Karanjit S. Siyan, Ph.D.
Tim Parker, Ph.D.

# TCP/IP UNLEASHED

Third Edition

SAMS

# TCP/IP UNLEASHED

Third Edition

**Karanjit S. Siyan, Ph.D.**
**Tim Parker, Ph.D.**

SAMS 201 West 103rd Street, Indianapolis, IN 46290

# TCP/IP Unleashed



International Standard Book Number: 0-672-32351-6

Library of Congress Catalog Card Number: 2001096703

Printed in the United States of America

First Printing: March 2002

05 04 03 02 4 3 2 1

## Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

## Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of programs accompanying it.

**Associate Publisher**
*Linda Engelman*

**Acquisitions Editor**
*Dayna Isley*

**Development Editor**
*Ginny Bess*

**Managing Editor**
*Charlotte Clapp*

**Project Editor**
*Elizabeth Finney*

**Copy Editor**
*Paula Lowell*

**Indexer**
*Larry Sweazy*

**Proofreader**
*Plan-it Publishing*

**Technical Editor**
*William Wagner*

**Team Coordinator**
*Lynne Williams*

**Interior Designer**
*Gary Adair*

**Cover Designer**
*Alan Clements*

**Page Layout**
*Rebecca Harmon*

# Contents at a Glance

# Contents

# About the Authors

**Karanjit S. Siyan, Ph.D.,** is Vice President of R&D for Higher Technology, a professional services company based in Hamden, Connecticut. He has authored international seminars on Solaris and SunOS, TCP/IP networks, PC Network Integration, Windows NT, Novell networks, and expert systems using Fuzzy Logic. He teaches advanced technology seminars in the United States, Canada, Europe, and the Far East. Dr. Siyan has published articles in *Dr. Dobb's Journal*, *The C Users Journal*, *Database Advisor*, and other research journals and is actively involved in Internet research. He has worked with computer networks for Unix, NetWare, Windows NT/2000, OS/2, and computer languages such as C/C++, Pascal, LISP, Algol, Ada, MSL, PL/1, Jovial, CMS-2, PL/SQL, Perl, VB, Java, and so on for many years. He has written compilers for a number of these languages. Karanjit holds a Ph.D. in Computer Science, a master's degree in Electrical Engineering and Computer Science, and a bachelor's degree in Electronics and Electrical Communication Engineering. He is a member of IEEE and ACM, and his career achievements are recorded in the Marquis Who's Who in the World, Who's Who in America, Who's Who in Finance and Industry, and Who's Who in Science and Engineering.

Before working as an independent consultant, Dr. Siyan worked as a senior member of the technical staff at ROLM Corporation, and as a software developer on numerous projects. As part of his consulting work, Dr. Siyan has written a number of custom compiler and operating system development tools. His interests include Unix-based, NetWare-based, and Windows NT/2000-based networks. He is actively involved in the application of many computer science disciplines such as networks, operating systems, programming languages, databases, expert systems, and computer security. Dr. Siyan holds certification credentials for a number of commercial operating system products, and has written numerous books.

**Tim Parker, Ph.D.**, founded Timothy Parker Consulting Incorporated (TPCI), a technical writing and consulting services firm, more than 20 years ago. He also specializes in technical training, including several courses about TCP/IP, software development, and hardware testing. He is the author of more than 2,500 articles and sixty books.

# Dedication

*To my wife for bringing so much joy in my life and to my parents for being there for me.*

*—Karanjit*

# Acknowledgments

One of the more pleasurable tasks of being an author is to thank the people responsible for the success of a book. I want to thank all those close to me: Harpreet, Ahal Singh, Tejinder Kaur, Harjeet, Jagjit, Kookie, and Dolly. Special thanks to Mother, Saint Germain, El Morya Khan, Djwal Kul, Kuthumi Lal Singh, Kuan-Yin, Bhagwan Krishna, and Babaji. Without their spiritual support, this book would not have been possible.

I want to thank Bob Sanregret and Anders Amundson, who initially got me interested in writing teaching materials about computers. I also want to thank the many people at Higher Technology and Learning Tree for their help and support on various projects. In particular I would like to thank Philip Weinberg, Lenny Bernstein, Dr. David Collins, Eric Garen, and John Moriarty.

I want to thank Dayna Isley for keeping the project on track and Ginny Bess, the development editor. I want to also thank Elizabeth Finney and William Wagner for their help in developing this book.

—Karanjit

# Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you would like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can e-mail or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

*Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.*

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.


Email: `networking@samspublishing.com`

Mail: Mark Taber
Associate Publisher
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

# Introduction

TCP/IP (Transmission Control Protocol/Internet Protocol) has become a de facto protocol not only for the Internet but also for intranets. On the intranet, TCP/IP was one choice among many, such as Novell's IPX/SPX, Microsoft's NetBIOS or NetBEUI, Apple's AppleTalk protocols. Although these other protocols can still be found on networks, TCP/IP has emerged as the dominant communications protocol.

What caused the change on intranets? It was gradual. As Windows NT/2000 and Novell NetWare servers evolved, they adopted TCP/IP. Microsoft included a TCP/IP stack with Windows 9x and NT/2000. One advantage of the adoption of a common protocol is that it promotes interoperability. For example, UNIX, Windows, and Macintosh computers can more seamlessly work together. The ability to connect seamlessly to the Internet, without having to convert network protocols, also helped move the momentum to TCP/IP. So today most networks, heterogeneous and Windows-specific, are TCP/IP based. IPX/SPX is a very small fraction of the market, and NetBEUI is used only on small, simple Windows networks.

The change has been for the better. TCP/IP is one of the most stable network protocols available, and it is evolving to meet the needs of the future. The security weaknesses of TCP/IP are well known and well researched. Standards such as IPSec have been created to make TCP/IP more secure. TCP/IP has been implemented on almost every type of hardware that needs to be networked. TCP/IP is an open standard, which means that all the specifications are published and readily available.

The growth of the Internet has helped push TCP/IP to the forefront of network protocols, of course. With the explosive growth of the World Wide Web, every computer now needs to understand TCP/IP in order to talk to an Internet service provider. And yet TCP/IP was in widespread use before the Web's popularity boost only a few years ago. Many of us were using TCP/IP for our e-mail, FTP, and Usenet services 20 years ago, working on old UNIX and CP/M systems that required us to understand more than how to click a few buttons in a GUI dialog box.

When the first edition of *TCP/IP Unleashed* and *Sams Teach Yourself TCP/IP in 14 Days* were published, they were in the minority: Books about TCP/IP were meant to be read by non-programmers or network administrators. Now, the shelves are crowded with TCP/IP books, many just derivatives of each other. The strange truth about TCP/IP is this: It is very simple, and at the same time very complex. If all you want to do is connect a Windows machine to a TCP/IP network, the steps are so simple they can be covered in a couple of pages of a book. Yet if you want to know the details behind TCP/IP's protocols and the utilities in the protocol suite, thousands of pages are required. Even more is

necessary if you want to develop TCP/IP-based applications. So there's obviously a need for all levels of books. This edition of *TCP/IP Unleashed* tries to appeal to the person who needs to know about the protocols and tools involved in TCP/IP, how to add machines to a TCP/IP network, and what issues you need to be aware of, but skims only lightly across the developer's side of the protocol.

A lot of information is in this book: We cover all the basic TCP/IP protocols in detail. You'll see how the protocols are used and how TCP builds upon IP, which itself builds upon a network protocol. You'll see how all kinds of applications use the basic TCP and IP protocols. Tools you use on large networks every day, such as DNS (Domain Name System), NFS (Network File System), and NIS (Network Information Service) all use TCP as their basic protocol. Other protocols used by administrators, such as SNMP (Simple Network Management Protocol) and ARP (Address Resolution Protocol), all build on the basic TCP protocols, too. Even things as simple as your e-mail use TCP protocols such as SMTP (Simple Mail Transfer Protocol). TCP/IP is pervasive throughout all our computer tasks. Understanding the protocol and its underlying operation can help you understand your computer's services better.

Throughout this book we've tried to take a non-programmer's approach to the material. Each protocol in the TCP/IP family has its own chapter, and there is special emphasis on the IP and TCP protocols themselves with additional chapters dedicated to them. This book shows you how the TCP/IP protocol works, how it interacts with the other protocols in the TCP/IP family, what the protocol does for you and your network, and how you can implement it easily on your machine.

Where appropriate, we give you step-by-step guides to install and configure the protocol or service. We show you the commands you will use and the output you can expect. You'll see how to implement TCP/IP on many different operating systems including Windows 9x, Windows NT, Windows 2000, NetWare, and the widely used Linux system. You'll see information about firewalls and security. In short, everything we could think of about TCP/IP, with the exception of programming applications, is in this book. We hope you enjoy it.

# Conventions Used in This Book

The following typographic conventions are used in this book:

- Code lines, commands, statements, variables, and any text you type or see onscreen appears in a `mono` typeface. **`Bold mono`** typeface is used to represent the user's input.

- Placeholders in syntax descriptions appear in an *`italic mono`* typeface. Replace the placeholder with the actual filename, parameter, or whatever element it represents.
- *Italics* highlight technical terms when they're being defined.
- The ➥ icon is used before a line of code that is really a continuation of the preceding line. Sometimes a line of code is too long to fit as a single line on the page. If you see ➥ before a line of code, remember that it's part of the line immediately above it.
- The book also contains Notes to help you spot important or useful information quickly. These notes contain tips and warnings as well. Some of these are helpful shortcuts to help you work more efficiently.

# TCP/IP Fundamentals

# PART I

## IN THIS PART

CHAPTER 1

# Introduction to Open Communications

*by Mark A. Sportack*

## IN THIS CHAPTER

Unarguably, TCP/IP (less commonly known as the Transmission Control Protocol/Internet Protocol) is the most successful communications protocol ever developed. The term *TCP/IP* is generically used to refer to a set of communication protocols, such as TCP for reliable delivery, UDP (User Datagram Protocol) for best-efforts delivery, IP for Internet Protocol, and other application services.

Evidence of the success of TCP/IP can be found on the Internet, the largest open network ever built. The Internet was specifically designed to facilitate defense-oriented research, and to enable the U.S. Government to continue communications despite the potentially devastating effects of a nuclear attack. TCP/IP was developed for this purpose.

Today, the Internet has grown into more commercial and consumer-oriented roles. Despite this radical change in its purpose, all of its original qualities (that is, openness, survivability, and reliability) continue to be essential. These attributes include reliable delivery of data with the use of explicit protocols such as TCP designed for that purpose, and the ability to automatically detect and avoid failures in the network. More importantly, though, TCP/IP is an open communications protocol. *Openness* means that communications are possible between *any* combination of devices, regardless of how physically dissimilar they might be.

This chapter explains how open communications emerged and the concepts that enable open data communications. It also introduces the two major models that make open communications a practical reality: the *Open Systems Interconnect* (OSI) *Reference Model* and the *TCP/IP Reference Model*. These models greatly facilitate the understanding of networks by dissecting them into their various functional components. These components are stratified into *layers.* A command of these layers, and the concept of open communications, will provide the context for a more meaningful examination of TCP/IP's various components and uses.

# Evolution of Open Networks

Networks, originally, were highly proprietary connectivity solutions that were an integral part of an equally proprietary, bundled computing solution. Companies that automated their data processing or accounting functions during the primitive days before personal computers had to commit to a single vendor for a turnkey solution.

In such a proprietary, single-vendor environment, the application software executed only in an environment supported by a single operating system. The operating system could execute only within the safety of the same vendor's hardware products. Even the users' terminal equipment and connectivity to the computer were part of the same, one-vendor, integrated solution.

During the reign of single-vendor, integrated solutions, the *Department of Defense* (DoD) identified a need for a robust and reliable data communications network that could interconnect all of its computers, as well as the computers owned by affiliated organizations such as universities, think tanks, and defense contractors. This might not sound like a big deal, but it was. In the early days of computing, manufacturers developed very tightly integrated platforms of hardware, software, and networks. A user on one platform would be very hard-pressed to share data with a user on a different computing platform. The logic behind this was fairly simple: The *Department of Defense* manufacturers wanted captive audiences.

Forcing all of the DoD's subcontractors and affiliated research organizations to migrate to a single vendor's brand of equipment was utterly impractical. Consequently, a means of communicating between dissimilar platforms was needed. The result was the creation of the world's first open communications protocol: the *Internet Protocol* (IP).

An *open network*, therefore, is one that enables communications and resource-sharing between dissimilar computers. Openness is achieved through the collaborative development and maintenance of technical specifications. These specifications, which are also known as *open standards*, are made publicly available.

## Layering the Communications Process

The key to enabling open communications lay in understanding all of the functions necessary for two end systems to communicate and share data with each other. Identifying these essential functions, and establishing the order in which they must occur, is the foundation for open communications. Two end systems can only communicate if they both agree on how to communicate. That is, they both must follow the same procedure for taking data from an application and packaging it for transit through a network. No detail, regardless of how small, can be taken for granted or left to chance.

Fortunately, there is a somewhat logical sequence of events that is necessary in a communications session. These events include, at a bare minimum, the following tasks:

- Data must be passed down from its application to a communications process (known as a *protocol*).
- The communications protocol must prepare the application data for transmission across some type of network. This usually means that the data must be broken down into more manageable pieces.
- The segmented data must then be enveloped in a data structure for passage through a network to a specific device (or devices). This means that the data must be wrapped in something that contains information that would enable any networked

computing device to identify where that envelope of data came from, and where it was going. This structure can be a frame, a packet, or a cell, depending on the protocol being used.

- These frames or packets must be converted into the physical bits for transmission. These bits can be transmitted as pulses of light on a fiber-optic network (such as FDDI) or as electronic states (on or off) for transmission across an electronic network (such as Ethernet, or any other network that transmits data in the form of electricity over metallic wire).

At the destination, or recipient machine, this process is reversed.

Other functions, too, may be needed during the communications session. These functions enable the source and destination machines to coordinate efforts and ensure that the data arrives safely. These functions include the following:

- Regulating the flow of transmitted data so that the receiving machine, and/or the network, are not inundated.
- Checking the received data to ensure that it was not damaged in transit. This is done using checksums, such as the *Cyclic Redundancy Checksum* (CRC). CRC is a polynomial-based mathematical checksum that is able to detect errors in transmission.
- Coordinating the retransmission of any data that either failed to arrive at its destination, or arrived damaged.
- Lastly, the recipient of the data must reassemble the segments into a form that the receiving application can recognize. From the receiving application's perspective, the data should be exactly the same as what the transmitting application sent out. In other words, the two applications appear to be passing data directly between each other. This is known as *logical adjacency*.

Perhaps the best tool for explaining the concepts of layered communications, including logical adjacency, is the OSI Reference Model.

# The OSI Reference Model

The *International Organization for Standardization* (ISO) developed the Open Systems Interconnection (OSI) Reference Model in 1978/1979 to facilitate the *open interconnection* of computer systems. An open interconnection is one that can be supported in a multivendor environment. This model established the global standard for defining the functional layers required for open communications between computers.

When the OSI Reference Model was developed almost 20 years ago, few commercial systems followed that model. Computer manufacturers at that time locked customers into proprietary, single-vendor architectures. Open communication was viewed as an invitation to competition. From the manufacturer's perspective, competition was undesirable. Consequently, all functions were integrated as tightly as possible. The notion of functional modularity, or layering, seemed antithetical to any manufacturer's mission.

It is important to note that the model has been so successful at achieving its original goals that it almost renders itself moot. The previous proprietary, integrated approach has disappeared. Open communications, today, are requisite. Curiously, very few products are fully OSI-compliant. Instead, its basic layered framework is frequently adapted to new standards. Nevertheless, the OSI Reference Model remains a viable mechanism for demonstrating the functional mechanics of a network.

Despite its successes, numerous misperceptions about the OSI Reference Model persist. Consequently, it is necessary to provide yet another overview of this model in this section. The overview identifies and corrects these misperceptions.

The first misperception is that the OSI Reference Model was developed by the *International Standards Organization* (ISO). It was not. The OSI Reference Model was developed by the International Organization for Standardization. This organization prefers to use a mnemonic abbreviation rather than an acronym. The mnemonic abbreviation is based on the Greek word, *isos*, which means equal or standard.

The OSI model categorizes the various processes necessary in a communications session into seven distinct functional layers. The layers are organized based on the natural sequence of events that occurs during a communications session.

Figure 1.1 illustrates the OSI Reference Model. Layers 1–3 provide network access, and Layers 4–7 are dedicated to the logistics of supporting end-to-end communications.

**FIGURE 1.1**
*The OSI Reference Model.*

| OSI Reference Model Layer Description | OSI Layer Number |
|---|---|
| Application | 7 |
| Presentation | 6 |
| Session | 5 |
| Transport | 4 |
| Network | 3 |
| Data Link | 2 |
| Physical | 1 |

## Layer 1: The Physical Layer

The bottom layer is called the *Physical Layer*. This layer is responsible for the transmission of the bit stream. It accepts frames of data from Layer 2, the Data Link Layer, and transmits their structure and content, typically serially, one bit at a time. The Physical Layer is also responsible for the reception of incoming streams of data, one bit at a time. These streams are then passed on to the Data Link Layer for reframing.

The Physical Layer, quite literally, sees only 1s and 0s. It has no mechanism for determining the significance of the bits it transmits or receives. It is solely concerned with the physical characteristics of electrical and/or optical signaling techniques. This includes the voltage of the electrical current used to transport the signal (the line encoding and framing not to be confused with the data character coding and framing), the media type and impedance characteristics, and even the physical shape of the connector that is used to terminate the media.

A common misperception is that OSI's Layer 1 includes anything that either generates or carries the data communications signals. This is not true. It is a *functional* model only. The Physical Layer is limited to just the processes and mechanisms needed to place signals onto the transmission media, and to receive signals from that media. Its lower boundary is the physical connector that attaches to the transmission media. It does *not* include the transmission media, though the different LAN (Local Area Network) standards, such as 10BASET and 100BASET, do refer to the media type used.

Transmission media include any means of actually transporting signals generated by the OSI's Layer 1 mechanisms. Some examples of transmission media are coaxial cabling, fiber-optic cabling, and twisted-pair wiring. The confusion seems to stem from the fact that the Physical Layer does provide specifications for the media's performance. These are the performance characteristics that are required, and assumed to exist, by the processes and mechanisms defined in the Physical Layer.

Consequently, transmission media remain outside the scope of the Physical Layer and are sometimes referred to as *Layer 0* of the OSI Reference Model.

## Layer 2: The Data Link Layer

The second layer of the OSI Reference Model is called the *Data Link Layer*. As all the layers do, it has two sets of responsibilities: transmit and receive. It is responsible for providing end-to-end validity of the data being transmitted, typically across a physical link.

On the transmit side, the Data Link Layer is responsible for packing instructions, data, and so forth into frames. A *frame* is a structure indigenous to the Data Link Layer that

contains enough information to ensure that the data can be successfully sent across a physical link (such as a Local Area Network) to its destination.

Successful delivery means that the frame reaches its intended destination intact. Thus, the frame must also contain a mechanism to verify the integrity of its contents upon delivery.

Two things must happen for guaranteed delivery to occur:

- The originating node must receive an acknowledgment of each frame received intact by the destination node.
- The destination node, prior to acknowledging receipt of a frame, must verify the integrity of that frame's contents.

Numerous situations can result in transmitted frames either not reaching the destination or becoming damaged and unusable during transit. The Data Link Layer is responsible for detecting and correcting any and all such errors. The Data Link Layer is also responsible for reassembling any binary streams that are received from the Physical Layer back into frames. Given that both the structure and content of a frame are transmitted, however, the Data Link Layer isn't really rebuilding a frame. Rather, it is buffering the incoming bits until it has a complete frame.

Layers 1 and 2 are required for each and every type of communication, regardless of whether the network is a LAN or WAN. Typically, network cards inside computers implement the Layer 1 and Layer 2 functionality.

## Layer 3: The Network Layer

The *Network Layer* is responsible for establishing the route to be used between the originating and destination computers. This layer lacks any native transmission error detection/correction mechanisms and, consequently, relies upon the link for reliable transmission service of the Data Link Layer. End-to-end reliability across several physical links is more of a function of the Transport Layer (Layer 4).

The Network Layer is used to establish communications with computer systems that lie beyond the local LAN segment. It can do so because it has its own routing addressing architecture, which is separate and distinct from the Layer 2 machine addressing. Such protocols are known as *routed* or *routable* protocols. Routable protocols include IP, Novell's IPX, and AppleTalk, although this book will focus exclusively on IP and its related protocols and applications.

Use of the Network Layer is optional. It is required only if the computer systems reside on different network segments that are separated by a router, or if the communicating applications require some service, feature, or capability of either the Network Layer or

the Transport Layer. For example, two hosts that are directly connected to the same LAN may communicate well using just that LAN's communications mechanisms (Layers 1 and 2 of the OSI Reference Model).

**Note**

In modern networks, the requirement to connect two networks is so common that the Network Layer is no longer considered optional. Earlier protocols such as NetBEUI that do not provide an explicit network layer have to be handled as special cases in modern network design consisting of interconnected networks.

## Layer 4: The Transport Layer

The *Transport Layer* provides a service similar to the Data Link Layer, in that it is responsible for the end-to-end integrity of transmissions. Unlike the Data Link Layer, however, the Transport Layer is capable of providing this function beyond the local LAN segment. It can detect packets that are discarded by routers and automatically generate a retransmit request. In other words, the Transport Layer provides true end-to-end reliability.

Another significant function of the Transport Layer is the resequencing of packets that may have arrived out of order. This can happen for a variety of reasons. For example, the packets may have taken different paths through the network, or some may have been damaged in transit. In any case, the Transport Layer is capable of identifying the original sequence of packets, and must put them back into that sequence before passing their contents up to the Session Layer.

## Layer 5: The Session Layer

The fifth layer of the OSI Model is the *Session Layer*. This layer is relatively unused—many protocols bundle this layer's functionality into their transport layers.

The function of the OSI Session Layer is to manage the flow of communications during a connection between two computer systems. This flow of communications is known as a *session*. It determines whether communications can be uni- or bi-directional. It also ensures that one request is completed before a new one is accepted.

The Session Layer also may provide some of the following enhancements:

- Dialog control
- Token management
- Activity management

A session, in general, allows two-way communications (*full duplex*) across a connection. Some applications may require alternate one-way communications (*half duplex*). The Session Layer has the option of providing two-way or one-way communications: an option called *dialog control*.

For some protocols, it is essential that only one side attempt a critical operation at a time. To prevent both sides from attempting the same operation, a control mechanism, such as the use of *tokens*, must be implemented. When using the token method, only the side holding a token is permitted to perform the operation. Determining which side has the token and how it is transferred between the two sides is known as *token management*. The token management functions are used in the ISO session layer protocols.

**Note**

The use of the word "token" here should not be confused with Token Ring operation. Token management is a much higher level concept at Layer 5 of the OSI model. IBM's Token Ring operation belongs to Layers 2 and 1 of the OSI model.

If you are performing a one-hour file transfer between two machines, and a network crash occurs approximately every 30 minutes, you might never be able to complete the file transfer. After each transfer aborts, you have to start all over again. To avoid this problem, you can treat the entire file transfer as a single activity with checkpoints inserted into the datastream. That way, if a crash occurs, the session layer can synchronize to a previous checkpoint. These checkpoints are called *synchronization points.*

There are two types of synchronization points: major and minor synchronization points. A major synchronized point inserted by any communicating side must be acknowledged by the other communicating side, whereas a minor synchronization point is not acknowledged. That portion of the session that is between two major synchronization points is called a *dialog unit*. The operation of managing an entire activity is called *activity management*. An activity can consist of one or more dialog units.

TCP/IP networks do not have a general Session Layer protocol. This is because some of the characteristics of the Session layer are provided by the TCP protocol. If TCP/IP applications require special session services they provide their own. An example of such a TCP/IP application service is the Network File System (NFS), which implements its own Session Layer service—the Remote Procedure Call (RPC) protocol.

**Note**

The Network File System (NFS) protocol also uses its own Session Layer service: the External Data Representation (XDR) protocol.

## Layer 6: The Presentation Layer

The *Presentation Layer* is responsible for managing the way that data is encoded. Not every computer system uses the same data-encoding scheme, and the Presentation Layer is responsible for providing the translation between otherwise incompatible data-encoding schemes, such as *American Standard Code for Information Interchange* (ASCII) and *Extended Binary Coded Decimal Interchange Code* (EBCDIC).

The Presentation Layer can be used to mediate differences in floating-point formats; number formats, such as one's complement and two's complement; and byte ordering; as well as to provide encryption and decryption services.

The Presentation Layer represents data with a common syntax and semantics. If all the nodes used and understood this common language, misunderstanding in data representation could be eliminated. An example of this common language is *Abstract Syntax Representation, Rev 1* (ASN.1), an OSI recommendation. In fact, ASN.1 is used by the Simple Network Management Protocol (SNMP) to encode its high-level data.

## Layer 7: The Application Layer

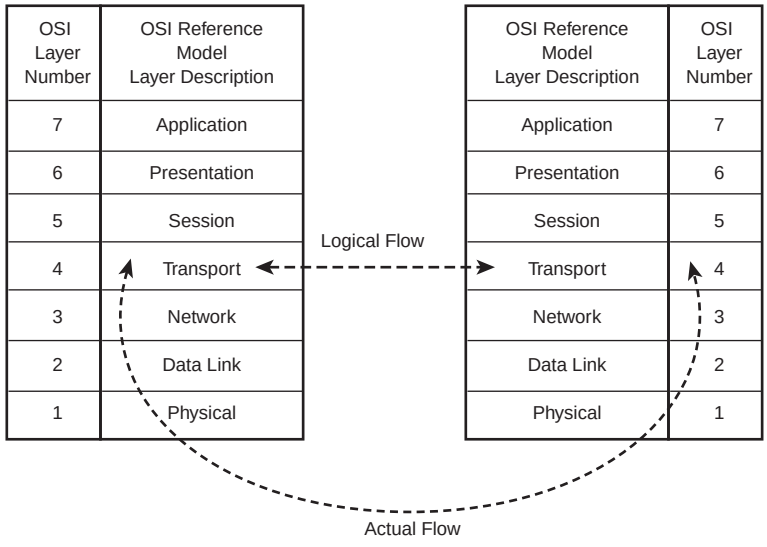
The top layer in the OSI Reference Model is the *Application Layer*. Despite its name, this layer does not include user applications. Rather, it provides the interface between those applications and the network's services.

This layer can be thought of as the reason for initiating the communications session. For example, an email client might generate a request to retrieve new messages from the email server. This client application automatically generates a request to the appropriate Layer 7 protocol(s) and launches a communications session to get the needed files.

# The Model's Usage

The vertical orientation of the stack is an acknowledgment of the functional flow of processes and data. Each layer has interfaces to its adjacent layers. In order to communicate, two systems must pass data, instructions, addresses, and so forth between the layers. The differences between the logical flow of communications and the actual flow of the session are illustrated in Figure 1.2.

**FIGURE 1.2**
*Actual versus logical flow of layered communications.*




**Note**

Although the Reference Model includes seven layers, not all layers are required for any given communications session. For example, communications across a single LAN segment can operate strictly at Layers 1 and 2 of the model, without requiring the other two communications layers.

Although communications flow vertically through the stack, each layer perceives itself to be capable of directly communicating with its counterpart layers on remote computers. To create this logical adjacency of layers, each layer of the originating machine's protocol stack adds a header. This header can be recognized and used by only that layer or its counterparts on other machines. The receiving machine's protocol stack removes the headers, one layer at a time, as the data is passed up to its application. This process is illustrated in Figure 1.3.

For example, segments of data are packaged by Layer 4 of an origination machine for presentation to Layer 3. Layer 3 bundles data received from Layer 4 into packets (that is, Layer 3 *packetizes* the segments), addresses them, and sends them to the destination machine's Layer 3 protocol by way of its own Layer 2. Layer 2 encases the packets with frames, complete with addressing that is recognized by the LAN. These frames are presented to Layer 1 for conversion into a stream of binary digits (bits) that are transmitted to the destination machine's Layer 1.

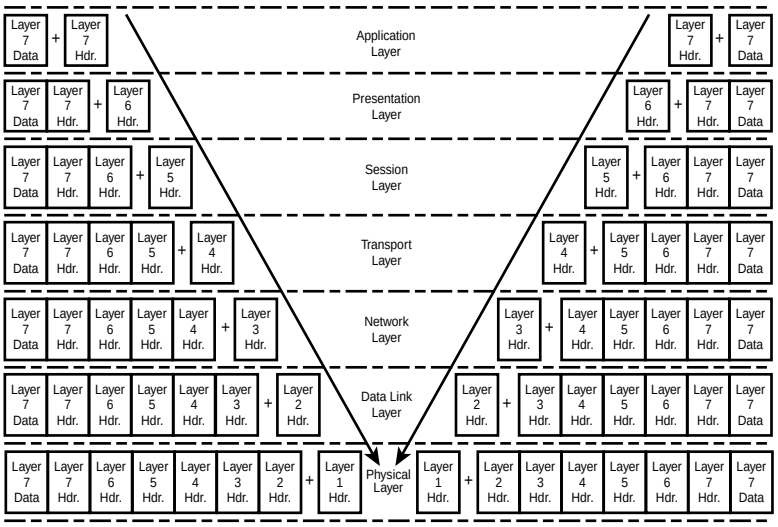



**FIGURE 1.3**
*Use of layered headers to support logical adjacency.*

The destination machine reverses this flow of handoffs, with each layer stripping off the headers that were added by their counterparts on the origination machine. By the time the data reaches the destination machine's Layer 4, the data is back in the same form into which the originating machine's Layer 4 put it. Consequently, the two Layer 4 protocols appear to be physically adjacent and communicating directly.

**Note**

Please note that most of today's networking protocols use their own layered models. These models vary in the degree to which they adhere to the separation of functions demonstrated by the OSI Reference Model. It is quite common for these models to collapse the seven OSI layers into five or fewer layers. It is also common for higher layers to not correspond perfectly to their OSI-equivalent layers.

In fact, each Layer 3 passes the data down to Layer 2, which, in turn, converts the frames to a bit stream. After the destination computer's Layer 1 device receives the bit stream, it is passed up to the Data Link Layer for reassembly into a frame. After the frame's receipt is successfully completed, the framing is stripped off and the packet that was

embedded in it is passed up to the recipient's Layer 3. It arrives in exactly the same form as that in which it was sent. To the Layer 3s, communications between them were virtually direct.

The fact that communication can appear to occur between adjacent layers (from the perspective of those layers) is a tribute to the success of the model.

Although the OSI Reference Model was originally intended as an architectural framework for open communications protocols, it has failed miserably in that capacity. In fact, the model has almost completely degenerated into just an academic construct. To its credit, the model is a marvelous device for explaining the concept of open communications, and the logical sequencing of necessary functions in a data communications session. A much more meaningful reference model is the TCP/IP Reference Model. This model describes the architecture of the IP suite of protocols, which are the focus of this book.

# The TCP/IP Reference Model

Unlike the OSI Reference Model, the TCP/IP model focuses more on delivering interconnectivity than on rigidly adhering to functional layers. It does this by acknowledging the importance of a hierarchical arrangement of functions, but still leaving protocol designers ample flexibility for implementation. Consequently, the OSI Reference Model is significantly better at explaining the mechanics of inter-computer communications, but TCP/IP has become the internetworking protocol of choice in the marketplace.

The flexibility of the TCP/IP Reference Model is shown in Figure 1.4 in comparison with the OSI Reference Model.

**FIGURE 1.4**
*Comparison of the OSI and TCP/IP Reference Models.*

| OSI Reference Model Layer Description | OSI Layer Number | TCP/IP Equivalent Layer Description |
|---|---|---|
| Application | 7 | Process/ Application |
| Presentation | 6 | |
| Session | 5 | |
| Transport | 4 | Host-to-Host |
| Network | 3 | Internet |
| Data Link | 2 | Network Access |
| Physical | 1 | |

The TCP/IP Reference Model, developed long after the protocol it explains, offers significantly more flexibility than its OSI counterpart because it emphasizes the hierarchical arrangement of functions, rather than strict functional layering.

# Dissecting the TCP/IP Model

The TCP/IP protocol stack includes four functional layers: Process/Application, Host-to-Host, Internet, and Network Access. These four loosely correlate to the seven layers of the OSI Reference Model without compromising functionality.

## The Process/Application Layer

The Application Layer provides protocols for remote access and resource sharing. Familiar applications such as Telnet, FTP, SMTP, HTTP, and many others all reside and operate in this layer and depend upon the functionality of the underlying layers. Similarly, any application (including homegrown and store-bought software) that requires communications over an IP network would also be described in this layer of the model.

## The Host-to-Host Layer

The IP Host-to-Host Layer correlates loosely to the OSI Reference Model's Session and Transport Layers. The functions supported in this layer include segmenting application data for transport across a network, performing mathematical tests to determine the integrity of received data, and multiplexing streams of data (both transmitted and received) for multiple applications simultaneously. This implies that the Host-to-Host Layer is capable of identifying specific applications, and can resequence data that might have been received out of order.

The Host-to-Host Layer currently consists of two protocol entities: *Transmission Control Protocol* (TCP) and *User Datagram Protocol* (UDP). A third entity has been defined to accommodate the increasingly transaction-oriented nature of the Internet. This protocol entity is tentatively called *Transaction/Transmission Control Protocol* (T/TCP). However, most application services on the Internet use the TCP and UDP protocols for the Host-to-Host layer.

## The Internet Layer

The Internet Layer of IPv4 consists of all the protocols and procedures necessary to allow data communications between hosts to traverse multiple networks. This means that the data-bearing packets must be routable. The Internet Protocol (IP) is responsible for making data packets routable.

The Internet Layer must also support other route management functions beyond just packet formatting. It must provide mechanisms for resolving Layer 2 addresses into Layer 3 addresses, and vice versa. These functions are provided by peer protocols to IP, which are described in Chapter 5, "ARP and RARP."

In addition, the Internet Layer must support routing and route management capabilities. These functions are also provided by external peer protocols, known as *routing protocols*. These protocols include Interior Gateway Protocols (IGP), and Exterior Gateway Protocols (EGP). These protocols are necessarily identified as peers because, even though they also reside at the Network Layer, they are not a native component of the IP suite of protocols. In fact, many of the routing protocols are capable of discovering and calculating routes in multiple routed protocol addressing architectures. Other examples of routed protocols with different address architectures include IPX and AppleTalk.

### The Network Access Layer

The Network Access Layer consists of all the functions needed to physically connect to, and transmit over, a network. The OSI Reference Model breaks down this set of functions into two layers: the Physical Layer and the Data Link Layer. The TCP/IP Reference Model, due to its creation after its namesake protocols, lumps these two layers together because the various IP protocols stop at the Internet Layer. IP assumes that all the lower-level functions are provided by either a LAN or serial connection.

## Summary

The basic network concepts presented in this chapter, their functions, and even their uses, are just the beginning. They are the proverbial building blocks that are explored in much greater detail throughout the remainder of this book. They have been presented to give you an appreciation for some of the fundamental terms and concepts of open networking.

Chapter 2, "TCP/IP and the Internet," builds on this framework by more closely examining the role of TCP/IP in the Internet, including the various mechanisms that are used to keep TCP/IP up-to-date. Both the OSI and TCP/IP Reference Models will be referred to throughout this book.

CHAPTER 2

# TCP/IP and the Internet

*by Neal S. Jamison*

## In This Chapter

TCP/IP allowed the Internet to evolve into what it is today, which subsequently has changed the way that we live and work in much the same way as its revolutionary forerunners such as the printing press, electricity, or the computer. This chapter discusses the development of the Internet, its caretakers, and future directions. It explores the process by which ideas go on to become standards, and it briefly introduces some of the more popular protocols and services, such as Telnet and HTTP (HyperText Transfer Protocol).

# A History of the Internet

The progression of events that led to the Internet goes back to the beginning of time. Cave wall drawings, smoke signals, the Pony Express—all forms of communication that made our forefathers think there had to be a better way. Next came the telegraph, the telephone, and transatlantic wireless messages—now we're getting somewhere. Then the first computers were invented. Massive, heat-generating calculators less powerful than our smallest handheld calculators today, these giants helped win wars and calculate the census. But very few were in existence, and no one could afford them, much less house them.

By the 1960s, transistors had replaced vacuum tubes, and the size and cost of computers were falling as the power and intelligence of these machines were rising. About this time, a group of scientists was working to allow computers to communicate. Leonard Kleinrock, then a Ph.D student at MIT, conceptualized the underlying technology of "packet switching," and published a paper on the subject in 1961. At that time the *Advanced Research Projects Agency* (ARPA) was looking for ways to improve communication within its own agency (as well as for the military), and Kleinrock's work provided the spark they needed. ARPA issued a *request for quotes* (RFQ) to solicit bids for creating the first packet-switched network. A small acoustics firm from Massachusetts known as Bolt Beranek and Newman (BBN) was awarded the contract, and thus the ARPANET was born. The year was 1969.

