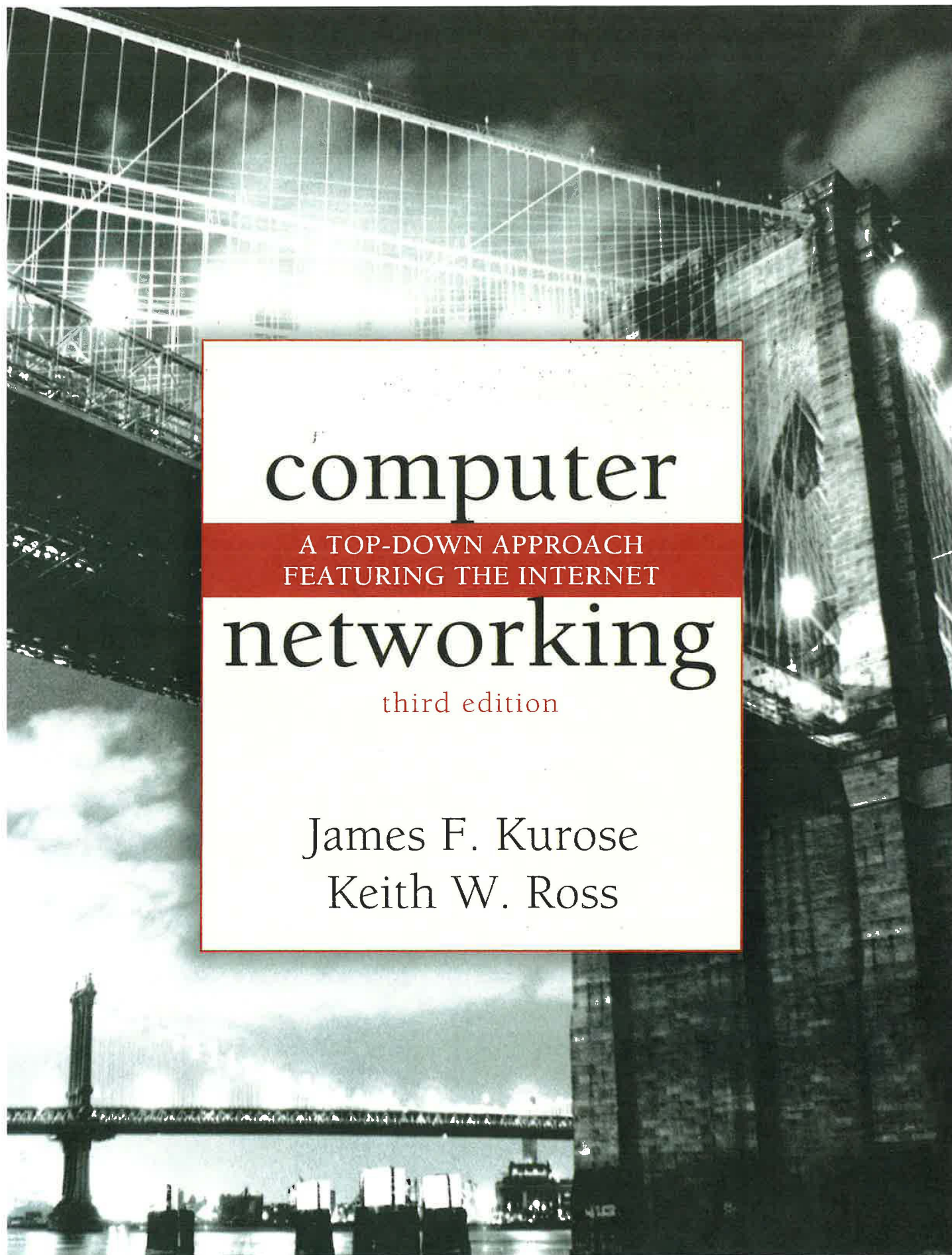




THIRD EDITION

Computer Networking

A Top-Down Approach Featuring the Internet

James F. Kurose

University of Massachusetts, Amherst



Keith W. Ross

Polytechnic University, Brooklyn



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

Managing Editor	Patty Mahtani
Executive Editor	Susan Hartman Sullivan
Assistant Editor	Elizabeth Paquin
Marketing Manager	Michelle Brown
Production Supervisor	Marilyn Lloyd
Project Management	Argosy Publishing, Inc.
Composition	Argosy Publishing, Inc.
Art	Pat Rossi Calkin and Argosy Publishing, Inc.
Art Development	Janet Theurer
Text and Cover Design	Joyce Cosentino Wells
Prepress and Manufacturing	Caroline Fell
Cover Photo:	©2004 Photodisc

Access the latest information about Addison-Wesley Computing titles from our World Wide Web site: <http://www.aw-bc.com/computing>

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

Kurose, James F

Computer networking : a top-down approach featuring the Internet / James F. Kurose, Keith W. Ross.—3rd ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-321-22735-2

1. Internet. 2. Computer networks I. Ross, Keith W., 1956- II. Title.

TK5105.875.I57 K88 2005

004.67'8—dc22

2004044284

Copyright © 2005 by Pearson Education, 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.

ISBN 0-321-22735-2

1 2 3 4 5 6 7 8 9 10-CRW-08 07 06 05 04

Table of Contents

Chapter 1	Computer Networks and the Internet	1
1.1	What Is the Internet?	2
1.1.1	A Nuts-and-Bolts Description	2
1.1.2	A Service Description	5
1.1.3	What Is a Protocol?	6
1.2	The Network Edge	9
1.2.1	End Systems, Clients, and Servers	9
1.2.2	Connectionless and Connection-Oriented Service	11
1.3	The Network Core	14
1.3.1	Circuit Switching and Packet Switching	14
1.3.2	Packet-Switched Networks: Datagram Networks and Virtual-Circuit Networks	21
1.4	Access Networks and Physical Media	24
1.4.1	Access Networks	25
1.4.2	Physical Media	31
1.5	ISPs and Internet Backbones	34
1.6	Delay and Loss in Packet-Switched Networks	37
1.6.1	Types of Delay	37
1.6.2	Queuing Delay and Packet Loss	40
1.6.3	Delay and Routes in the Internet	43
1.7	Protocol Layers and Their Service Models	44
1.7.1	Layered Architecture	45
1.7.2	Layers, Messages, Segments, Datagrams, and Frames	50
1.8	History of Computer Networking and the Internet	52
1.8.1	The Development of Packet Switching: 1961–1972	52
1.8.2	Proprietary Networks and Internetworking: 1972–1980	53
1.8.3	A Proliferation of Networks: 1980–1990	56
1.8.4	The Internet Explosion: The 1990s	56
1.8.5	Recent Developments	58
1.9	Summary	59
	Road-Mapping This Book	60
	Homework Problems and Questions	61

Problems	62
Discussion Questions	68
Ethereal Lab 1	69
Interview: Leonard Kleinrock	71

Chapter 2 Application Layer 73

2.1 Principles of Network Applications	74
2.1.1 Network Application Architectures	75
2.1.2 Processes Communicating	78
2.1.3 Application-Layer Protocols	81
2.1.4 What Services Does an Application Need?	82
2.1.5 Services Provided by the Internet Transport Protocols	84
2.1.6 Network Applications Covered in This Book	87
2.2 The Web and HTTP	87
2.2.1 Overview of HTTP	88
2.2.2 Nonpersistent and Persistent Connections	90
2.2.3 HTTP Message Format	93
2.2.4 User-Server Interaction: Cookies	98
2.2.5 HTTP Content	100
2.2.6 Web Caching	100
2.2.7 The Conditional GET	105
2.3 File Transfer: FTP	106
2.3.1 FTP Commands and Replies	108
2.4 Electronic Mail in the Internet	109
2.4.1 SMTP	112
2.4.2 Comparison with HTTP	115
2.4.3 Mail Message Formats and MIME	115
2.4.4 Mail Access Protocols	118
2.5 DNS—The Internet's Directory Service	123
2.5.1 Services Provided by DNS	123
2.5.2 Overview of How DNS Works	126
2.5.3 DNS Records and Messages	132
2.6 P2P File Sharing	136
2.7 Socket Programming with TCP	146
2.7.1 Socket Programming with TCP	147
2.7.2 An Example Client/Server Application in Java	149
2.8 Socket Programming with UDP	156
2.9 Building a Simple Web Server	164
2.9.1 Web Server Functions	164
2.10 Summary	169
Homework Problems and Questions	170

Problems	171
Discussion Questions	177
Socket Programming Assignments	178
Ethereal Labs	180
Interview: Tim Berners-Lee	181

