

Computer Networking

A Top-Down Approach Featuring the Internet



James F. Kurose ♦ Keith W. Ross

Computer Networking

A Top-Down Approach Featuring the Internet

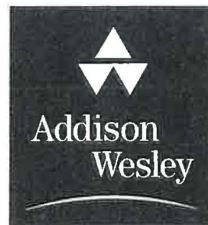
James F. Kurose

University of Massachusetts, Amherst



Keith W. Ross

Institute Eurécom



Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

Senior Acquisitions Editor	Susan Hartman
Assistant Editor	Lisa Kalner
Production Supervisor	Patty Mahtani
Art Editor	Helen Reebenacker
Executive Marketing Manager	Michael Hirsch
Composition	Pre-Press Company, Inc.
Technical Art	PD & PS
Copyeditor	Roberta Lewis
Proofreader	Holly McLean Aldis
Cover Design	Joyce Cosentino
Interior Design	Delgado Design
Design Manager	Regina Hagen
Cover Image	© 1999 PhotoDisc, Inc.

Access the latest information about Addison-Wesley titles from our World Wide Web site:
<http://www.awl.com/cs>

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs or applications.

Library of Congress Cataloging-in-Publication Data

Ross, Keith W., 1956-

Computer networking: a top-down approach featuring the Internet / Keith W. Ross,
 James F. Kurose.

p. cm.

Includes bibliographic references and index

ISBN 0-201-47711-4

1. Internet (Computer network) I. Kurose, James F.

TK5105.875.I57 R689 2001

004.6—dc21

00-025295

Copyright © 2001 by Addison Wesley Longman, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

1 2 3 4 5 6 7 8 9 10-MA-020100

Table of Contents

Chapter 1	Computer Networks and the Internet	1
1.1	What Is the Internet?	1
1.1.1	A Nuts and Bolts Description	1
1.1.2	A Service Description	4
1.1.3	Some Good Hyperlinks	5
1.2	What Is a Protocol?	6
1.2.1	A Human Analogy	7
1.2.2	Network Protocols	8
1.3	The Network Edge	9
1.3.1	End Systems, Clients, and Servers	9
1.3.2	Connectionless and Connection-Oriented Services	11
1.4	The Network Core	13
1.4.1	Circuit Switching, Packet Switching, and Message Switching	13
1.4.2	Routing in Data Networks	24
1.5	Access Networks and Physical Media	29
1.5.1	Access Networks	29
1.5.2	Physical Media	34
1.6	Delay and Loss in Packet-Switched Networks	38
1.6.1	Types of Delay	38
1.7	Protocol Layers and Their Service Models	44
1.7.1	Layered Architecture	45
1.7.2	The Internet Protocol Stack	49
1.7.3	Network Entities and Layers	53
1.8	Internet Backbones, NAPs, and ISPs	53
1.9	A Brief History of Computer Networking and the Internet	56
1.9.1	Development and Demonstration of Early Packet Switching Principles: 1961–1972	56
1.9.2	Internetworking, and New and Proprietary Networks: 1972–1980	57
1.9.3	A Proliferation of Networks: 1980–1990	60
1.9.4	Commercialization and the Web: The 1990s	61

1.10	Summary	62
	Homework Problems and Questions	64
	Problems	66
	Discussion Questions	68
	Interview: Leonard Kleinrock	70

Chapter 2 Application Layer 72

2.1	Principles of Application Layer Protocols	72
2.1.1	Application-Layer Protocols	73
2.1.2	What Services Does an Application Need?	78
2.1.3	Services Provided by the Internet Transport Protocols	80
2.1.4	Network Applications Covered in this Book	83
2.2	The World Wide Web: HTTP	84
2.2.1	Overview of HTTP	85
2.2.2	Nonpersistent and Persistent Connections	87
2.2.3	HTTP Message Format	90
2.2.4	User-Server Interaction: Authentication and Cookies	94
2.2.5	The Conditional GET	96
2.2.6	Web Caches	97
2.3	File Transfer: FTP	104
2.3.1	FTP Commands and Replies	105
2.4	Electronic Mail in the Internet	106
2.4.1	SMTP	109
2.4.2	Comparison with HTTP	111
2.4.3	Mail Message Formats and MIME	112
2.4.4	Mail Access Protocols	118
2.4.5	Continuous Media E-mail	123
2.5	DNS—The Internet's Directory Service	124
2.5.1	Services Provided by DNS	124
2.5.2	Overview of How DNS Works	127
2.5.3	DNS Records	132
2.5.4	DNS Messages	134
2.6	Socket Programming with TCP	136
2.6.1	Socket Programming with TCP	137
2.6.2	An Example Client/Server Application in Java	139
2.7	Socket Programming with UDP	146
2.8	Building a Simple Web Server	154
2.8.1	Web Server Functions	154
2.9	Summary	158
	Homework Problems and Questions	159
	Problems	161

Discussion Questions	162
Programming Assignments	163
Interview: Tim Berners-Lee	165

Chapter 3 Transport Layer 167

3.1	Transport-Layer Services and Principles	167
3.1.1	Relationship between Transport and Network Layers	169
3.1.2	Overview of the Transport Layer in the Internet	171
3.2	Multiplexing and Demultiplexing Applications	172
3.3	Connectionless Transport: UDP	177
3.3.1	UDP Segment Structure	180
3.3.2	UDP Checksum	181
3.4	Principles of Reliable Data Transfer	182
3.4.1	Building a Reliable Data Transfer Protocol	184
3.4.2	Pipelined Reliable Data Transfer Protocols	193
3.4.3	Go-Back-N (GBN)	196
3.4.4	Selective Repeat (SR)	201
3.5	Connection-Oriented Transport: TCP	207
3.5.1	The TCP Connection	207
3.5.2	TCP Segment Structure	210
3.5.3	Sequence Numbers and Acknowledgment Numbers	211
3.5.4	Telnet: A Case Study for Sequence and Acknowledgment Numbers	213
3.5.5	Reliable Data Transfer	215
3.5.6	Flow Control	221
3.5.7	Round Trip Time and Timeout	224
3.5.8	TCP Connection Management	226
3.6	Principles of Congestion Control	231
3.6.1	The Causes and the Costs of Congestion	231
3.6.2	Approaches toward Congestion Control	237
3.6.3	ATM ABR Congestion Control	239
3.7	TCP Congestion Control	240
3.7.1	Overview of TCP Congestion Control	241
3.7.2	Modeling Latency: Static Congestion Window	249
3.7.3	Modeling Latency: Dynamic Congestion Window	253
3.8	Summary	258
	Homework Problems and Questions	260
	Problems	261
	Discussion Question	268
	Programming Assignment	268
	Interview: Sally Floyd	269

Chapter 4	Network Layer and Routing	271
4.1	Introduction and Network Service Models	271
4.1.1	Network Service Model	273
4.1.2	Origins of Datagram and Virtual Circuit Service	279
4.2	Routing Principles	280
4.2.1	A Link State Routing Algorithm	282
4.2.2	A Distance Vector Routing Algorithm	286
4.2.3	Other Routing Algorithms	295
4.3	Hierarchical Routing	297
4.4	Internet Protocol	300
4.4.1	IPv1 Addressing	302
4.4.2	Transporting a Datagram from Source to Destination: Addressing and Routing	310
4.4.3	Datagram Format	314
4.4.4	IP Fragmentation and Reassembly	317
4.4.5	ICMP: Internet Control Message Protocol	319
4.5	Routing in the Internet	321
4.5.1	Intra-Autonomous System Routing in the Internet	322
4.5.2	Inter-Autonomous System Routing	329
4.6	What's Inside a Router?	332
4.6.1	Input Ports	333
4.6.2	Switching Fabrics	336
4.6.3	Output Ports	338
4.6.4	Where Does Queuing Occur?	338
4.7	IPv6	341
4.7.1	IPv6 Packet Format	342
4.7.2	Transitioning from IPv4 to IPv6	345
4.8	Multicast Routing	348
4.8.1	Introduction: The Internet Multicast Abstraction and Multicast Groups	348
4.8.2	The IGMP Protocol	350
4.8.3	Multicast Routing: The General Case	355
4.8.4	Multicast Routing in the Internet	362
4.9	Summary	367
	Homework Problems and Questions	368
	Problems	370
	Discussion Questions	374
	Programming Assignment	375
	Interview: José Joaquín García-Luna-Aceves	377

Chapter 5	Link Layer and Local Area Networks	379
5.1	The Data Link Layer: Introduction, Services	379
5.1.1	The Services Provided by the Link Layer	380
5.1.2	Adapters Communicating	383
5.2	Error Detection and Correction Techniques	385
5.2.1	Parity Checks	386
5.2.2	Checksumming Methods	389
5.2.3	Cyclic Redundancy Check (CRC)	389
5.3	Multiple Access Protocols and LANs	391
5.3.1	Channel Partitioning Protocols	394
5.3.2	Random Access Protocols	398
5.3.3	Taking-Turns Protocols	406
5.3.4	Local Area Networks (LANs)	407
5.4	LAN Addresses and ARP	409
5.4.1	LAN Addresses	409
5.4.2	Address Resolution Protocol	411
5.5	Ethernet	415
5.5.1	Ethernet Basics	417
5.5.2	CSMA/CD: Ethernet's Multiple Access Protocol	421
5.5.3	Ethernet Technologies	423
5.6	Hubs, Bridges, and Switches	427
5.6.1	Hubs	427
5.6.2	Bridges	429
5.6.3	Switches	437
5.7	IEEE 802.11 LANs	441
5.7.1	802.11 LAN Architecture	441
5.7.2	802.11 Media Access Protocols	442
5.8	PPP: The Point-to-Point Protocol	447
5.8.1	PPP Data Framing	449
5.8.2	PPP Link Control Protocol (LCP) and Network Control Protocols	451
5.9	Asynchronous Transfer Mode (ATM)	453
5.9.1	Principle Characteristics of ATM	454
5.9.2	ATM Physical Layer	456
5.9.3	ATM Layer	458
5.9.4	ATM Adaptation Layer	459
5.9.5	IP Over ATM	461
5.9.6	ARP and ATM	464
5.10	X.25 and Frame Relay	465
5.10.1	A Few Words About X.25	466
5.10.2	Frame Relay	467

5.11	Summary	471
	Homework Problems and Questions	472
	Problems	474
	Discussion Questions	480
	Interview: Robert Metcalfe	481

Chapter 6 Multimedia Networking 483

6.1	Multimedia Networking Applications	484
6.1.1	Examples of Multimedia Applications	484
6.1.2	Hurdles for Multimedia in Today's Internet	487
6.1.3	How Should the Internet Evolve to Better Support Multimedia?	488
6.1.4	Audio and Video Compression	489
6.2	Streaming Stored Audio and Video	491
6.2.1	Accessing Audio and Video from a Web Server	493
6.2.2	Sending Multimedia from a Streaming Server to a Helper Application	495
6.2.3	Real Time Streaming Protocol (RTSP)	497
6.3	Making the Best of the Best-Effort Service: An Internet Phone Example	501
6.3.1	The Limitations of a Best-Effort Service	501
6.3.2	Removing Jitter at the Receiver for Audio	503
6.3.3	Recovering from Packet Loss	506
6.3.4	Streaming Stored Audio and Video	510
6.4	RTP	510
6.4.1	RTP Basics	510
6.4.2	RTP Packet Header Fields	513
6.4.3	RTP Control Protocol (RTCP)	514
6.4.4	H.323	517
6.5	Beyond Best-Effort	522
6.5.1	Scenario 1: A 1 Mbps Audio Application and an FTP Transfer	523
6.5.2	Scenario 2: A 1 Mbps Audio Application and a High Priority FTP Transfer	524
6.5.3	Scenario 3: A Misbehaving Audio Application and an FTP Transfer	525
6.5.4	Scenario 4: Two 1 Mbps Audio Applications Over an Overloaded 1.5 Mbps Link	526
6.6	Scheduling and Policing Mechanisms	528
6.6.1	Scheduling Mechanisms	528
6.6.2	Policing: The Leaky Bucket	533
6.7	Integrated Services	536
6.7.1	Guaranteed Quality of Service	538
6.7.2	Controlled-Load Network Service	539

6.8	RSVP	539
6.8.1	The Essence of RSVP	540
6.8.2	A Few Simple Examples	542
6.8.3	Path Messages	544
6.8.4	Reservation Styles	544
6.8.5	Transport of Reservation Messages	548
6.9	Differentiated Services	549
6.9.1	Differentiated Services: A Simple Scenario	550
6.9.2	Traffic Classification and Conditioning	552
6.9.3	Per-Hop Behavior	554
6.9.4	A Beginning	556
6.10	Summary	556
	Homework Problems and Questions	558
	Problems	559
	Discussion Questions	561
	Interview: Henning Schulzrinne	563

Chapter 7 Security in Computer Networks 565

7.1	What is Network Security?	565
7.1.1	Secure Communication	565
7.1.2	Network Security Considerations in the Internet	567
7.2	Principles of Cryptography	569
7.2.1	Symmetric Key Cryptography	571
7.2.2	Public Key Encryption	575
7.3	Authentication: Who are You?	581
7.3.1	Authentication Protocol <i>ap1.0</i>	581
7.3.2	Authentication Protocol <i>ap2.0</i>	582
7.3.3	Authentication Protocol <i>ap3.0</i>	583
7.3.4	Authentication Protocol <i>ap3.1</i>	583
7.3.5	Authentication Protocol <i>ap4.0</i>	584
7.3.6	Authentication Protocol <i>ap5.0</i>	586
7.4	Integrity	588
7.4.1	Generating Digital Signatures	589
7.4.2	Message Digests	590
7.4.3	Hash Function Algorithms	593
7.5	Key Distribution and Certification	595
7.5.1	The Key Distribution Center	595
7.5.2	Public Key Certification	598
7.6	Secure E-Mail	602
7.6.1	Principles of Secure E-Mail	603
7.6.2	PGP	605