## The ARPANET

The original ARPANET consisted of four host computers—one each at UCLA, Stanford Research Institute, UC Santa Barbara, and the University of Utah. This small network, using the *Network Control Protocol* (NCP), provided its users with the ability to log in to a remote host, print to a remote printer, and transfer files. Ray Tomlinson, a BBN engineer, created the first e-mail program in 1971.

## TCP/IP

In 1974, just five short years after the birth of the ARPANET, Vinton Cerf and Robert Kahn invented the *Transmission Control Protocol* (TCP). TCP/IP, which was designed to be independent of the underlying computer and network, replaced the limited NCP in the early 1980s, and allowed the ARPANET to grow beyond anyone's expectations by allowing other heterogeneous ARPANET-like networks (or internets) to intercommunicate. Thus the Internet was born.



**Note**

An internet (with a lowercase "i") is a network of heterogeneous computers. The Internet (with an uppercase "I") is THE network connecting millions of computers and users.

The Department of Defense helped to promote the use of TCP/IP by selecting it as their standard protocol and requiring its use and support by defense contractors. Around this same time, developers at the University of California at Berkeley released the newest version of their Unix operating system, 4.2BSD (Berkeley Software Distribution), which was made freely available to everyone. This was a boon for TCP/IP due to its tight integration into 4.2BSD. BSD Unix became the basis for other Unix systems, which explains the predominance of TCP/IP in the Unix world.

TCP/IP provided the reliability that the Internet needed to get off the ground, and researchers and engineers began to add protocols and tools to the TCP/IP suite. FTP, Telnet, and SMTP have been around since the beginning. Newer TCP/IP tools include IMAP (Internet Mail Access Protocol), POP (Post Office Protocol), and of course, HTTP.

## The National Science Foundation (NSF)

Another network of great importance was the NSFNet. The National Science Foundation recognized the importance of work taking place with the ARPANET and decided to create a network of its own. The NSFNet connected a number of supercomputers with universities and government facilities. As NFSNet became more popular, the NSF increased the capability of the network by upgrading its "backbone" communication lines. Starting with 56k bit-per-second lines, progressing to T-1 (1.544Mbps) lines, and eventually to T-3 (43Mbps) lines, the NFSNet quickly became the fastest internet around.

Other networks that were created during this timeframe included the Because It's Time Network (BITNET) and the Computer Science Research Network (CSNET). The size of

the ARPANET was still growing at an alarming rate, with the number of hosts doubling every year.

In the late 1980s and early 1990s, NFSNet replaced the older and slower ARPANET, becoming the official backbone of the Internet.

## The Internet Today

In 1992, CERN (European Laboratory for Particle Physics) and Tim Berners-Lee demonstrated a concept called the World Wide Web (WWW), followed a year later by the release of a WWW client program called Mosaic. Together, these two events allowed the Internet to go from a text-only tool used by scientists and students to a graphical tool used today by many millions.

In April 1995, the NFSNet was retired and replaced with a competitive, commercial backbone. This relaxed the restrictions on connecting a host to the Internet, and opened it to a whole new type of user—the commercial user.

*Point-to-Point Protocol* (PPP) was created in 1994, and was becoming commonplace in 1995. PPP allowed TCP/IP over phone lines, which made it easier for home users to gain Internet access. This coincided nicely with an outcropping of *Internet service providers* (ISPs) who were willing and ready to connect home and business users, and the home-user boom was launched.

The Internet was (and still is!) growing at the alarming rate of 100 percent per year.

Today, a quick browse of the WWW allows us to see the significance of the evolution of the Internet. The Internet no longer is used only for academic and military communication and research. Today, we can shop and bank online. We can look up our favorite recipes or read books via the Web. The uses of the Internet today are endless.

# RFCs and the Standardization Process

Throughout the evolution of the Internet, ideas and notes have been presented in the form of documents called *Requests for Comments* (RFCs). These documents discuss many aspects of Internet-related computing and computer communication. The first RFC (RFC 1), titled “Host Software,” was written by Steve Crocker (a UCLA graduate student who wrote eight of the first 25 RFCs) in April 1969. These early RFCs provide fascinating reading to anyone interested in the history of the Internet. The documents that specify the Internet protocols, as defined by the IETF and IESG, are also published as RFCs.

The RFC Editor is the publisher of the RFCs and is responsible for the final editorial review of the documents. The Editor is discussed later in this chapter.

A good site for all types of RFC information is `http://www.rfc-editor.org/`.

**Note**

Jon Postel played a large role in the creation of the ARPANET, and thus the Internet. He participated in the creation of the Internet domain name system, which he administered for many years. That system is still in use today. Jon ran the *Internet Assigned Numbers Authority* (IANA) and was the RFC editor. Jon died in October 1998. RFC 2468 (Vinton Cerf, October 1998) is a tribute to Jon.



RFCs are the primary method for communicating new ideas on protocols, research, and standards. When a researcher comes up with a new protocol, research or tutorial on a topic, the researcher can submit this as an RFC document. Thus RFCs consist of Internet standards, proposals for new or revised protocols, implementation strategies, tutorials, collective wisdom, and so on.

Protocols that are to become standards in the Internet go through a series of states that describe their maturity levels. These levels are the proposed standard, draft standard, and standard. As each standard evolves, the amounts of scrutiny and testing increase. When a protocol completes this process, it is assigned a STD number, which is the number of the new standard.

As technology advances, some protocols are superseded by better ones or may be unused. These protocols are documented with the designation “historic.” The results of early protocol research and development work are documented in some RFCs. These protocols are designated “experimental.”

Protocols that are developed by other standards organizations, or by particular vendors and individuals, may be of interest or may be recommended for use in the Internet. The specifications of such protocols may be published as RFCs for the convenience and ease of distribution to the Internet community. These protocols are labeled “informational.”

Sometimes protocols may achieve widespread implementation without the approval or involvement of the IESG. This may be because of the evolution of protocols in the marketplace or because of political reasons. Some of these vendor protocols may even become very important to the Internet community because business organizations may have deployed them within their intranets. The position of the IAB is that it strongly recommends that the standards process be used in the evolution of the protocol to maximize interoperability and to prevent incompatible protocol requirements from arising.

Not all the protocols are required to be implemented in all systems because there is such a variety of possible systems where a protocol implementation may not make sense. For example, Internet devices such as gateways, routers, terminal servers, workstations, and multi-user hosts have a different set of requirements. Not all of them require the same protocols.

## Getting RFCs

RFCs can be obtained from several different repositories. You might want to start by referencing an RFC index. See the following section for a list of RFC indices.

You can obtain RFCs in several different ways, including via the Web, FTP, Telnet, and even e-mail. Table 2.1 lists several FTP-accessible RFC repositories.

**TABLE 2.1** FTP-Accessible RFC Repositories

| *Site* | *Login/Password Directory* |
|---|---|
| `ftp.isi.edu` | `anonymous/name@host.domain/in-notes` |
| `wuarchive.wustl.edu` | `anonymous/name@host.domain/doc/rfc` |

Several of the RFC repositories also provide RFCs via e-mail. For example, you can send a mail message to `nis-info@nis.nsf.net` with a blank subject line and the text of the message reading "send rfc*nnnn*.txt" where *nnnn* is the desired RFC number.

As expected, many of these repositories also provide the documents over the Web. A good starting point for accessing RFCs via the Web is `http://www.rfc-editor.org/`.

See `http://www.isi.edu/in-notes/rfc-retrieval.txt` for more information on obtaining RFCs.

## Indices of RFCs

A complete index of all RFC documents would take up far too many pages to include it here. However, several great indices are on the Web in ASCII text, HTML, and searchable format. These include the following:

- Text—`ftp://ftp.isi.edu/in-notes/rfc-index.txt`
- HTML—`ftp://ftpeng.cisco.com/fred/rfc-index/rfc.html`
- HTML by Protocol—`http://www.garlic.com/~lynn/rfcprot.htm`

An RFC search engine can be found at `http://www.rfc-editor.org/rfcsearch.html`.

## Humorous RFCs

It may surprise you to learn that not all RFCs are of a serious nature. Table 2.2 lists several of the less serious RFCs.

**TABLE 2.2** Humorous RFCs

| *RFC* | *Title* |
|---|---|
| 527 | ARPAWOCKY |
| 968 | 'Twas The Night Before Startup |
| 1097 | Telnet subliminal-message option |
| 1121 | Act one: the poems |
| 1149 | A standard for the Transmission of IP Datagrams on Avian Carriers |
| 1300 | Remembrances of Things Past |
| 1438 | IETF Statements of Boredom (SOBs) |
| 1882 | The 12 days of technology before Christmas |
| 1925 | The 12 networking truths |
| 1927 | Suggested additional MIME types for associating documents |



# A Brief Introduction to Internet Services

Without popular protocols and services such as HTTP, SMTP, and FTP, the Internet wouldn't be much more than a large number of computers connected into a worthless knot. This section briefly describes the popular (and most useful) of the Internet protocols and points you to other chapters in this book for more information.

## Whois and Finger

Whois is the service and protocol that allows us to locate information about Internet hosts and domains. By querying any of the available Whois database servers, Whois clients can gather information such as host and domain points-of-contact, geographical addresses, and more. Whois is also used by some organizations as a form of online personnel directory. This is especially common at universities.

Whois can be found on well-known TCP port 43 and is specified by RFC 954.

Finger is the service/protocol that allows us to gather information about Internet users. By "fingering" someone, you can get their e-mail address, find out whether he or she has mail or is currently online, and even read a little about what he or she is working on. Due to the nature of this service, some host administrators choose to disable Finger. Finger listens on TCP port 79 and is specified by RFC 1288.

For more information on these services, refer to Chapter 29, "Whois and Finger."

## File Transfer Protocol

*File Transfer Protocol* (FTP) is the service/protocol that enables the transfer of files across the Internet. It, too, is one of the earlier protocols, dating back to 1971. FTP is commonly used today for public file sharing (via anonymous FTP). FTP operates on well-known TCP ports 20 and 21 and is specified by RFC 959.

For more information) about FTP and other transfer protocols, see Chapter 30, "File Transfer Protocols."

## Telnet

Telnet is the terminal emulation program for the Internet. Put in simple terms, Telnet allows you to log in to remote hosts without worrying about terminal compatibility. Telnet was one of the very first protocols and services in the early Internet (see RFC 15). Telnet operates on well-known TCP port 23 and is specified by RFC 854.

For more information about Telnet, see Chapter 31, "Using Telnet."

## E-mail

*Simple Mail Transfer Protocol* (SMTP) is the Internet standard for e-mail. Many people use this protocol every day without even realizing it. SMTP is accompanied by other protocols and services, such as POP3 and IMAP4, which allow you to manipulate your mail on the mail server and download it to your local computer for reading. SMTP operates on well-known TCP port 25 and is specified in RFC 821.

For more information about SMTP and other e-mail-related protocols and services, see Chapter 35, "Internet Mail Protocols."

## The World Wide Web

HTTP is the language of the World Wide Web. In fact, HTTP can be credited for the mid-1990s explosion of the Internet. HTTP clients (like Mosaic and Netscape) emerged that allowed us to "see" the Web for the first time. This was quickly followed by the

appearance of Web servers presenting useful information. Today over six million HTTP-speaking Web sites are on the Internet. HTTP operates on well-known TCP port 80 and its current version, HTTP/1.1, is specified by RFC 2616.

For more information about HTTP and other Web-related services and protocols, refer to Chapter 36, "The HTTP (World Wide Web) Service."

## USENET News

The *Network News Transfer Protocol* (NNTP) is the protocol/service used to post, transfer, and retrieve USENET news messages. USENET, a history lesson in itself, is an Internet-sized bulletin board system made up of newsgroups—discussion forums covering almost any subject possible. NNTP operates on well-known TCP port 119 and is specified by RFC 977.

For more information about NNTP, see Chapter 37, "The NNTP (Network News) Service."



# Intranets and Extranets

After the commercialization of the Internet in 1991, corporations didn't take long to find new and improved ways to use the Internet and its services and technologies to save time, money, and gain strategic advantage. One of the largest applications of this technology to date may be the *intranet*.

## Intranets

An *intranet* is a local network controlled by an organization that uses TCP/IP as its primary communication protocol. The application services on the intranet are based around the standard services available on the Internet, such as HTTP, FTP, TELNET, SSH, and so on.

Simply put, an intranet is a finite, closed network of computers that uses Internet technologies to share data. An intranet may be a subset of the Internet with access controls in place to keep out the uninvited. Or for that matter, an intranet doesn't have to be on the Internet at all.

### Intranet Benefits

There are many benefits of implementing an organizational intranet. Due to the relatively low cost to implement and maintain, companies are realizing very high ROIs (return on investment). Benefits include the following:

- **Intranets are easy to use.** Because the user interface of the intranet is the Web browser, training costs are low to none.
- **Intranets facilitate the dissemination of information to employees.** Whether it is a one-page corporate memorandum or a 500-page telephone directory, intranets allow companies to share information with their employees with great efficiency.
- **Intranets reduce printing costs.** Not only is it logistically difficult to disseminate information in a large company or organization, it can also be very expensive.
- **Intranets add value to traditional documents.** Although looking up a name or product in a paper-based directory or catalog is not extremely difficult, it can be time consuming and tedious.
- **Intranets improve data accuracy.** Documents can become outdated soon after they are printed. Having copies of outdated and possibly inaccurate documents circulating can be dangerous.

These are just a few of the benefits of intranets. Companies that have deployed intranets have realized many more benefits than are listed here. By using Internet technologies and services such as "open source" Web server and browser software, all of these benefits can come at a relatively low cost.

## Example Applications of the Intranet

The ways in which your intranet can be applied to making your organization more efficient and productive are just about endless. Here are some common uses of intranet technology:

- **Human Resources:** More and more companies are circulating job advertisements, maintaining databases of employee skills, distributing documents such as 401k forms, timesheets, expense reports, and even paycheck direct-deposit slips, all via the intranet.
- **Project Management:** Spreadsheets and Gantt charts can be posted on the intranet where they can be viewed and updated by project managers. Status reports can be posted to the intranet where they are available for managers to review and comment on.
- **Inventory Tracking:** Inventory databases can be made available online either in their raw data format or as value-added applications.
- **Office File Management:** A robust intranet server running open Web technologies can put your file server out of business.

Although Intranets are useful to corporate users, external clients can also benefit from using the corporate intranet. This has led to the creation of extranets, discussed in the next section.

## Opening Our Intranets to Outsiders

Whereas intranets are intended to allow members of a company or organization to share information among themselves, an extranet goes one step further. *Extranets* are actually intranets that securely open their doors to an invited group of outsiders. One practical use of the extranet is to let companies share information with strategic partners, such as customers, shippers, or suppliers. In short, extranets are business-to-business networks.



Some uses of extranets follow:

- Conduct business using *Electronic Data Interchange* (EDI) or some other application.
- Collaborate with other organizations on projects.
- Share news or other information with partner organizations.

An example of an extranet in action is a major shipping company that allows a popular online bookseller access to its intranet so that the bookseller can arrange the shipping of its products to customers.

# The Internet of Tomorrow

As the popularity and use of the Internet expands, so, too, does Internet technology. Several initiatives are underway to improve upon these technologies. Three of the most promising initiatives are

- The Next Generation Internet Initiative (NGI)
- Very high-speed Backbone Network Service (vBNS)
- Internet2 (I2)

Each of these is described in the following sections.

## Next Generation Internet (NGI)

As called for by President Clinton in his 1998 State of the Union address, the *Next Generation Internet Initiative* is designed to fund and coordinate academia and federal agencies to design and build the next generation of Internet services.

For more information about the Next Generation Internet Initiative, refer to `http://www.ngi.gov/`.

## vBNS

The National Science Foundation has launched an effort to create an experimental Wide Area Network (WAN) backbone of tremendous speed. This backbone network, implemented by MCI WorldCom, is known as the *very high-speed Backbone Network Service* (vBNS). vBNS will serve as a platform for testing new, high-speed Internet technologies and protocols. It currently links several supercomputer centers and network access points at OC-12 speeds (622Mbps) and higher. In February 1999, MCI WorldCom announced the installation of an OC-48 (2.5Gbps) link between Los Angeles and San Francisco, California.

For more information on vBNS, refer to `http://www.vbns.net/`.

## Internet2 (I2)

*Internet2* is a testing-ground network created to allow universities, government, and industry participants to collaboratively develop advanced Internet technologies. Internet2 partners connect using the Abilene network, which boasts very high speeds of up to 9.6Gbps. I2 also uses the vBNS network described in the previous section.

To learn more about I2, refer to `http://www.internet2.edu/`. For more information on Abilene, refer to `http://www.ucaid.edu/abilene/`.

# Who's In Charge Anyway?

As large as the Internet has become and with all the high-technology initiatives going on to enhance it, you might think that the group in charge of the Internet stays very busy. Well, you are only partly correct: No one group is "in charge" of the Internet. There is no director or CEO, or even a president. The fact is that the Internet is still thriving under the 1960s anarchistic counterculture from which it was born. There are, however, several groups that help oversee the technologies of the Internet, the registration processes, and the other intricacies of running a major network.

## The Internet Society (ISOC)

The *Internet Society* is a professional membership society with more than 150 organizational and 6,000 individual members in over 100 countries. Together, these organizations and members oversee the issues that affect the Internet and its future. The ISOC is made up of several of the groups responsible for Internet infrastructure standards, including the *Internet Architecture Board* (IAB) and the *Internet Engineering Task Force* (IETF).

The ISOC is on the Web at `http://www.isoc.org/`.

## Internet Architecture Board (IAB)

Formerly known as the Internet Activities Board, the Internet Architecture Board is the technical advisory arm of the Internet Society. This small group (IAB members are nominated by the IETF and approved by the ISOC Board of Trustees) meets regularly to review and stimulate new ideas and proposals that need to be developed by the Internet Engineering Task Force and the Internet Engineering Steering Group.

The IAB Web site is at `http://www.iab.org/`.



## Internet Engineering Task Force (IETF)

The Internet Engineering Task Force is an open community of network designers, vendors, and researchers concerned with the evolution of the Internet. The IETF meets only three times each year, and accomplishes most of its work via electronic mailing lists. The IETF is broken into working groups, with each group assigned to one specific topic. IESG working groups include the *Hypertext Transfer Protocol* (HTTP) and the *Internet Printing Protocol* (IPP) working group.

The IETF is open to anyone, and can be found on the Web at `http://www.ietf.org/`.

## Internet Engineering Steering Group (IESG)

The Internet Engineering Steering Group is responsible for technical management of IETF activities and the Internet standards process. The IESG also ensures that everything goes according to the rules and procedures of the ISOC. The IESG gives final approval of specifications before they are adopted as Internet Standards.

You can locate more information about the IESG at `http://www.ietf.org/iesg.html`.

## Internet Assigned Numbers Authority (IANA)

The Internet Assigned Numbers Authority is responsible for assigning IP addresses and managing the domain name space. IANA also controls IP protocol port numbers and other parameters. IANA operates under the auspices of ICANN.

Visit IANA at `http://www.iana.org/`.

## Internet Corporation for Assigned Names and Numbers (ICANN)

The Internet Corporation for Assigned Names and Numbers was formed as part of an effort to internationalize the administration of domain names and IP address space. The

goal of the ICANN is to help transition the policy administration of Internet domains and addresses from government to private-sector. Currently, ICANN is involved in the *Shared Registry System* (SRS) through which the Internet domain registration process is being opened to fair competition. See the next section, "The InterNIC and Other Registrars," for more information on the SRS.

For more information on ICANN, visit its homepage at `http://www.icann.org/`.

## The InterNIC and Other Registrars

The InterNIC (short for Internet Network Information Center), operated by Network Solutions, Inc., has been the primary registrar for the top-level domains (`.com`, `.org`, `.net`, `.edu`) since 1993. The InterNIC is overseen by the *National Telecommunications & Information Administration* (NTIA), a subgroup of the Department of Commerce. The InterNIC has delegated some responsibility to other official registrars (such as the Department of Defense NIC, and the Asia-Pacific NIC). More recently there have been other initiatives that could break up the InterNIC even further. One such initiative, known as the *Shared Registry System* (SRS), strives to introduce fair and open competition to the domain registration process. Currently over 60 companies are acting as registrars under this initiative.

Table 2.3 lists some of the major registrars.

**TABLE 2.3** Major Internet Registrars

| *Name* | *URL* |
|---|---|
| InterNIC | `http://www.internic.net/` |
| Department of Defense NIC | `http://nic.mil/` |
| U.S. Federal Register | `http://nic.gov/` |
| Asia-Pacific NIC | `http://www.apnic.net` |
| RIPE (Réseaux IP Européens) | `http://www.ripe.net` |
| Council of Registrars (CORE) | `http://www.corenic.org/` |
| Register.com | `http://register.com` |

## The RFC Editor

The Requests for Comments are a series of notes and documents that, among other things, specify the standards of the Internet. For more information about RFCs, refer to the section "RFCs and the Standardization Process" earlier in this chapter.

The RFC Editor is the publisher of the Internet RFC documents and is responsible for the final editorial review of the documents.

For more information on the RFC Editor, see `http://www.rfc-editor.org/`.

## The Internet Service Providers

The commercialization of the Internet in the 1990s was greeted by numerous Internet service providers just waiting to help millions of household and business users onto the Internet. *ISPs* are businesses that set up Internet servers within their office space or computer rooms. These servers are equipped with modems and run either the Point-to-Point Protocol (PPP) or the Serial Line Internet Protocol (SLIP). These protocols allow remote users to dial in from their personal computers and connect to the Internet.

For a fee, an ISP will provide a remote user with connectivity to the Internet. Most common ISPs also provide an email account on their server, and some even provide a UNIX shell account.

Larger, wholesaler ISPs can provide businesses and other ISPs with high-speed networks such as ISDN, fractional T-1, and even higher.

Internet.com provides a database of ISPs, searchable by telephone area code. Visit this ISP buyer's guide at `http://thelist.internet.com/`.

# Summary

This chapter described the Internet through a look at its history, its evolution, and other related topics. Intranets and extranets, a modern use of Internet technology, were described as well.

The Request for Comments (RFC) process was explained, and links were provided to allow you to retrieve RFCs for further reading.

Some of the most popular Internet services, such as HTTP, Telnet, FTP, and SMTP, were described, and references were given to chapters later in this book to which you can turn for more information.

We looked into the organizations that have made the Internet what it is today and will continue to evolve it as technology and needs change. It's hard to tell what the future holds for the Internet. In just a few years, it has grown from a tiny experimental network used by a handful of scientists to a global internetwork of many millions of computers and users. One thing is for sure: With projects like the Next Generation Internet, I2, and vBNS, we've only just begun to see the capabilities and applications of this fascinating technology.

CHAPTER 3

# Overview of TCP/IP

*by Rima S. Regas*

## In This Chapter

TCP/IP is everywhere. It's not something physical that can only be in one place at a time. It's a set of protocols that allows anyone with a computer, modem, and an Internet service provider to access and share information over the Internet. In fact, users of AOL's Instant Messenger service and ICQ (also owned by AOL) account for over 750 million messages per day. This is an incredible amount of traffic, and most of it is transmitted over the Internet.

It is TCP/IP that allows these millions of transactions per day to occur (it's actually well into the billions because many more things are going on than email and instant messaging), mostly without a hitch. And it shows no signs of letting up anytime soon. TCP/IP is a stable, well-established, complete set of protocols, and this chapter takes a close look at exactly what makes it tick.

Both TCP and IP, two separate protocols that work hand-in-hand, perform chores that manage and guide the general mobility of data packets over the Internet. They both use special headers that define each packet's contents and, if there is more than one, how many others should be expected. TCP concerns itself with making the connections to remote hosts. IP, on the other hand, deals with addressing so that messages are directed to where they are intended. The following section takes a look at the benefits of TCP/IP.

# The Benefits of Using TCP/IP

TCP/IP enables cross-platform, or heterogeneous, networking. For example, a Windows NT/2000 network could contain Unix and Macintosh workstations or even networks mixed in it. TCP/IP also has the following characteristics:

- Good failure recovery
- The ability to add networks without interrupting existing services
- High error-rate handling
- Platform independence
- Low data overhead

Because TCP/IP was originally designed for Department of Defense–related purposes, what we now call features or characteristics were actually design requirements. The idea behind "Good Failure Recovery" was that if a portion of the network were disabled during an incursion or attack, its remaining pieces would still be able to function fully. Likewise is the capability of adding entire networks without any disruption to the services already in place. The ability to handle high error rates was built in so that if a packet of information got lost using one route, there would be a mechanism in place to ensure that it would reach its destination using another route. Platform independence

means that the networks and clients can be Windows, Unix, Macintosh, or any other platform or combination thereof. The reason TCP/IP is so efficient lies in its low overhead. Performance is key for any network. TCP/IP is unmatched in its speed and simplicity.

# TCP/IP Layers and Protocols

TCP and IP together manage the flow of data, both in and out, over a network. Whereas IP indiscriminately pumps packets into the ether, TCP is charged with making sure they get there. TCP is responsible for the following:

- Opening and closing a session
- Packet management
- Flow control
- Error detection and handling

## Architecture

TCP/IP is the environment that handles all these operations and coordinates them with remote hosts. TCP/IP was created using the DoD (Department of Defense) model, which is made up of four layers instead of the seven that make up the OSI model.

The primary difference between the OSI and the TCP/IP layer formats is that the Transport Layer does not guarantee delivery at all times. TCP/IP offers the *User Datagram Protocol* (UDP), a more simplified protocol, wherein all the layers in the TCP/IP stack perform specific duties or run applications.

### Application Layer

The Application Layer consists of protocols such as SMTP, FTP, NFS, NIS, LPD, Telnet, and Remote Login, all of which fall into areas that are familiar to most Internet users.

### Transport Layer

The Transport Layer consists of UDP (User Datagram Protocol) and TCP, where the former delivers packets with almost non-existent checking, and the latter provides delivery guarantees.

### Network Layer

The Network Layer is made up of the following protocols: ICMP, IP, IGMP, RIP, OSPF, EGP and BGP4. ICMP (Internet Control Message Protocol) is used for diagnostics and

to report on problems with the IP layer and for obtaining information about IP parameters. IGMP (Internet Group Management Protocol) is used for managing multicasting. RIP, OSPF, EGP, and BGP4 are examples of routing protocols.

### Link Layer

The Link Layer consists of ARP and RARP, and framing of data, which handle packet transmission.

## Transmission Control Protocol (TCP)

*Transmission Control Protocol* (TCP) is a protocol that provides a reliable stream delivery and connection service to applications. TCP uses sequenced acknowledgment and is able to retransmit packets as needed.

The TCP header appears as follows:

| 16-bit | | | | | | | 32-bit |
|---|---|---|---|---|---|---|---|
| Source Port | | | | | | | Destination Port |
| Sequence Number | | | | | | | |
| Acknowledgement Number (ACK) | | | | | | | |
| Offset Reserved | U | A | P | R | S | F | Window |
| Checksum | | | | | | | Urgent Pointer |
| Options and Padding | | | | | | | |

The parts of the header are detailed in the following sections.

### Source Port

The source port is the numerical value indicating the source port.

### Destination Port

The destination port is the numerical value indicating the destination port.

### Sequence Number

The sequence number is the number of the first data octet in any given segment.

## Acknowledgement Number (ACK)

When the ACK bit is set, this field contains the next sequence number that the sender of the segment is expecting to receive. This value is always sent.

## Data Offset

Data offset is the numerical value that indicates where the data begins, implying the end of the header by offset.

## Reserved

Reserved is not used, but it must be off (0).

## Control Bits

The control bits are as follows:

| | |
|---|---|
| U(URG) | Urgent pointer field significant |
| A(ACK) | Acknowledgement field significant |
| P(PSH) | Push function |
| R(RST) | Reset connection |
| S(SYN) | Synchronize sequence numbers |
| F(FIN) | No more data |

## Window

The window indicates the number of octets the sender is willing to take. This starts with the packet in the ACK field.

## Checksum

The checksum field is the 16-bit complement of the sum of all 16-bit words, restricted to the 1s column, in the header and text. If the result is an odd number of header and text octets, then the last octet is padded with zeros to form a 16-bit word that will checksum. Note that the pad is not sent as part of the segment.

## Urgent (URG) Pointer

The URG pointer field shows the value of the URG pointer in the form of a positive offset of the sequence number from the octet that follows the URG data or points to the end of urgent data.

## Options

Options may be sent at the end of a header, but must always be fully implemented and have a length that is any multiple of 8-bits. The two types of options are

- Type 1: A single octet of option-kind.
- Type 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option has a type, length and value field.

The length field includes the type, option length, as well as the value fields. The value field contains the data that makes up the options being sent. Table 3.1 shows how the octets are formatted in one of three classes. All the options are included in the checksum. An option can begin on any octet boundary as long as any remaining "dead" space is padded to meet the defined packet length.

**NOTE**

The list of options can be shorter than that designated by the data offset field because the contents of the header beyond the End-of-Option option must be padded with zeros (0).

**TABLE 3.1** Formatted Octets

| *Class* | *Length* | *Description* |
|---|---|---|
| 0 | - | End of options |
| 1 | - | No operation |
| 2 | 4 | Maximum segment size |

The Class 0 option indicates the end of the option list, ending all options, but not each individual option. Only use this option if the end of the options list does not coincide with the end of the header.

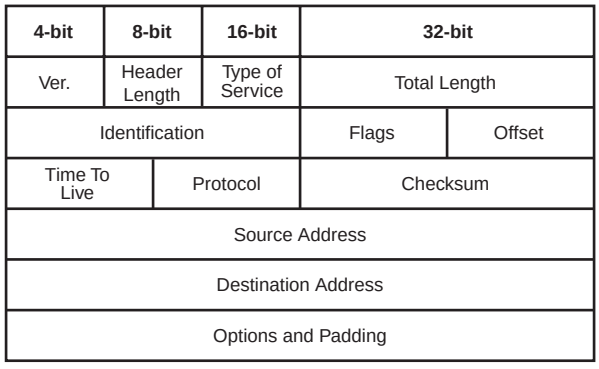
The Class 1 option is used to align an option with a word boundary. Other than that this option does not contain any useful information. Its main purpose is for aligning and formatting.

The Class 2 option indicates the maximum segment size that it will receive. It can only be included in the initial request and in segments where the SYN item bit is set. If this option is not used, there is no size limit.

# Internet Protocol (IP)

IP manages how packets are delivered to and from servers and clients.

The IP header appears as follows:




Each field contains information about the IP packet that it carries. The descriptions in the following sections should be helpful.



## Version Number

The version number indicates the version of IP that is in use for this packet. IP version 4 (Ipv4) is currently in widespread use.

## Header Length

The header length indicates the overall length of the header. The receiving machine then knows when to stop reading the header and start reading data.

## Type of Service

Mostly unused, the Type of Service field indicates the importance of the packet in a numerical value. Higher numbers result in prioritized handling.

## Total Length

Total length shows the total length of the packet in bytes. The total packet length cannot exceed 65,535 bytes or it will be deemed corrupt by the receiver.

## Identification

If there is more than one packet (an invariable inevitability), the identification field has an identifier that identifies its place in line, as it were. Fragmented packets retain their original ID number.

## Flags

The first flag, if set, is ignored. If the DF (Do Not Fragment) flag is set, under no circumstances can the packet be fragmented. If the MF (More Fragments) bit is turned on (1), there are packet fragments to come, the last of which is set to off (0).

## Offset

If the Flag field returns a 1 (on), the Offset field contains the location of the missing piece(s) indicated by a numerical offset based on the total length of the packet.

## Time To Live (TTL)

Typically 15 to 30 seconds, TTL indicates the length of time that a packet is allowed to remain in transit. If a packet is discarded or lost in transit, an indicator is sent back to the sending computer that the loss occurred. The sending machine then has the option of resending that packet.

## Protocol

The protocol field holds a numerical value indicating the handling protocol in use for this packet.

## Checksum

The checksum value acts as a validation checksum for the header.

## Source Address

The source address field indicates the address of the sending machine.

## Destination Address

The destination address field indicates the address of the destination machine.

## Options and Padding

The Options field is optional. If used, it contains codes that indicate the use of security, strict or loose source routing, routing records, and timestamping. If no options are used, the field is called *padded* and contains a 1. Padding is used to force a byte value that is rounded. Table 3.2 indicates the bit counts for the options available.

**TABLE 3.2** Bit Counts

| *Class* | *Number* | *Option* |
|---|---|---|
| 0 | 0 | End of option list |
| 0 | 2 | Military security |
| 0 | 3 | Loose source routing |
| 0 | 7 | Routing record* |
| 0 | 9 | Strict source routing |
| 2 | 4 | Timestamping |

**This option adds fields.*

TCP/IP provides its services via a "stack" of translation layers, which are called, of all things, TCP/IP. Because TCP and IP are separate protocols, they require a common environment for translation services. As mentioned earlier in this chapter, the TCP/IP stack has four layers as opposed to the OSI's seven layers. In a nutshell, they are

- Application
- Transport
- Network
- Link

Each of these is discussed in the following sections.



## Application Layer

The Application Layer combines a few services that the OSI separates into three layers. These services are end user-related: authentication, data handling, and compression. This is where email, Web browsers, telnet clients, and other Internet applications get their connections.

## Transport Layer

Again, unlike OSI, this layer is not responsible for guaranteeing the delivery of packets. Its primary responsibility is managing the transfer between the source and the destination. The OSI guarantees that packets are checked and, if they don't match, are dumped and re-requested from the source.

## Network Layer

The Network Layer deals strictly with packet-routing management. This layer is well-suited to making determinations on where to send packets based on the information it receives.

## Link Layer

The Link Layer manages the connection of the network and provides packet I/O over the network, but not at the application level.

Now that you have a clear idea of what TCP/IP is and what it can do (although hands-on experience is as close as your nearest Internet-connected computer), the following section moves on to the tangible benefits that TCP/IP provides you.

# Telnet

Telnet, short for TELecommunications NETwork, refers to both the application and the protocol itself, granting the name a dual role. Telnet provides users a way to log in and directly access their terminal across a network. This means actual, direct access to the remotely located computer. Telnet is provided on port 23.

Telnet requires that a Telnet server be located on the host machine, awaiting an authenticated login session from a remote location. Windows 9x/NT/2000, the BeOS, Linux, and other x86 platform–based operating systems require that a Telnet server be installed, configured, and running to accept incoming sessions. MacOS-based systems also require a Telnet server. Unix-based computers are the only systems that come with one and typically use an application called `telnetd` (the "d" denotes daemon, a server application). On the other end is a Telnet application that acts as an interface, either text-based or a GUI, for the session.

**Note**

Windows 2000 actually has a CLI Telnet application built in. If you click on a Telnet link or type **`Telnet`** in the console, it will appear. Therefore, Windows 2000 no longer requires the addition of a third-party Telnet server.

# File Transfer Protocol (FTP)

Whereas Telnet facilitates a live connection to the remote host, FTP is more passive, allowing you to move files back and forth from remotely located servers. This utility is ideal for Webmasters or anyone requiring to move large files from one location to another with no previously established "hot" connection. FTP is typically operated in what is called Passive Mode, which loads directory trees to the client and then disconnects, but periodically "tickles" the server to maintain an open port.

**Note**

Various FTP servers will be configured based on the particular Webmaster's tastes. Some allow anonymous users to access all areas of the server without restriction. Others limit access to previously authenticated users only. Still others limit anonymous access to very short timeout periods. If the user is not active, the server automatically disconnects, forcing him to reconnect if he wants to continue using the server.



On Unix-based systems, these programs are typically named `ftpd` (again, "d" meaning daemon) and `ftp` (the client application). FTP's default ports are 20 (for data transfers) and 21 (for command transfers). This makes FTP unique among TCP/IP protocols because commands and data can be transferred simultaneously with the data being transferred in real time, a feature that other protocols do not share.

All operating systems have FTP clients and servers in one form or another. All MacOS-based FTP applications are graphically oriented. Most Windows-based ones now are as well. The benefits to using a graphical FTP client is that all the commands, usually entered by hand, are now managed by the client, reducing the possibility of error and making sessions quicker and easier. On the other hand, because FTP servers do not require much management after the initial setup, they don't require a GUI.