Chapter 3 Transport Layer 183

3.1	Introduction and Transport-Layer Services	184
3.1.1	Relationship Between Transport and Network Layers	184
3.1.2	Overview of the Transport Layer in the Internet	187
3.2	Multiplexing and Demultiplexing	189
3.3	Connectionless Transport: UDP	196
3.3.1	UDP Segment Structure	199
3.3.2	UDP Checksum	200
3.4	Principles of Reliable Data Transfer	201
3.4.1	Building a Reliable Data Transfer Protocol	203
3.4.2	Pipelined Reliable Data Transfer Protocols	214
3.4.3	Go-Back-N (GBN)	217
3.4.4	Selective Repeat (SR)	221
3.5	Connection-Oriented Transport: TCP	228
3.5.1	The TCP Connection	228
3.5.2	TCP Segment Structure	231
3.5.3	Round-Trip Time Estimation and Timeout	236
3.5.4	Reliable Data Transfer	239
3.5.5	Flow Control	246
3.5.6	TCP Connection Management	249
3.6	Principles of Congestion Control	254
3.6.1	The Causes and the Costs of Congestion	254
3.6.2	Approaches to Congestion Control	260
3.6.3	Network-Assisted Congestion-Control Example: ATM ABR Congestion Control	261
3.7	TCP Congestion Control	264
3.7.1	Fairness	271
3.7.2	TCP Delay Modeling	275
3.8	Summary	284
	Homework Problems and Questions	285
	Problems	287
	Discussion Questions	294
	Programming Assignments	295
	Ethereal Lab: Exploring TCP	295
	Interview: Sally Floyd	297

Chapter 4	The Network Layer	299
4.1	Introduction	300
4.1.1	Forwarding and Routing	301
4.1.2	Network Service Models	304
4.2	Virtual Circuit and Datagram Networks	307
4.2.1	Virtual-Circuit Networks	307
4.2.2	Datagram Networks	310
4.2.3	Origins of VC and Datagram Networks	313
4.3	What's Inside a Router?	314
4.3.1	Input Ports	315
4.3.2	Switching Fabric	318
4.3.3	Output Ports	320
4.3.4	Where Does Queuing Occur?	320
4.4	The Internet Protocol (IP): Forwarding and Addressing in the Internet	323
4.4.1	Datagram Format	325
4.4.2	IPv4 Addressing	331
4.4.3	Internet Control Message Protocol (ICMP)	342
4.4.4	IPv6	344
4.5	Routing Algorithms	351
4.5.1	The Link-State (LS) Routing Algorithm	354
4.5.2	The Distance-Vector (DV) Routing Algorithm	358
4.5.3	Hierarchical Routing	366
4.6	Routing in the Internet	370
4.6.1	Intra-AS Routing in the Internet: RIP	371
4.6.2	Intra-AS Routing in the Internet: OSPF	374
4.6.3	Inter-Autonomous System Routing: BGP	378
4.7	Broadcast and Multicast Routing	384
4.7.1	Broadcast Routing Algorithms	385
4.7.2	Multicast	391
4.8	Summary	399
	Homework Problems and Questions	400
	Problems	403
	Discussion Questions	412
	Programming Assignment	412
	Ethereal Lab	413
	Interview: Vinton G. Cerf	414
Chapter 5	The Link Layer and Local Area Networks	417
5.1	Link Layer: Introduction and Services	419
5.1.1	The Services Provided by the Link Layer	419
5.1.2	Adapters Communicating	422

5.2	Error-Detection and -Correction Techniques	423
5.2.1	Parity Checks	425
5.2.2	Checksumming Methods	427
5.2.3	Cyclic Redundancy Check (CRC)	428
5.3	Multiple Access Protocols	430
5.3.1	Channel Partitioning Protocols	433
5.3.2	Random Access Protocols	435
5.3.3	Taking-Turns Protocols	442
5.3.4	Local Area Networks (LANs)	443
5.4	Link-Layer Addressing	445
5.4.1	MAC Addresses	445
5.4.2	Address Resolution Protocol (ARP)	447
5.4.3	Dynamic Host Configuration Protocol	451
5.5	Ethernet	455
5.5.1	Ethernet Frame Structure	456
5.5.2	CSMA/CD: Ethernet's Multiple Access Protocol	460
5.5.3	Ethernet Technologies	463
5.6	Interconnections: Hubs and Switches	465
5.6.1	Hubs	465
5.6.2	Link-Layer Switches	467
5.7	PPP: The Point-to-Point Protocol	477
5.7.1	PPP Data Framing	479
5.7.2	PPP Link-Control Protocol (LCP) and Network-Control Protocols	480
5.8	Link Virtualization: A Network as a Link Layer	482
5.8.1	Asynchronous Transfer Mode (ATM) Networks	483
5.8.2	Multiprotocol Label Switching (MPLS)	488
5.9	Summary	491
	Homework Problems and Questions	493
	Problems	494
	Discussion Questions	498
	Ethereal Lab	499
	Interview: Simon S. Lam	500

Chapter 6 Wireless and Mobile Networks 503

6.1	Introduction	504
6.2	Wireless Links and Network Characteristics	508
6.2.1	CDMA	509
6.3	Wi-Fi: 802.11 Wireless LANs	513
6.3.1	The 802.11 Architecture	514
6.3.2	The 802.11 MAC Protocol	517
6.3.3	The IEEE 802.11 Frame	523

6.3.4	Mobility in the Same IP Subnet	526
6.3.5	802.15 and Bluetooth	528
6.4	Cellular Internet Access	529
6.4.1	An Overview of Cellular Architecture	531
6.4.2	Cellular Standards and Technologies: A Brief Survey	532
6.5	Mobility Management: Principles	536
6.5.1	Addressing	538
6.5.2	Routing to a Mobile Node	540
6.6	Mobile IP	546
6.7	Managing Mobility in Cellular Networks	551
6.7.1	Routing Calls to a Mobile User	552
6.7.2	Handoffs in GSM	553
6.8	Wireless and Mobility: Impact on Higher-layer Protocols	556
6.9	Summary	559
	Homework Problems and Questions	559
	Problems	560
	Discussion Questions	562
	Ethereal Lab	562
	Interview: Charlie Perkins	563

Chapter 7 Multimedia Networking 565

7.1	Multimedia Networking Applications	566
7.1.1	Examples of Multimedia Applications	566
7.1.2	Hurdles for Multimedia in Today's Internet	569
7.1.3	How Should the Internet Evolve to Support Multimedia Better?	571
7.1.4	Audio and Video Compression	572
7.2	Streaming Stored Audio and Video	574
7.2.1	Accessing Audio and Video Through a Web Server	576
7.2.2	Sending Multimedia from a Streaming Server to a Helper Application	578
7.2.3	Real-Time Streaming Protocol (RTSP)	580
7.3	Making the Best of the Best-Effort Service: An Internet Phone Example	584
7.3.1	The Limitations of a Best-Effort Service	585
7.3.2	Removing Jitter at the Receiver for Audio	587
7.3.3	Recovering from Packet Loss	590
7.3.4	Streaming Stored Audio and Video	594
7.4	Protocols for Real-Time Interactive Applications	594
7.4.1	RTP	594
7.4.2	RTP Control Protocol (RTCP)	599
7.4.3	SIP	602
7.4.4	H.323	608

7.5	Distributing Multimedia: Content Distribution Networks	610
7.6	Beyond Best Effort	614
7.6.1	Scenario 1: A 1 Mbps Audio Application and an FTP Transfer	615
7.6.2	Scenario 2: A 1 Mbps Audio Application and a High-Priority FTP Transfer	616
7.6.3	Scenario 3: A Misbehaving Audio Application and an FTP Transfer	617
7.6.4	Scenario 4: Two 1 Mbps Audio Applications over an Overloaded 1.5 Mbps Link	619
7.7	Scheduling and Policing Mechanisms	620
7.7.1	Scheduling Mechanisms	621
7.7.2	Policing: The Leaky Bucket	625
7.8	Integrated Services and Differentiated Services	628
7.8.1	Intserv	628
7.8.2	Diffserv	631
7.9	RSVP	636
7.9.1	The Essence of RSVP	637
7.9.2	A Few Simple Examples	639
7.10	Summary	643
	Homework Problems and Questions	644
	Problems	645
	Discussion Questions	649
	Programming Assignment	649
	Interview: Henning Schulzrinne	651

Chapter 8 Security in Computer Networks 653

8.1	What Is Network Security?	654
8.2	Principles of Cryptography	657
8.2.1	Symmetric Key Cryptography	660
8.2.2	Public Key Encryption	664
8.3	Authentication	670
8.3.1	Authentication Protocol <i>ap1.0</i>	670
8.3.2	Authentication Protocol <i>ap2.0</i>	671
8.3.3	Authentication Protocol <i>ap3.0</i>	672
8.3.4	Authentication Protocol <i>ap3.1</i>	672
8.3.5	Authentication Protocol <i>ap4.0</i>	673
8.3.6	Authentication Protocol <i>ap5.0</i>	674
8.4	Integrity	678
8.4.1	Generating Digital Signatures	678
8.4.2	Message Digests	679
8.4.3	Hash Function Algorithms	681

8.5	Key Distribution and Certification	684
8.5.1	The Key Distribution Center	686
8.5.2	Public Key Certification	687
8.6	Access Control: Firewalls	691
8.6.1	Packet Filtering	692
8.6.2	Application Gateway	695
8.7	Attacks and Countermeasures	697
8.7.1	Mapping	697
8.7.2	Packet Sniffing	698
8.7.3	Spoofing	699
8.7.4	Denial-of-Service and Distributed Denial-of-Service Attacks	700
8.7.5	Hijacking	701
8.8	Security in Many Layers: Case Studies	702
8.8.1	Secure E-mail	703
8.8.2	Secure Sockets Layer (SSL) and Transport Layer Security (TLS)	708
8.8.3	Network-Layer Security: IPsec	712
8.8.4	Security in IEEE 802.11	716
8.9	Summary	721
	Homework Problems and Questions	722
	Problems	723
	Discussion Questions	725
	Interview: Steven M. Bellovin	726

Chapter 9 Network Management 729

9.1	What Is Network Management?	730
9.2	The Infrastructure for Network Management	734
9.3	The Internet-Standard Management Framework	738
9.3.1	Structure of Management Information: SMI	740
9.3.2	Management Information Base: MIB	743
9.3.3	SNMP Protocol Operations and Transport Mappings	745
9.3.4	Security and Administration	749
9.4	ASN.1	753
9.5	Conclusion	757
	Homework Problems and Questions	758
	Problems	759
	Discussion Questions	760
	Interview: Jeff Case	761

References	763
-------------------	------------

Index	797
--------------	------------

hotel room with a Web TV, and so on. As with IMAP, users can organize their messages in a hierarchy of folders on the remote server. In fact, many implementations of Web-based e-mail use an IMAP server to provide the folder functionality. In this case, access to the folders and messages is provided with scripts that run in an HTTP server; the scripts use the IMAP protocol to communicate with an IMAP server.

