and information about the machine. Then comes an entry for a non-UNIX system that is not capable of receiving mail itself, followed by an entry for a laser printer.

What is not shown (and is not in this file), are the IP addresses to use to look up the top level domains. These are needed to look up distant hosts, but since they are not part of the *cs.vu.nl* domain, they are not in this file. They are supplied by the root servers, whose IP addresses are present in a system configuration file and loaded into the DNS cache when the DNS server is booted. They have very long timeouts, so once loaded, they are never purged from the cache.

7.2.3. Name Servers

In theory at least, a single name server could contain the entire DNS database and respond to all queries about it. In practice, this server would be so overloaded as to be useless. Furthermore, if it ever went down, the entire Internet would be crippled.

To avoid the problems associated with having only a single source of information, the DNS name space is divided up into nonoverlapping **zones**. One possible way to divide up the name space of Fig. 7-25 is shown in Fig. 7-28. Each zone contains some part of the tree and also contains name servers holding the authoritative information about that zone. Normally, a zone will have one primary name server, which gets its information from a file on its disk, and one or more secondary name servers, which get their information from the primary name server. To improve reliability, some servers for a zone can be located outside the zone.

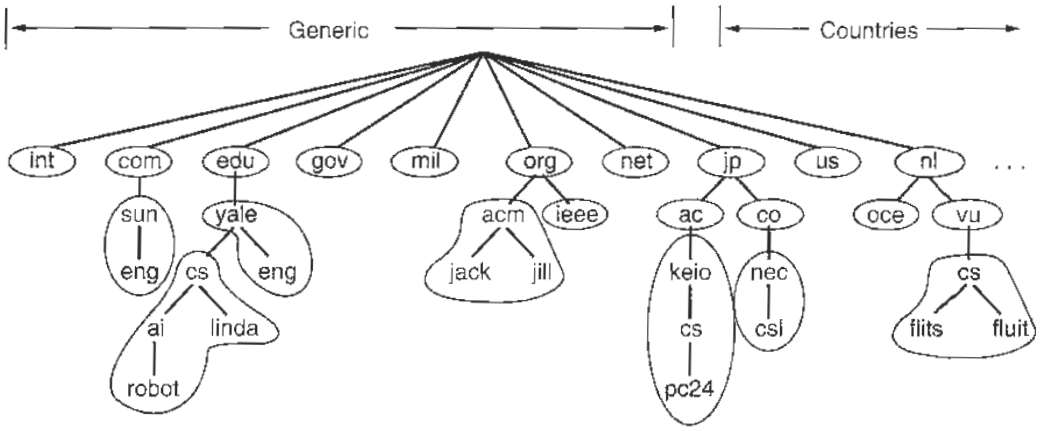


Fig. 7-28. Part of the DNS name space showing the division into zones.

Where the zone boundaries are placed within a zone is up to that zone's administrator. This decision is made in large part based on how many name

servers are desired, and where. For example, in Fig. 7-28, Yale has a server for *yale.edu* that handles *eng.yale.edu* but not *cs.yale.edu*, which is a separate zone with its own name servers. Such a decision might be made when a department such as English does not wish to run its own name server, but a department such as computer science does. Consequently, *cs.yale.edu* is a separate zone but *eng.yale.edu* is not.

When a resolver has a query about a domain name, it passes the query to one of the local name servers. If the domain being sought falls under the jurisdiction of the name server, such as *ai.cs.yale.edu* falling under *cs.yale.edu*, it returns the authoritative resource records. An **authoritative record** is one that comes from the authority that manages the record, and is thus always correct. Authoritative records are in contrast to cached records, which may be out of date.

If, however, the domain is remote and no information about the requested domain is available locally, the name server sends a query message to the top-level name server for the domain requested. To make this process clearer, consider the example of Fig. 7-29. Here, a resolver on *flits.cs.vu.nl* wants to know the IP address of the host *linda.cs.yale.edu*. In step 1, it seuds a query to the local name server, *cs.vu.nl*. This query contains the domain name sought, the type (*A*) and the class (*IN*).