# Trivial File Transfer Protocol (TFTP)

TFTP lives up to its name quite well. TFTP is the poor cousin of FTP in that it shares only a very small subset of the capabilities of FTP. It uses UDP, which, to use a similar metaphor, is TCP's poor relative. TFTP has no packet-monitoring capabilities and practically no error-handling capabilities. But then again, these limitations also reduce the process overhead. TFTP does not authenticate; it merely connects. As a built-in protection, TFTP can only move files that are publicly accessible.

Security is of great concern when employing TFTP. As a result, TFTP is typically used for embedded applications and copying configuration files for router configuration and in situations where space is of concern, and where security is handled in another fashion. TFTP is also used in a network computer environment where each machine is booted from a remote server and where TFTP can be easily embedded in the ROMs (Read Only Memory) on network cards.

# Simple Mail Transfer Protocol (SMTP)

SMTP is the de facto standard for transferring email over networks, primarily the Internet. All operating systems have email clients that can use SMTP and most, if not all, Internet service providers use SMTP as their outgoing mail service. There are also SMTP servers for all operating systems including, but not limited to, Windows 9x/NT/2K, MacOS, Unix and variants, Linux, BeOS, and even AmigaOS.

SMTP is designed to provide a transport for email messages under various network environments. In fact, SMTP is not really concerned with *how* it travels, just with getting the message to its destination.

SMTP has robust mail handling features that allow mail to be automatically routed based on certain criteria. SMTP has the ability to immediately notify a user of a nonexistent email address and to return mail to the sender when the mail remains undeliverable for a period of time (set by the system administrator of the server that hosts the message). SMTP uses TCP port 25.

# Network File System (NFS)

Sun Microsystems Inc. created NFS as an answer to problems with working in harmony across networks supporting many operating systems. NFS supports file sharing only and is now an integral part of many Unix-based operating systems. It is also well supported by most other operating systems.

NFS versions 1 and 2 used UDP as the primary transport protocol. Because UDP does not provide reliable delivery, reliable delivery had to be handled by NFS itself for unreliable links. Some of the early implementations of NFS had file corruption problems. Starting with NFS version 3, NFS can alternatively use TCP as the transport protocol. Actually, TCP support was available in NFS version 3 but was not optimized. With TCP as the transport protocol, NFS can use the reliability of TCP to provide better delivery over unreliable links. Consequently, NFS version 3 works better over WAN links and the Internet.

**Note**

An NFS version 4 has been proposed but most commercial implementations either implement NFS version 2 or version 3.

Pure NFS has no way of preventing various users from writing to a single file at the same time, allowing users to easily corrupt files at will without knowledge of other file activity. However, a file-locking mechanism implemented by a Network Lock Manager (NLM) protocol can be used in conjunction with NFS to enable file sharing and overwriting between multiple users.

NFS file access is seamless and transparent. After an NFS volume is mounted, it becomes part of the end user's system. There are no additional steps, beyond the export process, of course. Exporting is required to synchronize both server and clients to the NFS configuration. This system is neither simple nor administrator friendly.

# SNMP

SNMP provides a simple level of router monitoring and management via various protocols, such as UDP, IPX, or IP. Simple is an important word to remember in any discussion of SNMP. SNMP is nothing if not simple. First of all, it only supports four commands—`GET`, `GETNEXT`, `SET`, and `TRAP`. The first two provide access to reporting information, and the third allows you to remotely control certain functions of the routers. The `TRAP` command enables devices to report on problems or events within the device.

Network devices supply their specific information via a *Management Information Base* (MIB). This data, which defines the device to the SNMP manager, is fed to the SNMP Management Station, which in turn identifies each device and stores its specific data. All SNMP-compliant devices are managed from this station. Each device runs an SNMP Agent that provides the client side of the operations for the device. When the Management Station requests a `GET` command for port conditions, the agent returns that information.

SNMP is not meant to manage all network devices to a high level of detail. This is simple, day-to-day management that allows you to pay close attention to your devices without having to load half a dozen management interfaces. SNMP uses the UDP transport protocol for sending messages.

# How TCP/IP Fits into Your System

You've seen what services TCP/IP provides and also that TCP/IP is flexible and accepted industry-wide. The Internet uses TCP/IP, so there are no apparent limitations to bandwidth or to the size of a network based on TCP/IP. You've seen all the things that make TCP/IP what it is and why it's so good.

The only reason you would consider using it is if you are not already. If you are using a Windows- or Unix-based network, the chances are high that you are already using TCP/IP. If you are using earlier versions of NetWare such as NetWare 2 or 3, you are likely running IPX/SPX. NetWare 3 added TCP/IP support at the server and an IPX-to-TCP/IP gateway for Internet connectivity. Starting with NetWare 4, there was better support for TCP/IP. NetWare 5 and higher allows IPX to be replaced by TCP/IP and fully supports TCP/IP.

The same is true if you are considering an upgrade from AppleShare over AppleTalk (pre-6.x). This does not pertain to AppleShare IP because that server suite already implements TCP/IP. But cost here is a factor. TCP/IP offers a wide range of capabilities, servers, services, clients, and so on at little or no cost.

You can spend large amounts of money to have a TCP/IP intranet installed, but you can achieve the same for little money. First, you would consider a server operating system, for example, Linux. It's free, or nearly so if you purchase a distribution. RedHat, Debian, Mandrake, and Caldera are the most popular distributions. Linux remains free, but if you purchase from RedHat, Debian, Mandrake, or Caldera, they also sell you their service and support, special installation software, and other bells and whistles that would not normally come with Linux.

**Note**

RedHat Linux is available free for download from its Web site if you don't want to spend $60, but it will take an extremely long time over an analog modem because RedHat consumes over 160MB of space. It takes less time to buy a distribution.

You could also use your existing operating systems, because a number of servers are available for MacOS, Windows 9x/NT/2000/XP, and others. Some are free. Others can range from $30 to $2,500, depending on what it is and what you are licensing. You could also rewire, but your existing media should be good enough, unless you plan on changing from a bookkeeper to a 3D specialty animation shop with 24-hour service!

# The Intranet Concept

Three things drive TCP/IP as the protocol of choice in today's intranets: cost, speed, and extensibility. TCP/IP can be very inexpensive to implement. It can work alongside your older protocol (AppleTalk, IPX, and so on) until you migrate everything over; it works quickly and efficiently through solid and established protocols; and it can be added to on a whim via the convenience of packet switching.

The Internet is also a consideration because your company or network will now have access to this vast resource. Email is the most widely used application on the Internet today, far more even than browsing. Billions of messages are routed daily. Also, millions of Internet terminals allow people access to services that are posted to the Internet, such as sales data for a mobile workforce, an at-home workforce, or global communications.

There are many ways to connect to the Internet, but the dominant manner is dial-up and in some places DSL (Digital Subscriber Line) and Cable. Anyone with an analog modem can call an Internet service provider's modem and make a connection to the Internet. When connected, it's merely a matter of using the right resource. The dominant Data Link Layer protocol for dial-up is called Point-to-Point Protocol (PPP). An older protocol, Serial Line Interface Protocol (SLIP), allows you to make a serial connection to the Internet from a client. SLIP was later improved to provide compression of TCP/IP headers, and this led to a variation of SLIP called Compressed SLIP (CSLIP). Both SLIP and CSLIP have largely been replaced by PPP today.



# Summary

As you have seen in this chapter, there are many issues to consider in moving a packet from one place to another. This is why TCP and IP are tied so closely together. They both perform critical jobs that make the Internet work as well as it does. You learned that TCP/IP works in layers, and that each layer performs a particular job. If any link in the chain fails, the whole thing collapses. Fortunately, this doesn't happen very often.

Reliability is a very good reason for making TCP/IP available on all operating systems, in one form or another.

TCP/IP, though, is scalable and mobile. If there's a situation where TCP/IP can be leveraged, then you probably won't be surprised to find that connectivity already exists. Remember to always take into account all the active protocols that your internetwork already uses and make sure that a transition to TCP/IP will not cause irreparable harm to user access. Other than that, it's a piece of cake.

# Naming and Addressing

# PART II

## IN THIS PART

CHAPTER 4

# Names and Addresses in an IP Network

*by Mark A. Sportack*

## IN THIS CHAPTER

A critical prerequisite to internetworking is having an efficient address architecture that is adhered to by all users of that internetwork. Address architectures can take many different forms. Network addresses are always numeric, but they can be expressed in base 2 (binary), base 10 (decimal), or even base 16 (hexadecimal) number systems. They can be proprietary, or open for all to see and implement. Address architectures can be highly scalable, or intentionally designed to serve only small communities of users.

This chapter examines the address architecture implemented by the Internet Protocol (IP). As IP has evolved substantially over the past 20 years, so has its address architecture. This chapter describes the evolution of the IP address architecture and explains critical concepts, including classful IP addresses, classless interdomain routing (CIDR) addresses, subnetwork addresses or masks, and variable-length subnet masking (VLSM).

# IP Addressing

The *Internet Engineering Task Force* (IETF)—architects of both the Internet and IP—elected to use machine-friendly numeric addresses to identify IP networks and hosts. Thus, each network in the Internet would have its own unique numeric address—its network address. The administrator(s) of this network would also have to ensure that all the hosts in the network had their own unique host number.

The original version of IP, IP Version 4 (IPv4), uses a 32-bit binary (base 2) address. The address is commonly represented as four 8-bit decimal numbers separated by dots, also commonly referred to as dotted decimal notation. Each 8-bit number is called an *octet*. Binary numbers are machine friendly, but are not all user friendly. Thus, provisions were made to support the use of the more intuitive decimal (base 10) number system for internetwork addressing. The interrelationship between the binary and decimal number systems must be well understood because literally the entire IP address architecture is based on them. The relationship between binary and decimal numbers is examined in the next section, "Binary Versus Decimal Numbers."

The original 32-bit IPv4 address architecture meant that the Internet could support 4,294,967,296 possible IPv4 addresses—a number originally deemed ridiculously excessive. This number is arrived at by raising 2 to the power of 32. These addresses were squandered through a number of wasteful practices, including hoarding (but not using) large blocks of addresses, using classful addresses, assigning inappropriate subnet masks, and other practices. Thus, there is a shortage of addresses. Many of the more significant of these wasteful practices, and their subsequent fixes, will become more evident as you learn more about the IPv4 address architecture.

**Note**

A new version of IP is nearing completion. This version, IPv6, will feature radically different address architectures and use a colon hexadecimal representation. The IPv6 address will be 128 bits long, and use entirely new classifications that are designed to maximize their efficiency of use. Given that it will likely take several years for this new version of IP to be widely used, this book presents all examples using the IPv4 address architectures. Please see Chapter 12, "IP Version 6," for more information on IPv6.

## Binary Versus Decimal Numbers

In a base 2, or binary, number, the value represented by a 1 is determined by its position. This is not unlike the all too familiar base 10 system, in which the right-most digit enumerates ones; the second digit from the right enumerates tens; the third digit from the right enumerates hundreds, ad infinitum. Each digit signifies a ten-fold difference from the digit to the right.

Whereas the base 10 number system provides ten digits to represent different values (0 through 9), the base 2 number system only supports two valid digits: 0 and 1. Their position, too, determines the value that they signify. The right-most position, in decimal terms, is equal to 1. The next position to the left is equal to 2. The next position, 4, then 8, and so on. Each position to the left is 2 times the value of the position to the right.

The decimal value of a binary number is calculated by summing the decimal values of the number's digits that are populated with ones. Mathematically, each octet of an IPv4 address (there are four of them) can have a maximum value of 255, in the base 10 number system. A binary number equal to 255 consists of 8 bits, with all bits set equal to 1. Table 4.1 demonstrates this relationship between binary and decimal numbers.

**TABLE 4.1** Binary (11111111) Versus Decimal (255) Values of an Octet

| *Digit* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* |
|---|---|---|---|---|---|---|---|---|
| Binary | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Decimal Value of Digit | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

As you can see, each of the bits in the binary address is populated with a 1. Thus, calculating the decimal value of this binary number can be done by summing the decimal values of the eight columns: 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255.

Table 4.2 presents another example of the conversion between binary and decimal numbers. In this example, the fifth digit from the right is a zero. This position represents the decimal value 16. Thus, this binary number has a decimal value that is 16 less than 255: 128 + 64 + 32 + 8 + 4 + 2 + 1 = 239.

**TABLE 4.2** Binary (11101111) Versus Decimal (239) Values of an Octet

| *Position* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* |
|---|---|---|---|---|---|---|---|---|
| Binary | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| Decimal | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

This relationship between binary and decimal numbers is the foundation for the entire IP address architecture. Remember that four binary octets are in each IPv4 address. Every other aspect of IP's address architecture, including subnetwork masking, VLSM, and CIDR, is based on these number systems. Thus, you must understand the relationship between these basic numbering systems, and conversion between them, before you can understand the various ways that IP addressing can be implemented.

# IPv4 Address Formats

IP was standardized in September 1981. Its address architecture was as forward-looking as could be expected, given the state of computing at that time. The basic IP address was a 32-bit binary number that was compartmentalized into four 8-bit binary numbers, or octets.

To facilitate human usage, IP's machine-friendly binary addresses were converted into a more familiar number system: base 10. Each of the four octets in the IP address is represented by a decimal number, from 0 to 255, and is separated by dots (.). This is known as a *dotted-decimal* format. Thus the lowest possible value that can be represented within the framework of an IPv4 address is 0.0.0.0, and the highest possible value is 255.255.255.255. Both of these values, however, are reserved and cannot be assigned to individual end systems. The reason for this requires an examination of the way that the IETF implemented this basic address structure in its protocol.

The dotted-decimal IPv4 address was then broken down into classes, to accommodate large, medium, and small networks. The differences between the classes were the number of bits allocated to network versus host addresses. The five classes of IP addresses, identified by a single alphabetic character, are

- Class A
- Class B

- Class C
- Class D
- Class E

Each address consists of two parts: a network address and a host address. The five classes represent different compromises between the number of supportable networks and hosts.

## Class A Addresses

The Class A IPv4 address was designed to support extremely large networks. Because the need for very large-scale networks was perceived to be minimal, an architecture was developed that maximized the possible number of host addresses, but severely limited the number of possible Class A networks that could be defined.

A Class A IP address uses only the first octet to indicate the network address. The remaining three octets enumerate host addresses. The first bit of a Class A address is always a 0. This mathematically limits the possible range of the Class A address to less than or equal to 127, which is the sum of 64 + 32 + 16 + 8 + 4 + 2 + 1. The left-most bit's decimal value of 128 is absent from this equation. Thus, there can only ever be 127 possible Class A IP networks.

The last 24 bits (that is, three dotted-decimal numbers) of a Class A address represent possible host addresses. The range of possible Class A network addresses or host range is from 1.0.0.0 to 126.255.255.255.

Notice that only the first octet bears a network address number. The remaining three are used to create unique host addresses within each network number. As such, they are set to zeros when describing the range of network numbers.



**Note**

Technically, 127.0.0.0 is also a Class A network address, but it is reserved for loopback testing and cannot be assigned to a network. In retrospect, this design is poor because it wastes an entire class A address consisting of a large number of IP addresses.

Each Class A address can support 16,777,214 unique host addresses. This value is calculated by multiplying 2 to the $24^{th}$ power and then subtracting 2. Subtracting 2 is necessary because IP reserved the *all zeros* address for identifying the network and the *all ones* address for broadcasting within that network. The proportion of network to host octets is presented in the following table.

**Note**

Class A is generally used for very large networks such as the original ARPANET. Most of the class A addresses are allocated and it is very difficult, if not impossible, to get a class A network address assignment from the InterNIC.

Class A Address Architecture

| | Network Portion | Host Portion | | |
|---|---|---|---|---|
| Octet | 1 | 2 | 3 | 4 |

## Class B Addresses

The Class B addresses were designed to support the needs of moderate- to large-sized networks. The range of possible Class B host addresses is from 128.1.0.0 to 191.255.255.255.

The mathematical logic underlying this class is fairly simple. A Class B IP address uses two of the four octets to indicate the network address. The other two octets enumerate host addresses. The first two bits of the first octet of a Class B address are 10. The remaining six bits may be populated with either ones or zeros. This mathematically limits the possible range of the Class B address space to less than or equal to 191, which is the sum of 128 + 32 + 16 + 8 + 4 + 2 + 1.

The last 16 bits (2 octets) identify potential host addresses. Each Class B address can support 65,534 unique host addresses. This number is calculated by multiplying 2 to the $16^{th}$ power, and subtracting 2 (values reserved by IP). Mathematically, there can be only 16,382 Class B networks defined.

The proportion of network to host octets is presented in the following table.

Class B Address Architecture

| | Network Portion | | Host Portion | |
|---|---|---|---|---|
| Octet | 1 | 2 | 3 | 4 |

**Note**

Class B networks are popular in medium-sized networks. Many large organizations have been assigned Class B addresses.

## Class C Addresses

The Class C address space was intended to support lots of small networks. This address class can be thought of as the inverse of the Class A address space. Whereas the Class A space uses just one octet for network numbering, and the remaining three for host numbering, the Class C space uses three octets for networking addressing and just one octet for host numbering.

The first three bits of the first octet of a Class C address are 110. The first two bits sum to a decimal value of 192 (128 + 64). This forms the lower mathematical boundary of the Class C address space. The third bit equates to a decimal value of 32. Forcing this bit to a value of 0 establishes the upper mathematical boundary of the address space. Lacking the ability to use the third bit limits the maximum value of this octet to 255 – 32, which equals 223. Thus, the range of possible Class C host addresses is from 192.0.0.0 to 223.255.255.255.

The last octet is used for host addressing. Each Class C address can support a theoretical maximum of 256 unique host addresses (0 through 255), but only 254 are usable because 0 and 255 are not valid host numbers. There can be 2,097,150 different Class C network numbers.

**Note**

In the world of IP addressing, 0 and 255 are reserved host address values. IP addresses that have all of their host address bits set equal to 0 identify the local network. Similarly, IP addresses that have all their host address bits set equal to 255 are used to broadcast to all end systems within that network number, also known as a directive broadcast.



The proportion of network to host octets is presented in the following table.

Class C Address Architecture

| | Network Portion | | | Host Portion |
|---|---|---|---|---|
| Octet | 1 | 2 | 3 | 4 |

## Class D Addresses

The Class D address class was created to enable multicasting in an IP network. The Class D multicasting mechanisms are used for applications that use a "push" technology to send a message to a group of nodes. A multicast address is a unique network address that directs packets with that destination address to predefined groups of IP addresses. Thus, a single station can simultaneously transmit a single stream of datagrams that gets routed to multiple recipients simultaneously. This is much more efficient than creating a separate stream for each recipient. Multicasting has long been deemed a desirable feature in an IP network because it can substantially reduce network traffic. IPv6 also uses multicasting for many aspects of its operation.

The Class D address space, much like the other address spaces, is mathematically constrained. The first four bits of a Class D address must be 1110. Presetting the first three bits of the first octet to ones means that the address space begins at 128 + 64 + 32, which equals 224. Preventing the fourth bit from being used means that the Class D address is limited to a maximum value of 128 + 64 + 32 + 8 + 4 + 2 + 1, or 239. Thus, the Class D address space ranges from 224.0.0.0 to 239.255.255.255.

This range may seem odd, as the upper boundary is specified with all four octets. Ordinarily, this would mean that the octets for both host and network numbers are being used to signify a network number. There is a reason for this! The Class D address space isn't used for internetworking to individual end systems or networks.

Class D addresses are used for delivering multicast datagrams within a private network to groups of IP-addressed end systems. Thus, there isn't a need to allocate octets or bits of the address to separate network and host addresses. Instead, the entire address space can be used to identify groups of IP addresses (Classes A, B, or C). Today, numerous other proposals are being developed that would allow IP multicasting without the complexity of a Class D address space.

The proportion of network to host octets is presented in the following table.

Class D Address Architecture

| | Host Portion | | | |
|---|---|---|---|---|
| Octet | 1 | 2 | 3 | 4 |

## Class E Addresses

A Class E address has been defined but is reserved by the IETF for its own research. Thus, no Class E addresses have been released for use on the Internet. The first four bits of a Class E address are always set to ones, thus the range of valid addresses is from 240.0.0.0 to 255.255.255.255. Given that this class was defined for research purposes, and its use is limited to inside the IETF, examining it any further is not necessary.

## Inefficiencies in the System

Historically, the large gaps between the IP address classes have wasted a considerable amount of potential addresses. Consider, for example, a medium-sized company that requires 300 IP addresses. A single Class C address (254 addresses) is inadequate. Using two Class C addresses provides more than enough addresses, but results in two separate networks within the company. This increases the size of the routing tables across the Internet—one table entry is required for each of the address spaces, even though they belong to the same organization.

Alternatively, stepping up to a Class B address provides all the needed addresses within a single domain, but wastes 65,234 addresses. Too frequently, a Class B was handed out whenever a network supported more than 254 hosts. Thus, the Class B address space approached depletion more rapidly than the other classes.

Perhaps the most wasteful practice was that address spaces were handed out upon request. Any organization that wanted an address space simply requested one, and had to fill out a form describing the number of networks and hosts within the network. No attempts to verify these claims were made. Consequently, many organizations locked up substantial portions of the IPv4 address space as a hedge against some unseen, unspecified, future need.

Fortunately, this is no longer the case. Numerous extensions to IP have been developed that are specifically designed to improve the efficiency with which the 32-bit address space can be used. Three of the more important of these are

- Subnet masks
- Variable Length Subnet Masks (VLSM)
- CIDR

These very different mechanisms were designed to solve different problems. Subnet masks, both fixed and variable length, were developed to accommodate the multiple logical networks that might exist within a physical site that connects to the Internet. Masks are covered in more detail in the section "Variable Length Subnet Masks (VLSM)," later in this chapter.

CIDR was developed to eliminate the inefficiency inherent in the original, rigid, address classes. This enabled routers to more efficiently aggregate many different network addresses into a single routing table entry.

It is important to note that these two mechanisms are not mutually exclusive; they can, and should, be used together.

### Managing the Address Space

The Internet's stability is directly dependent on the uniqueness of publicly used network addresses. Thus, some mechanism was needed to ensure that addresses were, in fact, unique. This responsibility originally rested within an organization known as the InterNIC (Internet Network Information Center). This organization is now defunct and was succeeded by the Internet Assigned Numbers Authority (IANA). IANA, too, has been dismantled, and the new caretaker of the Internet's names and address numbers is the Internet Corporation for the Assignment of Names and Numbers (ICANN). ICANN is currently creating a competitive registry structure that will enable commercial entities to compete with each other in the registration of IP names and numbers.

One important goal is to ensure that duplication of publicly used addresses does not occur. Such duplication would cause instability on the Internet, and compromise its ability to deliver datagrams to networks using the duplicated addresses.

Although it is entirely possible for a network administrator to arbitrarily select unregistered IP addresses, this practice should not be condoned. Computers having such spurious IP addresses can only function properly within the confines of their network. Interconnecting networks with spurious addresses to the Internet incurs the risk of conflicting with an organization that has legitimate claim to that address space. Duplicated addresses will cause routing problems, and potentially hinder the Internet's ability to deliver datagrams to the correct network.

In a firewall environment, the private network can use non-unique addresses because network address translation is used so that the addresses on the private network are substituted with the address of the public interface on the firewall. In this case, the private addresses may be non-unique across the Internet. Certain addresses are reserved for private use. Specifically, one Class A address (10.0.0.0), 16 Class B addresses (172.16.0.0 – 172.31.0.0), and 256 Class C addresses (192.168.0.0 – 192.168.255.0) are designated for private use.

# Special IP Addresses

There are certain special cases for IP addresses. RFC 1700 describes a special notation that can be used to express special IP addresses, as shown here:

```
IP-address ::= { <Network-number>, <Host-number> }
```

The previous notation defines an IP address as consisting of a network number and a host number. The `<Network-number>` corresponds to the *netid* and the `<Host-number>` corresponds to the *hostid* used earlier in this chapter.

If the notation "-1" is used, then the field contains all 1 bits, and a value of "0" means that the field contains all 0 bits.

The following special IP addresses have been defined:

- {`<Network-number>`, 0}
- {`<Network-number>`, -1}
- { -1, -1}
- {0, 0}
- {0, `<Host-number>`}
- {127, <any>}

Addresses of all zeros and all ones in the `<Host-number>` field cannot be assigned as individual IP addresses. This means that if *N* bits are in the host number field, you can have only 2**_N_ IP address assignments.

These special addresses are discussed in the sections that follow. In addition to the previously listed special addresses, special addresses are to be used for broadcasting within subnets. The treatment of subnets is taken up in later chapters and is not discussed in this introductory chapter on IP addresses.

## Addressing "This" Network

A *hostid* value of all 0s are never assigned to an individual TCP/IP host. An IP address with a *hostid* value of 0 indicates the network itself. This special address is described using the following type:

```
{<Network-number>, 0}
```

Consider, the IP address of 137.53.0.0. This is a class B network. Recall that in a Class B network, the <Network-number> is 16 bits (2 bytes or first two decimal numbers in the dotted decimal address), and the <Host-number> is the remaining 16 bits of the IP address. Therefore, the special address 137.53.0.0 refers to the Class B network 137.53.

The two Class B addresses 137.53.0.0 and 141.85.0.0 have a "0.0" for the *hostid* field. These addresses indicate all the nodes in that network. The {<Network-number>, 0} cannot be used as a source or destination IP address.

## Directed Broadcast

The *hostid* value containing all 1s in the bit pattern indicates a *directed broadcast address*. A directed broadcast address can occur in the destination IP address of an IP datagram, but never as a source IP address. This special address is described using the following type:

```
{<Network-number>, -1}
```

A directed broadcast address will be seen by all nodes on that network. Therefore, for the network number 137.53, the broadcast address will be 137.53.255.255. The network number 137.53 is a Class B address and has 16 bits (2 bytes or first two decimal numbers in the dotted decimal address) in the hostid field. If 1s are used for the 16 bits of the *hostid*, they correspond to a decimal value of 255.255.

In the example of the network 137.53, the underlying network hardware supports broadcasts efficiently at the network hardware level. On other networks, such as point-to-point networks, hardware-level broadcast is not supported efficiently. In this case the broadcast has to be done by the IP protocol software by replicating the IP datagram across all possible links.

A directed broadcast is sent to a specific network identified in the <Network-number> field. Routers on the network configured to forward-directed broadcasts will send the IP datagram to the final router that connects the destination network <Network-number>. The destination network router will be obliged to send the IP datagram to all nodes on the network. If the destination network provides a hardware-level broadcast, it is used to broadcast the IP datagram.

Routers can be configured not to forward directed broadcasts in situations where broadcast to other networks is not to be permitted.

## Limited Broadcasts

Another type of broadcast, called the *local broadcast*, is represented by the value of 255.255.255.255. This special address is described using the following type:

```
{-1, -1}
```

The limited broadcast can be used in local area networks, where a broadcast never crosses a router boundary.

In a directed broadcast, the sender must supply the value of the `<Network-number>`. If a broadcast is to be done over the local network, you can use the limited broadcast that does not require knowledge of the `<Network-number>`.

A limited broadcast address can never appear as a source IP address; it can appear only as a destination IP address.

## All Zeros IP Address

An all zeros IP address is a special address of the following type:

```
{0, 0}
```

The `<Network-number>` field is 0, which means "this" network. The `<Host-number>` field is also 0, which means "this" node on the network. This address typically occurs when an IP node is trying to determine its IP address. An example of this is the BOOTP protocol, which can be used by nodes on a network to assign an IP address from a central BOOTP server.

When the IP node sends an initial request to the BOOTP server, the IP node does not know its IP address. It uses a value of 0.0.0.0 in the source IP address field to indicate that it could be "this" node (`<Host-number>` is 0) on "this" network (`<Network-number>` is 0). After the IP node learns its IP address, it does not use the 0.0.0.0 address.

The 0.0.0.0 address is also used in routing tables to indicate the network entry for the default router's (often called default gateway) IP address.

The IP address of 0.0.0.0 can only be used as a source IP address, and never as a destination IP address. Please note that some vintage systems still use an all-zero address as a local broadcast.

## IP Address on This Network

A <Network-number> value of 0 with the <Host-number> value of non-zero means the specific host number in "this" network.

```
{0, <Host-number>}
```

If a node on a network receives a packet whose destination IP address has the <Network-number> set to zero, but the <Host-number> matches the receiver's <Host-number> field, the receiver will accept the packet. The receiver interprets the <Network-number> value of zero to mean "this" network.

## Software Loopback

The number 127 is reserved for the *loopback address.* The loopback address is the special address described by:

```
{127, <any>}
```

Any packet sent by a TCP/IP application to an IP address of 127.*X*.*X*.*X*, where *X* is a number from 0 to 255, results in the packet coming back to the application without reaching the network media. The packet is copied from the transmit to receive buffer on the same computer. This is why the IP address 127.*X*.*X*.*X* is called a loopback address. The software loopback address can be used as a quick check to see that the TCP/IP software is properly configured. For example, you can use this with the "ping" tool to test that the IP Layer software is functioning at some basic level. The "ping" tool sends an ICMP echo packet to a destination IP address. The IP Layer at that address responds to an ICMP echo packet with an ICMP reply packet.

Another useful aspect of the software loopback address is when a TCP/IP client and service are running on the same machine. You can access the service on the same machine by using the loopback address. For example, you can use the Telnet or FTP client to connect to the Telnet or FTP server on the same machine using the following typical commands to gain access:

- `telnet 127.0.0.1`
- `ftp 127.0.0.1`

## Exception to IP Addressing

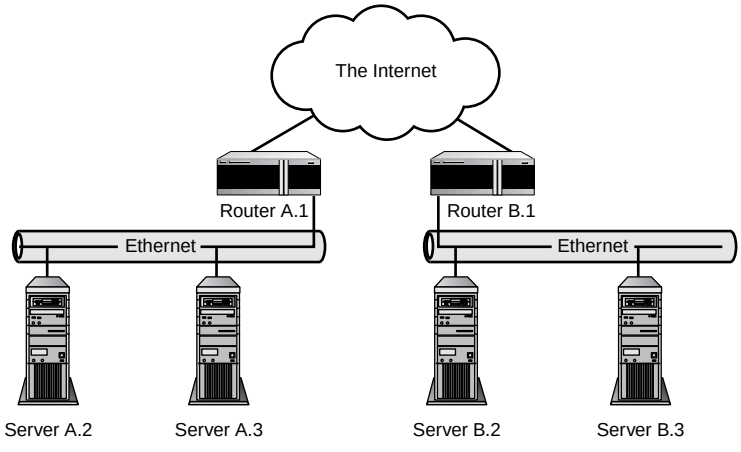
An important exception to the all 1s in the *hostid* field used for broadcast is TCP/IP software derived from 4.2 BSD Unix. Many commercial implementations were derived from BSD 4.2 Unix. If you are using old TCP/IP software, you may run into this exception.

BSD Unix used the convention of all 0s in the *hostid* field to indicate a broadcast address. At the time 4.2 BSD Unix was written, the RFCs were unclear about the convention used for broadcast addresses. This was clarified in later RFCs, which stated that all 1s in the *hostid* field should be used for broadcast addresses. 4.3 BSD Unix was modified to conform to the RFCs. Software derived from 4.2 BSD Unix, unless modified, may still use the all-0s broadcast convention. If hosts that use all-0s broadcast are placed on the same physical network as hosts that use all-1s broadcast, the broadcast mechanism will not work as expected. Symptoms of this will be failure of the TCP/IP applications on that network to work correctly.

# The Emergence of Subnetworks

The Internet originally used a two-level hierarchy, consisting of only network and host addresses. Figure 4.1 demonstrates a rather small and simple two-level network. This hierarchy assumed that each site would have only a single network. Therefore, each site would only need a single connection to the Internet. Initially, these assumptions were safe. However, over time, networked computing matured and expanded. By 1985, it was no longer safe to assume that an organization would only have a single network, nor that it would be satisfied with a single connection to the Internet.

**FIGURE 4.1**
*The Internet originally used a two-level hierarchy.*




As sites began to develop multiple networks, it became obvious to the IETF that some mechanism was needed to differentiate between the multiple logical networks that were emerging within sites of the Internet's second tier. Otherwise, there could be no efficient way to route data to specific end systems in sites with multiple networks. This is illustrated in Figure 4.2.

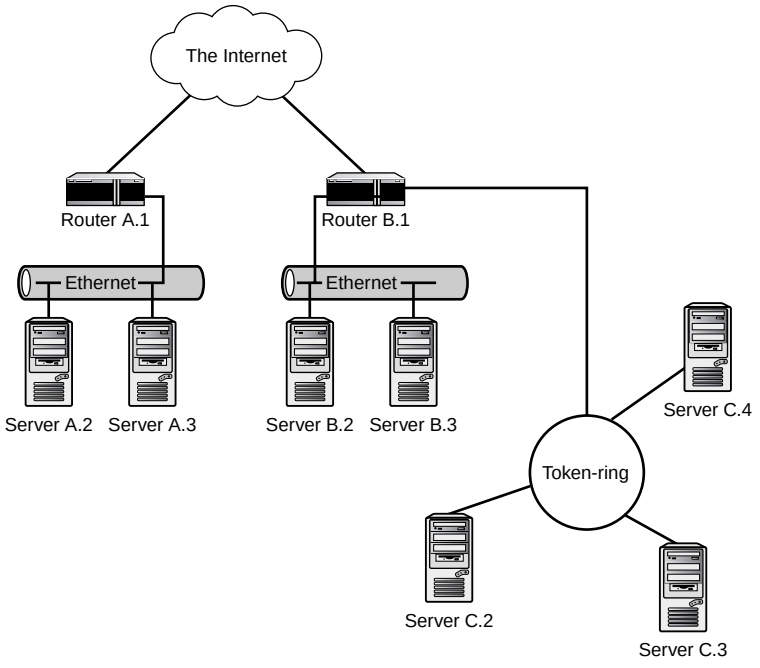



**FIGURE 4.2**
*The emergence of multiple networks per site violated the Internet's two-level hierarchy.*

One answer was to give each logical network, or subnetwork, its own IP address range. This would work, but would be a tremendously inefficient use of the IP address space. It wouldn't take very long for this approach to threaten to completely consume the remaining unassigned IP address ranges. A more immediate impact would be the expansion of routing tables in the Internet's routers. Each network would require its own routing table entry. Clearly, a better approach was needed.

The answer was to organize these logical networks hierarchically, and route between them. Sites with multiple logical networks should, from the Internet's perspective, be treated as a single network. Thus, they would share a common IP address range. However, they would need their own unique range of subnetwork numbers.

## Subnetting

In the mid-1980s, RFCs 917 and 950 were released. These documents proposed a means of solving the ever-growing problem posed by the relatively flat, two-level hierarchy of IP addressing. The solution was termed *subnetting*. The concept of subnetting is based on the need for a third level in the Internet's hierarchy. As internetworking technologies matured, their acceptance and use increased dramatically. As a result, it became normal for moderate and large-sized organizations to have multiple networks. Frequently, these networks were LANs. Each LAN may be treated as a subnet.

In such multiple-network environments, each subnetwork would interconnect to the Internet via a common point: a router. The actual details of the network environment are inconsequential to the Internet. They comprise a private network that is (or should be) capable of delivering its own datagrams. Thus, the Internet need only concern itself with how to reach that network's gateway router to the Internet. Inside the private network, the host portion of the IP address can be subdivided for use in identifying subnetworks.

Subnetting, as specified in RFC 950, enables the network number of any classful IP address (A, B, or C) to be subdivided into smaller network numbers. A subnetted IP address actually consists of three parts:

- Network address
- Subnetwork address
- Host address

The subnetwork and host addresses are carved from the original IP address's host address portion. Thus, your ability to subnet depends directly on the type of IP address being subnetted. The more host bits there are in the IP address, the more subnets and hosts you can create. However, these subnets decrease the number of hosts that can be addressed. You are, in effect, taking bits away from the host address to identify subnetwork numbers. Subnets are identified using a pseudo-IP address, known as a *subnet mask*.

A subnet mask is a 32-bit binary number that can be expressed in dotted-decimal form. The mask is used to tell end systems (including routers and other hosts) in the network how many bits of the IP address are used for network and subnetwork identification. These bits are called the *extended network prefix*. The remaining bits identify the hosts within the subnetwork. The bits of the mask that identify the network number are set to 1s and the host bits are set to 0s.

For example, a mask of 11111111.11111111.11111111.11000000 (255.255.255.192 in dotted-decimal notation) would yield 64 mathematically possible host addresses per subnet. The values of the right-most six bits, the ones set equal to zero, sum to 64 in the base 10 number system. Thus, you can uniquely identify 64 devices within this subnet. Only 62 of these addresses, however, are actually usable. The other two host addresses are reserved. The first host number in a subnet is always reserved for identifying the subnet itself. The last host number is also reserved, but is used for IP broadcasts within the subnet. Thus, you must always subtract 2 from the maximum number of hosts in a subnet to get the maximum number of *usable* host addresses per subnet.