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) prefer 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 `cis.poly.edu`—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 `www.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 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 the host is located in the Internet (that is, within which network, in the network of networks). Similarly, when we scan a postal address from bottom to top, we obtain more and more specific information about where the addressee 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 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 **DNS servers** and (2) an application-layer protocol that allows hosts to query the distributed database. The DNS servers are often UNIX machines running the Berkeley Internet Name Domain (BIND) software [BIND 2004]. The DNS protocol runs over UDP and uses port 53.

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 host, requests the URL `www.someschool.edu/index.html`. In order for the user's host to be able to send an HTTP request message to the Web server `www.someschool.edu`, the user's host must first obtain the IP address of `www.someschool.edu`. This is done as follows.

1. The same user machine runs the client side of the DNS application.
2. The browser extracts the hostname, `www.someschool.edu`, from the URL and passes the hostname to the client side of the DNS application.
3. The DNS client sends a query containing the hostname to a DNS server.
4. The DNS client eventually receives a reply, which includes the IP address for the hostname.
5. Once the browser receives the IP address from DNS, it can initiate 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 it. Fortunately, as we discuss below, the desired IP address is often cached in a “nearby” DNS server, which helps to reduce DNS network traffic as well as the average DNS delay.

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 canonical hostnames. 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

 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 systems 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 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.

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, the MX record (see below) 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.** DNS is also 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 each 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 the replicated servers. DNS rotation is also used for e-mail so that multiple mail servers can have the same alias name. Recently, content distribution companies such as Akamai [Akamai 2004] have used DNS in more sophisticated ways to provide Web content distribution (see Chapter 7).

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 the book by Abitz

and Liu [Abitz 1993]; see also the retrospective paper [Mockapetris 1988], which provides a nice description of the what and why of DNS.

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.

Suppose that some application (such as a Web browser or a mail reader) running in a user's host needs to translate a hostname to an IP address. The application will invoke the client side of DNS, specifying the hostname that needs to be translated. (On many UNIX-based machines, `gethostbyname()` is the function call that an application calls in order to perform the translation. In Section 2.7, we will show how a Java application can invoke DNS.) DNS in the user's host then takes over, sending a query message into the network. All DNS query and reply messages are sent within UDP datagrams to port 53. After a delay, ranging from milliseconds to seconds, DNS in the user's host receives a DNS reply message that provides the desired mapping. This mapping is then passed to the invoking application. Thus, from the perspective of the invoking application in the user's host, DNS is a black box providing a simple, straightforward translation service. But in fact, the black box that implements the service is complex, consisting of a large number of DNS servers distributed around the globe, as well as an application-layer protocol that specifies how the DNS servers and querying hosts communicate.

A simple design for DNS would have one DNS server that contains all the mappings. In this centralized design, clients simply direct all queries to the single DNS server, and the DNS server responds directly to the querying clients. Although the simplicity of this design is attractive, it is 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 DNS server crashes, so does the entire Internet!
- ◆ **Traffic volume.** A single DNS server would have to handle all DNS queries (for all the HTTP requests and e-mail messages generated from hundreds of millions of hosts).
- ◆ **Distant centralized database.** A single DNS server cannot be "close to" all the querying clients. If we put the single DNS 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
- ◆ **Maintenance.** The single DNS 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.

In summary, a centralized database in a single DNS 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.

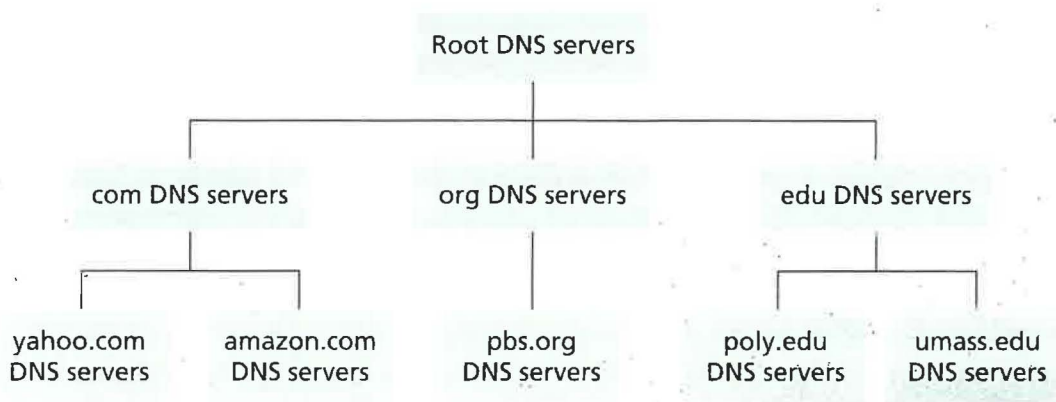


Figure 2.18 ♦ Portion of the hierarchy of DNS servers

A Distributed, Hierarchical Database

In order to deal with the issue of scale, the DNS uses a large number of servers, organized in a hierarchical fashion and distributed around the world. No single DNS server has all of the mappings for all of the hosts in the Internet. Instead, the mappings are distributed across the DNS servers. To a first approximation, there are three classes of DNS servers—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers—organized in a hierarchy as shown in Figure 2.18. To understand how these three classes of servers interact, suppose a DNS client wants to determine the IP address for the hostname `www.amazon.com`. To a first approximation, the following events will take place. The client first contacts one of the root servers, which returns IP addresses for TLD servers for the top-level domain `com`. The client then contacts one of these TLD servers, which returns the IP address of an authoritative server for `amazon.com`. Finally, the client contacts one of the authoritative servers for `amazon.com`, which returns the IP address for the hostname `www.amazon.com`. We'll soon examine this DNS lookup process in more detail. But let's first take a closer look at these three classes of DNS servers:

- ♦ **Root DNS servers.** In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America. A February 2004 map of the root DNS servers is shown in Figure 2.19; a list of the current root DNS servers is available via [Root-servers 2004]. Although we have referred to each of the 13 root DNS servers as if it were a single server, each “server” is actually a cluster of replicated servers, for both security and reliability purposes.
- ♦ **Top-Level Domain (TLD) servers.** These servers are responsible for top-level domains such as `com`, `org`, `net`, `edu`, and `gov`, and all of the country top-level domains such as `uk`, `fr`, `ca`, and `jp`. As of this writing (spring, 2004), the company Network Solutions maintains the TLD servers for the `com` top-level domain, and the company Educause maintains the TLD servers for the `edu` top-level domain.

◆ **Authoritative DNS servers.** Every organization with publicly accessible hosts (such as Web servers and mail servers) on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses. An organization's authoritative DNS server houses these DNS records. An organization can choose to implement its own authoritative DNS server to hold these records; alternatively, the organization can pay to have these records stored in an authoritative DNS server of some service provider. Most universities and large companies implement and maintain their own primary and secondary (backup) authoritative DNS server.

The root, TLD, and authoritative DNS servers all belong to the hierarchy of DNS servers, as shown in Figure 2.18. There is another important type of DNS, called the **local DNS server**. A local DNS server does not strictly belong to the hierarchy of servers but is nevertheless central to the DNS architecture. Each ISP—such as a university, an academic department, an employee's company, or a residential ISP—has a local DNS server (also called a default name server). When a host connects to an ISP, the ISP provides the host with the IP addresses of one or more of its local DNS servers (typically through DHCP, which is discussed in Section 5.4.3). You can easily determine the IP address of your local DNS server by accessing network status windows in Windows or UNIX. A host's local DNS server is typically “close to” the host. For an institutional ISP, the local DNS server may be on the same LAN as the host; for a residential ISP, it is typically separated from the host by



Figure 2.19 ◆ DNS root servers in 2004 (name, organization, location)

no more than a few routers. When a host makes a DNS query, the query is sent to the local DNS server, which acts a proxy, forwarding the query into the DNS server hierarchy as we'll discuss in more detail below.