7.7	Internet Commerce	608
7.7.1	Internet Commerce Using SSL	609
7.7.2	Internet Commerce Using SET	613
7.8	Network Layer Security: IPsec	616
7.8.1	Authentication Header (AH) Protocol	617
7.8.2	The ESP Protocol	618
7.8.3	SA and Key Management	619
7.9	Summary	620
	Homework Problems and Questions	621
	Problems	622
	Discussion Questions	623
	Interview: Philip Zimmermann	624

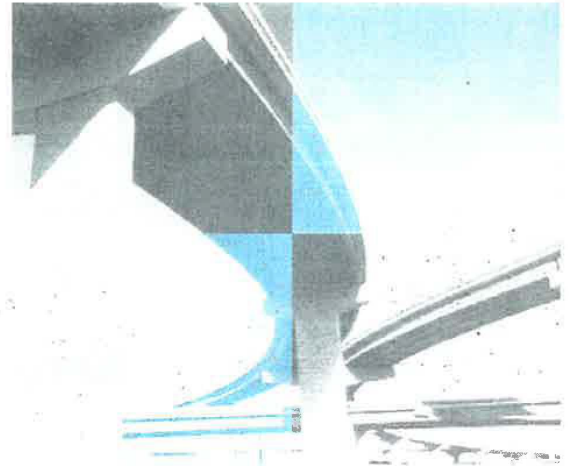
Chapter 8 Network Management 626

8.1	What Is Network Management?	626
8.2	The Infrastructure for Network Management	631
8.3	The Internet Network-Management Framework	634
8.3.1	Structure of Management Information: SMI	635
8.3.2	Management Information Base: MIB	638
8.3.3	SNMP Protocol Operations and Transport Mappings	641
8.3.4	Security and Administration	644
8.4	ASN.1	648
8.5	Firewalls	653
8.5.1	Packet Filtering	654
8.5.2	Application Gateways	655
8.6	Summary	657
	Homework Problems and Questions	658
	Problems	659
	Discussion Questions	659
	Interview: Jeff Case	660

References	663
Index	689

Chapter 1

Computer Networks and the Internet



1.1 ♦ What Is the Internet?

In this book we use the public Internet, a specific computer network (and one which probably most readers have used), as our principle vehicle for discussing computer networking protocols. But what is the Internet? We would like to be able to give you a one-sentence definition of the Internet, a definition that you can take home and share with your family and friends. Alas, the Internet is very complex, both in terms of its hardware and software components, as well as in the services it provides.

1.1.1 A Nuts and Bolts Description

Instead of giving a one-sentence definition, let's try a more descriptive approach. There are a couple of ways to do this. One way is to describe the nuts and bolts of the Internet, that is, the basic hardware and software components that make up the Internet. Another way is to describe the Internet in terms of a networking infrastructure that provides services to distributed applications. Let's begin with the nuts-and-bolts description, using Figure 1.1 to illustrate our discussion.

The public Internet is a world-wide **computer network**, that is, a network that interconnects millions of computing devices throughout the world. Most of these computing devices are traditional desktop PCs, Unix-based workstations, and so

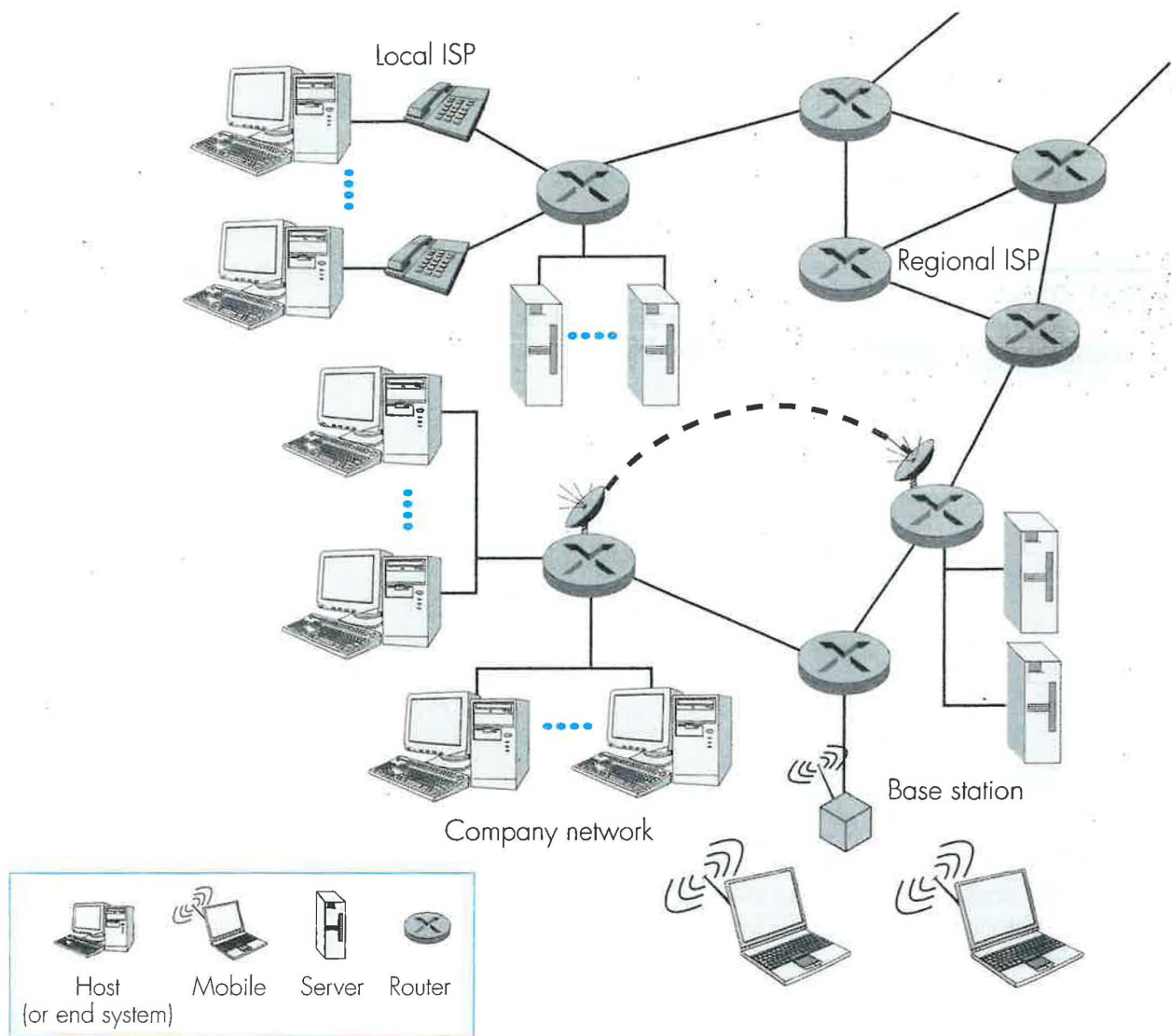


Figure 1.1 ♦ Some pieces of the Internet

called servers that store and transmit information such as Web (WWW) pages and e-mail messages. Increasingly, nontraditional computing devices such as Web TVs, mobile computers, pagers, and toasters are being connected to the Internet. (Toasters are not the only rather unusual devices to have been hooked up to the Internet; see the *The Future of the Living Room* [Greenberg 1997].) In the Internet jargon, all of these devices are called **hosts** or **end systems**. The Internet applications with

which many of us are familiar, such as the Web and e-mail, are **network application programs** that run on such end systems. We will look into Internet end systems in more detail in Section 1.3 and then delve deeply into the study of network applications in Chapter 2.

End systems, as well as most other “pieces” of the Internet, run **protocols** that control the sending and receiving of information within the Internet. **TCP** (the Transmission Control Protocol) and **IP** (the Internet Protocol) are two of the most important protocols in the Internet. The Internet’s principal protocols are collectively known as **TCP/IP**. We begin looking into protocols in Section 1.2. But that’s just a start—much of this book is concerned with computer network protocols!

End systems are connected together by **communication links**. We’ll see in Section 1.5 that there are many types of communication links. Links are made up of different types of **physical media**, including coaxial cable, copper wire, fiber optics, and radio spectrum. Different links can transmit data at different rates. The link transmission rate is often called the **link bandwidth** and is typically measured in bits/second.

Usually, end systems are not directly attached to each other via a single communication link. Instead, they are indirectly connected to each other through intermediate switching devices known as **routers**. A router takes information arriving on one of its incoming communication links and then forwards that information on one of its outgoing communication links. The **IP protocol** specifies the format of the information that is sent and received among routers and end systems. The path that transmitted information takes from the sending end system, through a series of communications links and routers, to the receiving end system is known as a **route** or **path** through the network. We introduce routing in more detail in Section 1.4 and study the algorithms used to determine routes, as well as the internal structure of a router itself, in Chapter 4.

Rather than provide a *dedicated* path between communicating end systems, the Internet uses a technique known as **packet switching** that allows multiple communicating end systems to share a path, or parts of a path, at the same time. The earliest ancestors of the Internet were the first packet-switched networks.

The Internet is really a network of networks. That is, the Internet is an interconnected set of privately and publicly owned and managed networks. Any network connected to the Internet must run the IP protocol and conform to certain naming and addressing conventions. Other than these few constraints, however, a network operator can configure and run its network (that is, its little piece of the Internet) however it chooses. Because of the universal use of the IP protocol in the Internet, the IP protocol is sometimes referred to as the **Internet dial tone**.

The topology of the Internet, that is, the structure of the interconnection among the various pieces of the Internet, is loosely hierarchical. Roughly speaking, from bottom-to-top, the hierarchy consists of end systems connected to local **Internet service providers (ISPs)** through **access networks**. An access network may be a so-called local area network within a company or university, a dial telephone line

with a modem, or a high-speed cable-based or phone-based access network. Local ISPs are in turn connected to regional ISPs, which are in turn connected to national and international ISPs. The national and international ISPs are connected together at the highest tier in the hierarchy. New tiers and branches (that is, new networks, and new networks of networks) can be added just as a new piece of Lego can be attached to an existing Lego construction. In the first half of 1996, approximately 40,000 *new* networks were added to the Internet [Network 1996]—an astounding growth rate.

At the technical and developmental level, the Internet is made possible through creation, testing, and implementation of **Internet standards**. These standards are developed by the Internet Engineering Task Force (IETF). The IETF standards documents are called **RFCs** (request for comments). RFCs started out as general request for comments (hence the name) to resolve architecture problems that faced the precursor to the Internet. RFCs, though not formally standards, have evolved to the point where they are cited as such. RFCs tend to be quite technical and detailed. They define protocols such as TCP, IP, HTTP (for the Web), and SMTP (for open-standards e-mail). There are more than 2,000 different RFCs.

The public Internet (that is, the global network of networks discussed above) is the network that one typically refers to as *the* Internet. There are also many private networks, such as certain corporate and government networks, whose hosts are not accessible from (that is, they cannot exchange messages with) hosts outside of that private network. These private networks are often referred to as **intranets**, as they often use the same Internet technology (for example, the same types of host, routers, links, protocols, and standards) as the public Internet.

1.1.2 A Service Description

The preceding discussion has identified many of the pieces that make up the Internet. Let's now leave the nuts-and-bolts description and take a more abstract, service-oriented view:

- ◆ The Internet allows **distributed applications** running on its end systems to exchange data with each other. These applications include remote login, file transfer, electronic mail, audio and video streaming, real-time audio and video conferencing, distributed games, the World Wide Web, and much, much more [AT&T Apps 1998]. It is worth emphasizing that the Web is not a separate network but rather just one of many distributed applications that use the communication services provided by the Internet. The Web *could* also run over a network besides the Internet. One reason that the Internet is the communication medium of choice for the Web, however, is that no other existing packet-switched network connects more than 43 million [Network 1999] computers together and has over 100 million users [Almanac 1998]. (By the way, determining the number of computers hooked up to the Internet is a very difficult task, as no one is responsible for maintaining a list of who's connected. When a new network is added to the Internet, its administrators do not need to report which end systems are con-

```
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24
Content-Type: image/gif
```

```
(data data data data data ...)
```

The client displays the object to the user but also saves the object in its local cache. Importantly, the client also caches the last-modified date along with the object. Third, one week later, the user requests the same object and the object is still in the cache. Since this object may have been modified at the Web server in the past week, the browser performs an up-to-date check by issuing a conditional GET. Specifically, the browser sends

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
If-modified-since: Mon, 22 Jun 1998 09:23:24
```

Note that the value of the `If-modified-since:` header line is exactly equal to the value of the `Last-Modified:` header line that was sent by the server one week ago. This conditional GET is telling the server to send the object only if the object has been modified since the specified date. Suppose the object has not been modified since 22 Jun 1998 09:23:24. Then, fourth, the Web server sends a response message to the client:

```
HTTP/1.0 304 Not Modified
Date: Wed, 19 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
```

(empty entity body)

We see that in response to the conditional GET, the Web server still sends a response message, but does not include the requested object in the response message. Including the requested object would only waste bandwidth and increase user-perceived response time, particularly if the object is large. Note that this last response message has in the status line `304 Not Modified`, which tells the client that it can go ahead and use its cached copy of the object.

2.2.6 Web Caches

A **Web cache**—also called a **proxy server**—is a network entity that satisfies HTTP requests on the behalf of a client. The Web cache has its own disk storage and keeps in this storage copies of recently requested objects. As shown in Figure 2.9, users

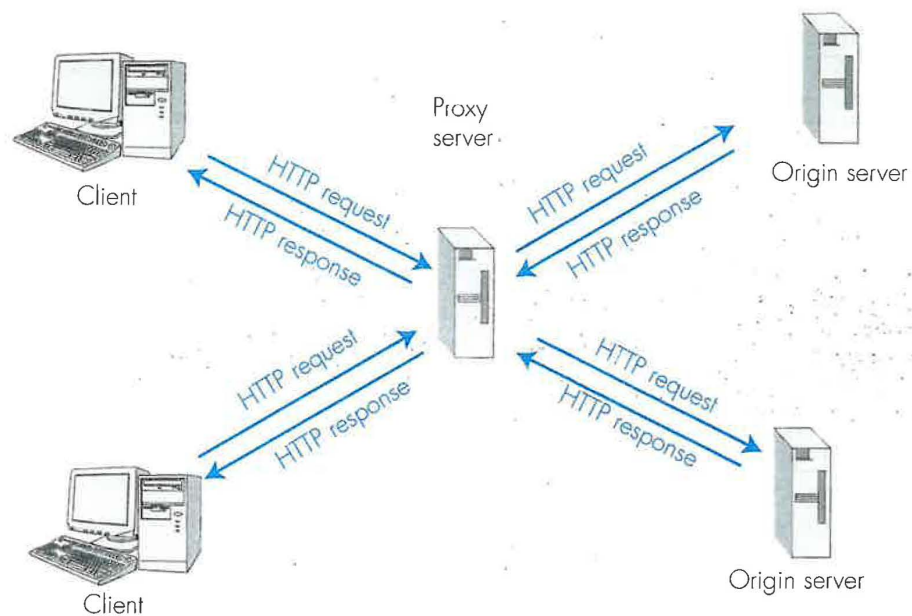


Figure 2.9 ♦ Clients requesting objects through a Web cache

configure their browsers so that all of their HTTP requests are first directed to the Web cache. (This is a straightforward procedure with Microsoft and Netscape browsers.) Once a browser is configured, each browser request for an object is first directed to the Web cache. As an example, suppose a browser is requesting the object `http://www.someschool.edu/campus.gif`.

- ♦ The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
- ♦ The Web cache checks to see if it has a copy of the object stored locally. If it does, the Web cache forwards the object within an HTTP response message to the client browser.
- ♦ If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that is, to `www.someschool.edu`. The Web cache then sends an HTTP request for the object into the TCP connection. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.
- ♦ When the Web cache receives the object, it stores a copy in its local storage and forwards a copy, within an HTTP response message, to the client browser (over the existing TCP connection between the client browser and the Web cache).

Note that a cache is both a server and a client at the same time. When it receives requests from and sends responses to a browser, it is a server. When it sends requests to and receives responses from an origin server it is a client.

So why bother with a Web cache? What advantages does it have? Web caches are enjoying wide-scale deployment in the Internet for at least three reasons. First, a Web cache can substantially reduce the response time for a client request, particularly if the bottleneck bandwidth between the client and the origin server is much less than the bottleneck bandwidth between the client and the cache. If there is a high-speed connection between the client and the cache, as there often is, and if the cache has the requested object, then the cache will be able to rapidly deliver the object to the client. Second, as we will soon illustrate with an example, Web caches can substantially reduce traffic on an institution's access link to the Internet. By reducing traffic, the institution (for example, a company or a university) does not have to upgrade bandwidth as quickly, thereby reducing costs. Furthermore, Web caches can substantially reduce Web traffic in the Internet as a whole, thereby improving performance for all applications. In 1998, over 75 percent of Internet traffic was Web traffic, so a significant reduction in Web traffic can translate into a significant improvement in Internet performance [Claffy 1998]. Third, an Internet dense with Web caches—such as, at institutional, regional, and national levels—provides an infrastructure for rapid distribution of content, even for content providers who run their sites on low-speed servers behind low-speed access links. If such a “resource-poor” content provider suddenly has popular content to distribute, this popular content will quickly be copied into the Internet caches, and high user demand will be satisfied.

To gain a deeper understanding of the benefits of caches, let us consider an example in the context of Figure 2.10. This figure shows two networks—the institutional network and the rest of the public Internet. The institutional network is a high-speed LAN. A router in the institutional network and a router in the Internet are connected by a 1.5 Mbps link. The origin servers are attached to the Internet, but located all over the globe. Suppose that the average object size is 100 Kbits and that the average request rate from the institution's browsers to the origin servers is 15 requests per second. Also suppose that the amount of time it takes from when the router on the Internet side of the access link in Figure 2.10 forwards an HTTP request (within an IP datagram) until it receives the IP datagram (typically, many IP datagrams) containing the corresponding response is two seconds on average. Informally, we refer to this last delay as the “Internet delay.”

The total response time—that is, the time from the browser's request of an object until its receipt of the object—is the sum of the LAN delay, the access delay (that is, the delay between the two routers), and the Internet delay. Let us now do a very crude calculation to estimate this delay. The traffic intensity on the LAN (see Section 1.6) is

$$(15 \text{ requests/sec}) * (100 \text{ Kbits/request}) / (10 \text{ Mbps}) = 0.15$$

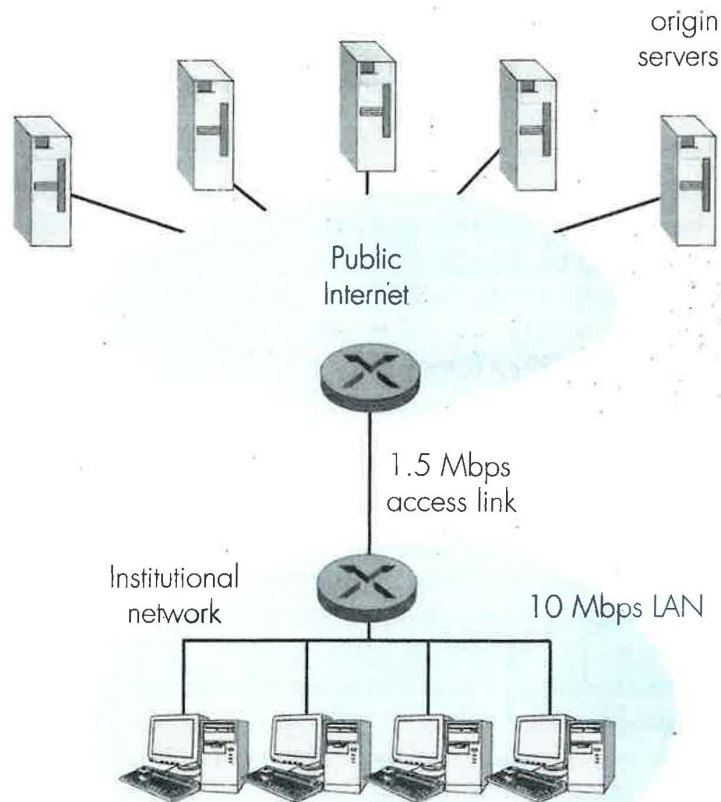


Figure 2.10 ♦ Bottleneck between an institutional network and the Internet

whereas the traffic intensity on the access link (from the Internet router to institution router) is

$$(15 \text{ requests/sec}) * (100 \text{ Kbits/request}) / (1.5 \text{ Mbps}) = 1$$

A traffic intensity of 0.15 on a LAN typically results in, at most, tens of milliseconds of delay; hence, we can neglect the LAN delay. However, as discussed in Section 1.6, as the traffic intensity approaches 1 (as is the case of the access link in Figure 2.10), the delay on a link becomes very large and grows without bound. Thus, the average response time to satisfy requests is going to be on the order of minutes, if not more, which is unacceptable for the institution's users. Clearly something must be done.

One possible solution is to increase the access rate from 1.5 Mbps to, say, 10 Mbps. This will lower the traffic intensity on the access link to 0.15, which trans-

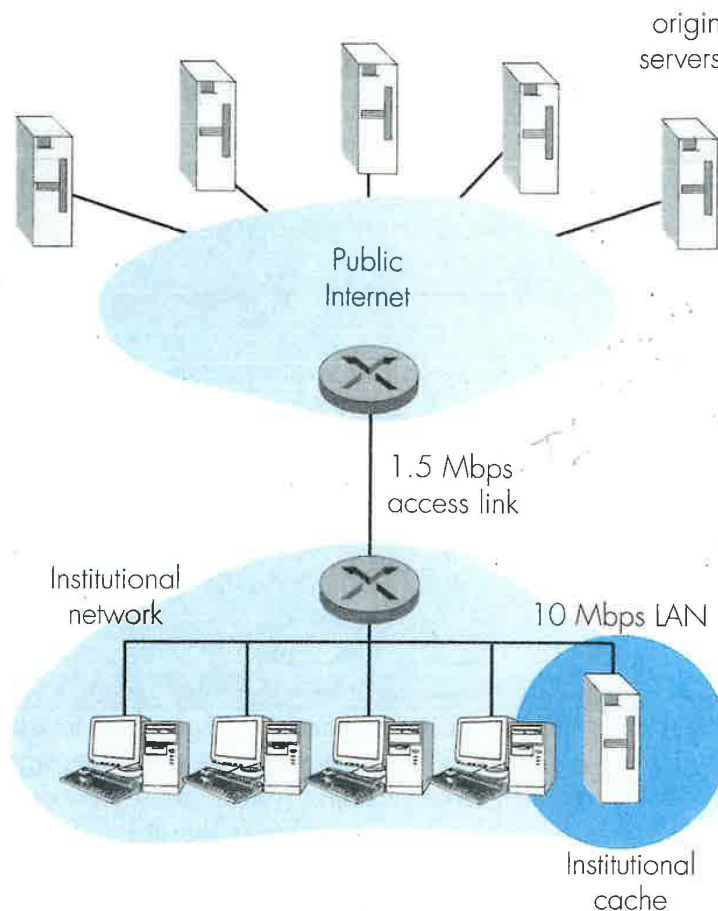


Figure 2.11 ♦ Adding a cache to the institutional network

lates to negligible delays between the two routers. In this case, the total response time will roughly be 2 seconds, that is, the Internet delay. But this solution also means that the institution must upgrade its access link from 1.5 Mbps to 10 Mbps, which can be very costly.

Now consider the alternative solution of not upgrading the access link but instead installing a Web cache in the institutional network. This solution is illustrated in Figure 2.11. Hit rates—the fraction of requests that are satisfied by a cache—typically range from 0.2 to 0.7 in practice. For illustrative purposes, let us suppose that the cache provides a hit rate of 0.4 for this institution. Because the clients and the cache are connected to the same high-speed LAN, 40 percent of the requests will be satisfied almost immediately, say, within 10 milliseconds, by the cache. Nevertheless, the remaining 60 percent of the requests still need to be satisfied by the

origin servers. But with only 60 percent of the requested objects passing through the access link, the traffic intensity on the access link is reduced from 1.0 to 0.6. Typically, a traffic intensity less than 0.8 corresponds to a small delay, say, tens of milliseconds, on a 1.5 Mbps link. This delay is negligible compared with the 2-second Internet delay. Given these considerations, average delay therefore is

$$0.4 * (0.010 \text{ seconds}) + 0.6 * (2.01 \text{ seconds})$$

which is just slightly greater than 1.2 seconds. Thus, this second solution provides an even lower response time than the first solution, and it doesn't require the institution to upgrade its link to the Internet. The institution does, of course, have to purchase and install a Web cache. But this cost is low—many caches use public-domain software that run on inexpensive servers and PCs.

Cooperative Caching

Multiple Web caches, located at different places in the Internet, can cooperate and improve overall performance. For example, an institutional cache can be configured to send its HTTP requests to a cache in a backbone ISP at the national level. In this case, when the institutional cache does not have the requested object in its storage, it forwards the HTTP request to the national cache. The national cache then retrieves the object from its own storage or, if the object is not in storage, from the origin server. The national cache then sends the object (within an HTTP response message) to the institutional cache, which in turn forwards the object to the requesting browser. Whenever an object passes through a cache (institutional or national), the cache keeps a copy in its local storage. The advantage of passing through a higher-level cache, such as a national cache, is that it has a larger user population and therefore higher hit rates.

An example of a cooperative caching system is the NLANR caching system, which consists of a number of backbone caches in the United States providing service to institutional and regional caches from all over the globe [NLANR 1999]. The NLANR caching hierarchy is shown in Figure 2.12 [Huffaker 1998]. The caches obtain objects from each other using a combination of HTTP and ICP (Internet Caching Protocol). ICP is an application-layer protocol that allows one cache to quickly ask another cache if it has a given document [RFC 2186]; a cache can then use HTTP to retrieve the object from the other cache. ICP is used extensively in many cooperative caching systems and is fully supported by Squid, a popular public-domain software for Web caching [Squid 2000]. If you are interested in learning more about ICP, you are encouraged to see [Luotonen 1998], [Ross 1998], and the ICP RFC [RFC 2186].

An alternative form of cooperative caching involves clusters of caches, often co-located on the same LAN. A single cache is often replaced with a cluster of caches when the single cache is not sufficient to handle the traffic or provide suffi-

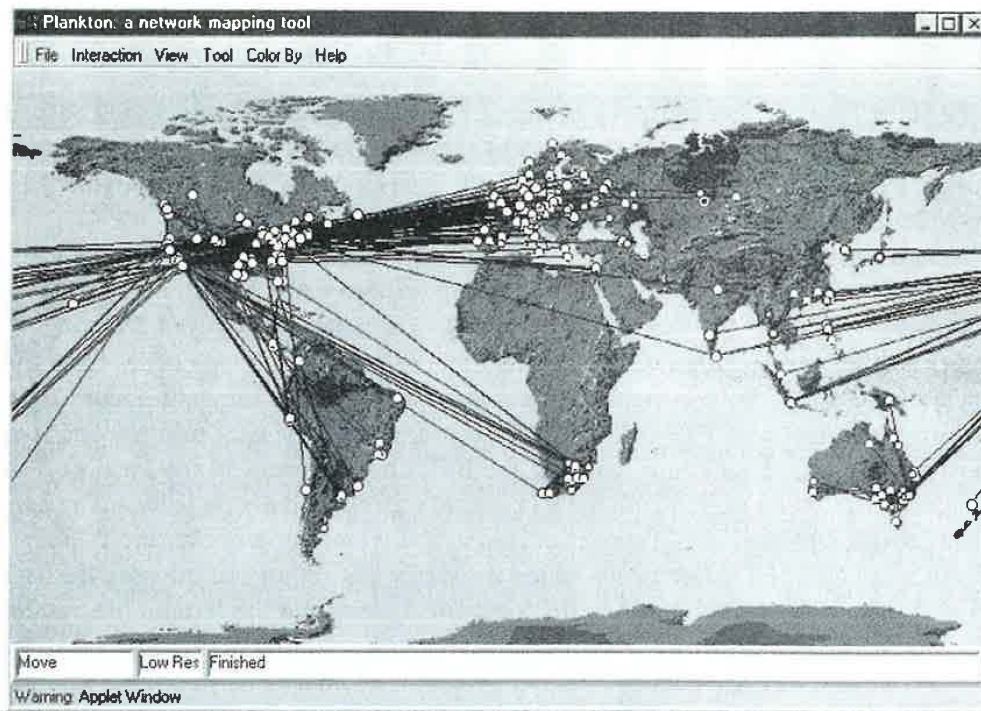


Figure 2.12 ♦ The NLANR caching hierarchy (Courtesy of [Huffaker and Jung 1998])

cient storage capacity. Although cache clustering is a natural way to scale as traffic increases, clusters introduce a new problem: When a browser wants to request a particular object, to which cache in the cache cluster should it send the request? This problem can be elegantly solved using hash routing. (If you are not familiar with hash functions, you can read about them in Chapter 7.) In the simplest form of hash routing, the browser hashes the URL, and depending on the result of the hash, the browser directs its request message to one of the caches in the cluster. By having all the browsers use the same hash function, an object will never be present in more than one cache in the cluster, and if the object is indeed in the cache cluster, the browser will always direct its request to the correct cache. Hash routing is the essence of the Cache Array Routing Protocol (CARP). Additional information is available if you are interested in learning more about hash routing or CARP [Valloppillil 1997; Luotonen 1998; Ross 1997, 1998].