The number of mathematically possible subnets, however, depends on what class of IP address is being subnetted. Each class reserves a different number of the available bits for the network number. Thus, each class offers a different number of bits that can be used for subnetting. Table 4.3 demonstrates the trade-off between the number of subnets and the number of hosts per subnet that can be carved from a Class B IP address. A

Class B address uses 16 bits for network number and 16 for host identification. As you peruse Table 4.3, you'll notice that the fewest number of bits you can allocate to the network prefix is 2, and the most is 14. The reason for this is simple. A network prefix of 1 bit will allow you to define only 2 subnet numbers: 0 and 1. Originally, the rules for subnetting prevented you from using a subnet address that consisted of all 0s, or all 1s. However, many commercial routers allowed the use of all 0s and all 1s for subnet addresses. A later RFC (1812), clarified this use. Today, the use of all 0s and all 1s are permitted for subnet addresses, notwithstanding the fact that many older books still refer to the use of all 0s and all 1s as not permissible.

Similarly, a network prefix of 2 bits only yields four usable subnet addresses. With a 2-bit binary subnet address field, the mathematically possible address combinations are 00, 01, 10, and 11.

**TABLE 4.3** Subnetting a Class B Address Space

| *Number of Bits in Network Addresses Prefix* | *Subnet Mask* | *Number of Usable Subnet* | *Number of Usable Host Addresses, Per Subnet* |
|---|---|---|---|
| 2 | 255.255.192.0 | 4 | 16,382 |
| 3 | 255.255.224.0 | 8 | 8,190 |
| 4 | 255.255.240.0 | 16 | 4,094 |
| 5 | 255.255.248.0 | 32 | 2,046 |
| 6 | 255.255.252.0 | 64 | 1,022 |
| 7 | 255.255.254.0 | 128 | 510 |
| 8 | 255.255.255.0 | 256 | 254 |
| 9 | 255.255.255.128 | 512 | 126 |
| 10 | 255.255.255.192 | 1,024 | 62 |
| 11 | 255.255.255.224 | 2,048 | 30 |
| 12 | 255.255.255.240 | 4,096 | 14 |
| 13 | 255.255.255.248 | 8,192 | 6 |
| 14 | 255.255.255.252 | 16,384 | 2 |

Obviously, the more bits that are allocated to identifying a subnet number, the fewer remain for host identification, and vice versa.

Class C addresses, too, can be subnetted. Because a Class C address allocates 24 bits for network addressing, only 8 bits remain for apportioning between subnet and host addressing. The trade-offs between subnet and host addressing in a Class C network are presented in Table 4.4.

**TABLE 4.4** Subnetting a Class C Address Space

| *Number of Bits in Network Prefix* | *Subnet Mask* | *Number of Usable Subnets Addresses* | *Number of Usable Hosts Addresses, Per Subnet* |
|---|---|---|---|
| 2 | 255.255.255.192 | 4 | 62 |
| 3 | 255.255.255.224 | 8 | 30 |
| 4 | 255.255.255.240 | 16 | 14 |
| 5 | 255.255.255.248 | 32 | 6 |
| 6 | 255.255.255.252 | 64 | 2 |

Although Tables 4.3 and 4.4 demonstrate the trade-offs between the numbers of possible subnets per mask, and hosts per subnet, they fall short of actually demonstrating how subnetting works. The best way to demonstrate subnetting is to actually subnet an IP address, which is done in the next section.

## A Subnetting Example

Subnetting is, perhaps, the most difficult aspect of the IP address architecture to comprehend. This is largely because it really only makes sense when viewed in binary numbers, which isn't very intuitive. For example, you need to subnet the Class C address 193.168.125.0. This is your base address, the one to which the Internet would calculate routes. If you need to carve this into eight subnets, you would need at least three of the 8 host bits to create a unique extended network prefix for each of the eight subnets. These addresses would be 000, 001, 010, 011, 100, 101, 110, and 111. The last octet is split: 3 bits are added to the network number to form the extended network prefix, and the remaining 5 bits are used to identify hosts. Table 4.5 demonstrates how subnetworks are formed.

In Table 4.5, the extended network prefixes (which consist of the IP network address and the subnetwork address) are in bold. The subnet address is in bold italic. The host addresses are in normal typeface, and separated from the extended network prefix with a hyphen. This makes it easier to see how a basic IP network address can be subdivided into subnetworks.

**TABLE 4.5** Forming Subnets

| *Network #* | *Binary Address* | *Decimal Address* |
|---|---|---|
| Base | 11000001.10101000.01111101.00000000 | 193.168.125.0 |
| Subnet 0 | 11000001.10101000.01111101.000-00000 | 193.168.125.0 |
| Subnet 1 | 11000001.10101000.01111101.001-00000 | 193.168.125.32 |
| Subnet 2 | 11000001.10101000.01111101.010-00000 | 193.168.125.64 |
| Subnet 3 | 11000001.10101000.01111101.011-00000 | 193.168.125.96 |
| Subnet 4 | 11000001.10101000.01111101.100-00000 | 193.168.125.128 |
| Subnet 5 | 11000001.10101000.01111101.101-00000 | 193.168.125.160 |
| Subnet 6 | 11000001.10101000.01111101.110-00000 | 193.168.125.192 |
| Subnet 7 | 11000001.10101000.01111101.111-00000 | 193.168.125.224 |

Each subnetwork number is defined with the first three bits of the last octet. The decimal values of these digits are 128, 64, and 32, respectively. The starting IP address (in decimal) for each subnet is presented in the third column. Not surprisingly, these increment in multiples of 32 (the right-most bit of the subnet number).

**Note**

Subnets 0 and 7, although mathematically possible, even if defined on a router, were once not possible. They are now permissible. Their subnet addresses are 000 and 111, respectively.

Hosts in each subnet would be defined by incrementing the remaining five bits in the last octet. There are 32 possible combinations of 0s and 1s. The highest and lowest values are reserved, yielding a usable maximum of 30 hosts per subnet. A device with an IP address of 193.168.125.193 would be the first host defined in Subnet 6. Subsequent hosts would be numbered up to 193.168.125.222, at which point the subnet would be fully populated. No further hosts could be added.

## Variable Length Subnet Masks (VLSM)

Although subnetting proved a valuable addition to the Internet addressing architecture, it did suffer from one fundamental limitation: You were limited to a single subnet mask for an entire network. Thus, after you selected a subnet mask (which dictated the number of hosts you could support per subnet number) you couldn't support subnets of a different size. Any requirement for larger-sized subnets meant you had to change the size of the

subnet mask for the entire network. Needless to say, this could be a complicated and time-consuming affair.

A solution to this problem arose in 1987. The IETF published RFC 1009, which specified how a subnetted network could use more than one subnet mask. Ostensibly, each subnet mask would be a different size. Otherwise, they wouldn't be different masks—their network prefix would be identical. The new subnetting technique was, therefore, called VLSM.

VLSM enables a more efficient use of an organization's IP address space, by enabling the network's administrator(s) to customize the size of a subnet mask to the specific requirements of each subnet. To illustrate this point, assume a base IP address of 172.16.0.0. This is a Class B address, which uses a 16-bit network number. Extending the network prefix by 6 bits results in a 22-bit extended network prefix. Mathematically, there are 64 usable subnet addresses and 1,022 usable host addresses per subnet.

This subnetting scheme would make sense if the organization needed more than 30 subnets populated with more than 500 hosts per subnet. However, if the organization consisted of a few large sub-organizations with more than 500 hosts each, and many smaller sub-organizations with just 40 or 50 host devices each, the majority of possible IP addresses would be wasted. Each organization, regardless of need, would be allocated a subnet with 1,022 host addresses. The smaller organizations would each waste approximately 950 host addresses. Given that a subnetted network could only use a single mask, of fixed and predetermined length, such address waste could not be avoided.

As a purely mathematical exercise, subnetting was an ideal solution to a vexing problem: the rapid depletion of the finite IP address space. Enabling networks to redefine the host field of an IP address into subnetwork and host addresses would greatly reduce the amount of wasted IP addresses. Unfortunately, in a real-world setting, the need for subnets is not homogeneous. It is not realistic to expect an organization, or its networks, to be divided into uniformly-sized subcomponents. It is much more likely that there will be organizations (and subnetworks) of all sizes. Thus, using a fixed-length subnet mask would result in wasted IP host addresses in each subnet defined, as was shown in the previous example.



**Note**

The size of an extended network prefix can be identified using a slash (/) followed by the number of bits used for the network and subnetwork addressing. Thus, 193.168.125.0/27 identifies a specific Class C address, with 27 bits used for the extended network prefix. This is also called Binary count notation.

The solution to this dilemma was to allow an IP address space to be subnetted flexibly, using different-sized subnet masks. Using the previous example, a network administrator could carve up a base IP address into different subnet masks. The few large organizations could continue to use the 22-bit extended network prefix, whereas the smaller organizations could be given a 25- or 26-bit extended network prefix. The 25-bit prefix would enable the creation of 126 host subnets, and the 26-bit prefix would permit subnets with up to 62 hosts each. This solution is VLSM.

# Classless Interdomain Routing (CIDR)

CIDR is a relatively recent addition to the IP address architecture. It was born of the crisis that accompanied the Internet's explosive growth during the early 1990s.

As early as 1992, the IETF became concerned with the Internet's ability to continue to scale upward in response to demand for Internet use. Their specific concerns were

- The exhaustion of the remaining unassigned IPv4 network addresses. The Class B space was in particular danger of depletion.
- The rapid, and substantial, increase in the size of the Internet's routing tables as a result of its growth.

All indications were that the Internet's rapid growth would continue, as more commercial organizations came online. In fact, some members of the IETF even predicted a "Date of Doom." This date, March 1994, was the projected date of the depletion of the Class B address space. Absent any other mechanism for addressing, the Internet's scalability would be seriously compromised. More ominously, the Internet's routing mechanisms might collapse under the weight of their ever-growing routing tables before the Date of Doom.

The Internet was becoming a victim of its own success. The IETF decided that, to avoid the collapse of the Internet, both short- and long-term solutions would be needed. In the long term, the only viable solution was a completely new IP, with greatly expanded address space and address architectures. Ultimately, this solution became known as IPng (Internet Protocol: Next Generation) or, more formally, as IP Version 6 (IPv6).

The more pressing, short-term needs were to slow down the rate of depletion of the remaining unassigned addresses. The answer was to eliminate the inefficient classes of addresses in favor of a more flexible addressing architecture. The result was CIDR. In September 1993, the plans for CIDR were released in RFCs 1517, 1518, 1519, and 1520.

CIDR had several key features that were invaluable in staving off depletion of the IPv4 address space. These features are

- The elimination of classful addressing
- Enhanced route aggregation
- Supernetting

The net effect of these innovations was the obsolescence of class-based addressing. Such addresses may still be found in use, but classless addresses (regardless of the negative connotation!) are much more efficient.

## Classless Addressing

Mathematically, the IPv4 address space still held a substantial number of available addresses. Unfortunately, many of these potential addresses were squandered because they were locked into assigned blocks, or classes, of assigned addresses. Eliminating classes wouldn't necessarily recover the addresses locked into those address spaces that were already assigned, but it would enable the remaining addresses to be used much more efficiently. Ostensibly, this stopgap effort would buy the time needed for IPv6 to be developed and deployed.

## Enhanced Route Aggregation

CIDR enables Internet routers (or any CIDR-compliant router) to more efficiently aggregate routing information. In other words, a single entry in a routing table can represent the address spaces of many networks. This greatly reduces the size of the routing tables needed in any given internetwork, and directly translates into an increased scalability.

CIDR was implemented in the Internet during 1994–1995, and was immediately effective in containing the expansion of the Internet routers' routing tables. It is doubtful that the Internet would have continued to grow had CIDR not been implemented.

## Supernetting

Another benefit of CIDR is the ability to supernet. *Supernetting* is nothing more than using contiguous blocks of Class C address spaces to simulate a single, albeit larger, address space. If you were to obtain enough contiguous Class C addresses, you could redefine the allocation of bits between network and host identification fields, and simulate a Class B address.

Supernetting is designed to alleviate the pressure on the rapidly depleting Class B address space by offering a more flexible alternative. The previous class-based address

architecture suffered from a tremendous disparity between its Class B and Class C networks. Networks that required more than the 254 hosts offered by a Class C had two choices, neither of which was highly desirable. These choices were

- Using multiple Class C addresses (which would have necessitated routing between the network domains)
- Stepping up to a Class B address, with its 65,534 usable host addresses

The simpler solution, frequently, was to use the Class B even though it wasted tens of thousands of IP addresses.

## How CIDR Works

CIDR was a dramatic break from tradition in that it completely abandoned the rigid classes of addresses. The original IPv4 address architecture used an 8-bit network number for Class A addresses, a 16-bit network number for Class B addresses, and a 24-bit number for Class C addresses. CIDR replaced these categories with a more generalized network prefix. This prefix could be of any length, rather than just 8, 16, or 24 bits. This allows CIDR to craft network address spaces according to the size of a network, rather than force-fitting networks into presized network address spaces.

Each CIDR-compliant network address is advertised with a specific bit mask. This mask identifies the length of the network prefix. For example, 192.125.61.8/20 identifies a CIDR address with a 20-bit network address. The IP address can be any mathematically valid address, regardless of whether that address was originally part of the Class A, B, or C range! CIDR-compliant routers look at the number after the / to determine the network number. Thus, the former Class C address 192.125.61.8 previously had a network number of 192.125.61 and a host number of 8. With a Class C address, you could provide addresses for a maximum of 254 hosts within the network. Using CIDR, the architectural limitations of the 8-bit boundaries between address components is eliminated. To better understand how this works, translating the decimal number to binary is necessary.

In binary, this network portion of this address is 11000000.01111101.00111101. The first 20 bits of this example identify the network number. Table 4.6 demonstrates the split of this address between network and host numbers.

**TABLE 4.6** A 20-bit CIDR Network Number

| | *Network Number* | *Host Number* |
|---|---|---|
| Binary Address | 11000000.01111101.0011 | 1101.00001000 |

Notice that the split between the network and host portions of the address falls in the middle of the third octet. The bits that aren't allocated to network number are used to identify hosts. Thus, an IPv4 address with a 20-bit network prefix has 12 bits left for host identification. Mathematically, this translates to 4,094 usable host addresses. Because none of the left-most bits are preset (which previously established the address class), virtually the entire range of addresses can be used in a CIDR network. Thus, a 20-bit network prefix can be assigned a value that was previously reserved for Class A, B, or C networks.

# Public Address Spaces

If your WAN will not be directly interconnected with the Internet, or to any other network, internetwork addresses could be arbitrarily selected. Generally speaking, arbitrarily selecting internetwork addresses is shortsighted and a gross dereliction of duties. That being said, Request for Comment (RFC) 1597 was released in May 1993, and posited a plan to the contrary.

## Requests for Comment (RFC) 1597 and 1918

RFC 1597 was made obsolete by RFC 1918 in February 1996. However, this new RFC only made extremely minor changes to the original specification. The most substantial of these changes was the abandonment of alphabetic classes, such as A, B, and C. Instead, RFC 1918 posited the use of the new CIDR-compliant addressing. As explained in the previous section, CIDR does not use class-based addresses. Instead, the number of bits reserved for the network number are identified with a slash (/) followed by the number of bits. Thus, a Class A address of 10 is identified as 10/8 because only 8 bits are used to designate network numbers. More significantly, a host address can be established on any bit boundary, as opposed to the previous classful addresses, which required the address to be created in multiples of 8 bits.

Three ranges of addresses that could be used for internal networking purposes only were identified and reserved. These ranges include one each of IPv4's Class A, B, and C addresses. They are as follows:

- 10.0.0.0–10.255.255.255
- 172.16.0.0–172.31.255.255
- 192.168.0.0–192.168.255.255

These ranges were reserved by IANA for use in private networks. One stipulation of RFC 1597 was that these addresses couldn't be used when directly accessing the Internet. Companies that used these addresses and subsequently found the need to access

the Internet faced a tough decision. They could renumber all their devices, to comply with IANA, or they could use a proxy server or a *network address translator* (NAT) as an intermediary between their intranet and the Internet. Using such devices would enable the company to keep their spurious addressing without compromising access to, and from, the Internet.

If you choose to implement RFC 1597's reserved addresses for your intranet, you must consider the long-term implications of that decision. Over time, you may need to interconnect with other company networks through an extranet, or to the Internet itself. In either event, you may not be able to guarantee the uniqueness of any given made-up address.

Finally, if you implement one of the address ranges reserved for private networks in RFC 1918, you must still guarantee the uniqueness of each device's address within your private network domain. The addresses won't be unique globally, but they must be unique locally.

## Private Network Addresses

In order to reduce the need for new IP addresses, RFC 1597 on Address Allocation for Private Internets was issued in March 1994. The authors of this RFC suggest a scheme for IP address assignments for private networks. Private networks are those not connected to the network or those that have hosts and services that have limited interaction with the connected Internet. Hosts on the network can be classified into the following categories:

- Hosts that do not require access to hosts in other enterprises or the Internet at large.
- Hosts that need access to a limited set of outside services such as email, FTP, netnews, remote login, and so on, which can be handled by an application-level gateway.
- Hosts that need Network Layer access outside the enterprise. This is provided via IP connectivity and includes router devices or hosts that are visible to the outside world.

Hosts within the first category can use IP addresses that are unique within the private network, but may not be unique across the Internet. Because there is no packet exchange with these hosts outside the private network, the duplicate IP address problem will never be seen.

For hosts in the second category that are insulated from the external Internet by an application-level gateway, having unique IP addresses across the Internet is unnecessary. This is because the application-level gateway hides the IP addresses of these hosts from the external network.

Hosts in the third category require IP addresses that are globally unique across the Internet.

If proper documentation on the network is maintained, identifying hosts in category 1 and 2 is relatively easy. These hosts do not need Network Layer connectivity outside the organization's network.

RFC 1597 gives the following examples of hosts in categories 1 and 2:

- A large airport that has many arrival/departure displays individually addressable via TCP/IP. It is very unlikely that these displays need to be directly accessible from other networks.
- Large organizations like banks and retail chains are switching to TCP/IP for their internal communication. Local workstations like cash registers, money machines, and equipment at clerical positions rarely need to have such connectivity.
- For security reasons, many enterprises use application-layer gateways (for example, firewalls) to connect their internal network to the Internet. The internal network usually does not have direct access to the Internet, and only one or more firewall hosts are visible from the Internet. In this case, the internal network can use non-unique IP numbers.
- If two enterprises communicate over their own private link, usually only a very limited set of hosts are reachable from the other enterprise over this link. Only those hosts need globally unique IP numbers.
- Interfaces of routers on an internal network usually do not need to be directly accessible from outside the enterprise.

Based on the request of the authors of RFC 1597, the ICANN has reserved the following three blocks of the IP address space for private networks that can be assigned for hosts in categories 1 and 2:

- 10.0.0.0 to 10.255.255.255—1 Class A network
- 172.16.0.0 to 172.31.255.255—16 Class B networks
- 192.168.0.0 to 192.168.255.255—256 Class C networks

An enterprise that decides to use IP addresses out of the previously listed address space can do so without any coordination with ICANN or an Internet registry. This address space can thus be used by many enterprises and would reduce the need for assigning new network numbers. Addresses within this private address space will be unique within the enterprise, but not unique across the network.

If an enterprise needs globally unique address space for applications and devices that are connected at the IP Layer, they will have to obtain such addresses from an Internet registry. The ICANN will not assign IP addresses from the private IP address space.

Because private addresses have no global meaning, routing information about private networks should not be propagated outside the enterprise. Routers in networks not using private address space, such as those of Internet Service Providers (ISPs), should have their filters set to reject routing information about private networks. Filtering out private address information should not be reported as a routing protocol error.

In order to use private address space, an enterprise needs to determine which hosts do not need to have Network Layer connectivity. If the configuration of the hosts change and Network Layer connectivity is needed to external networks, globally unique addresses must be assigned. That is, the IP address of these hosts must be changed from the private address space to the public address space. If an organization does not keep proper track of changing requirements for hosts, the organization can accidentally create a situation of duplicate IP address problems. Perhaps for this and other reasons, RFC 1627 was issued in July 1994.

The authors of RFC 1627 titled "Network 10 Considered Harmful (Some Practices Shouldn't be Codified)" argue that the recommendations in RFC 1597 reduce network reliability, security, and are costly in terms of potential problems. For example, if two organizations that are using private address spaces need to connect directly, at least one of them will have to change its IP address assignments. RFC 1627 states that "RFC 1597 gives the illusion of remedying a problem, by creating formal structure to a long-standing informal practice. In fact, the structure distracts us from the need to solve these very real problems and does not even provide substantive aid in the near-term." Additionally, RFC 1627 states that the ICANN has overstepped their mandate in recommending RFC 1597 without the benefit of the usual, public review and approval by the IETF or IAB.

So what should you, the network designer, do in the midst of this controversy about using private addresses? If you plan on using the recommendations of RFC 1597, do so with great care. Ensure that you have proper documentation and an understanding of which services are visible or non-visible to the outside world. Read RFC 1627 and understand the objections it raises.

## Internet Class C Address Allocation

The Class C addresses from 192.0.0 through 223.255.245 have been divided into eight blocks of addresses. Regional authorities are responsible for allocation of these blocks. Table 4.7 shows this address allocation.

**TABLE 4.7** Class C Address Allocation

| *Address range* | *Region* |
|---|---|
| 192.0.0-193.255.255 | Multi-regional. Includes addresses before the region-based address allocation scheme was introduced. |
| 194.0.0-195.255.255 | Europe |
| 196.0.0-197.255.255 | Used when there is a need to assign IP addresses not based on region. |
| 198.0.0-199.255.255 | North America |
| 200.0.0-201.255.255 | Central and South America |
| 202.0.0-203.255.255 | Pacific Rim |
| 204.0.0-205.255.255 | Used when there is a need to assign IP addresses not based on region. |
| 206.0.0-207.255.255 | Used when there is a need to assign IP addresses not based on region. |
| 208.0.0-223.255.255 | Available for assignment. |

The IP address allocation information in Table 4.7 can be useful if you have a protocol trace of packets coming into your network and you want to determine the region of the world from which the packets are originating.

# Configuring IP Addresses

Hosts on a TCP/IP network need to be configured using proper IP addresses. The actual configuration procedure depends on the operating system. The configuration procedure can be classified into the following categories:

- Command-line–based
- Menu interface
- Graphical User Interface–based
- Obtained dynamically when host boots from a central server

Table 4.8 shows examples of operating systems that fit into these categories.

**TABLE 4.8** Configuring IP Address and Other Parameters for Hosts

| *Method* | *Operating System/Protocol* |
|---|---|
| Command line | Unix, VMS, Router devices, MS-DOS |
| Menu interface | Router devices, Unix, NetWare servers, MS-DOS |
| Graphical User Interface | Unix, Microsoft Windows products |
| Dynamically assigned | DHCP and BOOTP protocols. Available on almost all major operating system platforms. |

Many systems offer a command that can be executed to modify the IP address. In these systems, the command line is often placed in the startup script for the operating system. The Menu interface is built using the extended line-drawing character set. You are prompted to enter the IP address and other IP parameters. The Graphical User Interface is for systems that offer a pixel-based graphical view. Examples of these are X-Windows and Microsoft Windows operating systems. The dynamically assigned IP address is used in conjunction with BOOTP or DHCP protocol. A device on starting up requests its IP address and other parameters from a central server that can deliver this information using either the BOOTP or the DHCP protocol.

The IP address information for the host is recorded in a number of places. When the IP address information is entered, it is cached in memory and available for use by the TCP/IP software. Alternatively, this information may also be recorded on a system file in the operating system. IP addresses of other systems can be discovered by consulting a special file, usually called the "hosts" file or by using the DNS protocol. The DNS service is typically used to determine a host's IP address given its symbolic DNS name (see Chapter 1, "Introduction to Open Communications"). In some systems, proprietary protocols can be used to discover an IP address on a network. An example of this is the WINS service used in Microsoft's operating systems.

# IP Version 6 Addresses

As noted earlier in this chapter, the 32-bit address space for IP addresses is running out as the Internet continues to grow. Notwithstanding RFC 1597, every IP network connection on the Internet requires a unique IP address. Some devices have more than one network connection, which results in the rapid consumption of assignable IP addresses.

It is estimated that the 32-bit IP addresses (also called IP version 4 addressing) can provide over 2,100,000 networks and over a total of 3,720 million hosts. However, the IP address space allocation scheme is not very efficient because new network numbers are generally required to connect networks through Network Layer relay devices such as routers.

IP addresses are fast becoming a scarce resource. To overcome this problem, the IETF has studied methods for overcoming the address space problem and making improvements to the protocol so that it can run more efficiently over newer networking technologies. After studying a variety of proposals for solving limitations of the current IP protocol (called IP version 4 (IPv4), after the version number field value of 4 in the IP header), the IETF has decided on the "IP Next Generation" protocol known as IPng or IP version 6 (IPv6). One of the design goals of IPv6 was to support at least one billion networks. To support this design criterion, IPv6 uses 128-bit addresses. This is four times the bit size of IPv4 addresses. IPv6 uses the concept of network numbers and host numbers, but extends this to several levels. The hierarchical addressing in IPv6 can support more efficient routing. The IPv6 address can contain an IPv4 address. This is done by using 32-bit IPv4 addresses in the lower bits of the IPv6 address and adding a fixed prefix of 96 bits. The 96-bit prefix consists of 80 zero bits followed by 16 zero or 16 one bits.

Unregistered IPv4 addresses can be used in IPv6 by adding a unique registered modifier so that the overall IPv6 address is unique.

IPv6 is designed to interoperate with IPv4 systems. This allows a period of coexistence of the two IP systems. The goal is to have current IPv4 systems replaced by IPv6 systems. Routers with both IPv6 and IPv4 support could be used to relay information between networks running the IPv4 protocol and the newer IPv6 protocol.

IPv6 includes support for mobile portable systems. This permits portable computer and other device users to connect anywhere within the network without having to perform manual configuration. Mobile systems will be automatically connected when joining the network at a new attachment point. In addition to automatic configuration for mobile systems, regular workstations are self-configuring. IPv6 supports encryption at the Internet Layer and better support for real-time traffic. Real-time data traffic requires that there be some guarantee of the maximum delay in transmitting datagrams across the network.

Because IPv6 addresses are 128 bits long, using dotted-decimal notation is not a convenient notation for writing IPv6 addresses. If the dotted-decimal notation were used for writing IPv6 addresses, you would have to write a string of 16 decimal numbers separated by dots!

Consider the following bit pattern representation of an IPv6 address:

```
01011000 00000000 00000000 11000000

11100011 11000011 11110001 10101010

01001000 11100011 11011001 00100111

11010100 10010101 10101010 11111110
```

You will notice that 128 bits are in the IPv6 address, and that writing it as a bit pattern is not very convenient. If the dotted-decimal notation were being used, this IPv6 address would be written as the following:

```
88.0.0.192.227.195.241.170.72.227.217.39.212.149.170.254
```

As you can see from the previous dotted-decimal value, it is not a compact notation. The designers chose to use a *colon hexadecimal notation* to write out the bit patterns. The hexadecimal values are written as 16 bits separated by the colon (:) character. The previous IPv6 address is written as the following:

```
5800:00C0:E3C3:F1AA:48E3:D927:D495:AAFE
```

Using the colon hexadecimal notation requires fewer digits and separation characters. The *colon hexadecimal notation* specifies two shorthand techniques that you can use to reduce the number of characters that you have to write.

The first technique is that you can skip leading zeros. Consider the following IPv6 address:

```
48A6:0000:0000:0000:0000:0DA3:003F:0001
```

By skipping the leading zeros, you can write this as the following simplified address:

```
48A6:0:0:0:0:DA3:3F:1
```

Note that 0000 is replaced by 0; 0DA3 is replaced by DA3; 0001 is replaced by 1. In each case, leading zeros are skipped to reduce the number of characters that you have to write.

The second technique uses *zero compression,* in which a single string of repeated zeros is replaced by a double colon (::). In the previous IPv6 address, there is a string of four 0s (0:0:0:0). These can be replaced by "::". The previous IPv6 address can therefore be written as the following:

```
48A6::DA3:3F:1
```

To expand the zero-compressed IPv6 address, align whatever is to the left of the colon with the left 16-bit words of the IPv6 address. Next, align whatever is to the right of the double colon with the right 16-bit words of the IPv6 address. Fill the remaining 16-bit words of the IPv6 address with 0s. As an example, consider the following zero-compressed IPv6 address:

```
5400:FD45::FFFF:3AFE
```

In the expanded form, this IPv6 address becomes the following:

```
5400:FD45:0:0:0:0:FFFF:3AFE
```

You can use the double colon only once in the notation so as to produce an unambiguous interpretation. The following IPv6 notation is illegal, because the number of zeros that were compressed is ambiguous.

```
5400::45A1::23A6
```

The third technique is to use the double colon either as a prefix or suffix. Consider the following IPv6 representation of the IPv4 address 170.1.1.1:

```
0:0:0:0:0:0:AA01:101
```

This can be represented as the following:

```
::AA01:101
```

# Summary

A solid understanding of IP's address architecture is a prerequisite to appreciating the fundamentals of internetworking with IP. The basics presented in this chapter should help you better appreciate the mechanics of internetworking with the IP. Many of the architectural devices presented in this chapter, including CIDR, subnet masking, and VLSM, are so widely used that not understanding them will compromise your ability to support and design internetworks. This chapter also gave you an understanding of private addresses and when to use them. The structure of IPv6 addresses was also presented.

In the subsequent chapters of this Part, other aspects of naming and addressing in IP networks are examined. These aspects include how IP addresses are correlated to local area network hardware addresses, as well as how user-friendly names are resolved into IP addresses.

CHAPTER 5

# ARP and RARP

*by Tim Parker*

## In This Chapter

IP addresses are the common identifiers for machines under TCP/IP, although the IP address alone is not enough to get a datagram to its target. Instead, the network system itself is involved, and the behavior of the network is usually specific to the network operating system and hardware type. In order to gain a good understanding of the manner in which data is routed from a source to a destination machine, you need to understand how a network interworks with its constituent machines. This chapter starts with a look at a typical network system, in particular Ethernet, and presents a look at how TCP/IP provides for a conversion of an IP address to a data link–specific address that the network can find. The conversion of IP address to data link address is performed by the Address Resolution Protocol (ARP). A complementary protocol called Reverse Address Resolution Protocol (RARP) converts data link addresses to IP addresses.

# Using Addresses

The purpose of an IP address is to help TCP/IP deliver a datagram to the proper destination. The three terms commonly used that relate to addressing are name, address, and route.

A *name* is a specific identification of a machine, a user, or an application. It is usually unique and provides an absolute target to which a datagram is to be delivered. An *address* typically identifies where the target is located, usually as its physical or logical location in a network. A *route* tells the system how to get a datagram to the correct address. Be careful when using the term address because it is often generically used with communications protocols to refer to many different things. It can mean the destination, a port of a machine, a memory location, an application, and more. In general, each layer of the OSI model has its own address. For example, the Data Link Layer address for Ethernet networks is the Ethernet hardware address that is unique for each network card, and the Network Layer address is the IP address for a TCP/IP node.

The recipient's login name is usually the key to the whole delivery process. From the username and machine name, a network software package called the *name server* resolves the address and the route, hiding that aspect of TCP/IP routing and delivery from the application. Besides making the addressing and routing transparent to the application, using a name server has another primary advantage: It gives the system or network administrator a lot of freedom to change the network as required, without having to update each user's machine individually. As long as an application can access a name server somewhere on the network, applications and users can ignore routing changes.

# Subnetwork Addressing

When you send a piece of data to another machine, you usually do it with the IP address. Whereas TCP/IP is designed to work with IP addresses, the actual network software and hardware is not. Instead, the network hardware uses a physical address encoded into the network hardware that identifies each machine. Making the correspondence between the IP address and a physical address is not normally part of the TCP/IP protocol's responsibility, so a number of special protocols have been developed for this task. These protocols are discussed in the next section, but first this section looks at the structure of the network's physical address.

On a single local area or wide area network, several pieces of information are necessary to ensure the correct delivery of datagrams. The primary pieces are the physical address and the data link address of the destination machine. These are important enough to warrant a closer look at each.

## Physical Addresses

Each device on a network has a unique *physical address*, sometimes called the *hardware address* or *Data Link Layer address*. For networking hardware, the addresses are usually encoded into the network interface card. The physical address is sometimes user-settable through switches or software. More often, they are not modifiable by users at all because a unique number is encoded into the card's Programmable Read-only Memory (PROM); manufacturers often work together to ensure there is no possibility of duplication of physical addresses. On any given network, there can be only one occurrence of each address; otherwise, the name server will have no way of unambiguously identifying the target machine. The length of the physical address varies depending on the networking system. For example, Ethernet and several other network schemes use 48 bits in each address. For communications to occur, two physical addresses are required: one each for the sending and receiving devices.

The IEEE is now handling the task of assigning universal physical addresses for subnetworks (a task previously performed by Xerox, which originally developed Ethernet). For each subnetwork, the IEEE assigns an *organization unique identifier* (OUI) that is 24-bits long, enabling the organization to assign the other 24 bits however it wants. Actually, two of the 24 bits assigned as an OUI are control bits, so only 22 bits identify the subnetwork. The format of the organization unique identifier is shown in Figure 5.1. The combination of 24 bits from the OUI and 24 locally assigned bits is called a *media access control* (MAC) address. Both the OUI and the local address are programmed into the network card. When a packet of data is assembled for transfer across an internetwork, there will be two sets of MACs—one from the sending machine and one for the receiving machine.

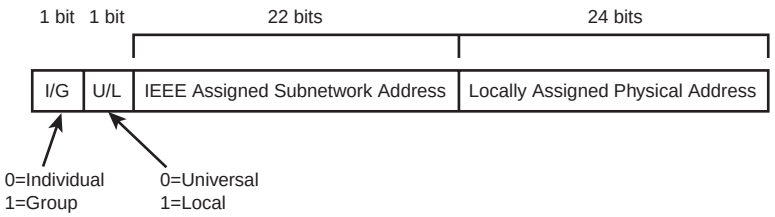



**FIGURE 5.1**
*The structure of the organization unique identifier.*

The least significant bit of the address (the lowest bit number or the bit to the left in the structure) is called the *individual* or *group* address bit. If the bit is set to 0, the rest of the address refers to an individual address; a setting of 1 means that the rest of the address field identifies a group address that needs further resolution. If the entire OUI is set to 1, all stations on the network are assumed to be the destination. This is a special convention supported by the OUI.

The second bit in the OUI structure is called the *local* or *universal* bit. If the second bit is set to 0, it has been set by the universal administration body. This is the setting for IEEE-assigned OUIs. If the second bit has the value of 1, the OUI has been locally assigned and would cause addressing problems if decoded as an IEEE-assigned address. Usually, a structure that has the second bit set to 1 is kept within a local or wide area network and not passed to other networks that may follow the IEEE addressing format.

**Note**

Older protocols such as DECNet Phase IV used locally assigned addresses where the locally assigned physical address was set to the DECNet network address.

The remaining 22 bits in the OUI structure make up the physical address of the subnetwork, as assigned by the IEEE. The second set of 24 bits identifies local network addresses and is administered locally. If an organization were to run out of physical addresses (about 16 million addresses are possible from 24 bits), the IEEE could assign a second subnetwork address.

## Logical Link Control Address

The IEEE Ethernet standards use another address called the Link Layer address (usually abbreviated as LSAP for *link service access point*). The Data Link Layer in IEEE networks is divided into an upper layer called the Logical Link Control (LLC) and a lower layer that contains the hardware dependencies called the MAC (Media Access Control) layer. The LLC layer protocol addresses are called Service Access Points (SAPs) or Logical Service Access Points (LSAPs).

The LSAP identifies the type of link protocol used in the Data Link Layer. As with the physical addresses, a datagram will carry both sending and receiving LSAPs.

## Network Frames

The layout of information in each transmitted packet of data differs depending on the protocol used by the network. However, it is instructive to examine one to see how the addresses previously mentioned, as well as other related information, are added to the datagram before it is sent out over the network. We can use Ethernet as an example because of its wide use with TCP/IP. It is quite similar to other systems as well, although the exact structures of the headers may differ. Remember that this is the way the network protocols package the TCP/IP-constructed headers, and has little to do specifically with TCP/IP. A typical Ethernet frame (the term for a network-ready datagram) is shown in Figure 5.2.