Let's take a look at a simple example. Suppose the host `cis.poly.edu` desires the IP address of `gaia.cs.umass.edu`. Also suppose that Polytechnic's local DNS server is called `dns.poly.edu` and that an authoritative DNS server for `gaia.cs.umass.edu` is called `dns.umass.edu`. As shown in Figure 2.20, the host `cis.poly.edu` first sends a DNS query message to its local DNS server, `dns.poly.edu`. The query message contains the hostname to be translated, namely, `gaia.cs.umass.edu`. The local DNS server forwards the query message to a root DNS server. The root DNS server takes note of the `edu` suffix and returns to the local DNS server a list of IP addresses for TLD servers responsible for `edu`. The local DNS server then resends the query message to one of these TLD servers. The TLD server takes note of the `umass.edu` suffix and responds with the

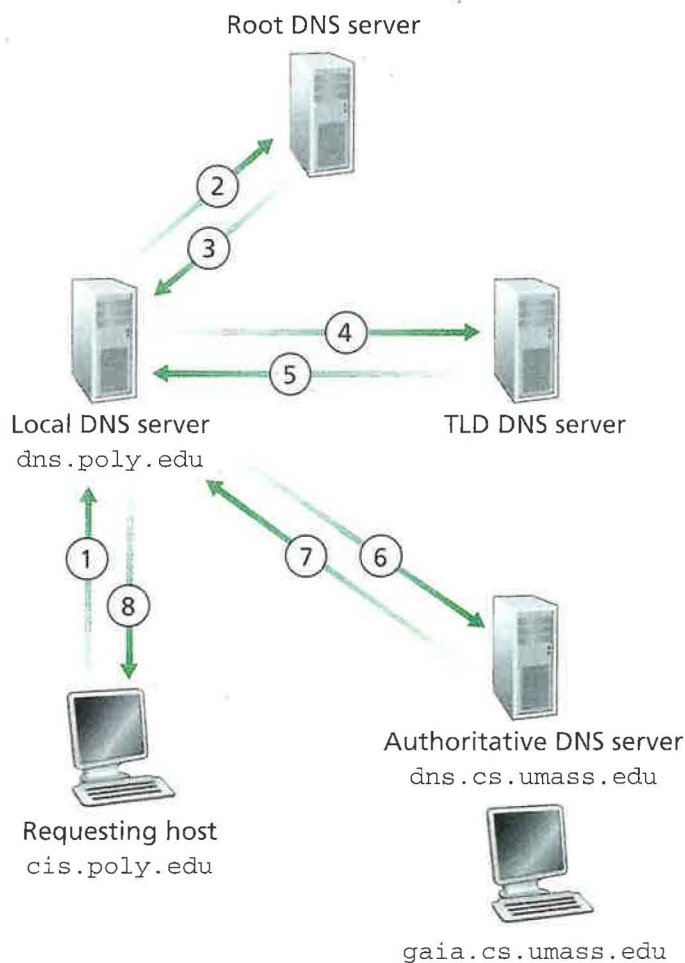


Figure 2.20 ♦ Interaction of the various DNS servers

IP address of the authoritative DNS server for the University of Massachusetts, namely, `dns.umass.edu`. Finally, the local DNS server resends the query message directly to `dns.umass.edu`, which responds with the IP address of `gaia.cs.umass.edu`. Note that in this example, in order to obtain the mapping for one hostname, eight DNS messages were sent: four query messages and four reply messages! We'll soon see how DNS caching reduces this query traffic.

Our previous example assumed that the TLD server knows the authoritative DNS server for the hostname. In general this is not always true. Instead, the TLD server may know only of an intermediate DNS server, which in turn knows the authoritative DNS server for the hostname. For example, suppose again that the University of Massachusetts has a DNS server for the university, called `dns.umass.edu`. Also suppose that each of the departments at the University of Massachusetts has its own DNS server, and that each departmental DNS server is authoritative for all hosts in the department. In this case, when the intermediate DNS server, `dns.umass.edu`, receives a query for a host with hostname ending with `cs.umass.edu`, it returns to `dns.poly.edu` the IP address of `dns.cs.umass.edu`, which is authoritative for all hostnames ending with `cs.umass.edu`. The local DNS server `dns.poly.edu` then sends the query to the authoritative DNS server, which returns the desired mapping to local DNS server, which in turn returns the mapping to the requesting host. In this case, a total of 10 DNS messages are sent!

The example shown in Figure 2.20 makes use of both **recursive queries** and **iterative queries**. The query sent from `cis.poly.edu` to `dns.poly.edu` is a recursive query, since the query asks `dns.poly.edu` to obtain the mapping on its behalf. But the subsequent three queries are iterative since all of the replies are directly returned to `dns.poly.edu`. In theory, any DNS query can be iterative or recursive. For example, Figure 2.21 shows a DNS query chain for which all of the queries are recursive. In practice, the queries typically follow the pattern in Figure 2.20: The query from the requesting host to the local DNS server is recursive, and the remaining queries are iterative.

DNS Caching

Our discussion thus far has ignored **DNS caching**, a critically important feature of the DNS system. In truth, DNS extensively exploits DNS caching in order to improve the delay performance and to reduce the number of DNS messages ricocheting around the Internet. The idea behind DNS caching is very simple. In a query chain, when a DNS server receives a DNS reply (containing, for example a mapping from a hostname to an IP address), it can cache the information in the reply in its local memory. For example, in Figure 2.20, each time the local DNS server `dns.poly.edu` receives a reply from some DNS server, it can cache any of the information contained in the reply. If a hostname/IP address pair is cached in a DNS server and another query arrives to the DNS server for the same hostname, the DNS server can provide the desired IP address, even

2.5.3 DNS Records and Messages

The DNS servers that together implement the DNS distributed database store **resource records (RR)**, including RRs that provide 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)

TTL is the time to live of the resource record; it determines when a resource should be removed from a cache. In the example records given below, we 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 DNS 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 the canonical name 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. Note that by using the MX record, a company can have the same aliased name for its mail server and for one of its other servers (such as its Web server). To obtain the canonical name for the mail server, a DNS client would query for an MX record; to obtain the canonical name for the other server, the DNS client would query for the CNAME record.

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

DNS Messages

Earlier in this section we referred to DNS query and reply messages. These are the only two kinds of DNS messages. Furthermore, both query and reply messages have the same format, as shown in Figure 2.22. The semantics of the various fields in a DNS message are 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 1-bit query/reply flag indicates whether the message is a query (0) or a reply (1). A 1-bit authoritative flag is set in a reply message when a DNS server is an authoritative server for a queried name. A 1-bit recursion-desired flag is set when a client (host or DNS server) desires that the DNS server perform recursion when it doesn't have the record. A 1-bit recursion-available field is set in a reply if the DNS 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 DNS 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, CNAME, 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 contains a resource record providing the canonical hostname of a mail server. The additional section contains a Type A record providing the IP address for the canonical hostname of the mail server.

How would you like to send a DNS query message directly from the host you're working on to some DNS server? This can easily be with the **nslookup program**, which is available from most Windows and UNIX platforms. For example, from a Windows host, open the Command Prompt and invoke the nslookup program by simply typing "nslookup". After invoking nslookup, you can send a DNS query to any DNS server (root, TLD, or authoritative). After receiving the reply message from the DNS server, nslookup will display the records included in the reply (in a human readable format). As an alternative to running nslookup from your own host, you can visit one of many Web sites that allow you to remotely employ nslookup. (Just type "nslookup" into a search engine and you'll be brought to one of these sites.)

Inserting Records into the DNS Database

The discussion above focused on how records are retrieved from the DNS database. You might be wondering how records get into the database in the first place. Let's look at how this is done in the context of a specific example. Suppose you have just created an exciting new startup company called Network Utopia. The first thing you'll surely want to do is register the domain name `networkutopia.com` at a registrar. A **registrar** is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database (as discussed below), and collects a small fee from you for its services. Prior to 1999, a single registrar,

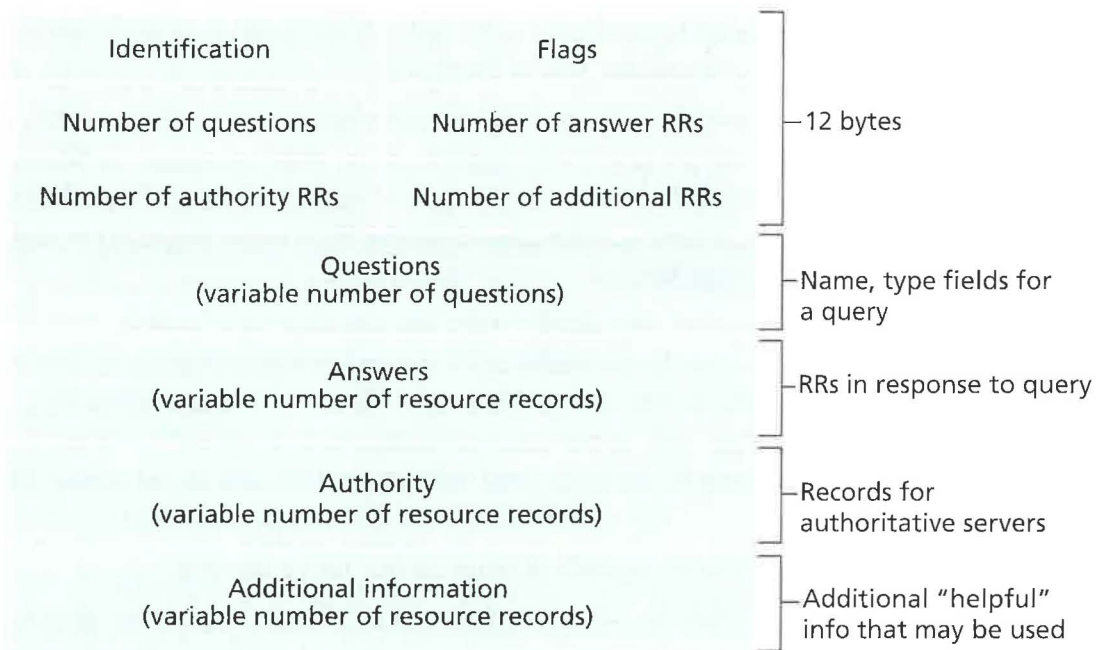


Figure 2.22 ♦ DNS message format

Network Solutions, had a monopoly on domain name registration for `com`, `net`, and `org` domains. But now there are many registrars competing for customers, and the Internet Corporation for Assigned Names and Numbers (ICANN) accredits the various registrars. A complete list of accredited registrars is available at <http://www.intenic.net>.