Web caching is a rich and complex subject. Web caching has also enjoyed extensive research and product development in recent years. Furthermore, caches are now being built to handle streaming audio and video. Caches will likely play an important role as the Internet begins to provide an infrastructure for the large-scale, on-demand distribution of music, television shows, and movies in the Internet.

user to pause/resume a message or to reposition within a message; furthermore, streaming over TCP often leads to poor reception (see Chapter 6). These inadequacies will hopefully be addressed in the upcoming years. Possible solutions are discussed in the literature [Gay 1997; Hess 1998; Schurmann 1996; and Turner 1999].

2.5 ♦ DNS—The Internet’s Directory Service

We human beings can be identified in many ways. For example, we can be identified by the names that appear on our birth certificates. We can be identified by our social security numbers. We can be identified by our driver’s license numbers. Although each of these identifiers can be used to identify people, within a given context one identifier may be more appropriate than another. For example, the computers at the IRS (the infamous tax collecting agency in the United States) prefers to use fixed-length social security numbers rather than birth-certificate names. On the other hand, ordinary people prefer the more mnemonic birth-certificate names rather than social security numbers. (Indeed, can you imagine saying, “Hi. My name is 132-67-9875. Please meet my husband, 178-87-1146.”)

Just as humans can be identified in many ways, so too can Internet hosts. One identifier for a host is its **hostname**. Hostnames—such as `cnn.com`, `www.yahoo.com`, `gaia.cs.umass.edu`, and `surf.eurecom.fr`—are mnemonic and are therefore appreciated by humans. However, hostnames provide little, if any, information about the location within the Internet of the host. (A hostname such as `surf.eurecom.fr`, which ends with the country code `.fr`, tells us that the host is probably in France, but doesn’t say much more.) Furthermore, because hostnames can consist of variable-length alphanumeric characters, they would be difficult to process by routers. For these reasons, hosts are also identified by so-called **IP addresses**. We will discuss IP addresses in some detail in Chapter 4, but it is useful to say a few brief words about them now. An IP address consists of four bytes and has a rigid hierarchical structure. An IP address looks like `121.7.106.83`, where each period separates one of the bytes expressed in decimal notation from 0 to 255. An IP address is hierarchical because as we scan the address from left to right, we obtain more and more specific information about where (that is, within which network, in the network of networks) the host is located in the Internet. (Similarly, when we scan a postal address from bottom to top, we obtain more and more specific information about where the residence is located.)

2.5.1 Services Provided by DNS

We have just seen that there are two ways to identify a host—by a hostname and by an IP address. People prefer the more mnemonic hostname identifier, while routers prefer fixed-length, hierarchically structured IP addresses. In order to reconcile these different preferences, we need a directory service that translates hostnames to

IP addresses. This is the main task of the Internet's **Domain Name System (DNS)**. The DNS is (1) a distributed database implemented in a hierarchy of **name servers** and (2) an application-layer protocol that allows hosts and name servers to communicate in order to provide the translation service. Name servers are often Unix machines running the Berkeley Internet Name Domain (BIND) software. The DNS protocol runs over UDP and uses port 53. On the Web site for this book, at <http://www.awl.com/kurose-ross> we provide interactive links to DNS programs that allow you to translate arbitrary hostnames, among other things.

DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP—to translate user-supplied host names to IP addresses. As an example, consider what happens when a browser (that is, an HTTP client), running on some user's machine, requests the URL `www.someschool.edu/index.html`. In order for the user's machine to be able to send an HTTP request message to the Web server `www.someschool.edu`, the user's machine must obtain the IP address of `www.someschool.edu`. This is done as follows. The same user machine runs the client-side of the DNS application. The browser extracts the hostname, `www.someschool.edu`, from the URL and passes the hostname to the client-side of the DNS application. As part of a DNS query message, the DNS client sends a query containing the hostname to a DNS server. The DNS client eventually receives a reply, which includes the IP address for the hostname. The browser then opens a TCP connection to the HTTP server process located at that IP address. We see from this example that DNS adds an additional delay—sometimes substantial—to the Internet applications that use DNS. Fortunately, as we shall discuss below, the desired IP address is often cached in a “nearby” DNS name server, which helps to reduce the DNS network traffic as well as the average DNS delay.



Principles in Practice

DNS: Critical network functions via the client-server paradigm

Like HTTP, FTP, and SMTP, the DNS protocol is an application-layer protocol since it (1) runs between communicating end using the client-server paradigm, and (2) relies on an underlying end-to-end transport protocol to transfer DNS messages between communicating end systems. In another sense, however, the role of the DNS is quite different from Web, file transfer, and e-mail applications. Unlike these applications, the DNS is not an application with which a user directly interacts. Instead, the DNS provides a core Internet function—namely, translating hostnames to their underlying IP addresses, for user applications and other software in the Internet. We noted earlier in Section 1.2 that much of the complexity in the Internet architecture is located at the “edges” of the network. The DNS, which implements the critical name-to-address translation process using clients and servers located at the edge of the network, is yet another example of that design philosophy.



DNS provides a few other important services in addition to translating hostnames to IP addresses:

- ◆ **Host aliasing.** A host with a complicated hostname can have one or more alias names. For example, a hostname such as `relay1.west-coast.enterprise.com` could have, say, two aliases such as `enterprise.com` and `www.enterprise.com`. In this case, the hostname `relay1.west-coast.enterprise.com` is said to be a **canonical hostname**. Alias hostnames, when present, are typically more mnemonic than a canonical hostname. DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- ◆ **Mail server aliasing.** For obvious reasons, it is highly desirable that e-mail addresses be mnemonic. For example, if Bob has an account with Hotmail, Bob's e-mail address might be as simple as `bob@hotmail.com`. However, the hostname of the Hotmail mail server is more complicated and much less mnemonic than simply `hotmail.com` (for example, the canonical hostname might be something like `relay1.west-coast.hotmail.com`). DNS can be invoked by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host. In fact, DNS permits a company's mail server and Web server to have identical (aliased) hostnames; for example, a company's Web server and mail server can both be called `enterprise.com`.
- ◆ **Load distribution.** Increasingly, DNS is also being used to perform load distribution among replicated servers, such as replicated Web servers. Busy sites, such as `cnn.com`, are replicated over multiple servers, with each server running on a different end system, and having a different IP address. For replicated Web servers, a *set* of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply. Because a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among all the replicated servers. DNS rotation is also used for e-mail so that multiple mail servers can have the same alias name. Recently, companies such as Akamai [Akamai 2000] have used DNS in more sophisticated ways to provide Web content distribution.

The DNS is specified in RFC 1034 and RFC 1035, and updated in several additional RFCs. It is a complex system, and we only touch upon key aspects of its operation here. The interested reader is referred to these RFCs and a book by Abitz and Liu [Abitz 1993].

2.5.2 Overview of How DNS Works

We now present a high-level overview of how DNS works. Our discussion will focus on the hostname-to-IP-address translation service. From the client's perspective, the DNS is a black box. The client sends a DNS query message into the black box, specifying the hostname that needs to be translated to an IP address. On many Unix-based machines, `gethostbyname()` is the library routine that an application calls in order to issue the query message. In Section 2.7, we will present a Java program that begins by issuing a DNS query. After a delay, ranging from milliseconds to tens of seconds, the client receives a DNS reply message that provides the desired mapping. Thus, from the client's perspective, DNS is a simple, straightforward translation service. But in fact, the black box that implements the service is complex, consisting of a large number of name servers distributed around the globe, as well as an application-layer protocol that specifies how the name servers and querying hosts communicate.

A simple design for DNS would have one Internet name server that contains all the mappings. In this centralized design, clients simply direct all queries to the single name server, and the name server responds directly to the querying clients. Although the simplicity of this design is attractive, it is completely inappropriate for today's Internet, with its vast (and growing) number of hosts. The problems with a centralized design include:

- ◆ **A single point of failure.** If the name server crashes, so too does the entire Internet!
- ◆ **Traffic volumes.** A single name server would have to handle all DNS queries (for all the HTTP requests and e-mail messages generated from millions of hosts).
- ◆ **Distant centralized database.** A single name server cannot be “close to” all the querying clients. If we put the single name server in New York City, then all queries from Australia must travel to the other side of the globe, perhaps over slow and congested links. This can lead to significant delays (thereby increasing the “world wide wait” for the Web and other applications).