**FIGURE 5.2**
*The structure of an Ethernet frame.*

| Preamble | Recipient Address | Sender Address | Type | Data | CRC |
|---|---|---|---|---|---|
| 64 Bits | 48 Bits | 48 Bits | 16 Bits | Variable Length | 32 Bits |

The preamble is 64 bits used primarily to synchronize the communication process and account for any random noise in the first few bits that are sent. The preamble is ignored as far as addressing and routing are concerned. At the end of the preamble's field is a sequence of bits called the *start frame delimiter* (SFD), which indicates that the frame follows immediately.

The recipient and sender addresses in the Ethernet frame structure use the IEEE 48-bit format, followed by a 16-bit type indicator that is used to identify the type of protocol used. The actual data (which is the assembled TCP/IP datagram) follows the type indicator. The Data field is between 64 and 1,500 bytes in length with standard Ethernet. If the data is less than 64 bytes in length, it is padded with zeroes until it is 46 bytes long. Padding to a minimum data length of 64 bytes is done to ensure that Ethernet collisions can be easily detected.

At the end of the Ethernet frame is the *cyclic redundancy check* (CRC) checksum count, used to ensure that the frame's contents have not been modified during the transmission process. Each machine along the transmission route calculates a CRC value for the frame and compares it to the value at the end of the frame. If the two match, the frame data is considered valid and can be sent farther along the network or into the subnetwork; if they differ, a modification to the frame must have happened, the frame is discarded, and a retransmit request may be sent.

In some protocols related to Ethernet, such as the IEEE 802.3, the overall layout of the frame is the same but slight variations in the contents are used. With 802.3, the 16 bits used by Ethernet to identify the protocol type are replaced with a 16-bit value for the length of the data block. Also, a new field appends the data area itself.

# IP Addresses

As mentioned before, TCP/IP uses a 32-bit address to identify any machine on a network and the network to which it is attached. IP addresses identify a machine's connection to the network, not the machine itself; this is an important distinction. Whenever a machine's location on the network is moved, the IP address must sometimes be changed, too, depending on the way in which the network is set up. The IP address is the set of numbers many people see on their workstations or terminals, such as 128.40.8.72, which uniquely identifies the device. IP addresses are four sets of 8 bits, for the total 32 bits. IP addresses are assigned only by the *Network Information Center* (NIC), although if a network is not connected to the Internet that network can determine its own numbering. The decimal notation used for IP addresses is properly called *dotted-quad notation* or just the *dotted-decimal notation*.

Four formats are used for the IP address, depending on the size of the network. The four formats, Class A through Class D, are shown in Figure 5.3. The class is identified by the first few bit sequences, shown in the figure as one bit for Class A and up to four bits for Class D. The class can be determined from the first three (high-order) bits. In fact, in most cases, the first two bits are enough, because there are few Class D networks.

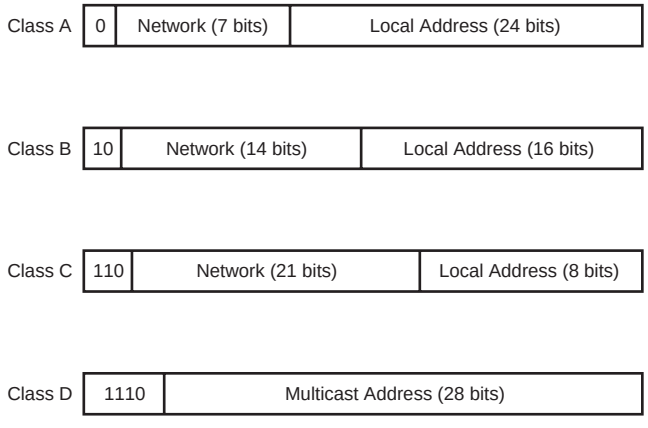



**FIGURE 5.3**
*The four IP address class structures.*

Class A addresses are for large networks that have many machines. The 24 bits for the local address (also frequently called the host address) are needed in these cases. The

network address is kept to seven bits, which limits the number of networks that can be identified.

Class B addresses are for intermediate-sized networks, with 16-bit local or host addresses and 14-bit network addresses.

Class C networks have only eight bits for the local or host address, limiting the number of devices to 256. There are 21 bits for the network address.

Finally, Class D networks are used for multicasting purposes, when a general broadcast to more than one device is required.

From an IP address, a network gateway can determine whether the data is to be sent out to the Internet (or other internetwork) or remain in the local area network. If the network address portion of the destination address is the same as the address of the network to which the sender is connected (directly connected network), the destination is on the directly connected network and the internetwork is avoided. All other network addresses are routed to a router (historically called the gateway on the Internet) so that they can leave the local network (*indirect host*).

It is possible for a machine (especially a router) to have more than one IP address if it is connected to more than one network. These machines are called *multihomed* because they have a unique address for each network to which they are connected. Two networks can have the same network address if they are connected by a gateway, which is an addressing problem because the gateway must be able to differentiate which network the physical address is on. This problem is handled by a special protocol—the *Address Resolution Protocol* (ARP)—that deals only with address resolution.

# Overview of the Address Resolution Protocol

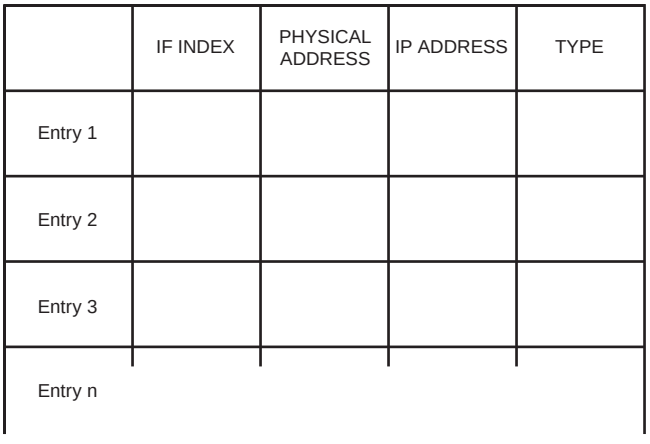
In order to send datagrams from one machine to another on a local or wide area network, the sender needs to know the destination machine's physical address, if it is not known. There needs to be some method to resolve the IP addresses (provided by applications) into the physical addresses of the hardware connecting each machine to the network.

The brute-force method of providing an IP address to physical address resolution is to build a table of conversions on each machine. Then, when an application sends data to another machine, the software can examine the conversion table for the physical address. This method has a variety of problems associated with it, which is why almost no one does it. The primary disadvantage is the need to constantly update the tables of addresses on each machine whenever there is a change.

The Address Resolution Protocol was developed to help solve this problem. ARP's task is to convert IP addresses to physical addresses (network and local) and in doing so, eliminate the need for applications to know anything about physical addresses. Put in its simplest terms, ARP is a conversion table of IP addresses and their corresponding physical addresses. This is called an *ARP table*. The ARP table is usually built dynamically, although static entries can be manually added for troubleshooting and fixing ARP corruption table problems.

The layout of an ARP table is shown in Figure 5.4. ARP also maintains a cache of entries in memory, called an *ARP cache*. Usually the ARP cache is searched for a match, then the ARP table is checked if one is not found in the cache. Dynamically generated ARP cache entries are timed-out after the ARP cache timeout has been exceeded, whereas static entries are not meant to be timed out. Static entries being timed-out indicates a corruption problem in the ARP cache tables.

**FIGURE 5.4**
*The ARP table layout, whereby each row in the table represents one device in the cache.*




## The ARP Cache

Each row in the ARP cache corresponds to one device, with the following four pieces of information stored for each device:

- IF Index—The physical port (interface)
- Physical Address—The physical address of the device
- IP Address—The IP address corresponding to the physical address
- Type—The type of entry to which this line corresponds

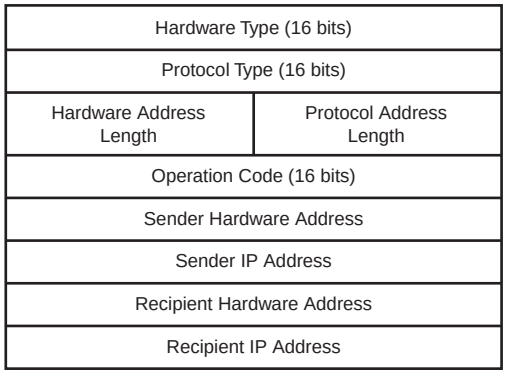
The type has one of four possible values. A value of 2 means the entry is invalid; a value of 3 means the mapping is dynamic (the entry may change); a value of 4 means static (the entry doesn't change); and a value of 1 means none of the above.

When ARP is handed an IP address to discover the corresponding physical address, it searches the ARP cache and ARP table for a match. If it finds one, it returns the physical address to whoever supplied the IP address. If ARP doesn't find a match for an IP address, it sends a message out on the network. The message, called an *ARP request*, is a broadcast that is received by all devices on the local network.

The ARP request contains the IP address of the intended recipient device. If a device recognizes the IP address as belonging to it, the device sends a reply message containing its physical address back to the machine that generated the ARP request, which places the information into its ARP table and cache for future use. In this manner, ARP can determine the physical address for any machine based on its IP address.

The layout of an ARP request or ARP reply is shown in Figure 5.5. When an ARP request is sent, all fields in the layout are used except the Recipient Hardware Address (which the request is trying to identify). In an ARP reply, all the fields are used.

**FIGURE 5.5**
*The ARP request and reply layout.*




The fields in the ARP request and reply can have several values. The remainder of this section presents a look at each of the fields in a little more detail to show their uses.

## Hardware Type

The hardware type identifies the type of hardware interface. Legal values are

| *Type* | *Description* |
|---|---|
| 1 | Ethernet |
| 2 | Experimental Ethernet |
| 3 | X.25 |
| 4 | Proteon ProNET (Token Ring) |
| 5 | Chaos |
| 6 | IEEE 802.X |
| 7 | ARCnet |
| 8 | Hyperchannel |
| 9 | Lanstar |
| 10 | Autonet Short Address |
| 11 | LocalTalk |
| 12 | LocalNet |
| 13 | Ultra link |
| 14 | SMDS |
| 15 | Frame Relay |
| 16 | Asynchronous Transmission Mode (ATM) |
| 17 | HDLC |
| 18 | Fibre Channel |
| 19 | Asynchronous Transmission Mode (ATM) |
| 20 | Serial Line |
| 21 | Asynchronous Transmission Mode (ATM) |

## Protocol Type

The protocol type identifies the type of protocol the sending device is using. With TCP/IP, these protocols are usually an EtherType, for which the legal values are as follows:

| *Decimal* | *Description* |
|---|---|
| 512 | XEROX PUP |
| 513 | PUP Address Translation |
| 1536 | XEROX NS IDP |
| 2048 | Internet Protocol (IP) |
| 2049 | X.75 |
| 2050 | NBS |

| *Decimal* | *Description* |
|---|---|
| 2051 | ECMA |
| 2052 | Chaosnet |
| 2053 | X.25 Level 3 |
| 2054 | Address Resolution Protocol (ARP) |
| 2055 | XNS |
| 4096 | Berkeley Trailer |
| 21000 | BBN Simnet |
| 24577 | DEC MOP Dump/Load |
| 24578 | DEC MOP Remote Console |
| 24579 | DEC DECnet Phase IV |
| 24580 | DEC LAT |
| 24582 | DEC |
| 24583 | DEC |
| 32773 | HP Probe |
| 32784 | Excelan |
| 32821 | Reverse ARP |
| 32823 | AppleTalk |
| 32824 | DEC LANBridge |

If the protocol is not EtherType, other values are allowed.

## Hardware Address Length

Hardware address length is the length of each hardware address in the datagram, given in bytes. For Ethernet networks, this value is set to 4 bytes.

## Protocol Address Length

Protocol address length is the length of the protocol address in the datagram, given in bytes. For the IP protocol, the protocol address length is set to 4 bytes.

## Operation Code (Opcode)

The Opcode indicates whether the datagram is an ARP request or an ARP reply. If it is an ARP request, the value is set to 1. If the datagram is an ARP reply, the value is set to 2.

### Sender Hardware Address

The sender hardware address is the hardware address of the sending device.

### Sender IP Address

The sender IP address is the IP address of the sending device.

### Recipient Hardware Address

The recipient hardware address is the hardware address of the recipient device.

### Recipient IP Address

The recipient IP address is the IP address of the recipient device.

## Details of ARP Operation

As an IP packet is sent through the network layers of nodes in a TCP/IP internet, the routing component of the Network Layer determines the IP address of the next router.

The routing component of the Network Layer determines whether the IP datagram is to be sent to a host on the local network or a host on a different network segment.

If the sender host determines that the destination is on the local network, it needs to find the destination's hardware address. If the destination is on a remote network (different network than the local network) then the sender needs to find the hardware address of the router port to which the IP datagram is to be forwarded.

For broadcast networks ARP is used to find the hardware address of the target node which is either the node on the local network or the router port on the local network.

ARP is implemented as part of the IP module in networks that need address resolution such as broadcast LANs (Ethernet, Token Ring and so on). The ARP protocol is directly encapsulated by the Data Link Layer protocol; it is not encapsulated by the IP protocol. This means that the ARP protocol cannot be routed; that is, it cannot cross a router boundary.

Before sending the ARP request, the ARP module tries to find the target address in its local cache, also called the ARP cache table. The ARP cache table keeps pairs of entries of IP addresses and the corresponding hardware address.

If the target IP address is found in the ARP cache table, it looks up the corresponding hardware address and returns it to the ARP module. The ARP module returns the hardware address to the network driver that made the request for discovering the target node's

hardware address. The network driver then transmits a Data Link Layer frame containing the IP datagram with the target node's hardware address placed in the destination hardware address field. In the scenario just discussed, an ARP request is never generated, because the target node's hardware address was discovered in the local cache.

What happens if the ARP cache does not contain the target node's hardware address? In this case, the ARP module generates a Data Link Layer frame containing the ARP request to discover the target node's hardware address. The ARP request is broadcast at the Data Link Layer to all nodes on the local network segment. As mentioned earlier, the ARP request/replies are confined to the local network segment and do not cross router boundaries.

When an ARP request is received, the Data Link Layer at the target node gives the packet to the Address Resolution module. The ARP request is broadcast at the Data Link level, so all nodes on the local network will receive it. However, only the node whose IP address corresponds to the Target IP address will respond with an ARP reply.

The steps performed by the ARP module at the receiving node are outlined here:

1. First, the node checks to see whether it recognizes the hardware type in the Hardware Type field of the ARP request packet. If it does recognize the hardware type, it optionally checks the hardware length Hlen in the ARP request packet.
2. The node then answers this question, "Do I speak the protocol in the Protocol Type field in ARP request packet?" If the answer is yes, the node optionally checks the protocol length Plen in the ARP request packet.
3. The Set Merge_flag is set to *false.*
4. If the pair <protocol type, sender protocol address> is already in the node's translation table, it updates the sender hardware address field of the entry with the new information in the packet and sets the Merge_flag to true.
5. The node then answers the question, "Am I the target protocol address?" This means "Is my IP address the same as that in the Target IP address field?" If the answer is yes, then the following occurs: If the merge flag is false, add the triplet <protocol type, sender protocol address, sender hardware address> to the translation table.
6. Finally, the node answers the question, "Is the Operation field value a request?" If the answer is yes, swap the hardware and protocol fields, putting the local hardware and protocol addresses in the sender fields. See the Operation field to ARP reply. Send the packet to the (new) target hardware address on the same hardware on which the request was received.

In the algorithm just described, the <protocol type, sender protocol address, sender hardware address> triplet is merged into the ARP cache table before the Operation field is examined in step 4. This step only occurs if a node has an entry for the sender's IP address in their ARP cache table.

Note that only the ARP cache table for nodes that have an entry for the sender IP address are modified.

Updating or adding ARP cache entries is done because the upper layers of the target node are likely to respond to the ARP request sender; when the target generates a Data Link Layer frame, it will consult its ARP cache table and discover that it already knows the hardware address of the sender. Thus an ARP request to discover the sender's hardware address is not required.

Also note also that if the target node already has an entry in its ARP cache table for the <protocol type, sender protocol address> pair, then the new hardware address replaces the old address. This could occur in any of the following situations:

- New network hardware was added in the ARP request sender to replace the existing one. In this case the hardware address of the node is different.
- The IP addresses were reassigned. The ARP request sender is another node that was assigned the IP address that is in the ARP cache table of the target. The hardware address of the ARP sender is different because it is a different node.
- There is a duplicate address problem. Another node claims to have the same IP address as the one in the target node's ARP cache table.

Although the first two situations are normal, the last situation of duplicate IP addresses is an error condition and is examined in detail in the sections on duplicate IP addresses in this chapter.

## The ARP Protocol Design

In theory, the length fields HLen and PLen in the ARP packet format are redundant, because the length of the hardware address and the protocol address can be determined by the value of the Hardware Type and Protocol Type fields. For instance, the Hardware Type field having a value of 1 indicates an Ethernet frame, and this implies an HLen value of 6 octets because Ethernet addresses are 6 octets long. Similarly, if the Protocol Type field indicates an IP protocol, the PLen value is 4 octets because IP addresses are 4 octets long.

The redundant HLen and PLen fields are included for optional consistency checking, and for the use of network monitoring tools.

The Operation field is used to determine whether this is a request or a reply to a previous request. Using a two-octet field is excessive considering the number of the operation codes that have been assigned thus far. In retrospect, a one-octet field would have been sufficient.

The Sender HA and Sender IP address fields are needed because these fields are potentially recorded in the ARP cache tables of the receiving nodes. In the ARP request the Target HA has no meaning because its value is not known. Note that the answer is actually returned in the Sender HA field in the ARP reply. The Target HA is included for completeness and network monitoring.

The Target IP field is necessary in the ARP request so that a machine can determine whether or not it is the target node and therefore needs to send a reply. The Target IP field is not necessarily needed in the ARP reply if one assumes a reply is only provoked by a request. It could be used for network monitoring, and to simplify the ARP request/reply processing algorithm.

## Network Monitoring with ARP

ARP can be used by a monitoring device to gain knowledge about the higher level protocol activity by just examining the ARP frame headers. For example, the network-monitoring device can determine which protocols are in use by examining the Protocol Type field value. A monitoring device could be designed along the lines suggested in this section.

When the monitoring device receives an ARP packet, it can enter the protocol type, sender protocol address, and sender hardware address in a table. Additionally, it can determine the length of the hardware and protocol address from the HLen and PLen fields of the ARP packet. Note that ARP requests are broadcast at the Data Link level, and a monitor receives all ARP requests.

If the Operation field indicates an ARP reply packet, and the target protocol address matches the protocol address of the monitor, the monitor sends an ARP reply like any other target node. The monitor will not receive any other ARP replies because an ARP reply is sent directly to the requesting host.

## Timeouts in the ARP Cache Table

ARP cache tables are often implemented with a timeout mechanism. Entries in the ARP cache table may have timestamp values associated with them. The following is a discussion of some of the timeout issues associated with ARP tables.

If a node is moved it is usually shut down before moving. Shutting down a node clears its ARP cache table so old entries will not cause confusion. Other nodes on the network will generally not be aware that a particular node has moved or been shut down.

If the node has been moved to a location on the same network segment, the ARP cache information about the moved node will still be valid because the IP address and hardware address of the moved node will not have changed.

If the node has been shut down, other nodes will not be able to reach this node. They will use their ARP cache information to access the node but be unable to make the connection. An implementation could use the failure to initiate a connection to a node to delete the information in the ARP cache table about the node. It could also try accessing the node by sending a few more ARP requests before giving up.

If the node has been moved to a location on a different network segment, it is behind a router boundary. Because IP addresses contain information about the network segment they are attached to, the IP address of the moved node must change even though its hardware address remains the same. Recall that ARP transmissions do not go beyond a router boundary. For all practical purposes the new node can be assumed to be shut down or unavailable.

An ARP implementation typically updates the timestamps when the address resolution entry is used for transmitting packets to a node. The timestamps are also updated when ARP requests are received from a node whose entry exists in the ARP cache table. If no packets are received from a node for a suitable length of time, called the ARP timeout value, the address resolution entry is discarded.

An ARP implementation may also have an independent process (also called a *daemon* process). This independent process checks the ARP cache table periodically and times-out old entries. In many TCP/IP implementations, a default timeout of 15 minutes is used, and in some implementations this value can be configured by a system administrator.

As a refinement, it is also possible for the ARP daemon to first send an ARP request to the node directly. If an ARP reply is not seen after a few retransmissions, the ARP entry is discarded. In this situation, the ARP request is sent directly and *not broadcast* because the hardware address of the node is known from the ARP cache table. This refinement is common in Linux implementations.

Many TCP/IP implementations allow you to make manual entries in the ARP cache table. Normally, there is no need to make manual entries in the ARP table, as the dynamic ARP operation determines the IP address and hardware associations. ARP entries made manually are not timed out and can be used to fix problems with incorrect

entries in the ARP table because of duplicate IP address problems or malfunctioning software. The tool used to view or make changes in the ARP table depends on the operating system: It can be a command-line tool or a graphical tool. A tool called ARP is available on both Windows NT/2000 and UNIX systems. The following shows the syntax for the ARP command for Windows 2000:

```
ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr]

-a      Displays current ARP entries by interrogating the current
       protocol data. If inet_addr is specified, the IP and Physical
       addresses for only the specified computer are displayed. If
       more than one network interface uses ARP, entries for each ARP
       table are displayed.
-g      Same as -a.
inet_addr   Specifies an internet address.
-N if_addr  Displays the ARP entries for the network interface specified
       by if_addr.
-d      Deletes the host specified by inet_addr. inet_addr may be
       wildcarded with * to delete all hosts.
-s      Adds the host and associates the Internet address inet_addr
       with the Physical address eth_addr. The Physical address is
       given as 6 hexadecimal bytes separated by hyphens. The entry
       is permanent.
eth_addr   Specifies a physical address.
if_addr    If present, this specifies the Internet address of the
       interface whose address translation table should be modified.
       If not present, the first applicable interface will be used.
Example:
> arp -s 157.55.85.212  00-aa-00-62-c6-09 .... Adds a static entry.
> arp -a                  .... Displays the arp table.
```

## ARP in Bridged Networks

Networks can be interconnected by using bridge devices that operate at OSI layer 2 and routers that operate at OSI layer 3. Bridges extend the range of LANs. ARP operation on LANs requires special discussion, which is the focus of this section.

When a TCP/IP connection fails, the TCP protocol, which is responsible for reliable delivery, attempts to recover from the failure by retransmitting packets. If the failure was caused by a physical link failure and other paths to the destination are available, the IP datagrams get routed around the failed link. If TCP is unable to recover from the failure, recovery continues at the lower layers. At the lower layers, the ARP module attempts to recover from the connection by broadcasting ARP requests. If the target node is on the local network segment, this is viable technique for recovering from failure. However, a complication arises when the several nodes start sending ARP request broadcasts. This could happen when a central host is no longer available because of host or link failure.

If 100 stations are connected to a central host on the network, and the connection to the host goes down, all 100 stations will start transmitting ARP broadcasts simultaneously. On some systems, ARP requests are sent every second. One hundred stations sending ARP broadcasts results in 100 broadcast packets being sent every second. This broadcast traffic will cause a significant amount of network traffic that would affect the network availability to other nodes on the network.

Broadcast traffic could even slow down nodes not currently accessing the network. This is because the network board and the associated software driver must examine every broadcast packet. In the case of ARP requests, the ARP module will also be executed for every broadcast request. Processing even 50 ARP requests/second can take a fair amount of processing power and have a significant slowdown effect on a node. As a result all nodes on the network will appear to slow down.

The repeated ARP broadcasts create further complications in a bridged network. Network segments separated by bridges are considered to be part of the same MAC domain. ARP requests/replies must be transmitted across these bridges. Propagation of broadcast traffic across bridges can have a multiplying effect and the network traffic could increase even more. Creating a large or complex network topology using bridges is therefore not desirable under these conditions.

# Duplicate Addresses and ARP

IP addresses on a network must be unique. How do you ensure this uniqueness? You ensure it by proper documentation and assignment of IP addresses. Because of the human element involved in assigning IP addresses, inadvertently assigning nodes on a network with the same IP address is possible. When this happens, chaos and confusion ensues. Networked applications work intermittently or sometimes mysteriously stop working altogether.

To help you understand the effect of duplicate IP addresses, the following sections discuss two scenarios:

- Duplicate IP addresses at TCP/IP clients
- Duplicate IP addresses at TCP/IP servers

## Duplicate IP Addresses at TCP/IP Clients

Consider the situation where two TCP/IP clients called “workstation 1” and “workstation 2” access a central TCP/IP server at IP address 144.19.74.102. The server’s hardware address is 080020021545.

Workstations 1 and 2 are using the same IP address of 144.19.74.1. Their hardware addresses are 0000C0085121 and 0000C0075106, respectively.

Workstation 1 starts an FTP session with the TCP/IP server. Before the FTP session starts, workstation 1 will issue an ARP broadcast request to discover the hardware address of the TCP/IP server. On receiving the ARP request, the TCP/IP server will add an entry for the ARP request sender (workstation 1) in its ARP cache table, and send an ARP reply to the sender. On receiving the ARP reply, workstation 1 will add an entry for the TCP/IP server in its local ARP cache table. At this point the ARP cache tables for the TCP/IP server and workstation 1 are as follows:

*TCP/IP server:*

| IP Address | Hardware Address |
|---|---|
| 144.19.74.1 | 0000C0085121 |

*Workstation 1:*

| IP Address | Hardware Address |
|---|---|
| 144.19.74.102 | 080020021545 |

In the second phase, workstation 2 attempts to establish a TCP/IP session (perhaps another FTP session) to the same TCP/IP server.

Before making the connection, workstation 2 must discover the TCP/IP server's hardware address. It attempts to do just that by sending an ARP request. On receiving the ARP request, the TCP/IP server will want to add an entry for the ARP request sender (workstation 2) in its ARP cache table, and send an ARP reply to the sender. The TCP/IP server will find that there is already an entry for 144.19.74.1 in its ARP cache table. As per RFC 826 on the Address Resolution Protocol, the TCP/IP server must replace the existing entry with the new one. Most TCP/IP implementations will perform this replacement silently without issuing any warning of a potential duplicate IP address problem. Some TCP/IP implementations will issue a warning of a potential duplicate IP address problem. The TCP/IP server will reply to the ARP request from workstation 2. On receiving the ARP reply, workstation 2 will add an entry for the TCP/IP server in its local ARP cache table. At this point the ARP cache tables for the TCP/IP server and workstation 2 are as follows:

*TCP/IP server:*

| IP Address | Hardware Address |
|---|---|
| 144.19.74.1 | 0000C0075106 |

*Workstation 2:*

| IP Address | Hardware Address |
|---|---|
| 144.19.74.102 | 080020021545 |

When the TCP/IP server wants to send data to workstation 1, it will look up the hardware address of workstation 1 at IP address 144.19.74.1 in its local ARP cache table and send the data to this hardware address. Unfortunately, this is workstation 2's hardware address, and the data will be sent to workstation 2. Workstation 1 on not receiving expected data will try to resend its last transmission, and the TCP/IP server will continue sending the response to workstation 2. Workstation 1 will therefore "hang" waiting for a response. It will eventually time-out, or the workstation may have to be rebooted to recover from the error. In any case, an application that was working fine some time ago may suddenly stop working, much to the consternation of the user and system administrator alike!

Workstation 2 on receiving unexpected data destined for workstation 1 will probably reject it, optionally generate an error message, and continue with its session with the TCP/IP server. Alternatively, workstation 2 can become confused with unexpected data and also "hang."

If the user at workstation 1 reboots the "hung" computer and tries to reconnect to the TCP/IP server, the ARP cache entry for workstation 2 at the TCP/IP server will be replaced with that for workstation 1. This will cause workstation 2 to "hang," and the situation could repeat until the users give up in frustration.

If the users at workstations 1 and 2 are not in communication with each other about the problems they are experiencing, each will experience his or her side of the story—that is, the workstation "hangs" unexpectedly in the middle of accessing the TCP/IP server. If the users do not access the TCP/IP server at the same time, they will never experience this problem and the duplicate IP address problem will go undetected. If the users experience the problem occasionally, they may dismiss the problem as a fluke on the network and never report it. Again, the problem will go undetected.

## Duplicate IP Addresses at TCP/IP Servers

Consider the situation where there are two TCP/IP servers called "server 1" and "server 2." They are both set to the duplicate IP address of 144.19.74.102. The servers' hardware addresses are 080020021545 and AA0004126750, respectively.

A workstation with the IP address 144.19.74.1 tries to access the TCP/IP server at 144.19.74.102. The workstation's hardware address is 0000C0085121.

The workstation tries to connect with server 2. Before making the TCP/IP connection, the workstation will issue an ARP broadcast request to discover the hardware address of the TCP/IP server. There are two TCP/IP servers at the same target IP address of 144.19.74.102. On receiving the ARP request, both servers 1 and 2 will add an entry for the ARP request sender (workstation) in their local ARP cache tables, and send an ARP reply to the sender. Two responses are sent on the network. The workstation accepts the first ARP reply and silently ignores the second response. If server 2's ARP reply reaches the workstation first, the workstation will add an entry for server 2 in its local ARP cache table. At this point the ARP cache tables for the servers and the workstation are as follows:

*Server 1:*

| IP Address | Hardware Address |
|---|---|
| 144.19.74.1 | 0000C0085121 |

*Server 2*:

| IP Address | Hardware Address |
|---|---|
| 144.19.74.1 | AA0004126750 |

*Workstation:*

| IP Address | Hardware Address |
|---|---|
| 144.19.74.102 | 080020021545 |

The workstation will then continue interacting with server 2, instead of server 1 as it had originally intended. If server 2 does not have the expected service, the connection will fail, and the user will experience an error message about the service not being supported at the server. This would be puzzling to the user, especially if the user had successfully accessed services at server 1 on prior occasions.

If server 2 has the service that the workstation expects, the user will attempt to log on to that service. If the service is Telenet or FTP, a user login and password will be required. If the user is not alert, he or she may not notice that the FTP server or Telnet server name/version number reported on the screen is different from the one he or she expects. If the user does not have a logon account with the same name and password as server 1, logon will be denied. If the logon account and password are the same on both servers, the user will log on to the wrong server. The user may not find the files or data he or she is expecting because he or she has logged on to the wrong server.

If the server is a *multihomed* host (a server with multiple network connections) that acts as a router, there may be the possibility of even more confusion because packets destined to outside networks may not follow the expected route or may not get delivered because the wrong server has accepted the packets for delivery.

## The ARP Duplicate Address Test

Many TCP/IP implementations broadcast an ARP request on startup. The ARP request is sent with the Target IP field in the ARP request set to that of the node that is starting up.

The purpose of this initial ARP request is to discover whether any nodes have a duplicate IP address. If an ARP reply is received, another node (the one generating the ARP reply) has the same IP address as that of the node that is booting up. The node that is starting up should report the duplicate IP address in a suitable manner. It is even possible for the node responding to the ARP request to do an extra check on the Sender IP field in the ARP request packet to see whether it is the same as its own IP address. If the Sender IP and Target IP address fields match, the responding node can generate a suitable alert about a duplicate IP address problem.

The initial ARP frame can only discover duplicate IP address problems on the same network segment. Because ARP packets do not cross router boundaries, they cannot detect duplicate IP address problems for network segments connected by routers.

Another function of the initial ARP broadcast is for nodes that have an entry for the node that is starting up to update their ARP cache table. If the node that is starting up has a different hardware address (because the network hardware was changed), all nodes receiving the ARP broadcast discard the old hardware address and update their ARP cache tables with the new hardware address.

# Proxy ARP

Earlier in this chapter, you saw that two networks connected through a gateway can have the same network address. The gateway has to determine to which of the networks the physical address or IP address of an incoming datagram corresponds. The gateway can do this with a modified ARP called *Proxy ARP* (sometimes called Promiscuous ARP).

Proxy ARP creates an ARP cache consisting of entries from both networks. The gateway has to manage the ARP requests and replies that cross the two networks. By combining two ARP caches into one, Proxy ARP adds flexibility to the resolution process and prevents excessive request and reply ARP datagrams whenever an address has to cross a network gateway.

# Reverse Address Resolution Protocol

If a device doesn't know its own IP address, there is no way to generate ARP requests and ARP replies. This is often the case when a device such as a diskless workstation is on the network. The only address the device is aware of is the physical address set on the network interface card.

A simple solution is the *Reverse Address Resolution Protocol* (RARP), which works as the reverse of ARP. RARP sends out the physical address of a destination and expects back an IP address. The reply containing the IP address is sent by a RARP server, a machine that can supply the information. Although the originating device sends the message as a broadcast, RARP rules stipulate that only the RARP server can generate a reply. Many networks assign more than one RARP server, both to spread the processing load and to act as a backup in case of problems.

RARP is often used in "diskless stations" that do not have local storage. Modern diskless stations may actually have a local disk, but this disk is used for speeding up the operation of the operating system and not for storing TCP/IP-related parameters, which are kept at a remote server. For example, the local disk may only be used for holding a local swap area needed to implement the operating system's virtual memory mechanism.

Diskless stations also have a copy of the operating system image kept on remote servers. On startup the diskless station downloads a copy of the operating system image from the remote server to its memory. Before the diskless station can download its operating system image using TCP/IP file transfer protocols, it needs an IP address. This IP address cannot be a random value; it must be unique and have the same prefix as the IP addresses of other nodes on that network segment. For example, if other stations on the network have IP addresses like 199.245.180.1, 199.245.180.5, 199.245.180.7, then the common prefix is "199.245.180". The diskless station must also have the same common prefix. A common prefix is needed because information on how to route to a network segment is embedded as a prefix value in the IP address.

The diskless station typically obtains its IP address by sending a request to servers on that network segment. Because it does not know the address information of the remote server, this request is sent as a Data Link Layer broadcast. One mechanism to obtain IP addresses from remote servers is the Reverse Address Resolution Protocol (RARP). The protocol is called Reverse ARP, because the information it seeks is opposite to that sought by the ARP protocol.

Diskless stations are used to reduce hardware costs, simplify node configuration and upgrades by keeping critical information in a central location, and reduce the possibility of viruses being introduced because the users cannot easily introduce programs through diskless workstations.

## RARP Operation

In RARP operation, the node that wants to discover its IP address (called the RARP client) broadcasts a request called the RARP request. The RARP request is broadcast at the Data Link Layer because the RARP client does not know the address (hardware or IP address) of the remote server. The remote server that processes the RARP request is called the RARP server. All nodes on the network segment will receive the RARP request broadcast, but only the nodes acting as RARP servers will respond.

If there is more than one RARP server, all RARP servers will attempt to process the RARP request. The RARP client will typically accept the first response it receives and silently ignore the rest.

The RARP server keeps a table of the IP address for nodes on the network segment. This table is indexed by a unique identifier that is specific to each machine. This unique identifier is sent in the RARP request. For diskless stations this unique identifier must be some hardware-specific parameter that it can easily read. The identifier cannot be a value stored in local storage because diskless stations do not have a local disk available for this purpose. The RARP designers (see STD 38, RFC 903) decided to use the hardware addresses as unique identifiers because they are unique for a network segment and can be easily read by the network drivers.

When a RARP server receives a RARP request, it consults its table of IP address and hardware address bindings (see Figure 5.6). If it finds an entry in the RARP table that matches the hardware address found in the RARP request, it returns the corresponding IP address in a RARP reply. The RARP reply is not broadcast because the RARP server looks up the hardware address of the RARP client from the RARP request.

The RARP request and reply packets use the same packet format as ARP packets. The difference between ARP and RARP is the values that are placed in the fields. When RARP packets are in an Ethernet frame, an EtherType value of 8035 hex is used.

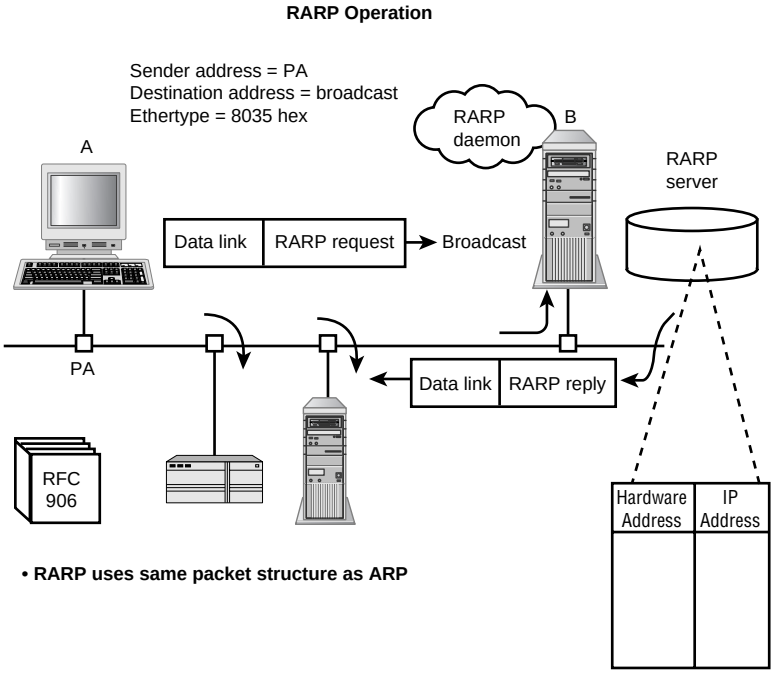



**FIGURE 5.6**
*RARP operation. (Courtesy of Learning Tree)*

The RARP request fields are filled in as follows.