When you register the domain name `networkutopia.com` with some registrar, you also need to provide the registrar with the names and IP addresses of your primary and secondary authoritative DNS servers. Suppose the names and IP addresses are `dns1.networkutopia.com`, `dns2.networkutopia.com`, `212.212.212.1` and `212.212.212.2`. For each of these two authoritative DNS servers, the registrar would then make sure that a type NS and a type A record are entered into the TLD `com` servers. Specifically, for the primary authoritative server for `networkutopia.com`, the registrar would insert into the DNS system the following two resource records:

```
(networkutopia.com, dns1.networkutopia.com, NS)
```

```
(dns1.networkutopia.com, 212.212.212.1, A)
```

You'll also have to make sure that the Type A resource record for your Web server `www.networkutopia.com` and the Type MX resource record for your mail server `mail.networkutopia.com` are entered into your authoritative DNS servers. (Until recently, the contents of each DNS server were 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.)

Once all of these steps are completed, people will be able to visit your Web site and send e-mail to the employees at your company. Let's conclude our discussion of DNS by verifying that this statement is true. This verification also helps to solidify what we have learned about DNS. Suppose Alice in Australia wants to view the Web page `www.networkutopia.com`. As discussed earlier, her host will first send a DNS query to her local DNS server. The local DNS server will then contact a TLD `com` server. (The local DNS server will also have to contact a root DNS server if the address of a TLD `com` server is not cached.) This TLD server contains the type NS and Type A resource records listed above, because the registrar had these resource records inserted into all of the TLD `com` servers. The TLD `com` server sends a reply to Alice's local DNS server, with the reply containing the two resource records. The local DNS server then sends a DNS query to `212.212.212.1`, asking for the Type A record corresponding to `www.networkutopia.com`. This record provides the IP address of the desired Web server, say, `212.212.71.4`, which the local DNS server passes back to Alice's host. Alice's browser can now initiate a TCP connection to the host `212.212.71.4` and send an HTTP request over the connection. Whew! There's a lot more going on than what meets the eye when one surfs the Web!

4.4.2 IPv4 Addressing

We now turn our attention to IPv4 addressing. Although you may be thinking that addressing must be a straightforward topic, hopefully by the end of this chapter you'll be convinced that Internet addressing is not only a juicy, subtle, and interesting topic but also one that is of central importance to the Internet. Excellent treatments of IPv4 addressing are [Semeria 1996] and the first chapter in [Stewart 1999].

Before discussing IP addressing, however, we'll need to say a few words about how hosts and routers are connected into the network. A host typically has only a single link into the network; when IP in the host wants to send a datagram, it does so over this link. The boundary between the host and the physical link is called an **interface**. Now consider a router and its interfaces. Because a router's job is to receive a datagram on one link and forward the datagram on some other link, a router necessarily has two or more links to which it is connected. The boundary between the router and any one of its links is also called an interface. A router thus has multiple interfaces, one for each of its links. Because every host and router is capable of sending and receiving IP datagrams, IP requires each host and router interface to have its own IP address. Thus, an IP address is technically associated with an interface, rather than with the host or router containing that interface.

Each IP address is 32 bits long (equivalently, 4 bytes), and there are thus a total of 2^{32} possible IP addresses. By approximating 2^{10} by 10^3 , it is easy to see that there are about 4 billion possible IP addresses. These addresses are typically written in so-called **dotted-decimal notation**, in which each byte of the address is written in its decimal form and is separated by a period (dot) from other bytes in the address. For example, consider the IP address 193.32.216.9. The 193 is the decimal equivalent of the first 8 bits of the address; the 32 is the decimal equivalent of the second 8 bits of the address, and so on. Thus, the address 193.32.216.9 in binary notation is

11000001 00100000 11011000 00001001

Each interface on every host and router in the global Internet must have an IP address that is globally unique (except for interfaces behind NATs, as discussed at the end of this section). These addresses cannot be chosen in a willy-nilly manner, however. A portion of an interface's IP address will be determined by the subnet to which it is connected.

Figure 4.15 provides an example of IP addressing and interfaces. In this figure, one router (with three interfaces) is used to interconnect seven hosts. Take a close look at the IP addresses assigned to the host and router interfaces; there are several things to notice. The three hosts in the upper-left portion of Figure 4.15, and the router interface to which they are connected, all have an IP address of the form 223.1.1.xxx. That is, they all have the same leftmost 24 bits in their IP address. The four interfaces are also interconnected to each other by a network *that contains no routers*. (This network could be, for example, an Ethernet LAN, in which case the

interfaces would be interconnected by an Ethernet hub or an Ethernet switch; see Chapter 5.) In IP terms, this network interconnecting three host interfaces and one router interface forms a **subnet** [RFC 950]. (A subnet also called an *IP network* or simply a *network* in the Internet literature.) IP addressing assigns an address to this subnet: 223.1.1.0/24, where the /24 notation, sometimes known as a **subnet mask**, indicates that the leftmost 24 bits of the 32-bit quantity define the subnet address. The subnet 223.1.1.0/24 thus consists of the three host interfaces (223.1.1.1, 223.1.1.2, and 223.1.1.3) and one router interface (223.1.1.4). Any additional hosts attached to the 223.1.1.0/24 subnet would be *required* to have an address of the form 223.1.1.xxx. There are two additional subnets shown in Figure 4.15: the 223.1.2.0/24 network and the 223.1.3.0/24 subnet. Figure 4.16 illustrates the three IP subnets present in Figure 4.15.

The IP definition of a subnet is not restricted to Ethernet segments that connect multiple hosts to a router interface. To get some insight here, consider Figure 4.17, which shows three routers that are interconnected with each other by point-to-point links. Each router has three interfaces, one for each point-to-point link and one for the broadcast link that directly connects the router to a pair of hosts. What subnets are present here? Three subnets, 223.1.1.0/24, 223.1.2.0/24, and 223.1.3.0/24 are similar to the subnets we encountered in Figure 4.15. But note that there are three additional subnets in this example as well: one subnet, 223.1.9.0/24, for the interfaces that connect routers R1 and R2; another subnet, 223.1.8.0/24, for the interfaces that