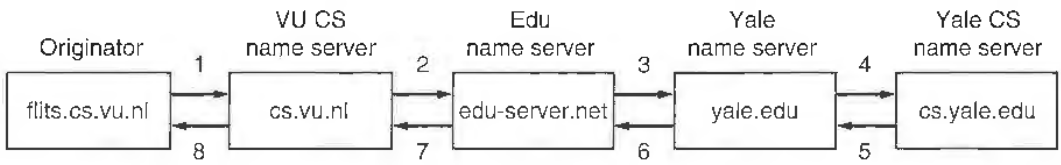


Fig. 7-29. How a resolver looks up a remote name in eight steps.

Let us suppose the local name server has never had a query for this domain before and knows nothing about it. It may ask a few other nearby name servers, but if none of them know, it sends a UDP packet to the server for *edu* given in its database (see Fig. 7-29), *edu-server.net*. It is unlikely that this server knows the address of *linda.cs.yale.edu*, and probably does not know *cs.yale.edu* either, but it must know all of its own children, so it forwards the request to the name server for *yale.edu* (step 3). In turn, this one forwards the request to *cs.yale.edu* (step 4), which must have the authoritative resource records. Since each request is from a client to a server, the resource record requested works its way back in steps 5 through 8.

Once these records get back to the *cs.vu.nl* name server, they will be entered into a cache there, in case they are needed later. However, this information is not authoritative, since changes made at *cs.yale.edu* will uot be propagated to all the caches in the world that may know about it. For this reason, cache entries should not live too long. This is the reason that the *Time_to_live* field is included in each resource record. It tells remote name servers how long to cache records. If a

certain machine has had the same IP address for years, it may be safe to cache that information for 1 day. For more volatile information, it might be safer to purge the records after a few seconds or a minute.

It is worth mentioning that the query method described here is known as a **recursive query**, since each server that does not have the requested information goes and finds it somewhere, then reports back. An alternative form is also possible. In this form, when a query cannot be satisfied locally, the query fails, but the name of the next server along the line to try is returned. This procedure gives the client more control over the search process. Some servers do uot implement recursive queries and always return the name of the next server to try.

It is also worth pointing out that when a DNS clieut fails to get a response before its timer goes off, it normally will try another server next time. The assumption here is that the server is probably down, rather than the request or reply got lost.

7.3. SNMP—SIMPLE NETWORK MANAGEMENT PROTOCOL

In the early days of the ARPANET, if the delay to some host became unexpectedly large, the person detecting the problem would just run the Ping program to bounce a packet off the destination. By looking at the timestamps in the header of the packet returned, the location of the problem could usually be pinpointed and some appropriate action taken. In addition, the number of routers was so small, that it was feasible to ping each one to see if it was sick.

When the ARPANET turned into the worldwide Internet, with multiple backbones and multiple operators, this solution ceased to be adequate, so better tools for network management were needed. Two early attempts were defined in RFC 1028 and RFC 1067, but these were short lived. In May 1990, RFC 1157 was published, defining version 1 of **SNMP (Simple Network Management Protocol)**. Along with a companion document (RFC 1155) on management iuformation, SNMP provided a systematic way of monitoring and mauaging a computer network. This framework and protocol were widely implemented in commercial products and became the de facto standards for network management.

As experience was gained, shortcomings in SNMP came to light, so an enhanced version of SNMP (SNMPv2) was defined (in RFCs 1441 to 1452) and started along the road to become an Internet standard. In the sections to follow, we will give a brief discussion of the SNMP (meaning SNMPv2) model and protocol.

Although SNMP was designed with the idea of its being simple, at least one author has managed to produce a 600-page book on it (Stallings, 1993a). For more compact descriptions (450-550 pages), see the books by Rose (1994) and Rose and McCloghrie (1995), both of whom were among the designers of SNMP. Other references are (Feit, 1995; and Hein and Griffiths, 1995).