RARP Request:

Data Link Destination Hardware Address = *Broadcast*
Data Link Source Hardware Address = RARP client HA
DataLink EtherType = 8035 hex
Operation = 3 (RARP Request)
Sender HA = RARP client HA
Sender IP = Undefined; Usually 0.0.0.0 used
Target HA = RARP client HA
Target IP = Undefined

RARP Reply:

Data Link Destination Hardware Address = RARP client HA
Data Link Source Hardware Address = RARP server HA
DataLink EtherType = 8035 hex

Operation = 4 (RARP Reply)
Sender HA = RARP server HA
Sender IP = RARP server IP address
Target HA = RARP client HA
Target IP = RARP client IP address; This is the answer

The Target HA address field in the RARP request is set to that of the RARP client. This is done as a convenience for the RARP server that does not then have to modify this field, because it contains the RARP client hardware address in the RARP reply.

The network driver can examine the EtherType field and know that this is a RARP packet (EtherType = 8035 hex) and send the request to the RARP module. The Operation code fields used for RARP are 3 for RARP request and 4 for RARP reply. These fields could also be used to distinguish an ARP packet from a RARP packet. However, this approach is not very efficient because the ARP module would have to examine the Operation field first, and if this indicated a RARP packet, the ARP module would then send it to the RARP module. By using the EtherType field protocol, demultiplexing can be done efficiently at a lower layer.

**Note**

In many implementations RARP is not automatically provided by the ARP or IP module. It must be run as a separate process, such as a daemon process, on the computer that is to act as a RARP server.

## RARP Storms

If the RARP server is unavailable because of a hardware link failure or because the RARP server is down, RARP clients will be unable to boot up. RARP clients will continually send out RARP broadcast requests. If many RARP clients simultaneously send out RARP request broadcasts, then a heavy network traffic load results.

The situation of not receiving a RARP reply is more serious than not receiving an ARP reply. Not receiving an ARP reply means that a particular host and its services are not available. The ARP client can still try to access other nodes on the network. However, if there is no RARP reply, the RARP client cannot boot up. Many RARP clients continue to send RARP broadcast requests indefinitely in the hope that the RARP server may eventually become available. The situation of repeated RARP broadcast requests is therefore more serious than that of ARP requests because it can result in greater network traffic.

## Primary and Backup RARP Servers

To make the availability of RARP services more reliable, multiple RARP servers can be used. With multiple RARP servers, all servers would attempt to answer to the RARP broadcast request simultaneously. Only one of the RARP replies is used by the RARP client; all other RARP replies are discarded as to not use up network bandwidth. One scheme to prevent multiple RARP replies is to designate one of the RARP servers as a primary and all others as secondary RARP servers.

On receiving a RARP request, the primary RARP server will respond to the request. The secondary servers do not respond, but note the arrival time of the RARP request. If the primary does not respond within a timeout interval, the secondary RARP server assumes that the primary RARP server is down, and responds to the RARP request.

If there are multiple secondary RARP servers, all secondary servers will attempt to respond at the same time. A refinement of this scheme is to have secondary RARP servers wait for a random time interval before responding. If the primary server is functioning normally, there will be no additional delays in sending the RARP reply. If the primary server is down, there may be a small random delay before one of the secondary RARP server responds.

# Using the ARP Command

Most implementations of TCP/IP (but not all) provide a way for you to check the ARP cache. Both UNIX and Windows NT/2000/XP implement the `arp` command. The `arp` command shows you all the entries in the machine's cache. To see the cache contents, issue the command with the `-a` (for all) option:

```
$ arp -a
brutus <205.150.89.3> at 0:0:d2:03:08:10
```

In this case, one machine called `brutus` has the IP address 205.150.89.3 and the MAC (Media Access Control) address 0:0:d2:03:08:10.

The `arp` command is seldom used except when a network administrator is trying to resolve the problem of duplicate IP addresses. If there are two machines with the same address (but with different MACs), they will show up in the ARP cache.

# Summary

In this chapter, you have seen the address resolution protocols in common use: ARP, Proxy ARP, and RARP. The chapter started off with a look at how Ethernet frames are

assembled on top of the TCP/IP datagrams. With all these layers of complexity in internetworking, you should be getting a good idea of how TCP/IP functions and how data is passed over networks to destination machines.

CHAPTER 6

# DNS: Name Services

*by Rima S. Regas and*
*Karanjit S. Siyan, Ph.D.*

## In This Chapter

In the early ARPANET when the network was small, users on a network were required to maintain a HOSTS configuration file. This file contained all the information the workstation needed in order to communicate with the other systems on the network. Problems were prevalent, however, because each machine's HOSTS file needed to be updated separately, by hand. There was little or no automated configuration, making updates tedious and time-consuming.

A HOSTS file contains information about which IPs are assigned to what name. When the computer needed to locate another computer on the network, it would look at the local HOSTS file. If the computer did not have an entry in the HOSTS file, it essentially did not exist. *Domain Name System* (DNS) changed this situation drastically. DNS allowed systems administrators to use a server as a DNS host. When the computers on the network needed to locate another server, for example, they would look to the HOSTS file.

Today, the HOSTS file is still used, but typically only to prevent a machine on a LAN from using the DNS to look up a local machine. It's much faster this way. In a nutshell, the computer's networking software is directed to look in the local HOSTS file before using the default DNS server. If a match is there, the client software proceeds to communicate with the remote host directly, greatly reducing the time involved in discovering the IP number via DNS.

# Domain Name System: The Concept

The technique used to give IP addresses names is *mnemonic*. The term is used to indicate that names are easier to recall than numbers. Although most people have an amazing capacity for remembering phone numbers, addresses, amounts, and other life-related figures, simply far too many IP numbers are in existence to make remembering easy. Hence the naming format. For example, C|Net's domain name is `www.cnet.com`. There is no vertical slash and there is no capitalization. In comparison to the name of the site, it's rather sterile. It works, though. Mnemonically, it's a perfect match for the site's name. C|Net is one of the largest and most visited sites in the world.

Before there was DNS, the idea was to have a system that would translate a name into its given number. When you translate or resolve a Web site's domain name (for example, `www.cnet.com`) and find its IP number, (it is 204.162.80.181, in case you were wondering), the IP number is the actual address. This is how the correct Internet content is delivered to your Web browser. This process requires a network of systems called DNS, or Domain Name Service (system, if you talk to certain people), servers.

Part of the job of maintaining a DNS system on the Internet involves providing the primary root domain servers to which the rest of the DNS servers would look to find the top-level domains registered on the Internet. DNS information is distributed across many DNS servers. If your DNS server cannot resolve the domain name to an IP address, that server contacts another DNS server that is expected to know more information about the name to be resolved. If that DNS server is unable to resolve the domain name, it can return a referral to another DNS server that might know the answer. This process is repeated until a timeout is reached. An error is then returned to the initiating IP and, if the client is capable, an error message is displayed. In the case of a Web site that cannot be found, the browser will display an error message stating that it was unable to locate the server or there was a DNS error.

The referral mechanism described in the previous paragraph is typically used by DNS servers when they query other DNS servers. This type of DNS query is called an *iterative* query. The DNS resolver, which is another name for a DNS client, resides on the node that issues the DNS query. The DNS resolver does not issue an iterative query. Instead, the DNS resolver issues another type of query called a *recursive* query. In a recursive query, the node issuing the query to the DNS server expects that DNS server to return the final answer or a response indicating that the name could not be resolved. When a DNS server receives a recursive query it does not return a referral to another DNS server; it is expected to do iterative queries to obtain the final answer.

Hierarchical by virtue of its organizational structure, DNS cascades from the top-level DNS root servers and propagates names and IPs to servers located all around the world. An example that clearly shows how the system is hierarchical is as simple as a DNS server that does not have a translation stored locally. When it cannot find the IP in its own databases, it seeks out another DNS server to see whether it has the IP address, and so on until it either times out or finds it. The following section takes a closer look at the structure because there is another, more important aspect to the hierarchy.

## DNS Hierarchical Organization

In the original RFC #819 (Request For Comments) document published in 1982, Zaw-Sing Su and Jon Postel laid out the concepts and plans for the DNS. Here's an example of the language they used.

> The set of domains forms a hierarchy. Using a graph theory representation, this hierarchy may be modeled as a directed graph. A directed graph consists of a set of nodes and a collection of arcs, where arcs are identified by ordered pairs of distinct nodes [1]. Each node of the graph represents a domain. An ordered pair (B, A), an arc from B to A, indicates that B is a subdomain of domain A, and B is a simple name unique within A.

This is not very easily understood, if comprehensible at all, so we will go through the concepts in a simpler fashion. The principle is simple. You have a top-level domain, like `COM` or `EDU`. `COM` and `EDU` domains represent the highest domain level and cannot be superseded. There is, however, one exception. In truth, there should be a period after `COM` (`COM.`). That period is an even higher domain than `COM` called the root domain. Top-level domains (such as `COM`, `EDU`, and `ORG`) are referred to as a Naming Universe because they contain the domains and subdomains that are placed below them in the hierarchy, much like the roots of a tree.

From the top level you move into the intermediate domains or the second-level domains. Examples of intermediate domains include `coke.com`, `whitehouse.gov`, and `disney.com`. These can be registered, but are usually only done so to cover a range of related domain names. Registering a complete domain in a subdomain, or Endpoint domain as Su and Postel referred to it, is common. Examples of these are `www.coke.com`, `www.whitehouse.gov`, and `www.disney.com`. Of course, these are all representational of domains in the InterNIC registration system. Other systems outside the United States use different conventions to indicate location and country of origin. For example, `www.bbc.co.uk` indicates that the BBC's Web site is a commercial concern (CO, which is similar to COM) and is located in the United Kingdom (UK).

Figure 6.1 shows Su and Postel's original Domain Hierarchy figure illustrating the branching possibilities.

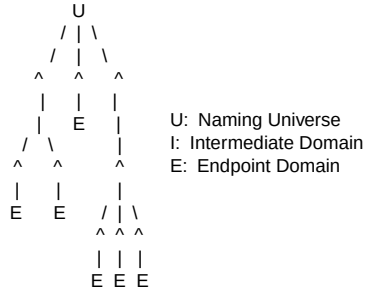



**FIGURE 6.1**
*An illustration of the original domain hierarchy.*

Subdomains can be just about anything. Examples of this are `www.netscape.com`, `home.netscape.com`, and `www1.netscape.com`. All of these could be very different places on the Internet and often will contain different information. Of course, all of these could point to one site and one page if the Webmaster so chose. That is the case with the Netscape site; it can be reached by all the listed URLs.

**Note**

As another example, Webopædia chose to also register `webopedia.com` as a subdomain because many people had trouble with the classic greek character æ.

# Delegating Authority

DNS is arranged in a way that allows servers subordinate to the root name servers to take over control of a given domain. A good example of this would be a local ISP that hosts Web sites for people and companies. When Company X registers its domain name (`www.companyx.com`) with InterNIC, it declares the Primary and Secondary DNS servers of its ISP as its DNS servers. InterNIC puts the information into its `.COM` root server and allows it to propagate.

**Note**

DNS servers periodically synchronize their local database with various other databases on other DNS servers and check for new entries on the root servers. This process is commonly referred to as *propagation*. Domain name registration is by no means instantaneous, but a newly registered domain name will propagate in roughly three to four days, making it available around the world.

In order for this system to work there needed to be a way of determining where a machine was located on the network. This need gave rise to the system hierarchy, a means of categorizing each machine in relation to its function. For example, if the machine in question was located at an educational institution, it would be located in the top-level domain known as `.EDU`. If the site was commercial or civilian in nature, it would be placed in the `.COM` top-level domain, and so on. This concept forms the root of the DNS hierarchy, but not the root name servers (we'll get to more on that shortly).

# DNS Distributed Database

The DNS distributed database architecture is quite powerful. As previously stated, authority is delegated to other servers that are better positioned to handle the traffic for a domain or subdomain. The manner in which the information is propagated to the servers is via a thorough distribution plan.

Each domain has an owner assigned to it. This is defined as part of the domain's *Start of Authority* (SOA), which is covered a little later in greater detail. The top-level domain, for example `COM.`, delegates authority for a domain name, such as `disney.com`, to the DNS servers specified to act as the primary DNS server. This relieves the top-level domain controller from being burdened with handling every single DNS query on the Internet.

After an SOA is associated with a domain controller, it can delegate subdomain control to other DNS servers, and so on. This is the manner in which delegation is distributed down the DNS server hierarchy from the topmost level to the bottom.

# Domains and Zones

Domains and zones are often paired together, but there is a subtle difference. A zone is the primary domain in a naming universe that is delegated to another DNS server for administrative purposes. Whereas `Disney.com` is a zone, `www.disney.com` is a host in that zone. A zone can consist of a single domain or several subdomains. For example, Disney could have subdomains called east-coast.Disney.com and west-coast.Disney.com. If these subdomains are delegated, they will form a separate zone called east-coast.Disney.com and west-coast.Disney.com, which will have their own DNS servers. If these subdomains are not delegated, they are part of the zone `Disney.com` and are handled by the DNS servers for the `Disney.com` server.

Administrative duties are delegated to the primary DNS server. The primary DNS server for Disney's site is `huey.disney.com` and its IP is 204.128.192.10. Because Huey is designated the primary DNS for Disney, it is also the primary DNS for the zone of `Disney.com`. The designated owner for this server is `root@huey.disney.com`, which is shown in the SOA as `root.huey.disney.com`.

**Note**

Be aware that a properly formatted SOA will change the administrator's email address to all periods, removing the @ symbol common to all email addresses. Some companies like to format email addresses with a separator preceding the @. This is problematic because, for example, an administrator's email address that is formatted as `host.admin@example.com` would be shown in the SOA as `host.admin.example.com`. Any system attempting to resolve that would assume `host@admin.example.com` and would receive a DNS error (assuming that it is not a valid address).

# Internet Top-Level Domains

There are a number of top-level domains (TLD), the most familiar of which are `COM`, `EDU`, `GOV`, `MIL`, `NET`, and `ORG`. There are, however, many, many more. Most of the unfamiliar ones are located in other countries. Each country has its own two-letter TLD. UK is the TLD for the United Kingdom (actually, the official TLD is GB, but this is not commonly used), NZ for New Zealand, JP for Japan, and so on. An example of a TLD in use would be `www.bbc.co.uk`.

**Note**

More domains are involved in foreign addresses than in the States. The `CO` indicates a commercial concern, much like the InterNICs `COM`.

**Caution**

There is nothing to prevent people from registering domains in other countries even though the server would not even be near the country indicated in the URL.

# Choosing a Name Server

You can select a name server on many platforms, such as Windows NT/2000/XP, UNIX, NetWare, Macintosh, and so on. Windows NT/2000 comes with DNS server implementations. Apple has a free DNS server called MacDNS. UNIX systems have the ever-present and reliable BIND to fall back on. There are, however, still a number of powerful third-party DNS servers available.

A good site to evaluate DNS servers is Dave Central (`www.davecentral.com`) and Tucows (`www.tucows.com`, and be prepared to select a local server affiliate). They generally review just about every piece of software that comes through there and give you a good idea of what is available.

# Name Service Resolution Process

When a client, such as a browser, puts in a request for a URL, it is passed to the local DNS server, which tries to parse the name into an IP address. If it is successful, it passes the data onward, completing the next leg of the journey, and takes the next request. If the

server is unable to locate the address, it has two options, depending on the way the server is configured. Those two are Recursive and Iterative.

## Recursive Queries

*Recursive* queries are the most typical. If the query comes into the server and the A record is located in the cache for the server, no additional searching is needed. If not, it must ask another server, so it moves up the ladder. This is where the *Time to Live* (TTL) comes in. If finding the A record takes too long, the query sputters out and dies in the process. The originating DNS server returns an `address not found` error.

## Iterative Queries

*Iterative* queries are forced to stay local for several reasons, the most common being that another DNS server is not available. The recursion feature may have been turned off in the DNS server you are accessing, but it is unlikely. The server will do its best to locate the best match it can possibly find in its cache. If it is not there, it's not there. An error is returned.

## Caching

As the DNS server goes about its business throughout the day, it picks up resources and stores them. These are Resource Records (RRs) that contain information about queried URLs. The TTL also comes into play here. The server will only cache as much as it is allocated and as long as it is valid.

## Reverse Resolution (Pointer) Queries

The typical query is forward looking, trying to match a URL with an IP. Reverse resolution is just the opposite, trying to match an IP to a URL. Several utilities will perform this type of lookup for you, but the best one for Windows is CyberKit from Luc Niejens, a freeware network utility that you can download at `www.ping.be/cyberkit/`.

## DNS Security

Clients can make secure updates to a dynamic DNS server so that their records are automatically updated without administrator intervention.

## Resource Records (RR)

All the DNS resource records have a similar format. Although many shortcut notations and abbreviations can be found in DNS files, these examples use the simplest nomenclature to eliminate confusion and ambiguity.

The first field in any DNS record is always either an IP address or a hostname. If it is missing, the name or address from the previous record is implied. *Note that all names and addresses end with a trailing "dot"(.)*. This signifies that the name or address is absolute rather than relative. *Absolute* addresses, also called *fully qualified domain names*, are relative to the root, whereas *relative* addresses are relative to a default domain (which may or may not be the root). This field may, optionally, be followed by a Time-to-Live (TTL) value, which indicates the length of time that the information in this field should be considered valid.

The second field indicates the address type. In today's DNS databases, the string "IN" most likely indicates an Internet address. This field is present for historical purposes and compatibility with older systems.

The third field is a string that indicates the type of resource record. This field is followed by optional parameters that are specific to the RR.

## Start of Authority (SOA)

An SOA record indicates that this DNS name server is the best source of information for the data within this DNS domain. SOA records serve to identify the email contact for the zone, to identify that changes have been made to a zone's database file, and to set timers on how often other servers update their copies of the zone information.

The record stores the name of the DNS system and the name of the person who is responsible for it. Here is an example of an SOA at the top of an RR:

```
; Start of Authority (SOA) record
dns.com.  IN SOA dns1.dns.com. owner.dns.com. (
         00000001  ; serial # (counter)
           10800  ; refresh (3 hours)
           3600  ; retry (1 hour)
          604800  ; expire (1 week)
           86400) ; TTL (1 day)
```

A couple of things to note here are that the owner is listed as `owner.dns.com`, which should be read as `owner@dns.com`. Also, there is an opening parenthesis following that. It is closed in the last line after the TTL value. This is important because the record is formatted this way to ease reading. This could easily be shown like the following and still be valid:

```
Dns.com. IN SOA dns1.dns.com. owner.dns.com.
➥(0000001 10800 3600 604800 86400)
```

The semicolon (;) is the comment operator here. Anything that comes after that will be ignored. The last item of note is that all numerical values are shown in seconds. The sub-items are discussed in the following sections.

## Serial Number

The *Serial Number* value identifies the active revision of the DNS database. When the database gets updated, this must be incremented so that secondary servers will know that they must update their database.

The Serial Number controls DNS zone transfers between primary and secondary DNS name servers. The secondary name server contacts a primary name server periodically and requests the serial number of the primary's zone data. If the primary zone's serial number is larger than the secondary's, the secondary's zone data is out of date and the secondary transfers a new copy of the zone data from the primary. The transfer of information from a primary to a secondary server is called a *zone transfer*.

The serial number value can be a simple counter, but more typically a timestamp value is used, because the timestamp when the change was done will always increase with newer changes to the primary server database.

## Refresh

*Refresh* tells any secondary name servers how often to check for updated information (10,800 seconds is three hours).

The number in the Refresh Interval field specifies how often the secondary DNS name servers check the zone data of the primary DNS name servers. The default refresh interval is three hours; if your secondaries accept notification of changes, you can safely increase it to a long value (a day or more) to reduce the polling. This feature is particularly valuable over slow or on-demand lines.

## Retry

If the secondary name server is unable to contact the primary name server, it will reattempt a connection every *retry* seconds (3600 seconds is one hour).

The Retry Interval field determines how often a secondary server should try to reconnect after a failed connection. The Retry Interval usually is shorter than the Refresh Interval. The default value is 60 minutes.

## Expire

If a secondary name server cannot contact the primary name server for *expire* seconds, the secondary will stop answering any queries about this domain. The theory here is that at some point, the data is so old as to be possibly harmful, and no answer is better than a bad answer (604,800 seconds is one week).

assigned to a single system, care must be taken. Some mailers, for example, do unpredictable things when asked to resolve an MX hostname that uses a CNAME record rather than an explicit A record.

You can use CNAME records to hide the implementation details of your network from the clients that connect to it. For alias names, you can use shortened forms of the hostnames. The hostname `public1.business.com`, for example, might have an alias of `www.business.com`.

## Pointer (PTR) Records

Pointer records map an IP address to a hostname in the DNS reverse lookup domain (in-addr.arpa). The pointer record specifies the IP Address and the Host Name corresponding to that IP address. For the hostname, you use the FQDN. Many Internet sites deny access to computers that do not have matching Address and Pointer records as a security measure, so be sure to keep your PTR records synchronized with your A records.

The reverse lookup query is sometimes called a PTR query and can also be used to discover the organization responsible for an IP address, because the result of a pointer query contains the FQDN, and the FQDN includes the domain name.

## Delegated Domains

Delegated domains remove the primary responsibility of administration for a domain from one server to a subservient one. The ICANN has administrative control over `.COM`. and delegates all domains registered under it to their respective controllers. In reality, most sites are hosted, which means the server space is sold to someone and the site owner does not have to maintain the hardware. That job is for the hosting service. Control of the domains that reside on hosted sites are often delegated to the DNS server administrator for the DNS servers of the host. There is, however, a free DNS server on which you can request space at `http://soa.granitecanyon.com`.

## Hardware Information (HINFO) Record

The HINFO record identifies a host's hardware type and operating system. The Internet standard abbreviations for identifying the operating system of a host are published in the Assigned Numbers RFC (RFC 1700 at the time of this writing) under the System Names List. Abbreviations for the hardware type of hosts also are defined in the Assigned Numbers RFC. The CPU Type identification number is in the Machine Names List. For a small sample of the types specified in the Assigned Numbers RFC, see Table 6.1.

The names are up to 40 characters long and contain uppercase letters and digits, including a hyphen (-) and a slash (/). The names must begin with a letter and end with a letter or digit. Some people choose not to provide this on publicly accessible DNS servers because it reveals information about a computer system that might make it easier for someone to break into a system, perhaps by attacking known security flaws for a particular system type. If you do use HINFO records, be sure to keep up with CERT security advisories for your systems.

**TABLE 6.1** RFC 1700 Standard Codes for Common Operating Systems

| | | |
|---|---|---|
| APOLLO | OPENVMS | TANDEM |
| AIX/370 | OS/2 | UNIX |
| DOMAIN | PCDOS | UNIX-BSD |
| DOS | SCO-OPEN-DESKTOP-2.0 | UNIX-PC |
| INTERLISP | SCO-OPEN-DESKTOP-3.0 | UNKNOWN |
| ITS | SCO-UNIX-3.2V4.0 | VM |
| LISP | SCO-UNIX-3.2V4.1 | VM/370 |
| MSDOS | SCO-UNIX-3.2V4.2 | VMS |
| MULTICS | SUN | WANG |
| MVS | SUN-OS-3.5 | WIN32 |
| NONSTOP | SUN-OS-4.0 | X11R3 |
| APOLLO | IBM-RS/6000 | SUN-4/200 |
| APPLE-MACINTOSH | INTEL-386 | SUN-4/390 |
| APPLE-POWERBOOK | M68000 | SYMBOLICS-3600 |
| CRAY-2 | MAC-II | UNKNOWN |
| DECSTATION | MAC-POWERBOOK | VAX |
| DEC-VAX | MACINTOSH | VAX-11/725 |
| DEC-VAXCLUSTER | MICROVAX | VAXCLUSTER |
| DEC-VAXSTATION | PDP-11 | VAXSTATION |
| IBM-PC/AT | SILICON-GRAPHICS | |
| IBM-PC/XT | SUN | |

## Integrated Services Digital Network (ISDN) Record

The ISDN record maps the hostname to an ISDN address. The ISDN phone number also can include the Direct Dial In (DDI) number. An ISDN subaddress is optional. ISDN records have several possible uses. The ISDN resource records can provide simple documentation of the correct addresses to use in static configurations of ISDN dial-up applications. In the future, they also may be used automatically by Internet routers.

## Mailbox (MB) Record

The MB record identifies a mailbox for the indicated host. You must provide the Mailbox name and the fully qualified domain name (FQDN) of the host that contains the mailbox. This record type is considered experimental and should not be used unless you are part of the Internet experiment.

## Mailgroup (MG) Record

The MG record identifies a mailbox that is a member of the group mailing list in the given DNS domain name. You must type the mailbox name as a FQDN (Fully Qualified Domain Name). This record type is considered experimental and should not be used unless you are part of the Internet experiment.

## Mail Information (MINFO) Record

The MINFO record provides information about a mailing list or mailbox. It has two fields: the Responsible Mailbox DNS field, which contains the fully qualified domain name (FQDN) of the mailbox that is responsible for the mailing list identified in the resource record; and the Error Mailbox DNS field, which contains the FQDN of the mailbox that is to receive error messages related to the mailing list. Although you can associate these records with a simple mailbox, they usually are used with a mailing list that uses different mailing addresses for administrative matters, such as subscriptions (usually the responsible mailbox), and for error messages, like `mail not deliverable` (the error mailbox).

## Mail Rename (MR) Record

The MR record renames a given mailbox. Type the FQDN of the new mailbox name in the Replacement Mailbox DNS Name field. You can use an alias name if a mailbox is moved from one host to another. This record type is considered experimental and should not be used unless you are part of the Internet experiment.

## Mail Exchange (MX) Record

An MX record indicates the hostname of a server that will process or forward mail for a given domain or host. The hostname of the mail exchanger must be a FQDN. The MX record format consists of the following fields:

- **For Domain.** Enter the host or domain name for which the mail exchanger will process mail.
- **Mail Exchange Server DNS Name.** Enter the FQDN of the mail exchange server.
- **Preference Number.** Enter the preference number of that particular mail exchanger.

The Preference Number is a number from 0 to 65,535 that indicates the mail exchange server's priority with respect to the other mail exchange servers for that destination. Lower preference numbers have higher priority, the highest priority being a preference number of 0. The absolute value of the preference is not significant except in relation to other mail exchanger values for the same destination. A mailer attempts delivery to the mail exchange server with the lowest preference number first. If delivery fails, the mail exchange server with the next highest preference number is tried next. If two or more mail exchange servers share the same preference number, the mailer must decide which one to try first. All mail exchange servers with a given preference number are tried before the mailer moves up to the next highest preference number, however. One normally uses a preference of 10 for the best mail server, with less preferred mail exchangers identified at 20, and then 30, or some similar scheme. Doing so enables you to insert a new mail exchanger between or before your existing mailers if needed.

Mail Exchange lookups are not recursive. After an MX record identifies the best destination for mail to a host, the only other lookup performed is for the A record of the mail exchanger. MX records for the mail exchanger itself are ignored unless the mail is addressed directly to that particular mail exchanger.

## Responsible Person (RP) Record

The RP record indicates who is responsible for the device identified by the specified DNS name. Although the SOA record indicates who is responsible for a zone and gives the best point of contact for DNS-related issues, different people usually are in charge of various hosts within a domain. The RP record enables you to indicate who should be contacted if a problem arises associated with something that has a DNS name. Typical problems might be that an important server has become unavailable or that another site is noticing suspicious activity from a host.

An RP record requires an email address and the FQDN to check for text records with more information. The email address is in standard DNS domain name format, with a period for the at symbol (@) used in most email addresses. The DNS text reference is a DNS domain name that acts as a pointer to a TXT resource record. You can use the Text resource record to which the RP record points to provide free-format information about the contact, such as full name and phone or pager number.

## Route Through (RT) Record

The RT record identifies an intermediate host that routes packets to a destination host that does not have direct connectivity itself. The Intermediate Host DNS Name must be a FQDN of an intermediate host that will accept packets for the destination. An intermediate host functions like a mail exchange server, in that there can be only one level of indirection, which means that the path to an intermediate host cannot be through another intermediate host.

## Text (TXT) Record

The TXT record contains information in straight text form, often a name and phone number of a contact or other relevant information. The text string must be less than 256 characters in length.

## Well Known Service (WKS) Record

The WKS record describes the services provided by a particular protocol on a particular interface, identified by the IP address. The services include Telnet and FTP. The Access Protocol indicates the protocol for the available service. This record generally isn't used and should be avoided (RFC 1123).

## X25 Record

The X25 record is similar to the A resource record, but maps the hostname to an X.121 address rather than an IP address. X.121 addresses are the standard form of addresses for X.25 networks. The PSDN (Public Switched Data Network) address begins with the four-digit DNIC (Data Network Identification Code) specified in X.121 guidelines established by the International Telephone and Telegraph Consultative Committee. Do not use the national prefixes as part of the address. The X25 resource record is designed for use in conjunction with the RT (route through) resource record.

# Summary

In this chapter you learned that a hierarchical structure controls the Domain Name System for the Internet. This system is expressly available to make names the primary form of addressing for the Internet's resources. You also learned that DNS uses a number of methods to determine the IP number of a domain name. It is important to remember that, like anything, the system can fail. If this material is not clear to you or you find yourself confused by its structure, allow your ISP to handle it. This is, after all, the way a Web site is reached or a resource located under another protocol.

CHAPTER 7

# WINS

*by Kurt Hudson*

## In This Chapter

The *Windows Internet Name Service* (WINS) is also called the NetBIOS Name Service because it handles the resolution of NetBIOS names to IP addresses. The Network Basic Input/Output System (NetBIOS) was invented by IBM and co-developed by IBM and Microsoft. Later, Microsoft acquired NetBIOS to use in its now obsolete LANManager product. Today, NetBIOS lives on in Microsoft network operating system products earlier than Windows 2000 (that is, NT 5.0). Networks running Windows 3.x, Windows 95, Windows 98, and Windows NT 4.0 or earlier utilize NetBIOS networking. Anyone supporting a network running those operating systems should familiarize himself with NetBIOS name resolution, especially WINS.

This chapter describes the configuration process for, and limitations of, NetBIOS names. NetBIOS name resolution techniques such as LMHOSTS files and WINS are discussed. A special emphasis is placed on WINS name resolution, WINS client and server configuration, and WINS administration.

**Note**

Microsoft has stripped NetBIOS out of Windows 2000 products. The company says that its future operating systems will only support NetBIOS naming for backward compatibility, but will not be dependent upon NetBIOS naming. For this reason, WINS is discussed for Windows 9x and Windows NT. Although a Windows 2000 server can be configured as a WINS server, this would normally be done for backward compatibility only.

# NetBIOS

Microsoft considers NetBIOS an *Application Programming Interface* (API). Essentially, an API is a layer of separation between components. The NetBIOS Layer resides between the Application and Transport Layers in the Microsoft TCP/IP protocol stack, as shown in Figure 7.1.

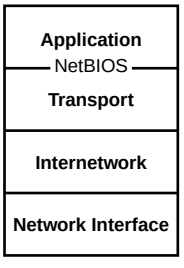



**FIGURE 7.1**
*NetBIOS in Microsoft's TCP/IP protocol stack.*

The function of NetBIOS is to provide independence between the Application and Transport Layers. The goal is to allow an application developer to write a network-enabled program without having to understand the underlying protocol. In addition, the NetBIOS Layer allows multiple transport protocols to be used with that application. The developer needs only to write the application to function with the NetBIOS Layer, not the specific protocol.

**Note**

Microsoft also includes an API called Windows Sockets (also known as WinSockets), which resides between the Application Layer and Transport Layer, like NetBIOS. NetBIOS and WinSocks are not integrated; they are two different paths of communication. WinSocks was created to allow Microsoft TCP/IP to utilize existing Internet utilities (primarily written for UNIX systems), which utilize the Sockets interface.



When a Microsoft network operating system is installed, the setup program requests that a computer name be configured. The computer name that is configured is actually the NetBIOS name. The NetBIOS name must be unique on the network; no other computer should be using the name that is entered. The maximum length of the NetBIOS name is 15 characters, and it cannot contain the following characters or symbols:

- backslash (\)
- space ( )
- hyphen (-)
- single quotation mark (‘)
- at sign (@)
- percent sign (%)
- exclamation point (!)
- ampersand (&)
- period (.)

All Microsoft network operating systems (prior to Windows 2000) that utilize TCP/IP have a NetBIOS computer name and a hostname. The hostname is configured through the properties of the TCP/IP protocol. By default, the hostname and the NetBIOS computer name are identical for the system. You should leave these two names the same; otherwise, an additional level of complexity is added when troubleshooting. For example,

utilities such as `ping` query the hostname, but mapping a drive to a Microsoft server utilizes the NetBIOS name. If the two names are different, troubleshooting a failed connection would be a little more difficult because each name and connection would have to be considered separately.

However, if the computer will be used on the Internet, the TCP/IP hostname has some additional character limitations that should be kept in mind when assigning the NetBIOS computer name. It can only include A–Z, a–z, 0–9, and the – (dash), and the first and last characters of the TCP/IP hostname must be alphanumeric (that is, A–Z, a–z, or 0–9).

On Microsoft operating systems, the NetBIOS name is the computer name, as shown in Figure 7.2. The figure illustrates the NetBIOS name for a Windows 98 computer, which applies identically to a Windows 95 system.

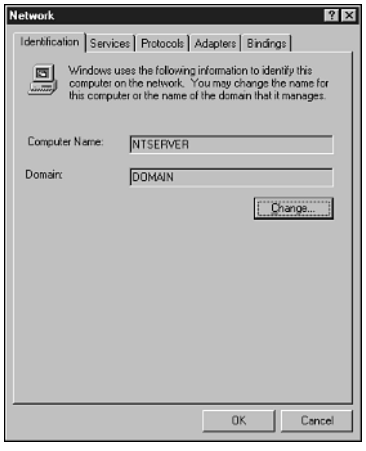



**FIGURE 7.2**
*The NetBIOS name is the computer name.*

To access this configuration dialog box, you can right-click the Windows 95/98 Network Neighborhood icon and select Properties from the resulting context menu. Then, click the Identification tab on the Network dialog box. In Windows NT 4, the same procedure applies except the tab is named General, which appears by default.

The TCP/IP hostname is part of the TCP/IP configuration on Microsoft operating systems, as illustrated in the Microsoft Windows NT screen shot shown in Figure 7.3. The configuration settings for TCP/IP in Windows 95/98 are covered in Chapter 23, "Configuring TCP/IP for Windows 95 and Windows 98." The TCP/IP configuration settings for Windows NT are described in Chapter 25, "Configuring TCP/IP for Windows NT."

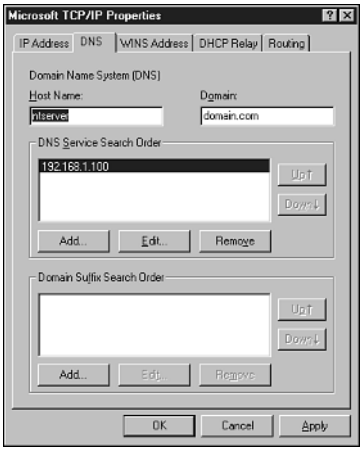



**FIGURE 7.3**
*TCP/IP hostname configuration in Windows NT Server.*



NetBIOS is such an integral part of the Microsoft network configuration that the TCP/IP protocol for Microsoft operating systems is known by the acronyms NBT and NetBT. NBT and NetBT both stand for NetBIOS over TCP/IP. The networking components in a Microsoft operating system will not function correctly if NetBIOS is removed from the system.