- ◆ **Maintenance.** The single name server would have to keep records for all Internet hosts. Not only would this centralized database be huge, but it would have to be updated frequently to account for every new host. There are also authentication and authorization problems associated with allowing any user to register a host with the centralized database.

In summary, a centralized database in a single name server simply *doesn't scale*. Consequently, the DNS is distributed by design. In fact, the DNS is a wonderful example of how a distributed database can be implemented in the Internet.

In order to deal with the issue of scale, the DNS uses a large number of name servers, organized in a hierarchical fashion and distributed around the world. No one name server has all of the mappings for all of the hosts in the Internet. Instead, the mappings are distributed across the name servers. To a first approximation, there are three types of name servers: local name servers, root name servers, and authoritative name servers. These name servers, again to a first approximation, interact with each other and with the querying host as follows:

- ◆ **Local name servers.** Each ISP—such as a university, an academic department, an employee’s company, or a residential ISP—has a local name server (also called a default name server). When a host issues a DNS query message, the message is first sent to the host’s local name server. The IP address of the local name server is typically configured by hand in a host. (On a Windows 95/98 machine, you can find the IP address of the local name server used by your PC by opening the Control Panel, and then selecting “Network,” then selecting an installed TCP/IP component, and then selecting the DNS configuration folder tab.) The local name server is typically “close to” the client; in the case of an institutional ISP, it may be on the same LAN as the client host; for a residential ISP, the name server is typically separated from the client host by no more than a few routers. If a host requests a translation for another host that is part of the same local ISP, then the local name server will be able to immediately provide the requested IP address. For example, when the host `surf.eurecom.fr` requests the IP address for `baie.eurecom.fr`, the local name server at Eurecom will be able to provide the requested IP address without contacting any other name servers.
- ◆ **Root name servers.** In the Internet there are a dozen or so root name servers, most of which are currently located in North America. A February 1998 map of the root servers is shown in Figure 2.18. When a local name server cannot immediately satisfy a query from a host (because it does not have a record for the hostname being requested), the local name server behaves as a DNS client and queries one of the root name servers. If the root name server has a record for the hostname, it sends a DNS reply message to the local name server, and the local name server then sends a DNS reply to the querying host. But the root name server may not have a record for the hostname. Instead, the rootname server knows the IP address of an “authoritative name server” that has the mapping for that particular hostname.
- ◆ **Authoritative name servers.** Every host is registered with an authoritative name server. Typically, the authoritative name server for a host is a name server in the host’s local ISP. (Actually, each host is required to have at least two authoritative name servers, in case of failures.) By definition, a name server is authoritative for a host if it always has a DNS record that translates the host’s hostname to that host’s IP address. When an authoritative name server is queried by a root server,

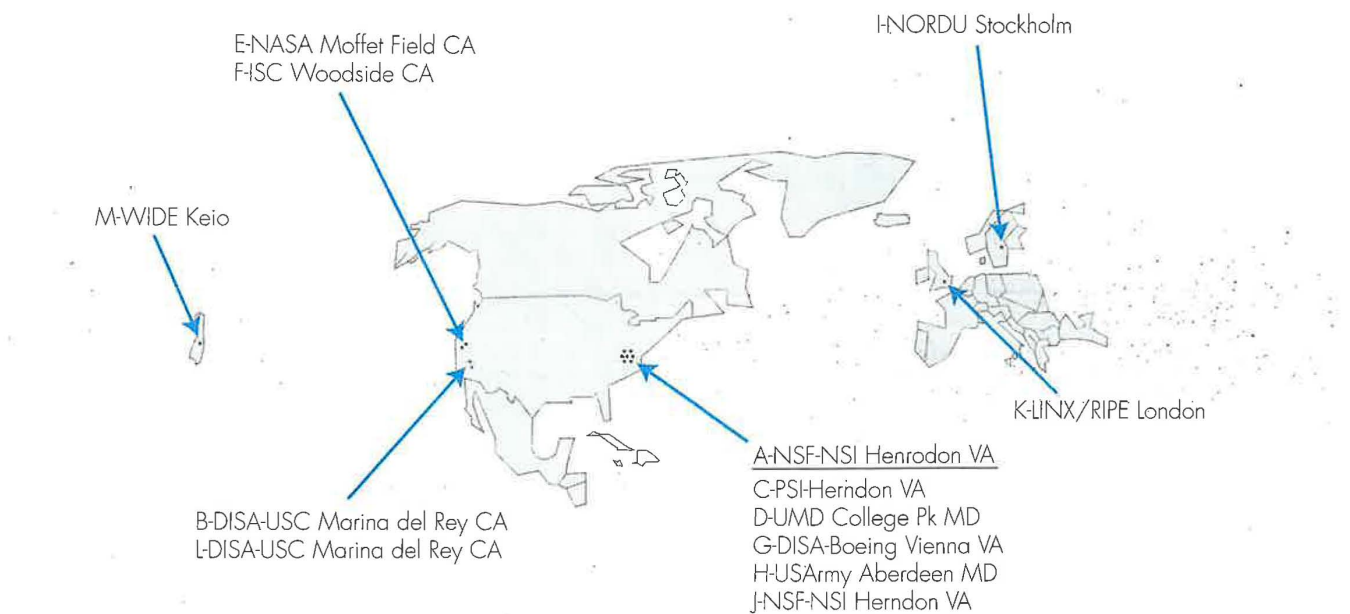


Figure 2.18 ♦ A February 1998 map of the DNS root servers. Data from the WIA alliance Web site (<http://www.wia.org/>)

the authoritative name server responds with a DNS reply that contains the requested mapping. The root server then forwards the mapping to the local name server, which in turn forwards the mapping to the requesting host. Many name servers act as both local and authoritative name servers.

Let's take a look at a simple example. Suppose the host `surf.eurecom.fr` desires the IP address of `gaia.cs.umass.edu`. Also suppose that Eurecom's local name server is called `dns.eurecom.fr` and that an authoritative name server for `gaia.cs.umass.edu` is called `dns.umass.edu`. As shown in Figure 2.19, the host `surf.eurecom.fr` first sends a DNS query message to its local name server, `dns.eurecom.fr`. The query message contains the hostname to be translated, namely, `gaia.cs.umass.edu`. The local name server forwards the query message to a root name server. The root name server forwards the query message to the name server that is authoritative for all the hosts in the domain `umass.edu`, for example, to `dns.umass.edu`. The authoritative name server then sends the desired mapping to the querying host, via the root name server and the local name server. Note that in this example, in order to obtain the mapping for one hostname, six DNS messages were sent: three query messages and three reply messages.

Our discussion up to this point has assumed that the root name server knows the IP address of an authoritative name server for *every* hostname. This assumption may

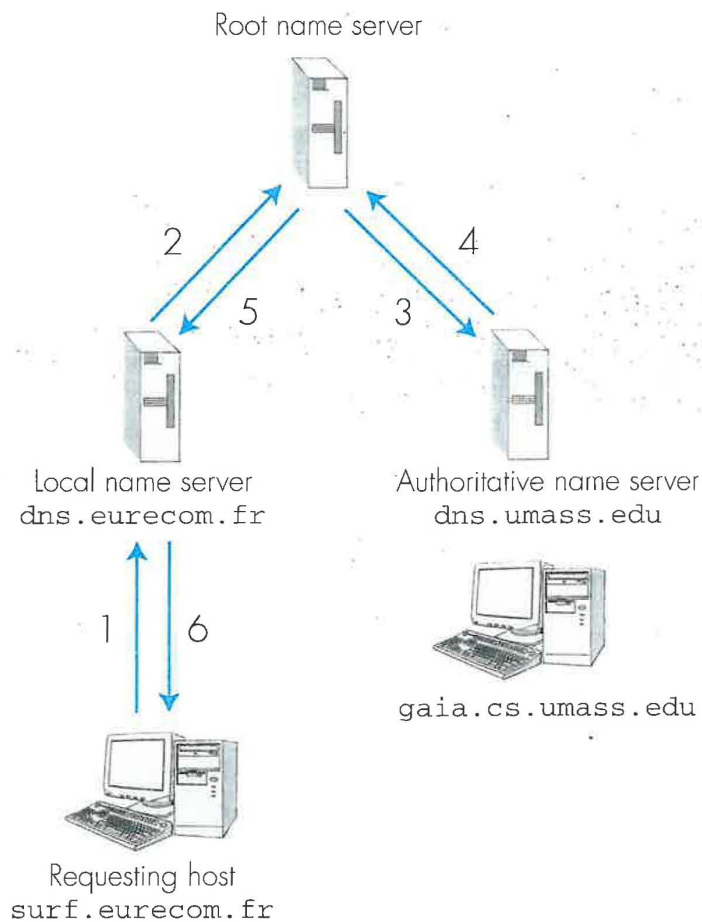


Figure 2.19 ♦ Recursive queries to obtain the mapping for `gaia.cs.umass.edu`

be incorrect. For a given hostname, the root name server may only know the IP address of an intermediate name server that in turn knows the IP address of an authoritative name server for the hostname. To illustrate this, consider once again the above example with the host `surf.eurecom.fr` querying for the IP address of `gaia.cs.umass.edu`. Suppose now that the University of Massachusetts has a name server for the university, called `dns.umass.edu`. Also suppose that each of the departments at the University of Massachusetts has its own name server, and that each departmental name server is authoritative for all the hosts in the department. As shown in Figure 2.20, when the root name server receives a query for a host with hostname ending with `umass.edu`, it forwards the query to the name server `dns.umass.edu`. This name server forwards all queries with hostnames ending with `.cs.umass.edu` to the name server `dns.cs.umass.edu`, which is

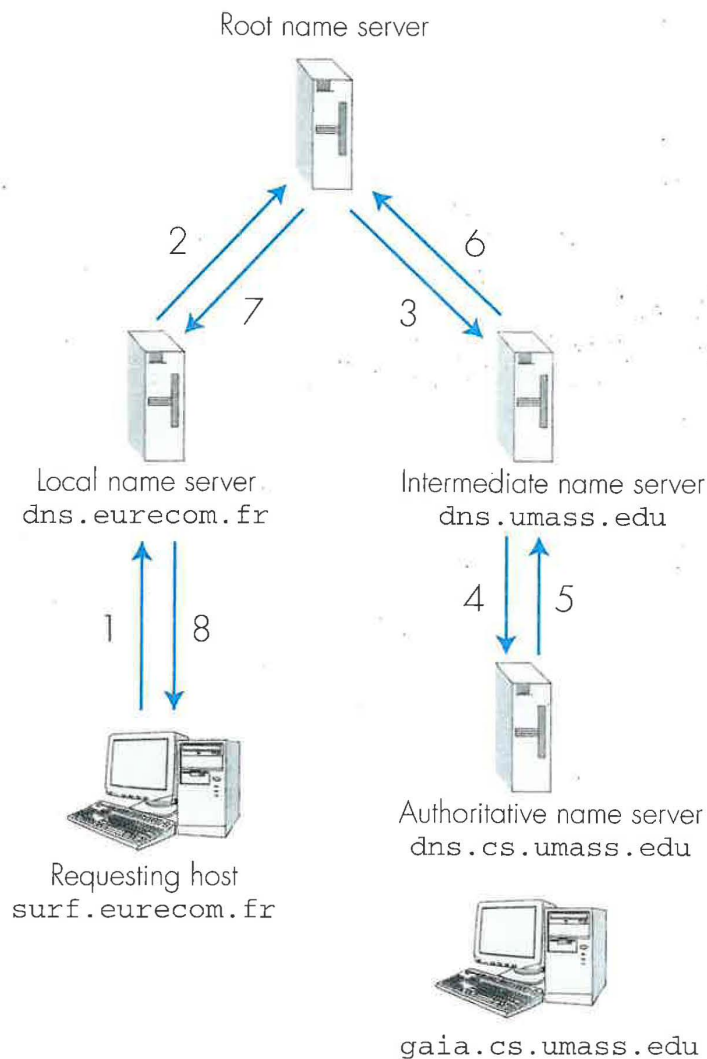


Figure 2.20 ♦ Recursive queries with an intermediate name server between the root and authoritative name servers

authoritative for all hostnames ending with `.cs.umass.edu`. The authoritative name server sends the desired mapping to the intermediate name server, `dns.umass.edu`, which forwards the mapping to the root name server, which forwards the mapping to the local name server, `dns.eurecom.fr`, which forwards the mapping to the requesting host! In this example, eight DNS messages are sent. Actually, even more DNS messages can be sent in order to translate a single hostname—there can be two or more intermediate name servers in the chain between the root name server and the authoritative name server!

The examples up to this point assumed that all queries are so-called **recursive queries**. When a host or name server A makes a recursive query to a name server B, then name server B obtains the requested mapping *on behalf* of A and then forwards the mapping to A. The DNS protocol also allows for **iterative queries** at any step in the chain between requesting host and authoritative name server. When a name server A makes an iterative query to name server B, if name server B does not have the requested mapping, it immediately sends a DNS reply to A that contains the IP address of the next name server in the chain, say, name server C. Name server A then sends a query directly to name server C.

In the sequence of queries that are required to translate a hostname, some of the queries can be iterative and others recursive. Such a combination of recursive and iterative queries is illustrated in Figure 2.21. Typically, all queries in the query chain are recursive except for the query from the local name server to the root name server, which is iterative. (Because root servers handle huge volumes of queries, it is preferable to use the less burdensome iterative queries for root servers.)

Our discussion this far has not touched on one important feature of the DNS: **DNS caching**. In reality, DNS extensively exploits caching in order to improve the delay performance and to reduce the number of DNS messages in the network. The idea is very simple. When a name server receives a DNS mapping for some hostname, it caches the mapping in local memory (disk or RAM) while passing the message along the name server chain. Given a cached hostname/IP address translation pair, if another query arrives to the name server for the same hostname, the name server can provide the desired IP address, even if it is not authoritative for the hostname. In order to deal with ephemeral hosts, a cached record is discarded after a period of time (often set to two days). As an example, suppose that `surf.eurecom.fr` queries the DNS for the IP address for the hostname `cnn.com`. Furthermore, suppose that a few hours later, another Eurecom host, say, `baie.eurecom.fr`, also queries DNS with the same hostname. Because of caching, the local name server at Eurecom will be able to immediately return the IP address of `cnn.com` to the requesting host without having to query name servers on another continent. Any name server may cache DNS mappings.

2.5.3 DNS Records

The name servers that together implement the DNS distributed database, store **resource records (RR)** for the hostname to IP address mappings. Each DNS reply message carries one or more resource records. In this and the following subsection, we provide a brief overview of DNS resource records and messages; more details can be found in [Abitz 1993] or in the DNS RFCs [RFC 1034; RFC 1035].

A resource record is a four-tuple that contains the following fields:

(Name, Value, Type, TTL)

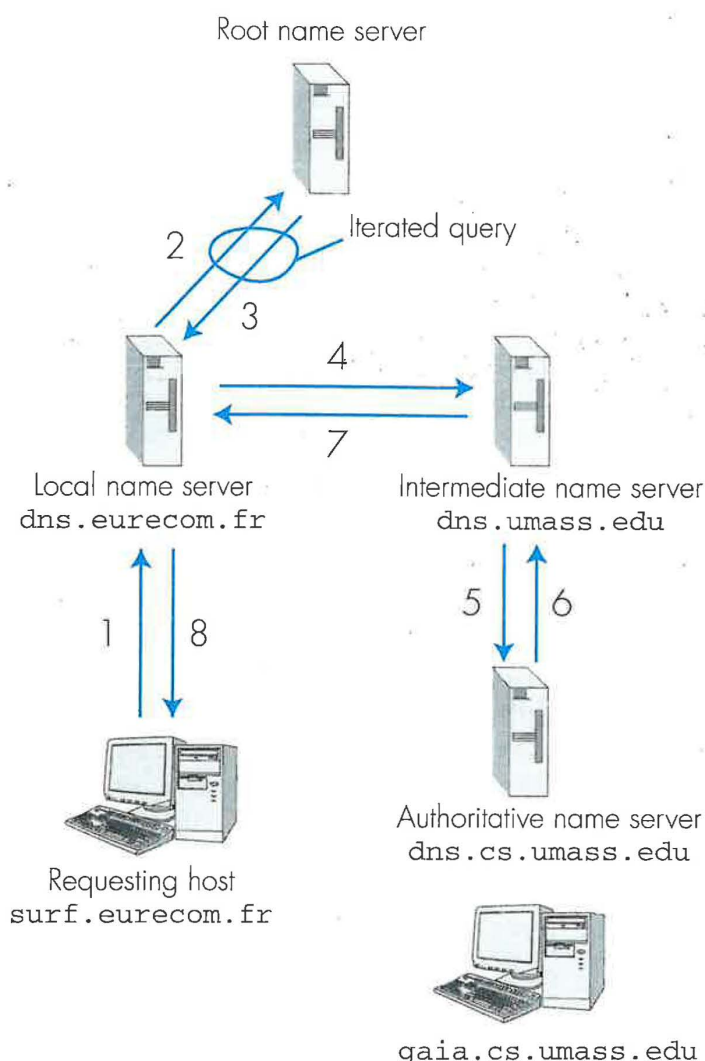


Figure 2.21 ♦ A query chain with recursive and iterative queries

TTL is the time to live of the resource record; it determines the time at which a resource should be removed from a cache. In the example records given below, we will ignore the TTL field. The meaning of Name and Value depend on Type:

- ♦ If Type=A, then Name is a hostname and Value is the IP address for the hostname. Thus, a Type A record provides the standard hostname to IP address mapping. As an example, (relay1.bar.foo.com, 145.37.93.126, A) is a Type A record.

- ◆ If `Type=NS`, then `Name` is a domain (such as `foo.com`) and `Value` is the hostname of an authoritative name server that knows how to obtain the IP addresses for hosts in the domain. This record is used to route DNS queries further along in the query chain. As an example, `(foo.com, dns.foo.com, NS)` is a Type NS record.