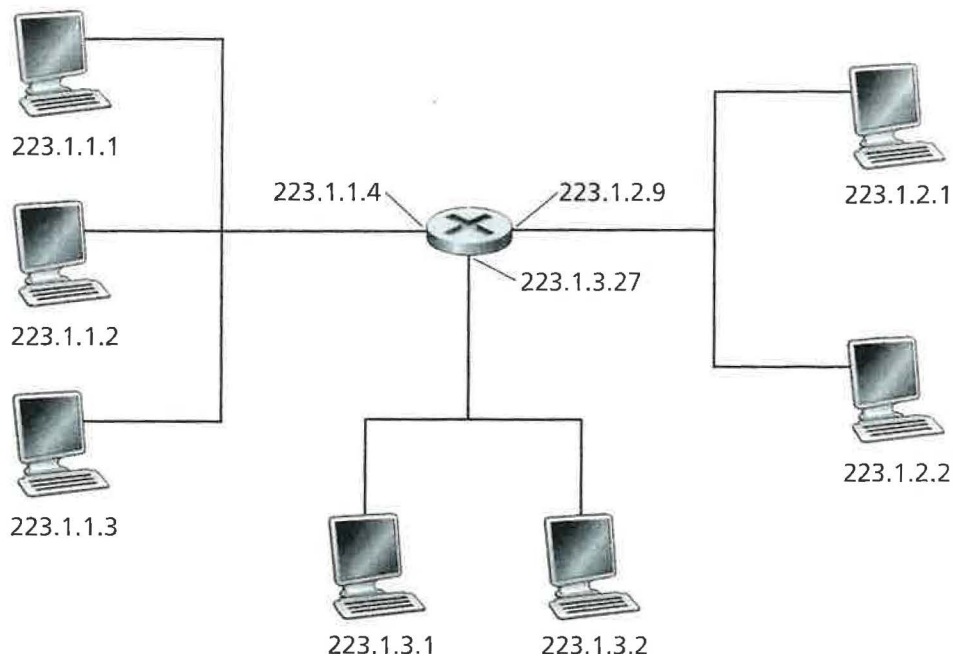


Figure 4.15 ♦ Interface addresses and subnets

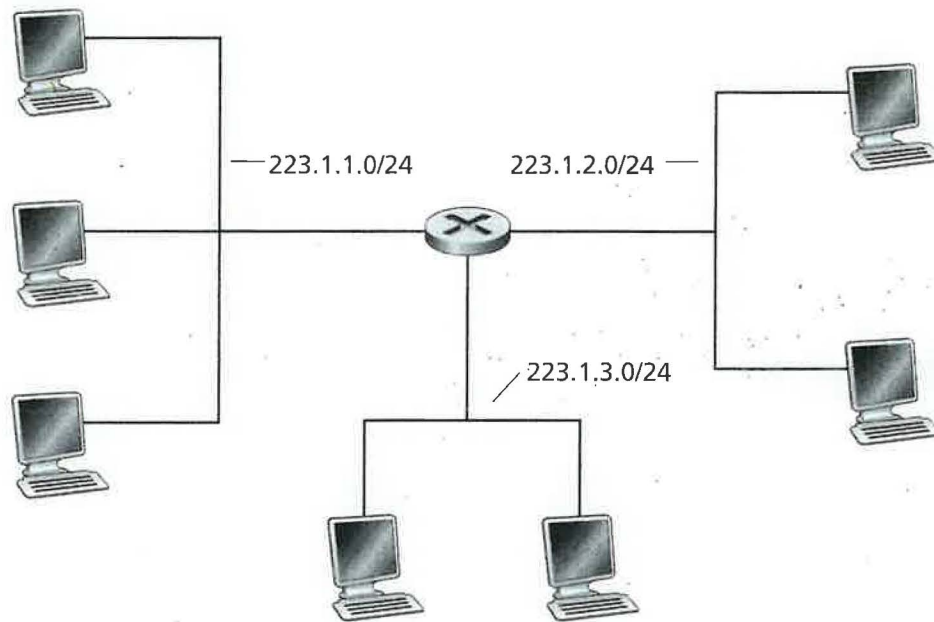


Figure 4.16 ♦ Subnet addresses

connect routers R2 and R3; and a third subnet, 223.1.7.0/24, for the interfaces that connect routers R3 and R1. For a general interconnected system of routers and hosts, we can use the following recipe to define the subnets in the system.

*To determine the subnets, detach each interface from its host or router, creating islands of isolated networks, with interfaces terminating the endpoints of the isolated networks. Each of these isolated networks is called a **subnet**.*

If we apply this procedure to the interconnected system in Figure 4.17, we get six islands or subnets.

From the discussion above, it's clear that an organization (such as a company or academic institution) with multiple Ethernet segments and point-to-point links will have multiple subnets, with all of the devices on a given subnet having the same subnet address. In principle, the different subnets could have quite different subnet addresses. In practice, however, their subnet addresses often have much in common. To understand why, let's next turn our attention to how addressing is handled in the global Internet.

The Internet's address assignment strategy is known as **Classless Interdomain Routing (CIDR)**—pronounced *cider* [RFC 1519]. CIDR generalizes the notion of subnet addressing. As with subnet addressing, the 32-bit IP address is divided into two parts and again has the dotted-decimal form *a.b.c.d/x*, where *x* indicates the number of bits in the first part of the address.

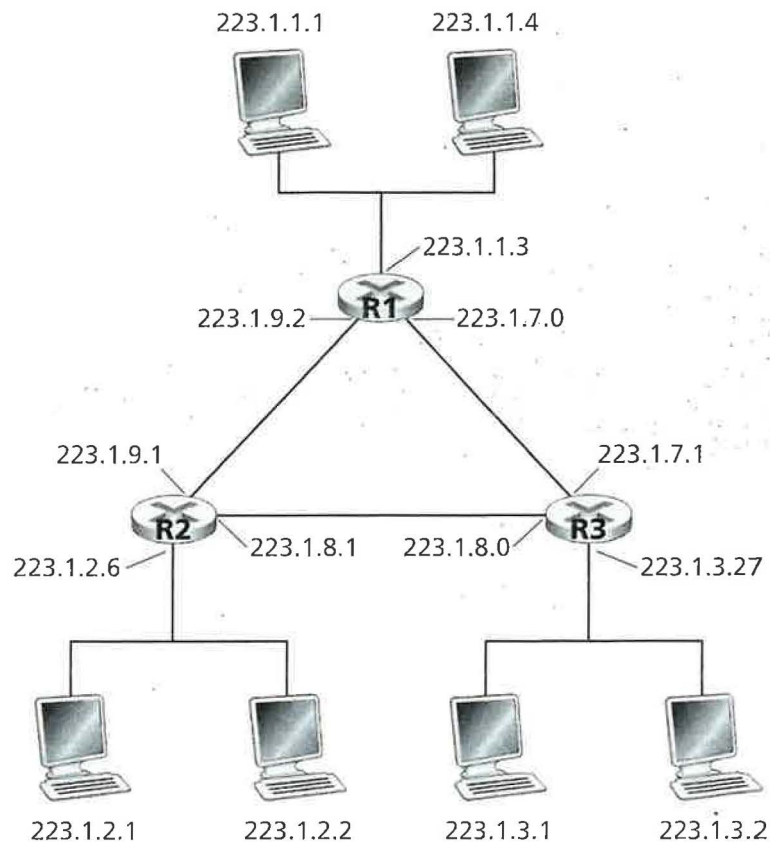


Figure 4.17 ♦ Three routers interconnecting six subnets

The x most significant bits of an address of the form $a.b.c.d/x$ constitute the network portion of the IP address, and are often referred to as the **prefix** (or *network prefix*) of the address. An organization is typically assigned a block of contiguous addresses, that is, a range of addresses with a common prefix (see Principles in Practice sidebar). In this case, the IP addresses of devices within the organization will share the common prefix. When we cover the Internet's BGP routing protocol in Section 4.6, we'll see that only these x leading prefix bits are considered by routers outside the organization's network. That is, when a router outside the organization forwards a datagram whose destination address is inside the organization, only the leading x bits of the address need be considered. This considerably reduces the size of the forwarding table in these routers, since a *single* entry of the form $a.b.c.d/x$ will be sufficient to forward packets to *any* destination within the organization.

The remaining $32-x$ bits of an address can be thought of as distinguishing among the devices *within* the organization, all of which have the same network prefix. These are the bits that will be considered when forwarding packets at routers *within* the organization. These lower-order bits may (or may not) have an additional subnetting structure, such as that discussed above. For example, suppose the first 21

bits of the CIDRized address *a.b.c.d/21* specify the organization's network prefix and are common to the IP addresses of all devices in that organization. The remaining 11 bits then identify the specific hosts in the organization. The organization's internal structure might be such that these 11 rightmost bits are used for subnetting within the organization, as discussed above. For example, *a.b.c.d/24* might refer to a specific subnet within the organization.

Before CIDR was adopted, the network portions of an IP address were constrained to be 8, 16, or 24 bits in length, an addressing scheme known as **classful addressing**, since subnets with an 8-, 16-, and 24-bit subnet addresses were known as class A, B, and C networks, respectively. The requirement that the subnet portion of an IP address be exactly 1, 2, or 3 bytes long turned out to be problematic for supporting the rapidly growing number of organizations with small and medium-sized subnets. A class C (/24) subnet could accommodate only up to $2^8 - 2 = 254$ hosts (two of the $2^8 = 256$ addresses are reserved for special use)—too small for many organizations. However, a class B (/16) subnet, which supports up to 65,534 hosts, was too large. Under classful addressing, an organization with, say, 2,000 hosts was typically allocated a class B (/16) subnet address. This led to a rapid depletion of the class B address space and 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 addresses that could not be used by other organizations.

We would be remiss if we did not mention yet another type of IP address, the IP broadcast address 255.255.255.255. When a host emits a datagram with destination address 255.255.255.255, the message is delivered to all hosts on the same subnet. Routers optionally forward the message into neighboring subnets as well (although they usually don't). In Chapter 5 we'll look at an example of how IP broadcast is used when discussing the DHCP protocol.

Having now studied IP addressing in detail, we need to know how hosts and subnets get their addresses in the first place. Let's begin by looking at how an organization gets a block of addresses for its devices, and then look at how a device (such as a host) is assigned an address from within the organization's block of addresses.

Obtaining a Block of Addresses

In order to obtain a block of IP addresses for use within an organization's subnet, a 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-sized contiguous 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 subnet part of these addresses for your convenience.)

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

PRINCIPLES IN PRACTICE

This example of an ISP that connects eight organizations to the Internet nicely illustrates how carefully allocated CIDRized addresses facilitate routing. Suppose, as shown in Figure 4.18, 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 address block 200.23.16.0/20 there are in fact eight other organizations, each with their own subnets. This ability to use a single prefix to advertise multiple networks is often referred to as **address aggregation** (also **route aggregation** or **route summarization**).