# NetBIOS Name Resolution

Just like hostnames, NetBIOS names must be resolved to IP addresses when used on a TCP/IP network. There are several ways in which a NetBIOS name can be resolved to an IP address. Microsoft operating systems use the following methods of resolving NetBIOS names to IP addresses:

- *Name cache*—Microsoft clients maintain a NetBIOS name cache that contains the computer names and IP addresses that the client previously resolved. You can view the contents of the name cache by typing **`NBTSTAT -c`** at the command prompt of a Microsoft networked operating system using TCP/IP.
- *WINS server*—The WINS server maintains a database of computer names to IP addresses. Clients query the WINS server to obtain computer-name-to-IP-address resolution. This is also referred to as a *NetBIOS Name Server* (NBNS).
- *Broadcast*—A Microsoft client can broadcast a name query on the local segment to determine whether a system on that segment owns the computer name the client is attempting to resolve.
- *LMHOSTS file*—A static file that can be posted centrally or on individual systems. This file is a list of IP addresses and their corresponding computer names.

- *HOSTS file*—Another static file that can be posted centrally or on individual systems. This file, maintained in the same format as the 4.2 Berkeley Software Distribution (BSD) `UNIX\etc\hosts` ASCII text file, is a list of Internet hostnames and *Fully Qualified Domain Names* (FQDN) and their corresponding IP addresses.
- *DNS server*—The DNS server can be used by the WINS clients for WINS name resolution. This option, configured on the client system, allows NetBIOS name resolution to be routed through a DNS server, instead of a WINS server.

The order in which the client system attempts to resolve a NetBIOS name is determined by its NetBIOS node type. The four NetBIOS node types are

- *B-node*—This is called a Broadcast node because a computer configured as a B-node only broadcasts on the local segment for NetBIOS name resolution. Actually, Microsoft systems use an enhanced B-node, meaning they will check for an LMHOSTS file entry if the name is not discovered on the local segment.
- *P-node*—Systems configured for point-to-point (P-node) call a WINS server for name resolution, but do not broadcast on the local segment.
- *M-node*—Mixed node (M-node) systems first broadcast on the local segment, and then they contact a WINS server for name resolution.
- *H-node*—Hybrid node (H-node) systems contact WINS servers first for name resolution. If no answer is provided, the H-node system will broadcast on the local segment.

All Microsoft clients first check the name cache before they attempt any other form of NetBIOS name resolution. The default node type for Microsoft operating systems is either B-node or H-node. Enhanced B-node is the default for all systems that are not configured with a WINS server address. H-node is the default for all systems that are configured with the address of a WINS server. Client configuration is explained later in this chapter.

### Modifying Node Type Via the Registry

If you want to modify the NetBIOS node type to point-to-point or mixed, you will have to edit the Windows Registry. For more information on configuring TCP/IP in Windows NT (including modification of the NetBIOS node type), see the Microsoft support Web site at `http://www.microsoft.com` or obtain a copy of the Microsoft TechNet CD. The article that describes Windows 95/98 settings is titled "MS TCP/IP and Windows 95 Networking." The article that describes TCP/IP settings for Windows NT can be found by searching for "Q120642." These articles also describe many other configuration parameters for Microsoft TCP/IP.

There are actually several types of NetBIOS names that each Microsoft operating system may use. These name types are made unique by appending a hexadecimal identifier to the computer name of the system. Each unique name describes a type of service that the computer supports. For example, every Microsoft network operating system has a workstation name, which allows it to communicate on the network. If the operating system also provides services on the network, such as file sharing or printer sharing, it will have a server name as well. Table 7.1 lists the common NetBIOS name types a Microsoft operating system may use.



**TABLE 7.1** NetBIOS Names with Hexadecimal Identifiers

| *NetBIOS Name and Hex ID* | *Description* |
|---|---|
| Computer name[00h] | Workstation service on a WINS client |
| Computer name[03h] | Messenger service on a WINS client |
| Computer name[20h] | Server service on a WINS client |
| User name[03h] | The registered user name of the currently logged-on user |
| Domain name[1Bh] | Indicates the Primary Domain Controller (PDC) functioning as the Domain Master Browser |

As Table 7.1 implies, each Microsoft operating system on the network typically registers multiple NetBIOS computer names on the network. Name registration is handled in one of two ways: If the system is a WINS client, it registers each of its NetBIOS names with the WINS server. If the system is not a WINS client, it broadcasts the names on the local segment. If another system does not respond to the registration broadcast informing the system that those names are already in use, the names are considered registered. If another computer on the local segment was already using any of those NetBIOS names, the system attempting to register them would automatically remove itself from the network by shutting down its networking services. Then, it would be up to the system administrator to configure a different NetBIOS name for one of these systems.

# Dynamic NetBIOS Name Resolution

WINS is the only dynamic method of resolving NetBIOS names to IP addresses on Microsoft TCP/IP networks. A WINS server receives name registration requests from WINS clients when they start up. When WINS clients shut down, they send a name release to the WINS server. This way, the WINS database is kept current with the names of computers active on the network.

WINS is a server service that is run on Windows NT servers. The service uses an Access database to store the name registration information and provide name resolution services.

# Benefits of Using WINS

There are several ways in which a WINS server improves NetBIOS name resolution on a Microsoft network. The following list describes the benefits of using WINS:

- WINS reduces name resolution broadcast traffic. WINS clients contact the WINS server directly instead of broadcasting for name resolution on the local segment.
- WINS clients can reach WINS servers on remote segments. Routers typically filter broadcast traffic, so broadcasts for name resolution are limited to the local subnet. WINS clients contact the WINS server via a directed request to the WINS server IP address. Although allowing a router to forward name resolution broadcasts by enabling UDP ports 137 and 138 is possible, it is not recommended because of the resulting increase in network traffic.
- WINS is dynamic. WINS clients register with the WINS server when they start up. An administrator does not have to type in the IP address to NetBIOS name resolutions.
- WINS offers better browse list maintenance. Microsoft systems maintain large browse lists, which are lists of resources available on the network such as shared files and printers. The WINS server helps to document the available resources on the network by collecting the various NetBIOS names from each computer. If the network relied only on broadcast traffic, the browse lists would be incomplete on multisegment networks.

# How WINS Works

A WINS server gathers and maintains a central database of the NetBIOS names that are active on the network. The WINS server handles name registration, name release, and name resolution requests from WINS clients. As previously stated, the WINS server relies on the WINS clients to register their names with the WINS server when those clients start up. The clients are also expected to release their names when they shut down. Because all WINS clients register and release their names through the WINS server, the WINS server can accurately provide IP address to NetBIOS name resolution when clients request it.

## Name Registration

WINS clients are required to register their NetBIOS names with the WINS server when they start up. The name registration request is sent directly to the WINS server, and the WINS server either registers the name or refuses the name registration.

If the name is not in use by another system, the WINS server will register the name and send an acknowledgment packet to the client system. The acknowledgment packet contains the registered name and the *Time To Live* (TTL) for that name. The TTL is the amount of time that the name will be reserved for that client until that name is either reregistered or released. If the WINS client does not renew the name before the TTL expires, the name will be removed from the WINS database and made available for another system to register. The TTL ensures that systems that register names and then shut down abnormally, or lose power, do not occupy NetBIOS names when they are no longer on the network.

If a WINS client attempts to register a NetBIOS name that is already in use in the WINS database, the WINS server returns a wait for acknowledgment packet to the client. The WINS server then verifies that the name is actually in use on the network by sending a challenge packet to the listed owner of that computer name. If the WINS server receives a response from the system that originally registered the name, the WINS server denies the second computer's request to register that name. Should the previously registered owner of the name not respond, the WINS server queries it two more times. If the previous owner does not respond to all queries, the WINS server releases the name and gives it to the requesting computer.



**Note**

If the WINS client does not receive a response from its primary WINS server during the startup process after three tries, one of the alternate or secondary WINS servers will be contacted. If the primary and secondary WINS servers fail to respond, the WINS client uses a broadcast to register its NetBIOS name.

## Name Renewal

The TTL is used by the WINS client to determine when the NetBIOS name should be renewed with the WINS server. The default TTL for a WINS name is six days, or 518,400 seconds. WINS clients attempt to renew their name after one-eighth of the TTL has expired. If the WINS server cannot be contacted at that time, the client will attempt to contact a secondary or alternate WINS server to renew the name. After the name has been renewed, the client will wait until half the TTL has passed before attempting to renew the name again. If the client is unsuccessful in all of its attempts to renew the name, the NetBIOS name will be released.

## Name Query and Response

As explained earlier, WINS clients default to the Hybrid NetBIOS node type, which means these clients query the WINS server when they need name resolution services. For example, if a WINS client is attempting to contact the computer named HOSTX, the WINS client first checks its name cache for an existing name resolution. If a mapping is not found in the local name cache, the WINS client contacts the WINS server to determine the IP address for HOSTX. If the primary WINS server cannot be located after three repeated requests, the WINS client attempts to call a secondary WINS server. If all WINS servers are unavailable, or any responding WINS server does not have a mapping for the NetBIOS name, the WINS client will broadcast for name resolution on the local subnet.

## Name Release

WINS clients are required to provide name release requests when they shut down. These name release requests are sent directly to the WINS server for each name the client has registered. The name release packet includes the NetBIOS name and corresponding IP address that is being released.

The server responds to the WINS client's name release request with a name release request response packet. This response packet can either be a positive or negative acknowledgment. If the WINS server finds a conflict in the database, such as the computer attempting to release the name is not the computer that registered the name, then the response packet will be negative. This does not make any difference to the WINS client that is shutting down because it ignores the contents of the release response packet and shuts down whether the response is negative or positive.

# Configuring WINS Clients

Configuring a Windows NT or Windows 95 computer to use WINS is a matter of entering the IP address of the WINS server. Both operating systems can be configured for WINS via the TCP/IP protocol properties dialog box (using the WINS Configuration tab in Windows 95 and Windows 98 and the WINS tab in Windows NT). Figure 7.4 illustrates the WINS configuration for a Windows 98 system.

To enable WINS resolution on the client, you need only enter the IP address of a single WINS server. If you want the client to have access to multiple WINS servers, you can enter additional IP addresses into the configuration dialog box. On Windows NT systems you can enter a primary and secondary WINS server only through the interface. By default, the secondary WINS server is only contacted when the client cannot find the primary WINS server on the network.

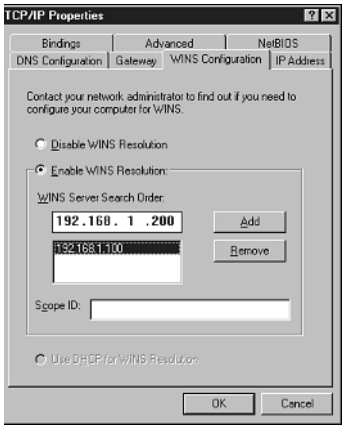



**FIGURE 7.4**
*Configuring WINS in Windows 98.*



The following operating systems can be configured as WINS clients:

- Windows NT Server 4.0, 3.5x
- Windows NT Workstation 4.0. 3.5x
- Windows 95/98
- Windows for Workgroups 3.11 with Microsoft TCP/IP-32
- Microsoft Network Client 3.0 for MS-DOS
- LAN Manager 2.2c for MS-DOS

As previously described, configuring a client with a WINS address automatically changes the node type from B-node to H-node. You can confirm this change by typing **`IPCONFIG /ALL`** at the command prompt of a Windows NT system. To check the IP configuration of a Windows 95/98 system, you must type **`WINIPCFG`** at the command prompt. To see the node type, click the More Info button on the resulting dialog box.

## Configuring WINS for Proxy Agents

Windows 95, Windows 98, and Windows NT can be configured to provide WINS proxy services. When a WINS client is configured to provide WINS proxy services, it is called a WINS Proxy Agent. The WINS Proxy Agent contacts the WINS server for name resolution on behalf of non-WINS clients on the local segment. This allows non-WINS clients to benefit from WINS NetBIOS name resolution services.

The process is not much different from the normal WINS name resolution process. When a non-WINS client broadcasts for name resolution on the local subnet, the WINS Proxy Agent receives the request and forwards it to the WINS server. If the WINS server is able to provide a mapping for the NetBIOS name to IP address, the WINS Proxy Agent forwards the answer to a non-WINS client.

One important item that you should keep in mind when configuring systems to be WINS Proxy Agents is that they increase network traffic because they repeat name resolution requests. Microsoft recommends that you have no more than two WINS Proxy Agents per subnet where non-WINS clients exist. If all the computers on a network were configured as WINS Proxy Agents, a single broadcast for name resolution would be repeated once for every computer on the network.

## Configuring NT 4 Systems

To configure a Windows NT 4 system to be a WINS Proxy Agent, you must complete the following steps:

1. Open `REGEDIT.EXE` or `REGEDT32.EXE` to edit the Windows NT 4 Registry.
2. Locate the Parameters sub-key in the following path: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NetBT`.
3. Double-click the EnableProxy key and set its value to `1`.
4. Close the Registry Editor.
5. Restart the computer.

## Configuring Windows 95 and Windows 98 Systems

To configure a Windows 95 or Windows 98 computer to be a WINS Proxy Agent, do the following:

1. Open `REGEDIT.EXE` to edit the Windows 95/98 Registry.
2. Locate the MSTCP sub-key in the following path: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\VxD`.
3. Highlight the MSTCP sub-key by clicking it once.
4. Select Edit, New, and String Value from the Registry Editor toolbar.
5. Type **`EnableProxy`** as the new value.
6. Double-click the new EnableProxy string value to open the Edit String dialog box.
7. Enter **`1`** as the Value data and click OK.
8. Close the Registry Editor and reboot the system.

# Installing a WINS Server

The WINS server service can be installed on a Windows NT server. After it is installed, the Windows NT server becomes a WINS server. The WINS server service software ships as part of the Windows NT Server software, but is not part of the default NT Server installation.

To install the WINS server service on a Windows NT 4 system:

1. Right-click the Network Neighborhood icon and select Properties from the resulting context menu. The Network dialog box opens.
2. Click the Services tab.
3. Click the Add button.
4. From the list of services, select the Windows Internet Name Service and click OK.
5. You are asked for the path to the Windows NT source files. Ensure that the path is correct, and then click OK.
6. When the service is done installing, click Close and restart the computer as requested.



After the system restarts:

1. Confirm installation by opening the Control Panel (click Start, choose Settings, then choose Control Panel).
2. Double-click the Services icon in the Control Panel to view a list of installed services.

If the Windows Internet Name Service was successfully installed, it will be part of this list. The Windows Internet Name Service should be near the end of the list of installed services. From this interface, you can stop, start, pause, and configure the startup configuration for the WINS service and many others.

# WINS Administration and Maintenance

After the WINS server service is installed, the WINS Manager application should appear under the Administrative Tools section of the Windows NT Start menu. Click Start, Programs, Administrative Tools, and then select WINS Manager.

The WINS Manager application allows you to perform several administrative tasks on the WINS server. For example, you can add static mappings to the WINS database for non-WINS clients. You can also back up the WINS database, configure WINS replication partners, and view the WINS database from the WINS Manager application.

## Adding Static Entries

If you have non-WINS clients on your network and you want the WINS server to be able to resolve the non-WINS client's computer names to IP addresses, you can configure static mappings. A static mapping is one that you add to the WINS database manually to allow WINS clients to resolve the IP address to the NetBIOS name of non-WINS clients. To add a static mapping, follow these steps:

1. Open WINS Manager.
2. Click the Mappings menu and select Static Mappings.
3. Click Add Mappings.
4. In the Add Static Mappings dialog box, type the Name and IP address for the computer you want to statically add to the database.
5. Choose the correct name type for the computer. You can choose from the following options:

   *Unique*—Used for a single computer name to IP address mapping.

   *Group*—Used as a "Normal Group" name. The IP address of individual members of a group is not stored because broadcast name packets are used to communicate with group members.

   *Domain Name*—Used for entering a Domain Name. This option is for listing Windows NT domain controllers.

   *Internet Group*—Enables you to group resources, such as printers, for browsing purposes.

   *Multihomed*—Used to support computers that have multiple IP addresses, but only one computer name. Typically, these systems have multiple network cards, but use a single computer name. This entry supports such a configuration by mapping multiple IP addresses to the same NetBIOS computer name.
6. Click Add.
7. Repeat the process for each additional entry you require. When you are finished, click Close.

Because the Domain Name, Internet Group, and Multihomed entries allow for more than one entry, an additional dialog box appears when these categories are selected. All of

these options allow for a maximum of 25 total entries each. WINS client computers will be unable to register names that have been statically configured on the WINS server.

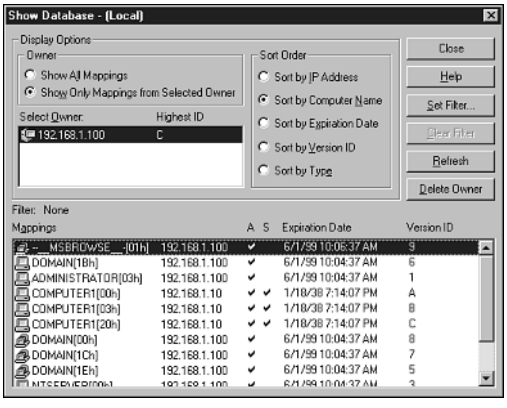
# Maintaining the WINS Database

Several maintenance tasks can be performed on the WINS database; these tasks include backing up the database, restoring the database, compressing the database, and reconciling database records.

You can view the records of the WINS database from the WINS Manager. To view the WINS database records, click the Mappings menu and select Show Database. You have the option of viewing all database entries or just those of the selected owner. An owner is a WINS server. When multiple WINS servers are connected as replication partners, each partner can view the database records of the other. The configuration of WINS replication partners is detailed later in this chapter. You can also sort the database entries by IP Address, Computer Name, Expiration Date, Version ID, or Type (see Figure 7.5).



**FIGURE 7.5**
*The WINS database.*




In the Mappings window, you can determine whether the entries are active or static. If an entry is static, then the S column has a check mark in it. Likewise, if an entry is active, then the A column has a check mark in it. Entries that do not have check marks for active or static cannot be verified and are awaiting removal.

## Initiating Scavenging

When the database records seem to be incorrect, you can try to get the WINS server to reconcile the entries. For example, if you notice that computer names no longer on the network remain in the WINS database, you can initiate scavenging through the WINS server. After you initiate scavenging, the database should be cleared of names that were

released or entries that were added by other WINS servers. If you notice that several entries need to be removed prior to the automatic scavenging of the database, you might want to initiate scavenging via the Mappings menu in the WINS manager.

## Purging the Database

Sometimes the WINS database maintains entries that should have been purged. The database is supposed to remove entries that have expired, but occasionally an entry will remain in the database after it should have been removed. If you have tried initiating scavenging and still see entries that should be removed, there are a couple of ways to purge database entries manually. You can purge the entire database by selecting Delete Owner from the Show Database dialog box. Follow these steps:

1. From the WINS Manager application, click Mappings, and then click Show Database. The Show Database dialog box appears.
2. From the Show Database dialog box, select the Owner window, then click the name or IP address of the WINS computer that you want to purge.
3. Click the Delete Owner button.
4. Confirm this action by clicking Yes in the WINS Manager warning dialog box. The message tells you that entries cannot be reconciled by this WINS database until those entries are rebuilt.

Of course, purging the entire database removes all the entries in the database. After the database is purged, each WINS client must re-register its name with the WINS database. This is done automatically if the computers on the network are rebooted.

Another way to clear entries from the WINS database is to remove the specific records individually. The `WINSCL.EXE` utility allows you to remove specific, individual records from the WINS database. You can obtain this utility from the Windows NT 4.0 Resource Kit, which is sold in bookstores. You can also purchase the Windows NT 4.0 Resource Kit CD directly from Microsoft.

## Controlling Automatic Intervals

The WINS database automatically performs many database maintenance tasks at set intervals. You can modify the interval for these automated activities via the WINS Server Configuration dialog box. To access this dialog box, click the Server menu in the WINS Manager application, and then select Configuration (see Figure 7.6).

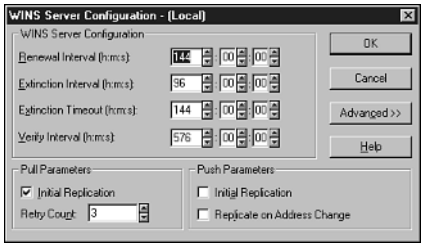



**FIGURE 7.6**
*The WINS Server Configuration dialog box.*

You can manage the following intervals via the WINS Server Configuration section:

- *Renewal Interval*—The time-to-live for a registered name. A WINS client must renew its name before this time interval expires. By default, the entry is set at 144 hours (six days).
- *Extinction Interval*—The time span (default 144 hours) between the release of a name and the expiration of that name. When this time period has elapsed, the computer name is marked as *extinct*.
- *Extinction Timeout*—The time between the marking of an entry as *extinct* and its actual removal from the database. The extinction timeout is 144 hours by default.
- *Verify Interval*—Entries that came from other WINS servers are verified every 576 hours (24 days) by default. Only the entries from other WINS servers are verified in this manner.

## Advanced WINS Server Configuration

If you click the Advanced button on the WINS Server Configuration dialog box, you can access the Advanced WINS Server Configuration dialog box. In this dialog box, you can control WINS server logging, backup, and other options. The following options are available:

- *Logging Enabled*—A log of database changes is kept in the `%system_root%/system32/wins` folder by default. This file is used by WINS to recover transactions after a failure.
- *Log Detailed Events*—Causes WINS to log every WINS action in detail. If you are experiencing difficulty with WINS server registration, you might want to enable this option. Because this option slows down the WINS server, it should only be enabled when you are troubleshooting registration problems. Detailed events can be found in the System Log of the Event Viewer application.
- *Replicate Only With Partners*—By default the WINS server does not replicate with other WINS servers that are not listed as replication partners. If this option is deselected, the WINS server can be configured to push or pull entries to or from an unlisted WINS server.

- *Backup On Termination*—Backs up the WINS database each time the WINS Manager application is closed.
- *Migrate On/Off*—Allows WINS clients to overwrite static entries. This option is used when static entries have been configured for non-WINS clients and those clients are then made WINS clients. This option allows the new WINS clients to dynamically register their computer names (overwriting their existing static entries). Select this option when upgrading non-Windows NT computer systems to Windows NT.
- *Starting Version Count*—The value used to track the version of the latest database. If the WINS database becomes corrupt beyond repair, you can set this version number to zero to refresh the database.
- *Database Backup Path*—The location for database backups. It is used for restoration in the event of a WINS or system failure. One caveat: Do not specify a network directory to back up the WINS database to; in the event of a network outage, you may be unable to access or restore the WINS database.

## Backing Up the WINS Database

Every 24 hours the WINS database is automatically backed up. However, you must specify a backup directory for this action to occur. Follow these steps:

1. Click the Mappings menu and select Back Up Database.
2. Type (or select by browsing) the location for the backup directory.
3. Click OK.

## Backing Up the WINS Registry Entries

As part of routine WINS maintenance, you should also plan on backing up the Registry Entries for the WINS server by following these steps:

1. Open `REGEDIT.EXE` or `REGEDT32.EXE` to edit the Windows NT 4 Registry.
2. Locate the WINS key in the following path:
   `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\`.
3. Click the Registry menu and select Save Key. The Save Key dialog box appears.
4. Type (or select by browsing) the location for the backup directory or use the same backup directory from the previous section and click Save.
5. Close the Registry Editor.

## Restoring the WINS Database

If you find that WINS is no longer functioning properly, suspect that the WINS database is corrupt. WINS automatically restores the database if it determines that the database is corrupt. You can stop and restart the WINS service in hopes that the corruption will be detected by following these steps:

1. Open the Control Panel (click Start, choose Settings, then choose Control Panel).
2. Double-click the Services icon.
3. Highlight the Windows Internet Name Service from the list of services.
4. Click Stop.
5. Click Start.

If WINS detects the database corruption and restores the database, you can look in the Event Viewer application to confirm the restoration. If WINS does not detect the corruption and restore the database after a restart, you can restore the database manually. To do so, follow these steps:

1. Click the Mappings menu and select Restore Database.
2. Type (or select) the location of the backup.
3. Click OK.

If the option to restore the database is grayed out, the WINS database has never been backed up. In this case, you would have to delete the WINS database (Delete Owner) as described earlier and rebuild it. To rebuild a deleted database, all clients must reregister with WINS. This happens automatically when the client systems are rebooted.

## Compressing the WINS Database

Whenever the WINS database (`wins.mdb`) reaches a size of 30MB or more, you should compress it. To compress the WINS database manually use the `JETPACK.EXE` utility from a command prompt. The WINS database is located in the `C:\WINNT\SYSTEM32\WINS` directory by default. Be sure to stop the WINS service before compacting the database, and restart the service when it is complete. Here is the command-line structure for compacting the WINS database:

```
jetpack wins.mdb temporary_name.mdb
```

You can perform the entire process from the command line. The full command syntax to stop the WINS service, compact the database, and restart the WINS service is as follows:

```
net stop wins
Jetpack wins.mdb temp1.mdb
net start wins
```

You can type any temporary name for the temporary database file. The temporary file is automatically deleted by the system when the compression is complete.

## WINS Replication Partners

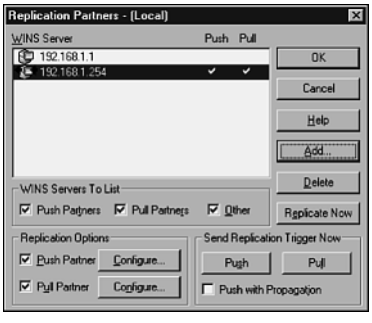
On networks with multiple WINS servers, having the WINS servers share their databases with one another is often wise. Sharing the databases ensures maximum name resolution capabilities for each WINS server because they have access to the mappings of all WINS servers. WINS servers that share their databases with each other are called *replication partners*.

The replication process is done as a push or pull operation. When the replication is configured as a push, the replication is done whenever a certain number (set by the administrator) of changes to the database have occurred. Pull replication is configured on a schedule, which is also set by the administrator.

Typically, only pull replication is configured on slow wide area network links because it can be scheduled around peak usage periods. Push and pull replications are typically configured for local area networks where connections are usually fast. The push ensures that too many changes do not occur to the database before an update is issued. In addition, the pull system ensures updates are issued on a regular basis no matter how few changes have occurred.

To configure database replication, open the WINS Manager application, click the Server menu, and then choose Replication Partners (see Figure 7.7).

**FIGURE 7.7**
*WINS replication partners.*




The Replication Partner dialog box allows you to configure pull and/or push replication partners. The Replicate Now button starts both push and pull replication. The Push with Propagation option sends changes to replication partners and triggers a chain reaction, which not only updates the replication partner, but also any and all replication partners configured under that partner.

## WINS Implementation Recommendations

Although only one WINS server is required to implement WINS on the network, Microsoft recommends that you implement a minimum of two WINS servers. The second WINS server provides a level of fault tolerance in the event that the first WINS server fails. In addition to fault tolerance, a second WINS server could provide some load balancing. To ensure that the load is equally balanced between the WINS servers, configure half of the clients to access one WINS server primarily and the other half to access the other WINS server primarily. Then, configure the opposite WINS server as a secondary or alternate for each WINS client to get the benefits of fault tolerance. In addition, be sure that the WINS servers act as replication partners.

In order to handle the load properly, you should add additional WINS servers for every 10,000 WINS clients served. If you want to increase the performance of a single WINS server, you can turn off logging. If the WINS server crashes, you will lose recent changes to the database, but the server performs better with logging off. If you want to increase performance by adding hardware to the WINS server, consider using a multiprocessor system. Microsoft states that WINS servers perform up to 25% better with each additional processor.



# Integrating WINS and DNS Name Resolution Services

In addition to using a WINS server, the Microsoft clients can contact a DNS server for NetBIOS name resolution. Windows NT 4.0 systems have an Enable DNS for Windows Resolution check box (see Figure 7.8) to allow you to enable this feature. If this option is enabled, the WINS client calls the DNS server to resolve NetBIOS names after failing to resolve the name via the WINS server, broadcast, or LMHOSTS/HOSTS file.

Microsoft DNS servers can also be configured to use WINS to help with name resolution (see Figure 7.9). The Microsoft DNS server only calls the WINS server to resolve the host part of a computer on the network. This means that the DNS server must be able to resolve the domain name and any sub-domain names before contacting the WINS server.

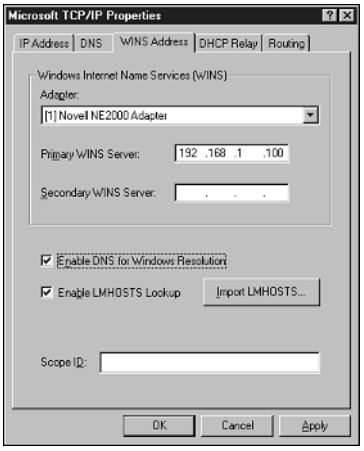



**FIGURE 7.8**
*Enabling the WINS client to contact the DNS server.*

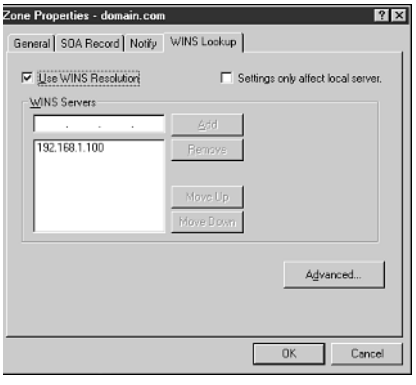



**FIGURE 7.9**
*Enabling the DNS server to contact the WINS server.*

# DHCP Serving WINS Options

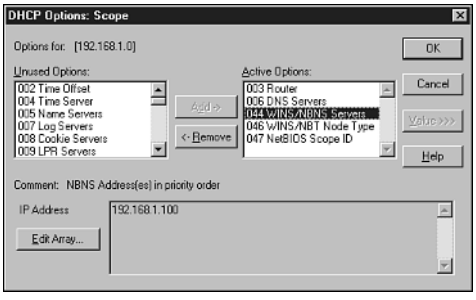
A DHCP server can be configured to provide more than just an IP address and subnet mask to a DHCP client. The DHCP client can also receive WINS options from the DHCP server. Specifically, the DHCP client can be configured with a primary and secondary WINS server address and a NetBIOS node type.

The four types of NetBIOS nodes were described earlier in this chapter. Remember, the NetBIOS node types are the B-node, P-node, M-node, and H-node options. Figure 7.10 shows the configuration dialog box for implementing these options on a Microsoft DHCP server. You will learn more about configuring DHCP in Chapter 8, "Automatic Configuration."

**NOTE**

IP addresses configured in primary and secondary WINS servers take priority over corresponding parameters previously configured via DHCP.

**FIGURE 7.10**
*DHCP servers can provide WINS configuration options.*



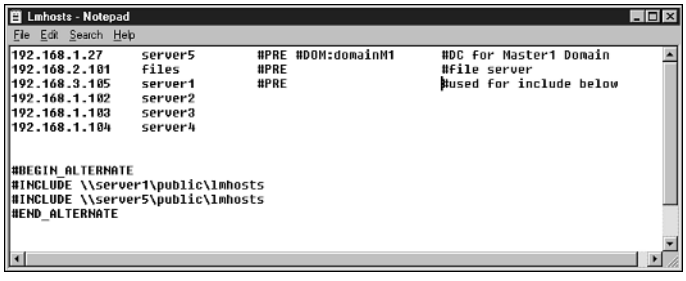



# NetBIOS Name Resolution Via LMHOSTS

The LMHOSTS file can also perform NetBIOS name to IP address resolution. However, the LMHOSTS file must be manually created and maintained. Essentially, it is a static mapping file that matches computer names to IP addresses. Microsoft has included sample LMHOSTS files in the Windows NT and Windows 95/98 operating systems. You can find the `LMHOSTS.SAM` file under the `C:\WINDOWS` directory in Windows 95/98 operating systems. In Windows NT operating systems, the `LMHOSTS.SAM` is in the `C:\WINNT\SYSTEM32\DRIVERS\ETC` directory.

The `LMHOSTS.SAM` file describes in detail how to create an LMHOSTS file. However, in order for the LMHOSTS file to be active it must have the name LMHOSTS without a file extension. Figure 7.11 illustrates a sample LMHOSTS file.

**FIGURE 7.11**
*LMHOSTS file example.*



In addition to mapping IP addresses to NetBIOS computer names, the LMHOSTS file has special predefined tags, which allow for added functionality. For instance, an entry in the LMHOSTS file followed by a #PRE tag is placed in the NetBIOS name cache when the computer starts up. This increases the speed of name resolution because Microsoft clients always check the name cache before attempting any other type of name resolution.

Table 7.2 lists the special tags that can be set in an LMHOSTS file.

**TABLE 7.2** LMHOSTS Tags and Descriptions

| *LMHOSTS Tag* | *Description* |
|---|---|
| #PRE | Entries with a #PRE tag are preloaded into the NetBIOS name cache. |
| #DOM:domain_name | #DOM identifies the domain name. This entry is essential for domain controller activity such as database synchronization and browsing. |
| #INCLUDE<*filename*> | #INCLUDE<*filename*> sets alternate files to be used for name resolution. These files can be the LMHOSTS files on remote computers and can be referenced with a UNC path. If you use a remote computer's LMHOSTS file, you must have already provided a mapping to that computer for this entry to work. |
| #BEGIN_ALTERNATE<br>#END_ALTERNATE | #BEGIN_ALTERNATE and #END_ALTERNATE allow multiple #INCLUDE entries to exist in an LMHOSTS file. This enables the LMHOSTS file to parse multiple remote LMHOSTS files as if they were local entries. |
| #MH | #MH is for multihomed computers that may have multiple IP addresses mapping to a single computer name. |
| #comment | Comments can be placed in the LMHOSTS file after any entry or as a standalone line. |

The #PRE tag is probably the most significant LMHOSTS file tag because it can have the greatest impact on the speed of name resolution on the network. The second most important tag is #DOM, which identifies the domain controllers. Implementing the #PRE and #DOM entries in the LMHOSTS file on all domain controllers improves the performance of administrative communications between domain controllers. In addition, if client computers have LMHOSTS files that add the domain controller's IP addresses into their name cache, the login process is faster because the client system already has a mapping for the domain controller's IP address.

**Note**

The HOSTS file and LMHOSTS file are very similar. The focus of the LMHOSTS file is NetBIOS name resolution and the focus of the HOSTS file is hostname resolution, as explained in Chapter 6, "DNS: Name Services."

# Summary



NetBIOS names are used in Microsoft networks prior to Windows 2000. On a TCP/IP network, these NetBIOS names, like hostnames, must be resolved to IP addresses in order for communication to take place between two computers. There are a few different methods for resolving IP addresses to NetBIOS names, but the only dynamic method is to use WINS.

WINS functions as a client/server application built in to Microsoft operating systems. The WINS client registers its NetBIOS name and IP address with the WINS server when it starts up. A WINS client releases its name from the WINS server when the WINS client shuts down. This allows the WINS server to maintain a current database of NetBIOS names in use on the network. When WINS clients want to contact other computers on the network, they call the WINS server for NetBIOS name to IP address resolution.

In order for a client system to become a WINS client, the IP address of at least one WINS server must be entered for the client. The entry is made in the TCP/IP configuration dialog box for the WINS client. The IP addresses of multiple WINS servers can be configured on a WINS client system to provide fault tolerance. The WINS client will contact an alternate or secondary WINS server when the primary WINS server is unavailable.

Non-WINS clients can also take advantage of WINS services. WINS Proxy Agents help non-WINS clients to utilize WINS services. WINS Proxy Agents take local subnet name resolution broadcasts and forward them to the WINS server. WINS Proxy Agents return the WINS server response to the non-WINS clients. This means that non-WINS clients can benefit from WINS server name resolution via a WINS Proxy Agent.

Many maintenance and administrative tasks are automatically accomplished by a WINS server. However, you must set the WINS server to perform backups via the WINS Manager application. The WINS server can automatically restore a corrupted database (if there is a backup) after it is restarted. However, database restoration can be done manually through the WINS Manager application. The WINS server database can be compressed through the command-line utility `JETPACK.EXE`.

WINS replication partners are WINS servers that are configured to exchange WINS database information. WINS replication partners can be push partners and/or pull partners. Pull partners are best for slow WAN links. On fast LAN connections you can enable both push and pull to ensure that your WINS databases have the most current information.

LMHOSTS files can help with NetBIOS name resolution in several ways. Although they are not dynamic, like WINS, and must be manually maintained, they do provide some benefits. Entries in the LMHOSTS files can be added directly to the NetBIOS name cache by tagging them with `#PRE`. In addition, domain controllers can be identified in the LMHOSTS file by the tag `#DOM`.