- ◆ If `Type=CNAME`, then `Value` is a canonical hostname for the alias hostname `Name`. This record can provide querying hosts the canonical name for a hostname. As an example, `(foo.com, relay1.bar.foo.com, CNAME)` is a CNAME record.
- ◆ If `Type=MX`, then `Value` is a hostname of a mail server that has an alias hostname `Name`. As an example, `(foo.com, mail.bar.foo.com, MX)` is an MX record. MX records allow the hostnames of mail servers to have simple aliases.

If a name server is authoritative for a particular hostname, then the name server will contain a Type A record for the hostname. (Even if the name server is not authoritative, it may contain a Type A record in its cache.) If a server is not authoritative for a hostname, then the server will contain a Type NS record for the domain that includes the hostname; it will also contain a Type A record that provides the IP address of the name server in the `Value` field of the NS record. As an example, suppose a root server is not authoritative for the host `gaia.cs.umass.edu`. Then the root server will contain a record for a domain that includes the host `cs.umass.edu`, for example, `(umass.edu, dns.umass.edu, NS)`. The root server would also contain a Type A record, which maps the name server `dns.umass.edu` to an IP address, for example, `(dns.umass.edu, 128.119.40.111, A)`.

2.5.4 DNS Messages

Earlier in this section we alluded to DNS query and reply messages. These are the only two kinds of DNS messages. Furthermore, both request and reply messages have the same format, as shown in Figure 2.22.

The semantics of the various fields in a DNS message is as follows:

- ◆ The first 12 bytes is the *header section*, which has a number of fields. The first field is a 16-bit number that identifies the query. This identifier is copied into the reply message to a query, allowing the client to match received replies with sent queries. There are a number of flags in the flag field. A one-bit query/reply flag indicates whether the message is a query (0) or a reply (1). A one-bit authoritative flag is set in a reply message when a name server is an authoritative server for a queried name. A one-bit recursion-desired flag is set when a client (host or name server) desires that the name server perform recursion when it doesn't have the record. A

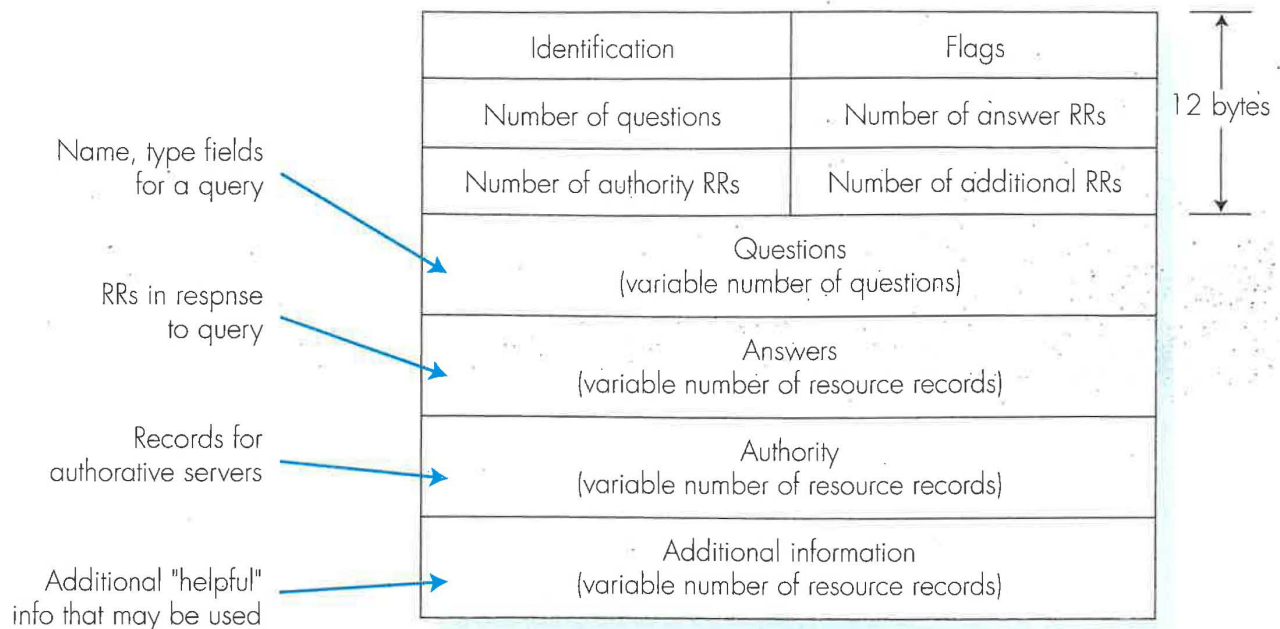


Figure 2.22 ♦ DNS message format

one-bit recursion-available field is set in a reply if the name server supports recursion. In the header, there are also four “number of” fields. These fields indicate the number of occurrences of the four types of “data” sections that follow the header.

- ♦ The *question section* contains information about the query that is being made. This section includes (1) a name field that contains the name that is being queried, and (2) a type field that indicates the type of question being asked about the name (for example, a host address associated with a name—type A, or the mail server for a name—type MX).
- ♦ In a reply from a name server, the *answer section* contains the resource records for the name that was originally queried. Recall that in each resource record there is the **Type** (for example, A, NS, CSNAME, and MX), the **Value**, and the **TTL**. A reply can return multiple RRs in the answer, since a hostname can have multiple IP addresses (for example, for replicated Web servers, as discussed earlier in this section).
- ♦ The *authority section* contains records of other authoritative servers.
- ♦ The *additional section* contains other “helpful” records. For example, the answer field in a reply to an MX query will contain the hostname of a mail server associated with the alias name Name. The additional section will contain a Type A record providing the IP address for the canonical hostname of the mail server.

The discussion above has focused on how data is retrieved from the DNS database. You might be wondering how data gets into the database in the first place? Until recently, the contents of each DNS server was configured statically, for example, from a configuration file created by a system manager. More recently, an `UPDATE` option has been added to the DNS protocol to allow data to be dynamically added or deleted from the database via DNS messages. RFC 2136 specifies DNS dynamic updates.

DNSNet provides a nice collection of documents pertaining to DNS [DNSNet 1999]. The Internet Software Consortium provides many resources for **BIND**, a popular public-domain name server for Unix machines [BIND 1999].

2.6 ♦ Socket Programming with TCP

This and the subsequent sections provide an introduction to network application development. Recall from Section 2.1 that the core of a network application consists of a pair of programs—a client program and a server program. When these two programs are executed, a client and server process are created, and these two processes communicate with each other by reading from and writing to sockets. When creating a network application, the developer's main task is to write the code for both the client and server programs.

There are two sorts of client/server applications. One sort is a client/server application that is an *implementation* of a protocol standard defined in an RFC. For such an implementation, the client and server programs must conform to the rules dictated by the RFC. For example, the client program could be an implementation of the FTP client, defined in RFC 959, and the server program could be an implementation of the FTP server, also defined in RFC 959. If one developer writes code for the client program and an independent developer writes code for the server program, and both developers carefully follow the rules of the RFC, then the two programs will be able to interoperate. Indeed, most of today's network applications involve communication between client and server programs that have been created by independent developers. (For example, a Netscape browser communicating with an Apache Web server, or an FTP client on a PC uploading a file to a Unix FTP server.) When a client or server program implements a protocol defined in an RFC, it should use the port number associated with the protocol. (Port numbers were briefly discussed in Section 2.1. They will be covered in more detail in the next chapter.)

The other sort of client/server application is a *proprietary* client/server application. In this case the client and server programs do not necessarily conform to any existing RFC. A single developer (or development team) creates both the client and server programs, and the developer has complete control over what goes in the code. But because the code does not implement a public-domain protocol, other indepen-

poor utilization of the assigned address space. For example, the organization that used a class B address for its 2,000 hosts was allocated enough of the address space for up to 65,534 interfaces—leaving more than 63,000 unused addresses that could not be used by other organizations.

In 1993, the IETF standardized on **Classless Interdomain Routing (CIDR**—pronounced the same as “cider”) [RFC 1519]. With so-called CIDRized network addresses, the network part of an IP address can be any number of bits long, rather than being constrained to 8, 16, or 24 bits. A CIDRized network address has the dotted-decimal form *a.b.c.d/x*, where *x* indicates the number of leading bits in the 32-bit quantity that constitutes the network portion of the address. In our example above, the organization needing to support 2,000 hosts could be allocated a block of only 2,048 host addresses of the form *a.b.c.d/21*, allowing the approximately 63,000 addresses that would have been allocated and unused under classful addressing to be allocated to a different organization. In this case, the first 21 bits specify the organization’s network address and are common in the IP addresses of all hosts in the organization. The remaining 11 bits then identify the specific hosts in the organization. In practice, the organization could further divide these 11 rightmost bits using a procedure known as **subnetting** [RFC 950] to create its own internal networks within the *a.b.c.d/21* network.

Assigning Addresses

Having introduced IP addressing, a question that immediately comes to mind is how a host gets its own IP address. We have just learned that an IP address has two parts, a network part and a host part. The host part of the address can be assigned in several different ways, including:

- ◆ **Manual configuration.** The IP address is configured into the host (typically in a file) by the system administrator.
- ◆ **Dynamic Host Configuration Protocol (DHCP)** [RFC 2131]. DHCP is an extension of the BOOTP [RFC 1542] protocol, and is sometimes referred to as Plug and Play. With DHCP, a DHCP server in a network (for example, in a LAN) receives DHCP requests from a client and, in the case of dynamic address allocation, allocates an IP address back to the requesting client. DHCP is used extensively in LANs and in residential Internet access.

Obtaining a network address is not as simple. An organization’s network administrator might first contact its ISP, which would provide addresses from a larger block of addresses that had already been allocated to the ISP. For example, the ISP may itself have been allocated the address block 200.23.16.0/20. The ISP, in turn could divide its address block into eight equal-size smaller address blocks and give one of these address blocks out to each of up to eight organizations that are supported by

this ISP, as shown below. (We have underlined the network part of these addresses for visual convenience.)

ISP's block	<u>11001000</u> 00010111 00010000 00000000	200.23.16.0/20
Organization 0	<u>11001000</u> 00010111 00010000 00000000	200.23.16.0/23
Organization 1	<u>11001000</u> 00010111 00010010 00000000	200.23.18.0/23
Organization 2	<u>11001000</u> 00010111 00010100 00000000	200.23.20.0/23
...		
Organization 7	<u>11001000</u> 00010111 00011110 00000000	200.23.30.0/23



Principles in Practice

This example of an ISP that connects eight organizations into the larger Internet also nicely illustrates how carefully allocated CIDRized addresses facilitate hierarchical routing. Suppose, as shown in Figure 4.19, that the ISP (which we'll call Fly-By-Night-ISP) advertises to the outside world that it should be sent any datagrams whose first 20 address bits match 200.23.16.0/20. The rest of the world need not know that within the

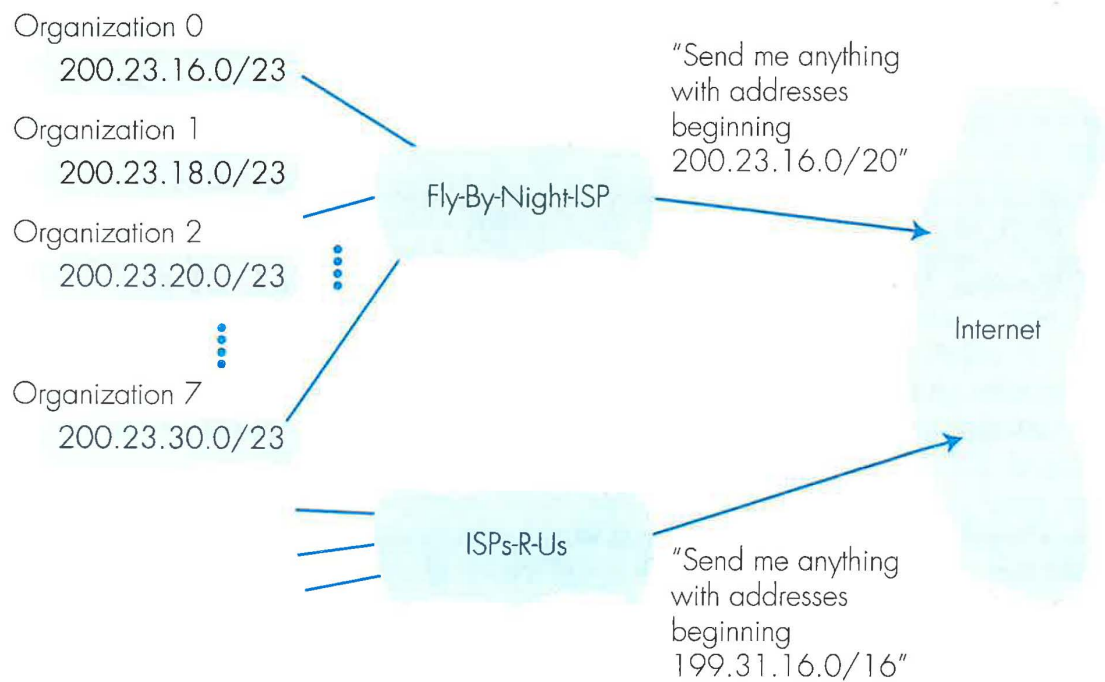


Figure 4.19 ♦ Hierarchical addressing and route aggregation

address block 200.23.16.0/20 there are in fact eight other organizations, each with their own networks. This ability to use a single network prefix to advertise multiple networks is often referred to as **route aggregation** or **route summarization**.