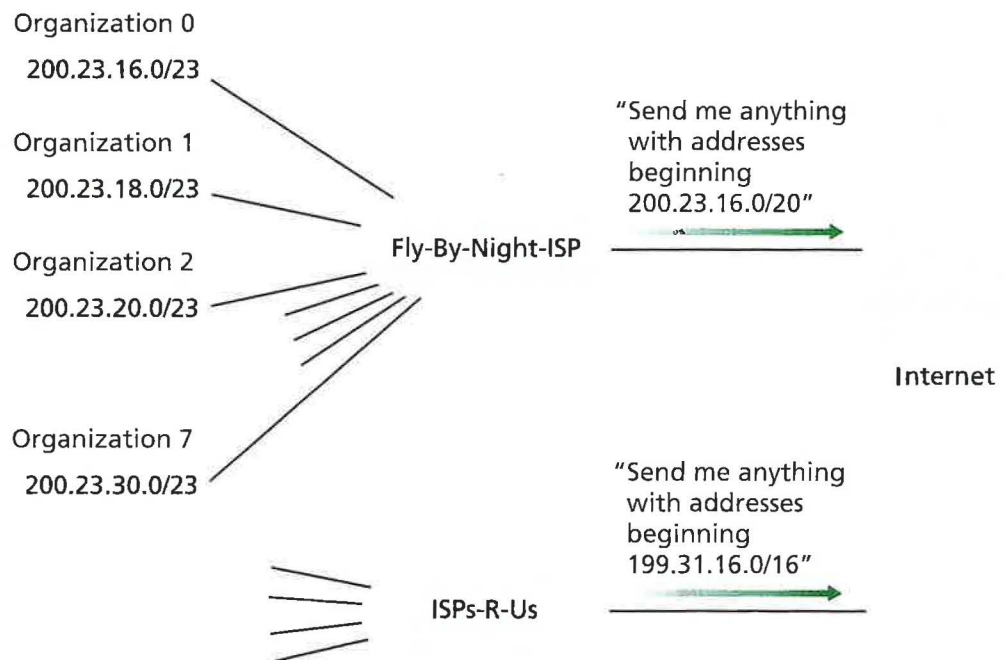


Figure 4.18 ♦ Hierarchical addressing and route aggregation

Address 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 Fly-by-Night-ISP acquires ISPs-R-U's and then has Organization 1 connect to the Internet through its subsidiary ISPs-R-U's? As shown in Figure 4.18, the subsidiary ISPs-R-U's owns the address block 199.31.0.0/16, but Organization 1'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-U's address block. But this is a costly solution, and Organization 1 might well be re-assigned to another subsidiary 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 4.19, Fly-By-Night-ISP continues to advertise the address block 200.23.16.0/20 and ISPs-R-U's continues to advertise 199.31.0.0/16. However, ISPs-R-U's 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-U's) and want to route to an address in the block 200.23.18.0/23, they will use longest prefix matching (see Section 4.2.2), and route toward ISPs-R-U's, as it advertises the longest (most specific) address prefix that matches the destination address.

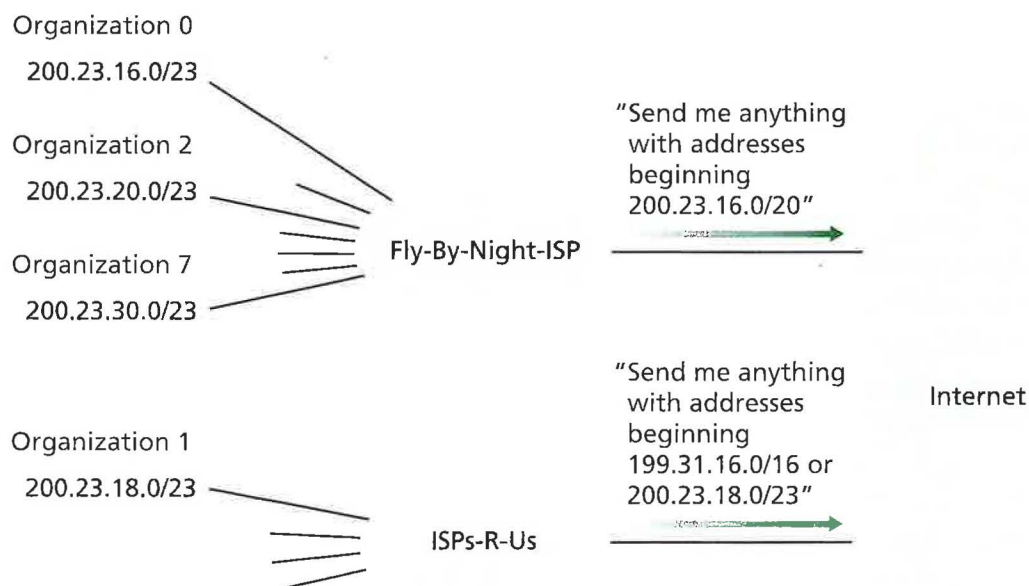


Figure 4.19 ♦ ISPs-R-U's has a more specific route to Organization 1

While obtaining a set of addresses from an ISP is one way to get a block of addresses, it is not the only way. Clearly, there must also be a way for the ISP itself to get a block of addresses. Is there a global authority that has ultimate responsibility for managing the IP address space and allocating address blocks to ISPs and other organizations? Indeed there is! IP addresses are managed under the authority of the Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN 2004], based on guidelines set forth in RFC 2050. The role of the nonprofit ICANN organization [NTIA 1998] is not only to allocate 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 ICANN allocates addresses to regional Internet registries (for example, ARIN, RIPE, APNIC, and LACNIC, which together form the Address Supporting Organization of ICANN [ASO-ICANN 2004]), which handle the allocation/management of addresses within their regions.

Obtaining a Host Address

Once an organization has obtained a block of addresses, it can assign individual IP addresses to the host and router interfaces in its organization. For the router interface addresses, the system administrator manually configures the IP addresses into the router (often remotely, with a network management tool). There are two ways in which a host can be assigned an IP address:

- ◆ *Manual configuration.* A system administrator manually configures the IP address into the host (typically in a file).
- ◆ *Dynamic Host Configuration Protocol (DHCP) [RFC 2131].* DHCP allows a host to obtain (be allocated) an IP address automatically, as well as to learn additional information, such as its subnet mask, the address of its first-hop router (often called the default gateway), and the address of its local DNS server.

Because of DHCP's ability to automate the network-related aspects of connecting a host into a network, it is often referred to as a **plug-and-play protocol**. This capability makes it *very* attractive to the network administrator who would otherwise have to perform these tasks manually! DHCP is also enjoying widespread use in residential Internet access networks and in wireless LANs, where hosts join and leave the network frequently.

A network administrator can configure DHCP so that a given host receives a persistent IP address, so that each time the host joins the network it is assigned the same IP address. But many organizations and residential ISPs do not have enough IP addresses for all of their hosts. When this is the case, DHCP is used to assign each of its connecting hosts a **temporary IP address**. As an example, consider a residential ISP that has 2,000 customers, but no more than 400 of these customers are ever online at the same time. To handle all of its 2,000 customers, the ISP doesn't need a block of 2,000 addresses. Instead, by using a DHCP server to assign addresses

dynamically, it only needs a block of 512 addresses (for example, a block of the form a.b.c.d/23). As the hosts join and leave, the DHCP server needs to update its list of available IP addresses. Each time a host joins, the DHCP server allocates an arbitrary address from its current pool of available addresses; each time a host leaves, an address is returned to the pool.

Another important reason why DHCP has found such widespread use is the advent of mobile computing. Consider, for example, the student who carries a laptop from a dormitory room to a library to a classroom. It is likely that in each location, the student will be connecting into a new subnet and hence will need a new IP address at each location. DHCP is ideally suited to this situation, as there are many users coming and going, and addresses are needed for only a limited amount of time.

The DHCP protocol actually straddles the boundary between the network and link layers in the five-layer Internet protocol stack. We'll therefore delay a detailed discussion of how the DHCP service is implemented until Chapter 5, which covers the link layer.

Network Address Translation (NAT)

Given our discussion about Internet addresses and the IPv4 datagram format, we're now well aware that every IP-capable device needs an IP address. With the proliferation of small office, home office (SOHO) subnets, this would seem to imply that whenever a SOHO wants to install a LAN to connect multiple machines, a range of addresses would need to be allocated by the ISP to cover all of the SOHO's machines. If the subnet grew bigger (for example, the kids at home have not only their own computers, but have bought handheld PDAs, IP-capable phones, and networked Game Boys as well), a larger block of addresses would have to be allocated. But what if the ISP had already allocated the contiguous portions of the SOHO network's current address range? And what typical homeowner wants (or should need) to know how to manage IP addresses in the first place? Fortunately, there is a simpler approach to address allocation that has found increasingly widespread use in such scenarios: **network address translation (NAT)** [RFC 2663; RFC 3022].