Route aggregation works extremely well when addresses are allocated in blocks to ISPs and then from ISPs to client organizations. But what happens when addresses are not allocated in such a hierarchical manner? What would happen, for example, if Organization 1 becomes discontent with the poor service provided by Fly-By-Night-ISP and decides to switch over to a new ISP, say, ISPs-R-Us? As shown in Figure 4.19, ISPs-R-Us owns the address block 199.31.0.0/16, but Organization 2's IP addresses are unfortunately outside of this address block. What should be done here? Certainly, Organization 1 could renumber all of its routers and hosts to have addresses within the ISPs-R-Us address block. But this is a costly solution, and Organization 1 might well choose to switch from ISPs-R-Us to yet another ISP in the future. The solution typically adopted is for Organization 1 to keep its IP addresses in 200.23.18.0/23. In this case, as shown in Figure 4.20, Fly-By-Night-ISP continues to advertise the address block 200.23.16.0/20 and ISPs-R-Us continues to advertise 199.31.0.0/16. However, ISPs-R-Us now *also* advertises the block of addresses for Organization 1, 200.23.18.0/23. When other routers in the larger Internet see the address blocks 200.23.16.0/20 (from Fly-By-Night-ISP) and 200.23.18.0/23 (from ISPs-R-Us) and want to route to an address in the block 200.23.18.0/23, they will use a **longest prefix matching** rule, and route toward ISPs-R-Us, as it advertises the longest (more specific) address prefix that matches the destination address.

Let's conclude our discussion of addressing by considering how an ISP itself gets a block of addresses. IP addresses are managed under the authority of The Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN 2000] based on guidelines set forth in RFC 2050. The role of the nonprofit ICANN organization [NTIA 1998] is to allocate not only IP addresses, but also to manage the DNS root servers. It also has the very contentious job of assigning domain names and resolving domain name disputes. The actual assignment of addresses is now managed by regional Internet registries. As of mid-2000, there are three such regional registries: the American Registry for Internet Number (ARIN, which handles registrations for North and South America, as well as parts of Africa. ARIN has recently taken over a number of the functions previously provided by Network Solutions), the Reseaux IP Europeens (RIPE, which covers Europe and nearby countries), and the Asia Pacific Network Information Center (APNIC).

Before leaving our discussion of addressing, we want to mention that mobile hosts may change the network to which they are attached, either dynamically while in motion or on a longer time scale. Because routing is to a network first, and then to a host within the network, this means that the mobile host's IP address must change when the host changes networks. Techniques for handling such issues are now under development within the IETF and the research community [RFC 2002; RFC 2131].

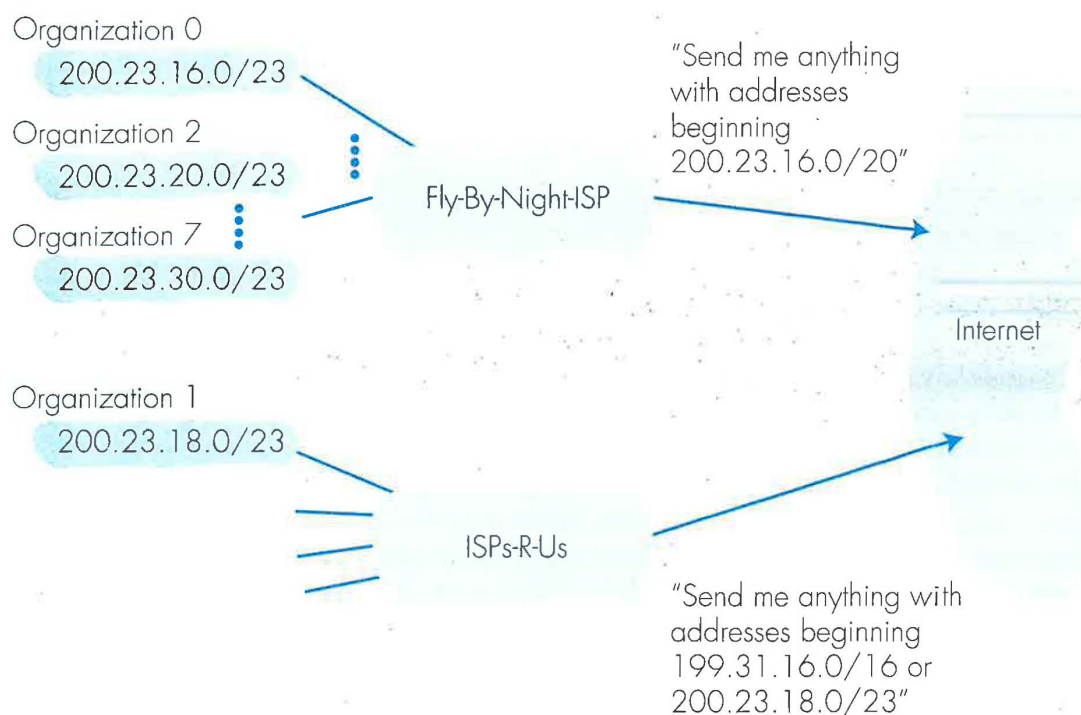


Figure 4.20 ♦ ISPs-R-Us has a more specific route to Organization 1

4.4.2 Transporting a Datagram from Source to Destination: Addressing and Routing

Now that we have defined interfaces and networks and have a basic understanding of IP addressing, we take a step back and examine how hosts and routers transport an IP datagram from source to destination. To this end, a high-level view of an IP datagram is shown in Figure 4.21. Every IP datagram has a source address field and a destination address field. The source host fills a datagram's source address field with its own 32-bit IP address. It fills the destination address field with the 32-bit IP address of the final destination host to which the datagram is being sent. The data field of the datagram is typically filled with a TCP or UDP segment. We'll discuss the remaining IP datagram fields a little later in this section.

Once the source host creates the IP datagram, how does the network layer transport the datagram from the source host to the destination host? The answer to this question depends on whether the source and destination reside on the same network (where the term "network" is used here in the precise, addressing sense discussed in Section 4.4.1). Let's consider this question in the context of the network shown in Figure 4.22. First suppose host *A* wants to send an IP datagram to host *B*, which

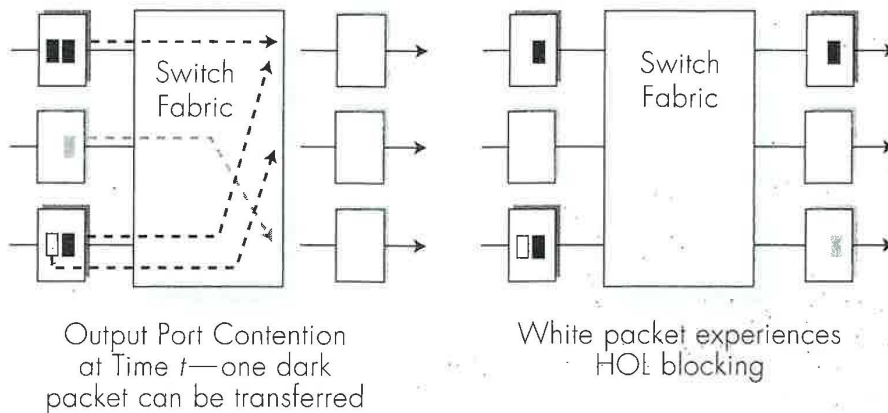


Figure 4.39 ♦ HOL blocking at an input queued switch

that packet in the lower-left queue, even though there is *no* contention for the middle-right output port (the destination for the white packet). This phenomenon is known as **head-of-the-line (HOL) blocking** in an input-queued switch—a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) because it is blocked by another packet at the head of the line. [Karol 1987] shows that due to HOL blocking, the input queue will grow to unbounded length (informally, this is equivalent to saying that significant packet loss will occur) under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity. A number of solutions to HOL blocking are discussed in [McKeown 1997b].

4.7 ♦ IPv6

In the early 1990s, the Internet Engineering Task Force began an effort to develop a successor to the IPv4 protocol. A prime motivation for this effort was the realization that the 32-bit IP address space was beginning to be used up, with new networks and IP nodes being attached to the Internet (and being allocated unique IP addresses) at a breathtaking rate. To respond to this need for a large IP address space, a new IP protocol, IPv6, was developed. The designers of IPv6 also took this opportunity to tweak and augment other aspects of IPv4, based on the accumulated operational experience with IPv4.

The point in time when IPv4 addresses would have been completely allocated (and hence no new networks could have attached to the Internet) was the subject of considerable debate. Based on current trends in address allocation at the time, the estimates of the two leaders of the IETF's Address Lifetime Expectations working group were that addresses would become exhausted in 2008 and 2018, respectively

[Solensky 1996]. In 1996, the American Registry for Internet Numbers (ARIN) reported that all of the IPv4 class A addresses had been assigned, 62 percent of the class B addresses had been assigned, and 37 percent of the class C addresses had been assigned [ARIN 1996]. Although these estimates and numbers suggested that a considerable amount of time might be left until the IPv4 address space became exhausted, it was realized that considerable time would be needed to deploy a new technology on such an extensive scale, and so the “Next Generation IP” (IPng) effort [Bradner 1996; RFC 1752], was begun. An excellent online source of information about IPv6 is The IP Next Generation Homepage [Hinden 1999]. An excellent book is also available on the subject [Huitema 1997].

4.7.1 IPv6 Packet Format

The format of the IPv6 packet is shown in Figure 4.40. The most important changes introduced in IPv6 are evident in the packet format:

- ◆ *Expanded addressing capabilities.* IPv6 increases the size of the IP address from 32 to 128 bits. This ensures that the world won’t run out of IP addresses. Now, every grain of sand on the planet can be IP-addressable. In addition to unicast and multicast addresses, a new type of address, called an **anycast address**, has also been introduced, which allows a packet addressed to an anycast address to be delivered to any one of a group of hosts. (This feature could be used, for example, to



Figure 4.40 ♦ IPv6 packet format

send an HTTP GET to the nearest of a number of mirror sites that contain a given document.)

- ◆ *A streamlined 40-byte header.* As discussed below, a number of IPv4 fields have been dropped or made optional. The resulting 40-byte fixed-length header allows for faster processing of the IP datagram. A new encoding of options allows for more flexible options processing.
- ◆ *Flow labeling and priority.* IPv6 has an elusive definition of a “**flow**.” RFC 1752 and RFC 2460 state that this allows “labeling of packets belonging to particular flows for which the sender requests special handling, such as a non-default quality of service or real-time service.” For example, audio and video transmission might likely be treated as a flow. On the other hand, the more traditional applications, such as file transfer and e-mail might not be treated as flows. It is possible that the traffic carried by a high-priority user (for example, someone paying for better service for their traffic) might also be treated as a flow. What is clear, however, is that the designers of IPv6 foresee the eventual need to be able to differentiate among the “flows,” even if the exact meaning of a flow has not yet been determined. The IPv6 header also has eight-bit Traffic Class field. This field, like the TOS field in IPv4, can be used to give priority to certain packets within a flow, or it can be used to give priority to datagrams from certain applications (for example, ICMP packets) over datagrams from other applications (for example, network news).

The IPv6 datagram format is shown in Figure 4.40. As noted above, a comparison of Figure 4.40 with Figure 4.24 reveals the simpler, more streamlined structure of the IPv6 datagram. The following fields are defined in IPv6:

- ◆ *Version.* This four-bit field identifies the IP version number. Not surprisingly, IPv6 carries a value of “6” in this field. Note that putting a “4” in this field does not create a valid IPv4 datagram. (If it did, life would be a lot simpler—see the discussion below regarding the transition from IPv4 to IPv6.)
- ◆ *Traffic class.* This eight-bit field is similar in spirit to the ToS field we saw in IP version 4.
- ◆ *Flow label.* As discussed above, this 20-bit field is used to identify a “flow” of datagrams.
- ◆ *Payload length.* This 16-bit value is treated as an unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte packet header.
- ◆ *Next header.* This field identifies the protocol to which the contents (data field) of this datagram will be delivered (for example, to TCP or UDP). The field uses the same values as the Protocol field in the IPv4 header.
- ◆ *Hop limit.* The contents of this field are decremented by one by each router that forwards the datagram. If the hop limit count reaches zero, the datagram is discarded.

- ◆ *Source and destination address.* The various formats of the IPv6 128-bit address are described in RFC 2373.
- ◆ *Data.* This is the payload portion of the IPv6 datagram. When the datagram reaches its destination, the payload will be removed from the IP datagram and passed on to the protocol specified in the next header field.

The discussion above identified the purpose of the fields that *are* included in the IPv6 datagram. Comparing the IPv6 datagram format in Figure 4.40 with the IPv4 datagram format that we saw earlier in Figure 4.24, we notice that several fields appearing in the IPv4 datagram are no longer present in the IPv6 datagram:

- ◆ *Fragmentation/Reassembly.* IPv6 does not allow for fragmentation and reassembly at intermediate routers; these operations can be performed only by the source and destination. If an IPv6 datagram received by a router is too large to be forwarded over the outgoing link, the router simply drops the datagram and sends a “Packet Too Big” ICMP error message (see below) back to the sender. The sender can then resend the data, using a smaller IP datagram size. Fragmentation and reassembly is a time-consuming operation; removing this functionality from the routers and placing it squarely in the end systems considerably speeds up IP forwarding within the network.
- ◆ *Checksum.* Because the transport layer (for example, TCP and UDP) and data link (for example, Ethernet) protocols in the Internet layers perform checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed. Once again, fast processing of IP packets was a central concern. Recall from our discussion of IPv4 in Section 4.4.1, that since the IPv4 header contains a TTL field (similar to the hop limit field in IPv6), the IPv4 header checksum needed to be recomputed at every router. As with fragmentation and reassembly, this too was a costly operation in IPv4.
- ◆ *Options.* An options field is no longer a part of the standard IP header. However, it has not gone away. Instead, the options field is one of the possible “next headers” pointed to from within the IPv6 header. That is, just as TCP or UDP protocol headers can be the next header within an IP packet, so too can an options field. The removal of the options field results in a fixed length, 40-byte IP header.

A New ICMP for IPv6

Recall from our discussion in Section 4.4, that the ICMP protocol is used by IP nodes to report error conditions and provide limited information (for example, the echo reply to a ping message) to an end system. A new version of ICMP has been defined for IPv6 in RFC 2463. In addition to reorganizing the existing ICMP type and code definitions, ICMPv6 also added new types and codes required by the new IPv6 functionality. These include the “Packet Too Big” type, and an “unrecognized

IPv6 options” error code. In addition, ICMPv6 subsumes the functionality of the Internet Group Management Protocol (IGMP) that we will study in Section 4.8. IGMP, which is used to manage a host’s joining and leaving of so-called multicast groups, was previously a separate protocol from ICMP in IPv4.

4.7.2 Transitioning from IPv4 to IPv6

Now that we have seen the technical details of IPv6, let us consider a very practical matter: how will the public Internet, which is based on IPv4, be transitioned to IPv6? The problem is that while new IPv6-capable systems can be made “backwards compatible,” that is, can send, route, and receive IPv4 datagrams, already deployed IPv4-capable systems are not capable of handling IPv6 datagrams. Several options are possible.

One option would be to declare a “flag day”—a given time and date when all Internet machines would be turned off and upgraded from IPv4 to IPv6. The last major technology transition (from using NCP to using TCP for reliable transport service) occurred almost 20 years ago. Even back then [RFC 801], when the Internet was tiny and still being administered by a small number of “wizards,” it was realized that such a flag day was not possible. A flag day involving hundreds of millions of machines and millions of network administrators and users is even more unthinkable today. RFC 1933 describes two approaches (which can be used either alone or together) for gradually integrating IPv6 hosts and routers into an IPv4 world (with the long-term goal, of course, of having all IPv4 nodes eventually transition to IPv6).

Probably the most straightforward way to introduce IPv6-capable nodes is a **dual-stack** approach, where IPv6 nodes also have a complete IPv4 implementation as well. Such a node, referred to as an IPv6/IPv4 node in RFC 1933, has the ability to send and receive both IPv4 and IPv6 datagrams. When interoperating with an IPv4 node, an IPv6/IPv4 node can use IPv4 datagrams; when interoperating with an IPv6 node, it can speak IPv6. IPv6/IPv4 nodes must have both IPv6 and IPv4 addresses. They must furthermore be able to determine whether another node is IPv6-capable or IPv4-only. This problem can be solved using the DNS (see Chapter 2), which can return an IPv6 address if the node name being resolved is IPv6-capable, or otherwise return an IPv4 address. Of course, if the node issuing the DNS request is only IPv4-capable, the DNS returns only an IPv4 address.

In the dual-stack approach, if either the sender or the receiver is only IPv4-capable, an IPv4 datagram must be used. As a result, it is possible that two IPv6-capable nodes can end up, in essence, sending IPv4 datagrams to each other. This is illustrated in Figure 4.41. Suppose node A is IPv6 capable and wants to send an IP datagram to node F, which is also IPv6-capable. Nodes A and B can exchange an IPv6 packet. However, node B must create an IPv4 datagram to send to C. Certainly, the data field of the IPv6 packet can be copied into the data field of the IPv4 datagram and appropriate address mapping can be done. However, in performing the conversion from IPv6 to IPv4, there will be IPv6-specific fields in the IPv6 datagram (for example, the flow identifier field) that have no counterpart in IPv4. The

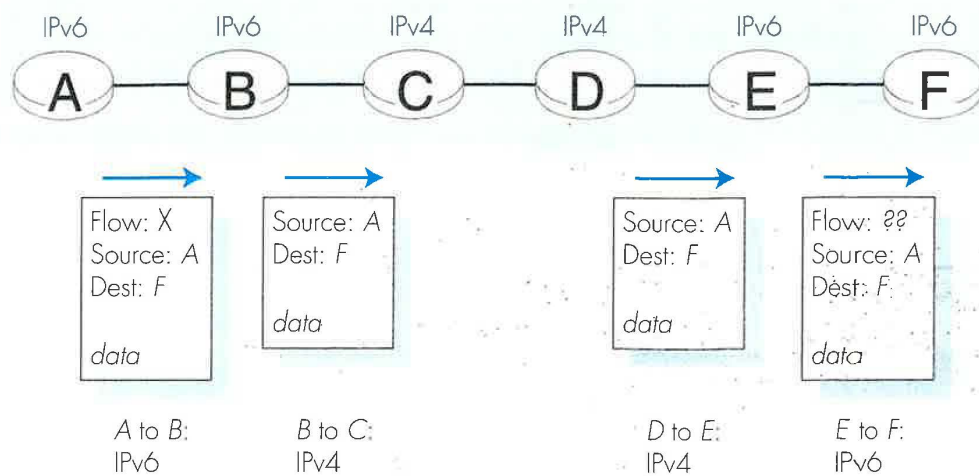


Figure 4.41 ♦ A dual-stack approach

information in these fields will be lost. Thus, even though E and F can exchange IPv6 datagrams, the arriving IPv4 datagrams at E from D do not contain all of the fields that were in the original IPv6 datagram sent from A.

An alternative to the dual-stack approach, also discussed in RFC 1933, is known as **tunneling**. Tunneling can solve the problem noted above, allowing, for example, E to receive the IPv6 datagram originated by A. The basic idea behind tunneling is the following. Suppose two IPv6 nodes (for example, B and E in Figure 4.41) want to interoperate using IPv6 datagrams, but are connected to each other by intervening IPv4 routers. We refer to the intervening set of IPv4 routers between two IPv6 routers as a **tunnel**, as illustrated in Figure 4.42. With tunneling, the IPv6 node on the sending side of the tunnel (for example, B) takes the *entire* IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram. This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel (for example, E) and sent to the first node in the tunnel (for example, C). The intervening IPv4 routers in the tunnel route this IPv4 datagram among themselves, just as they would any other datagram, blissfully unaware that the IPv4 datagram itself contains a complete IPv6 datagram. The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 datagram (it is the destination of the IPv4 datagram!), determines that the IPv4 datagram contains an IPv6 datagram, extracts the IPv6 datagram, and then routes the IPv6 datagram exactly as it would if it had received the IPv6 datagram from a directly connected IPv6 neighbor.

We end this section by mentioning that there is currently some doubt about whether IPv6 will make significant inroads into the Internet in the near future (2000–2002) or even ever at all [Garber 1999]. Indeed, at the time of this writing, a number of North American ISPs have said they don't plan to buy IPv6-enabled

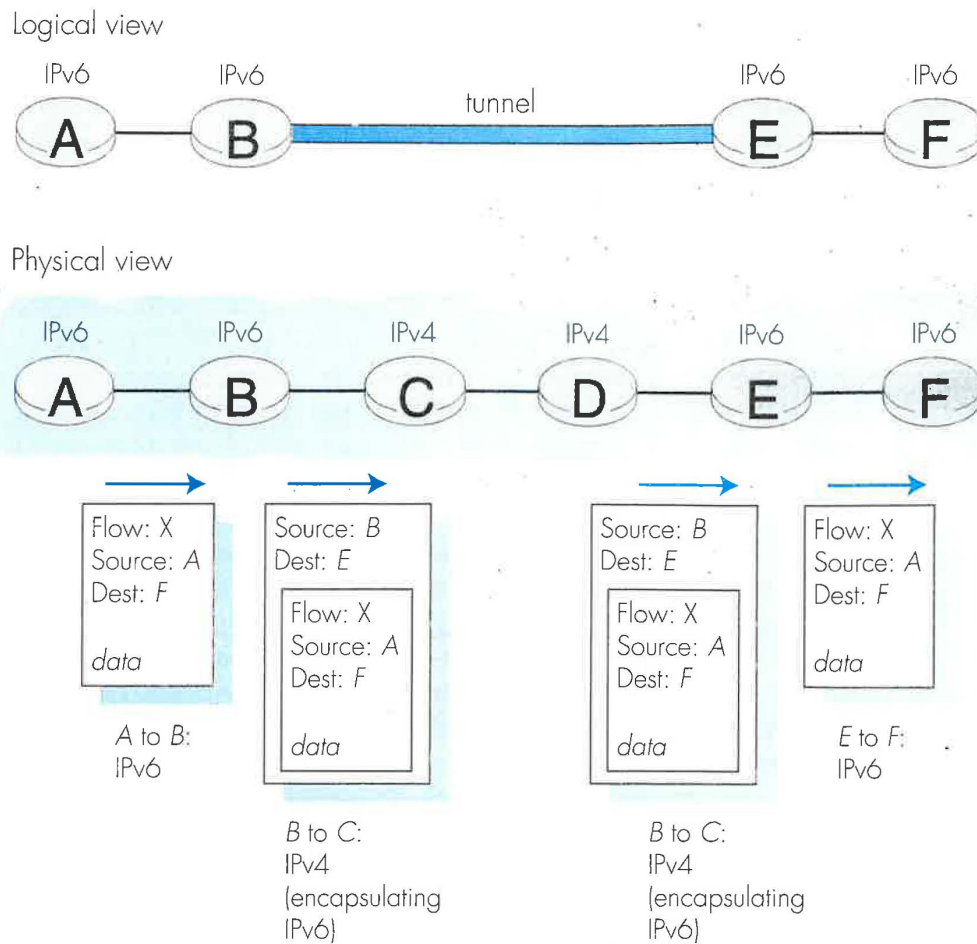


Figure 4.42 ♦ Tunneling

networking equipment. These ISPs say that there is little customer demand for IPv6's capabilities when IPv4, with some patches (such as CIDR, see Section 4.4.1, and network address translator [RFC 1631] boxes), is working well enough. On the other hand, there appears to be more interest in IPv6 in Europe and Asia.

One important lesson that we can learn from the IPv6 experience is that it is enormously difficult to change network-layer protocols. Since the early 1990s, numerous new network-layer protocols have been trumpeted as the next major revolution for the Internet, but most of these protocols have had limited penetration to date. These protocols include IPv6, multicast protocols (Section 4.8), and resource reservation protocols (Section 6.9). Indeed, introducing new protocols into the network layer is like replacing the foundation of a house—it is difficult to do without tearing the whole house down or at least temporarily relocating the house's residents. On the other hand, the Internet has witnessed rapid deployment of new protocols at the application layer.

The classic example, of course, is HTTP and the Web; other examples include audio and video streaming and chat. Introducing new application layer protocols is like adding a new layer of paint to a house—it is relatively easy to do, and if you choose an attractive color, others in the neighborhood will copy you. In summary, in the future we can expect to see changes in the Internet's network layer, but these changes will likely occur on a time scale that is much slower than the changes that will occur at the application layer.

4.8 ♦ Multicast Routing

The transport- and network-layer protocols we have studied so far provide for the delivery of packets from a single source to a single destination. Protocols involving just one sender and one receiver are often referred to as **unicast protocols**.

A number of emerging network applications require the delivery of packets from one or more senders to a *group of receivers*. These applications include bulk data transfer (for example, the transfer of a software upgrade from the software developer to users needing the upgrade), streaming continuous media (for example, the transfer of the audio, video, and text of a live lecture to a set of distributed lecture participants), shared data applications (for example, a whiteboard or teleconferencing application that is shared among many distributed participants), data feeds (for example, stock quotes), WWW cache updating, and interactive gaming (for example, distributed interactive virtual environments or multiplayer games such as Quake). For each of these applications, an extremely useful abstraction is the notion of a **multicast**: the sending of a packet from one sender to multiple receivers with a single send operation.

In this section we consider the network-layer aspects of multicast. We continue our primary focus on the Internet here, as multicast is much more mature (although it is still undergoing significant development and evolution) in the Internet than in ATM networks. We will see that as in the unicast case, routing algorithms again play a central role in the network layer. We will also see, however, that unlike the unicast case, Internet multicast is *not* a connectionless service—state information for a multicast connection must be established and maintained in routers that handle multicast packets sent among hosts in a so-called multicast group. This, in turn, will require a combination of signaling and routing protocols in order to set up, maintain, and tear down connection state in the routers.

4.8.1 Introduction: The Internet Multicast Abstraction and Multicast Groups

From a networking standpoint, the multicast abstraction—a single send operation that results in copies of the sent data being delivered to many receivers—can be im-