Figure 4.20 shows the operation of a NAT-enabled router. The NAT-enabled router, residing in the home, has an interface that is part of the home network on the right of Figure 4.20. Addressing within the home network is exactly as we have seen above—all four interfaces in the home network have the same subnet address of 10.0.0/24. The address space 10.0.0.0/8 is one of three portions of the IP address space that is reserved in [RFC 1918] for a private network or a **realm** with private addresses, such as the home network in Figure 4.20. A *realm with private addresses* refers to a network whose addresses only have meaning to devices within that network. To see why this is important, consider the fact that there are hundreds of thousands of home networks, many using the same address space, 10.0.0.0/24. Devices within a given home network can send packets to each other using 10.0.0.0/24 addressing. However, packets forwarded *beyond* the home network into the larger

global Internet clearly cannot use these addresses (as either a source or a destination address) because there are hundreds of thousands of networks using this block of addresses. That is, the 10.0.0.0/24 addresses can only have meaning within the given home network. But if private addresses only have meaning within a given network, how is addressing handled when packets are sent to or received from the global Internet, where addresses are necessarily unique? The answer lies in understanding NAT.

The NAT-enabled router does not *look* like a router to the outside world. Instead the NAT router behaves to the outside world as a *single* device with a *single* IP address. In Figure 4.20, all traffic leaving the home router for the larger Internet has a source IP address of 138.76.29.7; and all traffic entering the home must have a destination address of 138.76.29.7. In essence, the NAT-enabled router is hiding the details of the home network from the outside world. (As an aside, you might wonder where the home network computers get their addresses and where the router gets its single IP address. Often, the answer is the same—DHCP! The router gets its address from the ISP's DHCP server, and the router runs a DHCP server to provide addresses to computers within the NAT-DHCP-router-controlled home network's address space.)

If all datagrams arriving at the NAT router from the WAN have the same destination IP address (specifically, that of the WAN-side interface of the NAT router), then how does the router know the internal host to which it should forward a given datagram? The trick is to use a **NAT translation table** at the NAT router, and to include port numbers as well as IP addresses in the table entries.

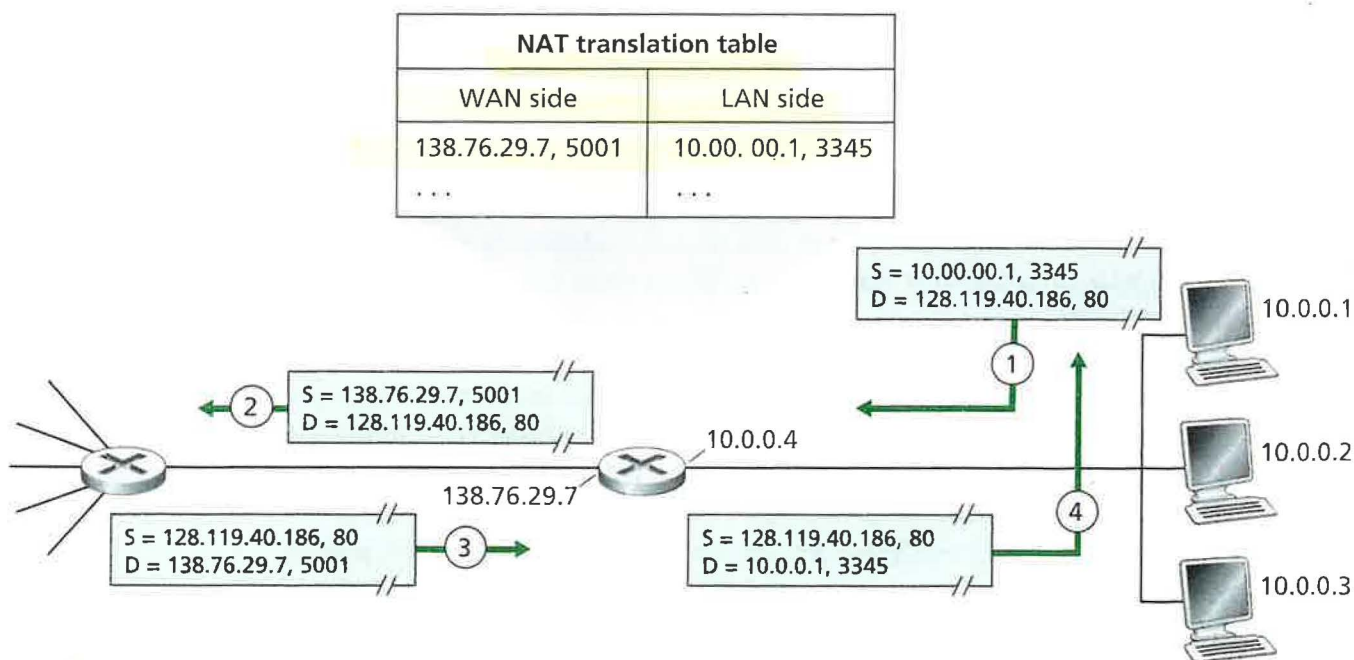


Figure 4.20 ♦ Network address translation

Consider the example in Figure 4.20. Suppose a user sitting in a home network behind host 10.0.0.1 requests a Web page on some Web server (port 80) with IP address 128.119.40.186. The host 10.0.0.1 assigns the (arbitrary) source port number 3345 and sends the datagram into the LAN. The NAT router receives the datagram, generates a new source port number 5001 for the datagram, replaces the source IP address with its WAN-side IP address 138.76.29.7, and replaces the original source port number 3345 with the new source port number 5001. When generating a new source port number, the NAT router can select any source port number that is not currently in the NAT translation table. (Note that because a port number field is 16 bits long, the NAT protocol can support over 60,000 simultaneous connections with a single WAN-side IP address for the router!) NAT in the router also adds an entry to its NAT translation table. The Web server, blissfully unaware that the arriving datagram containing the HTTP request has been manipulated by the NAT router, responds with a datagram whose destination address is the IP address of the NAT router, and whose destination port number is 5001. When this datagram arrives at the NAT router, the router indexes the NAT translation table using the destination IP address and destination port number to obtain the appropriate IP address (10.0.0.1) and destination port number (3345) for the browser in the home network. The router then rewrites the datagram's destination address and destination port number, and forwards the datagram into the home network.

NAT has enjoyed widespread deployment in recent years. But we should mention that many purists in the IETF community loudly object to NAT. First, they argue, port numbers are meant to be used for addressing processes, not for addressing hosts. (This violation can indeed cause problems for servers running on the home network, since, as we have seen in Chapter 2, server processes wait for incoming requests at well-known port numbers.) Second, they argue, routers are supposed to process packets only up to layer 3. Third, they argue, the NAT protocol violates the so-called end-to-end argument; that is, hosts should be talking directly with each other, without interfering nodes modifying IP addresses and port numbers. And fourth, they argue, we should use IPv6 (see Section 4.4.4) to solve the shortage of IP addresses, rather than recklessly patching up the problem with a stopgap solution like NAT. But like it or not, NAT has become an important component of the Internet.

Yet another major problem with NAT is that it interferes with P2P applications, including P2P file sharing applications and P2P voice-over-IP applications. Recall from Chapter 2 that in a P2P application, any participating Peer A should be able to initiate a TCP connection to any other participating Peer B. The essence of the problem is that if Peer B is behind a NAT, it cannot act as a server and accept TCP connections (unless the NAT is specifically configured for the P2P application). As we'll see in the homework problems, this NAT problem can be circumvented if Peer A first contacts Peer B through an intermediate Peer C which is not behind a NAT and to which B has established an on-going TCP connection. Peer A can then ask Peer B, via Peer C, to initiate a TCP connection directly back to Peer A. Once the

direct P2P TCP connection is established between Peers A and B, the two peers can exchange messages or files. This hack, called **connection reversal**, is actually used by many P2P applications. But if both Peer A and Peer B are behind their own NATs, then, for all practical purposes, it is impossible to establish a TCP connection between the two peers without application-specific NAT configuration.

Our discussion of NAT here has been necessarily brief. There are many other important aspects of NAT, including static versus dynamic NAT, and the effects of NAT on higher-layer protocols and on security. For more details, and a discussion of the pros and cons of NAT, see [Cisco NAT 2004; Phifer 2000].

4.4.3 Internet Control Message Protocol (ICMP)

Recall that the network layer of the Internet has three main components: the IP protocol, discussed in the previous section; the Internet routing protocols (including RIP, OSPF, and BGP), which are covered in Section 4.6; and ICMP, which is the subject of this section.

ICMP, specified in RFC 792, is used by hosts and routers to communicate network-layer information to each other. The most typical use of ICMP is for error reporting. For example, when running a Telnet, FTP, or HTTP session, you may have encountered an error message such as “Destination network unreachable.” This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your Telnet, FTP, or HTTP application. That router created and sent a type-3 ICMP message to your host indicating the error.

ICMP is often considered part of IP but architecturally lies just above IP, as ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP datagram with ICMP specified as the upper-layer protocol, it demultiplexes the datagram’s contents to ICMP, just as it would demultiplex a datagram’s content to TCP or UDP.

ICMP messages have a type and a code field, and contain the header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated in the first place (so that the sender can determine the datagram that caused the error). Selected ICMP message types are shown in Figure 4.21. Note that ICMP messages are used not only for signaling error conditions.

The well-known ping program sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply. Most TCP/IP implementations support the ping server directly in the operating system; that is, the server is not a process. Chapter 11 of [Stevens 1990] provides the source code for the ping client program. Note that the client program needs to be able to instruct the operating system to generate an ICMP message of type 8 code 0.