CHAPTER 8

# Automatic Configuration

*by Karanjit S. Siyan, Ph.D.*

## In This Chapter

Configuring TCP/IP parameters for each workstation on a large network can be a difficult and time-consuming task, particularly when the TCP/IP parameters, such as IP addresses and subnet masks, need to be changed. The changes can occur because of a major restructuring of the network or because the network has a large number of mobile users with portable computers that can be connected to any of the network segments. The network connections can be direct physical connections or wireless connections. Because the TCP/IP parameters for computers depend on the network segment they connect to, appropriate values must be set up whenever a computer is connected to a different network segment.

Understanding the consequences of TCP/IP parameter changes requires knowledgeable network administrators. Several auto-configuration protocols, such as Boot Protocol (BOOTP) and Dynamic Host Configuration Protocol (DHCP), have been developed by the Internet Engineering Task Force (IETF) for TCP/IP internetworks.

# Dynamic Configuration Using BOOTP

BOOTP is designed to use User Datagram Protocol (UDP) and Internet Protocol (IP) to carry configuration information for computers wanting to configure themselves. Computers making the request are called *BOOTP clients*, and computers that respond to the BOOTP client requests are called *BOOTP servers* (see Figure 8.1).

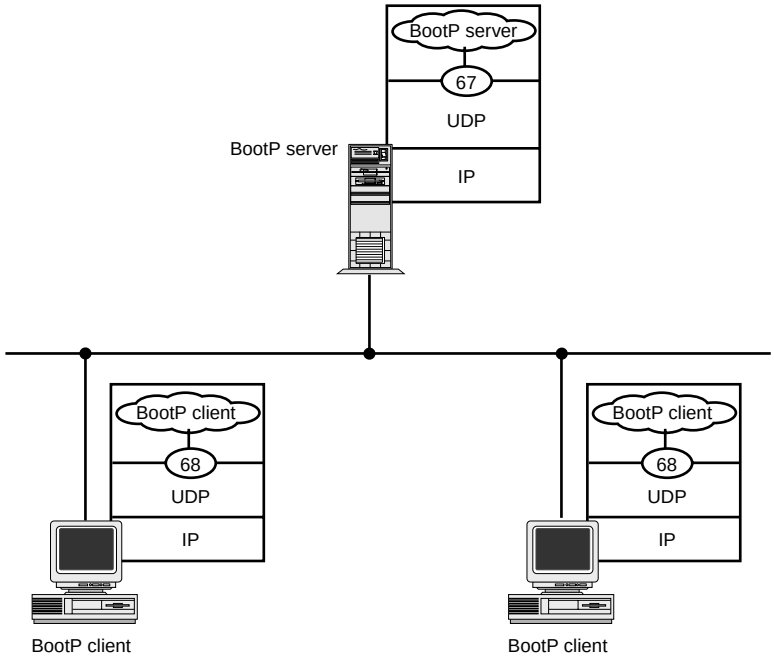
The purpose of the BOOTP client request is to discover the IP parameter settings for the computer running the BOOTP client.

BOOTP is discussed in RFC 951 on the "Bootstrap Protocol" and is updated by RFC 2132, RFC 1532, RFC 1542, and RFC 1395. The following sections describe the operation of BOOTP.

## BOOTP Request/Reply IP Addresses

The BOOTP messages are encapsulated in a UDP header that identifies the port number used by the BOOTP processes. The UDP datagram is encapsulated in an IP header. What source and destination IP address values are used in the IP header? This is an interesting question because when the BOOTP client makes its request, it must use a source and destination IP address. Typically, the BOOTP client does not know its IP address; it uses a source IP address value of 0.0.0.0. If the BOOTP client knows its IP address, it uses this address in the BOOTP client request.

**FIGURE 8.1**
*BOOTP clients and servers.*




Does the BOOTP client know the IP address of the BOOTP server? Typically, no. This is especially true in situations where the BOOTP client is a diskless workstation and has no way of knowing the BOOTP server address. In situations where the workstation can be configured with the BOOTP server address, the BOOTP client can use the BOOTP server address.

When the BOOTP client does not recognize the BOOTP server's IP address, it uses a limited IP broadcast. The IP limited broadcast has the following value:

255.255.255.255

**Note**

The limited or local broadcast address is received by all IP nodes including BOOTP servers on the local network.

What if the BOOTP server is on a different IP subnet, beyond a router boundary? A limited broadcast cannot cross a router boundary. In this case, the router must be configured with a BOOTP relay agent that checks to see whether the UDP destination port number of 67 is being used in the limited broadcast datagram. (UDP port 67 is reserved for BOOTP/DHCP.) If this is the case, the limited broadcast is forwarded to the router's network interfaces (see Figure 8.2).

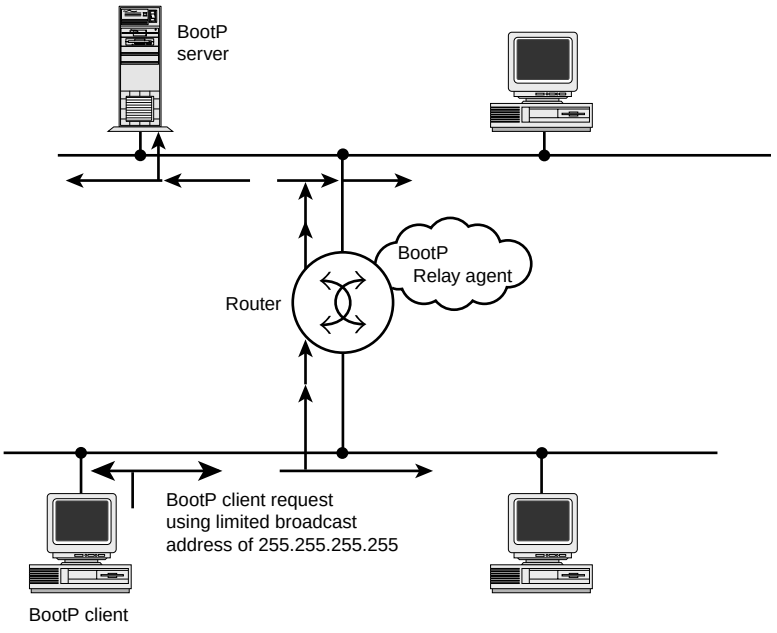



**FIGURE 8.2**
*Forwarding of BOOTP messages across a router boundary.*

When the BOOTP server is on the local network, it can respond to the BOOTP client request message. Does the BOOTP reply message from the BOOTP server use the IP address of the client? In other words, can the BOOTP server send a directed reply? The answer depends on the BOOTP server implementation.

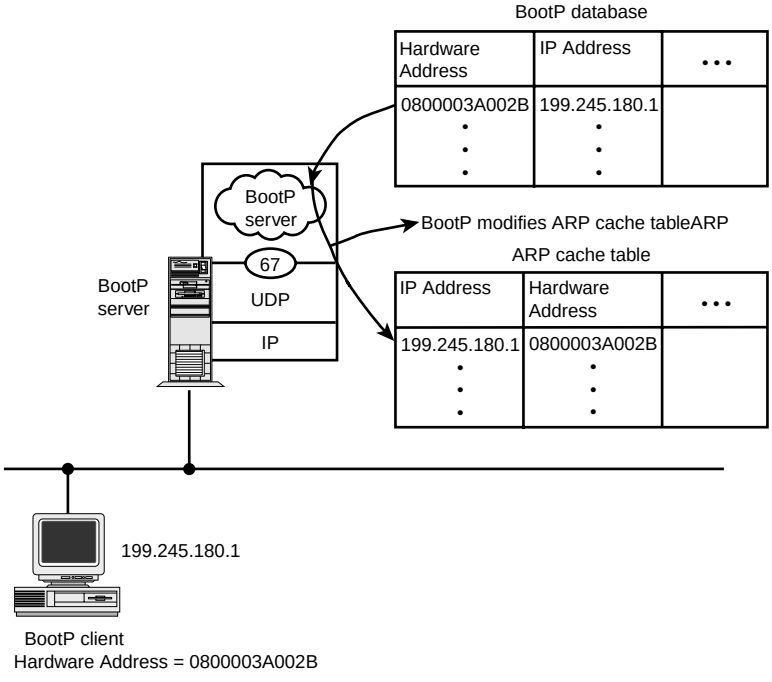
A problem is associated with sending a directed BOOTP reply to the BOOTP client. The BOOTP server knows the BOOTP client's IP address by looking it up in its BOOTP configuration database. Why can't the BOOTP server use this IP address to send a directed reply to the BOOTP client? The answer to this question lies in the fact that when the BOOTP server sends an ARP request message to discover the hardware address of the BOOTP client, the BOOTP client is unable to respond. Why? The answer is because the BOOTP client does not know its IP address yet. The BOOTP server

knows the hardware address of the BOOTP client from the Data Link Layer header in the BOOTP client message. A BOOTP server implementation may add an entry in the ARP cache table for the BOOTP client (see Figure 8.3). If it does, the BOOTP server can use a directed reply.

**Note**

The BSD Unix BOOTP implementation uses the ARP modification technique.

**FIGURE 8.3**
*Adjusting of the ARP cache table by a BOOTP server.*




If a BOOTP server does not adjust the ARP cache table, it must broadcast its reply.

## Handling the Loss of BOOTP Messages

BOOTP uses UDP and IP protocols. These protocols cannot guarantee delivery of BOOTP messages. As a result BOOTP messages can become lost, delayed, duplicated, or delivered out of sequence. Because IP provides only a checksum to guard against errors in the IP header and not the data field, a BOOTP implementation may require that the UDP checksum be enabled. The UDP checksum can guard against error in the entire BOOTP message.

A BOOTP client sends requests using the IP Don't Fragment (DF) flag set to 1 to simplify the processing of BOOTP replies, and in case it does not have enough memory to reassemble fragmented datagrams.

BOOTP uses a timeout and retransmission policy to handle message loss. When a BOOTP client sends its request, it also starts a timer. A BOOTP reply within the timeout interval cancels the timer, but if the timer expires before a BOOTP reply is received, the BOOTP retransmits its request.

When BOOTP clients start at the same time, they can flood the network with BOOTP broadcast requests. To prevent this from occurring, the BOOTP specification recommends using a random delay starting with a random timeout interval from zero to four seconds. Successive retransmissions cause the doubling of the random timeout interval until a large value, such as 60 seconds, is reached. When this upper limit is reached, the random timeout interval is not doubled, but random time delays within this interval are still used. Randomization is used to avoid simultaneous BOOTP client requests that could overload the server and cause packet collisions in an ethernet network. Doubling the random timeout interval avoids overloading the network with too many BOOTP client broadcast requests.

## BOOTP Message Format

Figure 8.4 shows the BOOTP message format, and Table 8.1 discusses the meanings of the fields in the BOOT message.

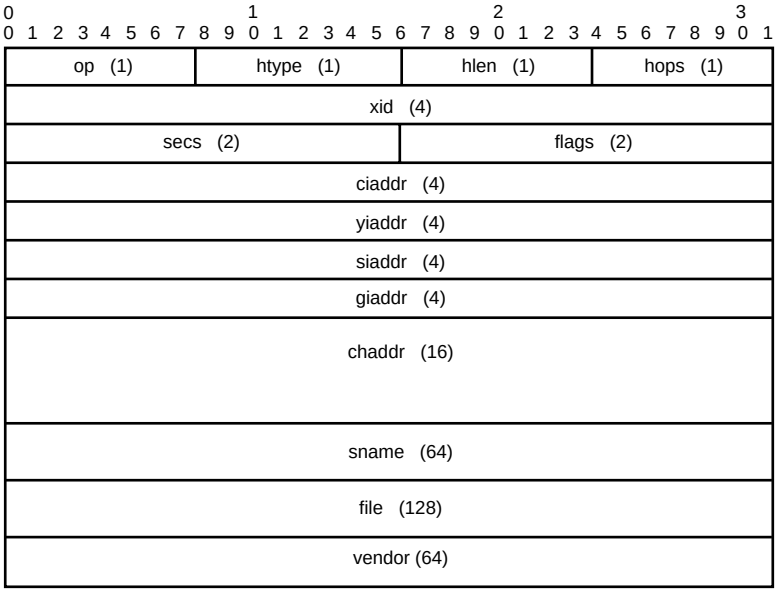



**FIGURE 8.4**
*BOOTP message format.*

**TABLE 8.1** DHCP Fields

| *Field* | *Octets* | *Description* |
|---|---|---|
| op | 1 | Message op code (message type). A value of 1 means it is a BOOTREQUEST message, and a value of 2 means it is a BOOTREPLY message. |
| htype | 1 | Hardware address type. The values are the same as those used for the ARP packet format. For example, a value of 1 is used for 10 Mbps ethernet. |
| hlen | 1 | Hardware address length in octets. Ethernet and token-ring hardware address length is 6 bytes. |
| hops | 1 | Set to zero by the BOOTP client. This is optionally used by relay agents running on routers when they forward BOOTP messages. |
| xid | 4 | Transaction ID. A randomly generated number chosen by the BOOTP client when it generates a BOOTP message. The BOOTP server uses the same Transaction ID in its BOOTP messages to the client. The Transaction ID enables the BOOTP clients and servers to associate BOOTP messages with the corresponding responses. |
| secs | 2 | Filled by the BOOTP client. It is the seconds elapsed since client started trying to boot. |
| unused | 2 | Not used. |
| ciaddr | 4 | The BOOTP client's IP address. It is filled by the BOOTP client in a BOOTPREQUEST message to verify the use of previously allocated configuration parameters. If the client does not recognize its IP address, this field is set to 0. |
| yiaddr | 4 | The BOOTP client's "your IP address." Returned by the BOOTP server. |
| siaddr | 4 | The server's IP address. Inserted by the BOOTP client if the client wants to contact a BOOTP server. The BOOTP server's IP address may have been initialized as part of the TCP/IP configuration database at the BOOTP client. The value returned by the BOOTP server may be the address of the next server to contact as part of the boot process. For example, this may be the address of a server that holds the operating system boot image. |
| giaddr | 4 | The IP address of the router that runs the BOOTP relay agent. |
| chaddr | 16 | The BOOTP client's hardware address. A value of 16 octets is used to enable different network hardware types. Ethernet and token-ring hardware use only 6 octets. |
| sname | 64 | An optional server hostname if recognized by the BOOTP client. It is a null-terminated string. |

**TABLE 8.1** continued

| *Field* | *Octets* | *Description* |
|---|---|---|
| file | 128 | The boot filename. It is a null-terminated string. If the BOOTP client wants to boot with an image of the operating system downloaded from a network device, it can specify a generic name, such as "unix," for booting a UNIX image.<br>The BOOTP server can hold more specific information about the exact operating system image needed for that workstation. This reply from the BOOTP server contains the fully qualified directory pathname. |
| vendor | 64 | An optional vendor-specific area. This could be hardware type/serial on request, or "capability"/remote file system handle on reply. This information may be set aside for use by a third phase bootstrap or kernel. |

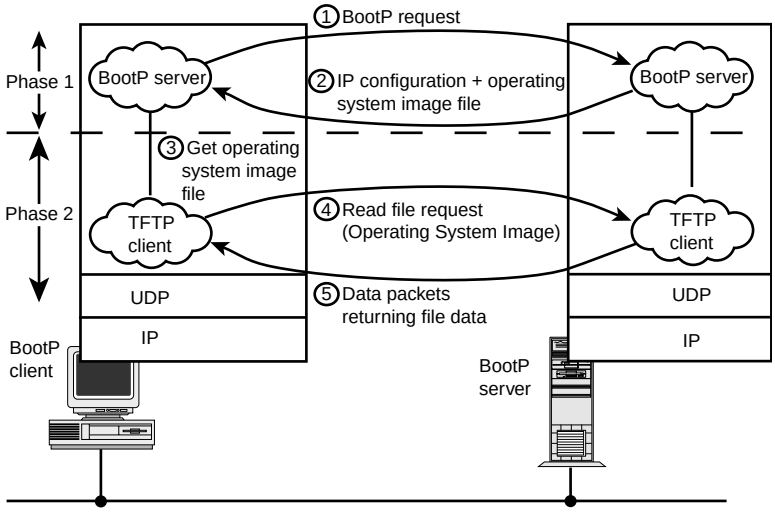
## Phases of the BOOTP Procedure

Contrary to what might be suggested by its name, BOOTP is not used for actually transferring the operating system image to the BOOTP client. The transfer of the operating system image is handled by a separate protocol called Trivial File Transfer Protocol (TFTP), which runs on the UDP transport. Diskless workstations need to download the operating system image. Other workstations that are only using BOOTP to assign IP addresses from a central location don't download the operating system image.

Figure 8.5 shows the phases of the complete bootstrap procedure required to download an operating system image. In step 1, the BOOTP client makes a request to obtain the IP configuration image. In step 2, the BOOTP server returns the IP configuration information and the name of the operating system image file to download. In step 3, the BOOTP client passes a request to obtain the operating system image to the TFTP client. In step 4, the TFTP client issues a request to the TFTP server to obtain the operating system image. In step 5, the TFTP server returns a series of data packets containing the operating system image. After obtaining the operating system image, the BOOTP loads the operating system image in memory and initializes itself.

Keeping the BOOTP phase separate from the transfer of the operating system image has the advantage that the BOOTP server need not run on the same machine that stores the operating system images (TFTP server). Additionally, it enables BOOTP to be used in situations where only the configuration image is needed from the BOOTP server and there is no need to transfer the operating system image.

**FIGURE 8.5**
*Typical bootstrap procedure.*




There may be situations in which an administrator may want to set up a workstation to boot up with a different operating system type depending on the user. In this case, the BOOTP request message can contain a generic label in the filename field, such as "unix," to download a Unix operating system image or "default" to download some other default operating system image. On seeing the generic name of the operating system image that has been requested, the BOOTP server can consult its configuration database and map the generic name to its fully qualified filename and return this name in the BOOTP reply. The BOOTP client can pass this fully qualified name to its local TFTP client for downloading the operating system image.



## The Vendor Specific Area Field

The BOOTP message format describes several common IP parameters. Many vendors need to describe additional parameters for their devices. The *Vendor Specific Area* field in BOOTP messages is used for these additional parameters.

The Vendor Specific Area field in the BOOTP message contains vendor-specific data for the BOOTP client. This field is optional, and its usage depends on the type of additional configuration information needed by the BOOTP client. The first four octets contain a "magic cookie" value that defines the format of the rest of the vendor field. These four octets are expressed using a dotted-decimal notation—they should not be confused as IP addresses. The magic cookie value of 99.130.83.99 (hexadecimal dotted notation 63.82.53.63) is an arbitrary agreed-upon value that specifies a standard format for the vendor-specific area. The magic cookie value is followed by a list of items. Each item is identified by the following:

- A 1-octet tag field or type field
- An optional 1-octet length field that specifies the length of the data field that follows
- A multi-octet data field

The BOOTP item tags were defined in RFC 1497 on "BOOTP Vendor Information Extensions." With the introduction of the DHCP protocol, the BOOTP vendor extensions and the DHCP options field now have a common format. This common format is described in RFC 2132 in "DHCP Options and BOOTP Vendor Extensions." Common examples of the vendor-specific tag values are listed in Table 8.2.

**TABLE 8.2** Common Vendor-Specific Tags for BOOTP

| *Tag* | *Length* | *Meaning (octets)* |
|---|---|---|
| 0 | - | Used only for padding. The pad option can be used to cause subsequent fields to align on word boundaries. |
| 1 | 4 | Subnet Mask field. |
| 2 | 4 | Time Offset field. Specifies time offset of the client's subnet in seconds from Coordinated Universal Time (UTC). The offset is expressed as a signed 32-bit integer. |
| 3 | N | Router option. Specifies a list of IP addresses for routers on the client's subnet. Routers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 4 | N | Time Server option. Specifies a list of RFC 868 time servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 5 | N | Name Server option. Specifies a list of Internet Engineering Note (IEN) 116 name servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 6 | N | Domain Name Server option. Specifies a list of Domain Name System (STD 13, RFC 1035) name servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |

**TABLE 8.2** continued

| *Tag* | *Length* | *Meaning (octets)* |
|---|---|---|
| 7 | N | Log Server option. Specifies a list of MIT-LCS UDP log servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 8 | N | Cookie Server option. Specifies a list of RFC 865 cookie servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 9 | N | LPR Server option. Specifies a list of RFC 1179 line printer servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 10 | N | Impress Server option. Specifies a list of Imagen Impress servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 11 | N | Resource Location Server option. Specifies a list of RFC 887 Resource Location servers available to the client. Servers should be listed in order of preference. The minimum length for the router option is 4 octets, and the length must always be a multiple of 4. |
| 12 | N | Host Name option. Specifies the name of the client. The name may or may not be qualified with the local domain name. See RFC 1035 for character set restrictions. Minimum length of the data field is 1 octet. |
| 13 | 2 | Boot File Size option. Specifies the length in 512-octet blocks of the default boot image for the client. The file length is specified as an unsigned 16-bit integer. |
| 128-254 | Undefined | Reserved for site-specific options. Interpretation depends on implementation. |
| 255 | - | End option. Marks the end of valid information in the vendor field. Subsequent octets should be filled with the pad option. |

# Dynamic Configuration Using DHCP

Dynamic Host Configuration Protocol (DHCP) is an extension of the BOOTP protocol that provides greater flexibility in IP address management. DHCP can be used for the dynamic configuration of essential TCP/IP parameters for hosts (workstations and servers) on a network. The DHCP protocol consists of two elements:

- A mechanism for allocating IP addresses and other TCP/IP parameters
- A protocol for negotiating and transmitting host-specific information

The TCP/IP host requesting the TCP/IP configuration information is called the *DHCP client*, and the TCP/IP host that supplies this information is called a *DHCP server*.

The DHCP protocol is described in RFC 2131 on "Dynamic Host Configuration Protocol." The following sections discuss the operation of DHCP.

## Understanding IP Address Management with DHCP

DHCP uses the following three methods for IP address allocation:

- Manual allocation
- Automatic allocation
- Dynamic allocation

In the manual allocation method, the DHCP client's IP address is set manually by the network administrator at the DHCP server, and DHCP is used to convey to the DHCP client the value of the manually configured IP address.

In the automatic allocation method, no manual assignments of IP addresses need to be made. The DHCP client is assigned an IP address when it first contacts the DHCP server. The IP address assigned using this method is permanently assigned to the DHCP client and is not reused by another DHCP client.

In the dynamic allocation method, DHCP assigns an IP address to a DHCP client on a temporary basis. The IP address is on loan or "leased" to the DHCP client for a specified duration. On the expiry of this lease, the IP address is revoked, and the DHCP client is required to surrender the IP address. If the DHCP client still needs an IP address to perform its functions, it can request another IP address.

The dynamic allocation method is the only one of the three methods that affords automatic reuse of an IP address. An IP address need not be surrendered by the DHCP client on the expiry of the lease. If the DHCP client no longer needs an IP address, such as when the computer is being gracefully shut down, it can release the IP address to the DHCP server. The DHCP server then can reissue the same IP address to another DHCP client making a request for an IP address.

The dynamic allocation method is particularly useful for DHCP clients that need an IP address for temporary connection to a network. For example, consider a situation where 300 users have portable computers on a network and a class C address has been assigned to a network. This enables the network to have 253 nodes on the network (255–2 special addresses = 253). Because computers connecting to a network using TCP/IP are required to have unique IP addresses, all the 300 computers cannot be simultaneously connected to the network. However, if at most only 200 physical connections are on the network, using a class C address by reusing IP addresses that are not in use should be possible. Using DHCP's dynamic IP address allocation, IP address reuse is possible.

Dynamic IP address allocation is also a good choice for assigning IP addresses to new hosts that are being permanently connected and where IP addresses are scarce. As old hosts are retired, their IP addresses can be immediately reused.



**Note**

Regardless of which of the three methods of IP address allocation is used, you can still configure IP parameters at a central DHCP server once, instead of repeating the TCP/IP configuration for each computer.

## The DHCP IP Address Acquisition Process

Upon contacting a DHCP server, a DHCP client goes through several internal states during which it negotiates the use of an IP address and the duration of use. How a DHCP client acquires the IP address can best be explained in terms of a state transition diagram (also called a finite state machine). Figure 8.6 shows the state transition diagram that explains the interaction between the DHCP client and DHCP server.

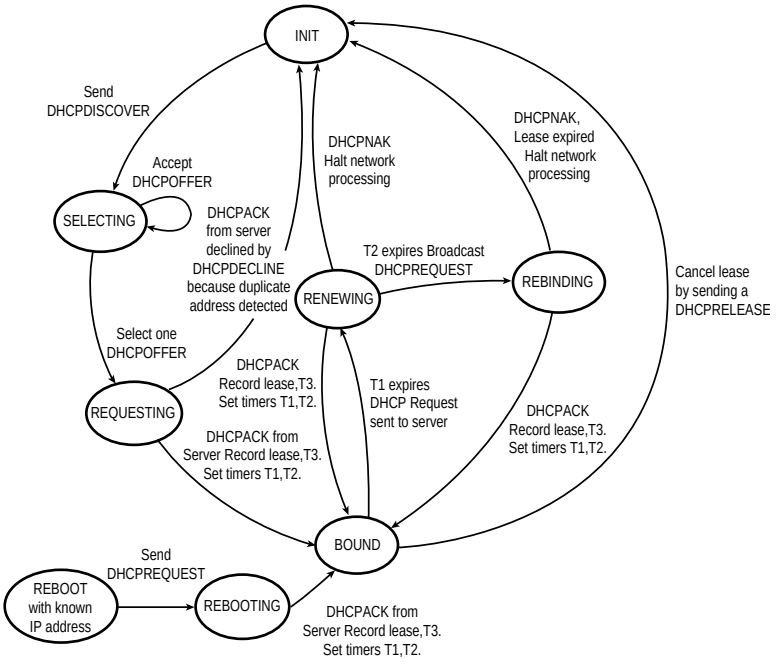



**FIGURE 8.6**
*DHCP state transition diagram showing DHCP client/server interaction.*

When the DHCP client is first started, it begins in the INIT (initialize) state. At this point the DHCP client does not recognize its IP parameters, and so sends a DHCPDISCOVER broadcast. The DHCPDISCOVER is encapsulated in a UDP/IP packet. The destination UDP port number is set to 67 (decimal), the same as that for a BOOTP server, because the DHCP protocol is an extension of the BOOTP protocol. A local IP broadcast address of 255.255.255.255 is used in the DHCPDISCOVER packet. If DHCP servers are not on the local network, the IP router must have DHCP-relay agent support to forward the DHCPDISCOVER request to other subnetworks. DHCP-relay agent support is discussed in RFC 1542.

Before sending the DHCPDISCOVER broadcast packet, the DHCP client waits for a random time interval between 1 to 10 seconds. This is to prevent DHCP clients from starting at the same time if they are powered up at the same time (as sometimes happens after a power failure).

After sending the DHCPDISCOVER broadcast, the DHCP client enters the SELECTING state. In this state, the DHCP client receives DHCPOFFER messages from the DHCP servers that have been configured to respond to the DHCP client. The time period over which the DHCP client waits to receive DHCPOFFER messages is implementation

dependent. The DHCP client must select one DHCPOFFER response if it receives multiple DHCPOFFER responses. After selecting a DHCPOFFER message from a server, the DHCP client sends a DHCPREQUEST message to the selected DHCP server. The DHCP server responds with a DHCPACK.

The DHCP client may optionally perform a check on the IP address sent in the DHCPACK to verify that the address is not in use. On a broadcast network, the DHCP client can send an ARP request for the suggested IP address to see whether there is an ARP response. An ARP response would imply that the suggested IP address is already in use, in which case the DHCPACK from the sever is ignored, a DHCPDECLINE is sent, and the DHCP client enters the INIT state and retries to get a valid unused IP address. When the ARP request is broadcast on the local network, the client uses its own hardware address in the sender hardware address field of the ARP packet, but sets a value of 0 in the sender IP address field. A sender IP address of 0 is used rather than the suggested IP address, so as not to confuse ARP caches on other TCP/IP hosts (in case the suggested IP address is already in use).

When the DHCPACK from the DHCP server is accepted, three timer values are set, and the DHCP client moves into the BOUND state. The first timer, T1, is the lease renewal timer; the second timer, T2, is the rebinding timer; and the third timer, T3, is the lease duration. The DHCPACK always returns the value of T3, which is the duration of the lease. The values of timers T1 and T2 can be configured at the DHCP server, but if they are not set, default values are used based on the duration of the lease. The following list shows the default values used for T1 and T2:

T1 = renewal timer

T2 = rebinding timer

T3 = duration of lease

T1 = 0.5 x T3

T2 = 0.875 x T3

The actual time at which the timer values expire is computed by adding to the timer value, the time at which the DHCPREQUEST that generated the DHCPACK response was sent. If the time at which the DHCP request was sent was T0, then the expiration values are computed as follows:

Expiration of T1 = T0 + T1

Expiration of T2 = T0 + T2

Expiration of T3 = T0 + T3

**Note**

RFC 2131 recommends that a "fuzz" factor be added to the timers T1 and T2 to prevent several DHCP clients from expiring their timers at the same time.

At the expiration of timer T1, the DHCP client moves from the BOUND state to the RENEWING state. In the RENEWING state a new lease for the allocated IP address must be negotiated by the DHCP client from the DHCP server that originally allocated the IP address. If the original DHCP server does not renew the release, it sends a DHCPNAK message, and the DHCP client moves into the INIT state and tries to obtain a new IP address. If the original DHCP server sends a DHCPACK message, this message contains the new lease duration. The DHCP client sets its timer values and moves to the BOUND state.

If the timer T2 expires while DHCP client is waiting in the RENEWING state for a DHCPACK or DHCPNAK message from the original DHCP server, the DHCP client moves from the RENEWING state to the REBINDING state. The original DHCP server may not respond because it's down or a network link is down. Note from the previous equations that T2 > T1, so the DHCP client waits for the original DHCP server to renew the release for a duration of T2-T1.

At the expiration of a timer T2, a broadcast DHCPREQUEST is sent on the network to contact any DHCP server to extend the lease, and the DHCP client is in the REBINDING state. A broadcast DHCPREQUEST is sent because the DHCP client assumes, after spending T2-T1 seconds in the RENEWING state, that the original DHCP server is not available, and the DHCP client tries to contact any configured DHCP server to respond to it. If a DHCP server responds with a DHCPACK message, the DHCP client renews its lease (T3), sets the timers T1 and T2, and moves back to the BOUND state. If no DHCP server is able to renew the release after the expiration of timer T3, the lease expires and the DHCP client moves to the INIT state. Note that by this time, the DHCP client has tried to renew the lease, first with the original DHCP server and then with any DHCP server on the network.

When the lease expires (timer T3 expires), the DHCP client must surrender the use of its IP address and halt network processing with that IP address.

The DHCP client does not always have to wait for the expiration of the lease (timer T3) to surrender the use of an IP address. It could voluntarily relinquish control of an IP address by canceling its lease. For example, a user with a portable computer may connect to the network to perform a network activity. The DHCP server on the network may set the duration of the lease for one hour. Assume that the user finishes the network tasks in

30 minutes, and now wants to disconnect from the network. When the user gracefully shuts down his computer, the DHCP client sends a DHCPRELEASE message to the DHCP server to cancel its lease. The surrendered IP address is now available for use by another DHCP client.

If DHCP clients are run on computers that have a disk, the IP address allocated can be stored on the disk, and when the computer reboots, it can make a request for the same IP address. This is shown in Figure 8.6 in the state labeled "REBOOT with known IP address."

## DHCP Packet Format

The DHCP packet format is shown in Figure 8.7. The DHCP messages use a fixed format for all the fields, except the options field that has a minimum size of 312 octets. Readers who are familiar with the BOOTP protocol recognize that with the exception of the flags field and the options field, the message formats for DHCP and BOOTP are identical. In fact the DHCP server can be configured to answer BOOTP requests. The configuration details are operating system–specific.



**FIGURE 8.7**
*DHCP packet format.*

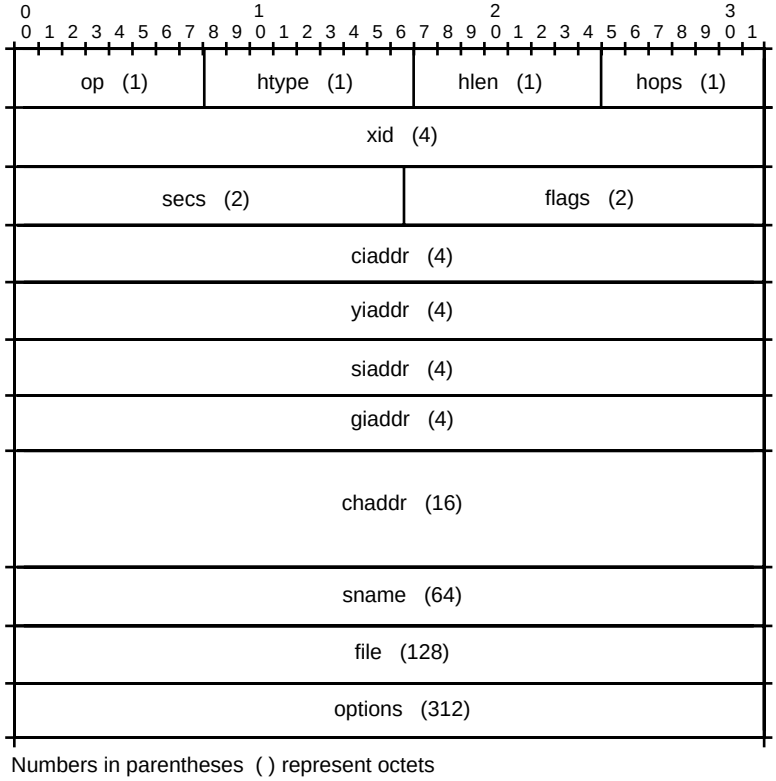


Table 8.3 provides an explanation of the fields used in the DHCP protocol. Only the leftmost bit of the DHCP options field is used (see Figure 8.8). The other bits in the options field must be set to 0.

**FIGURE 8.8**
*DHCP options field.*

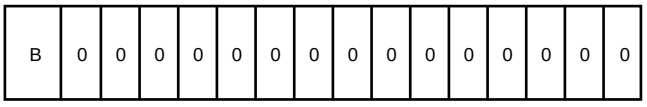



**TABLE 8.3** DHCP Fields

| *Field* | *Octets* | *Description* |
|---|---|---|
| op | 1 | Message op code (message type). A value of 1 means it is a BOOTREQUEST message, and a value of 2 means it is a BOOTREPLY message. |
| htype | 1 | The hardware address type. The values are the same as that used for the ARP packet format. For example, a value of 1 is used for 10 Mbps ethernet. |
| hlen | 1 | The hardware address length in octets. Ethernet and token-ring hardware adders length is 6 bytes. |
| hops | 1 | Set to 0 by the DHCP client. This is optionally used by relay agents running on routers when they forward DHCP messages. |
| xid | 4 | A Transaction ID. A randomly generated number chosen by the DHCP client when it generates a DHCP message. The DHCP server uses the same Transaction ID in its DHCP messages to the client. The Transaction ID enables the DHCP clients and servers to associate DHCP messages with the corresponding responses. |
| secs | 2 | Filled by the DHCP client. It is the seconds elapsed since client started trying to boot. |
| flags | 2 | Used to indicate whether this is a broadcast message. If so, the leftmost bit has a value of 1. All other bits must remain zero. |
| ciaddr | 4 | The DHCP client's IP address. It is filled by the DHCP client in a DHCPREQUEST message to verify the use of previously allocated configuration parameters. If the client does not know its IP address, this field is set to 0. |
| yiaddr | 4 | The DHCP client's "your IP address" returned by the DHCP server. |

**TABLE 8.3** continued

| *Field* | *Octets* | *Description* |
|---|---|---|
| siaddr | 4 | The server's IP address. If the DHCP client wants to contact a specific DHCP server, it inserts the server's IP address in this field. The DHCP server's IP address may have been discovered in prior DHCPOFFER and DHCPACK messages returned by the server. The value returned by the DHCP server may be the address of the next server to contact as part of the boot process. For example, this may be the address of a server that holds the operating system boot image. |
| giaddr | 4 | The IP address of the router that runs the relay agent. |
| chaddr | 16 | The DHCP client's hardware address. A value of 16 octets is used to enable different network hardware types. Ethernet and token-ring hardware use only 6 octets. |
| sname | 64 | An optional server hostname if known by the DHCP client. It is a null-terminated string. |
| file | 128 | The boot filename. It is a null-terminated string. If the DHCP client wants to boot with an image of the operating system downloaded from a network device, it can specify a generic name, such as "unix," for booting a Unix image in a DHCPDISCOVER. The DHCP server can hold more specific information about the exact operating system image needed for that workstation. This image name can be returned as a fully qualified directory path name in the DHCPOFFER message from the DHCP server. |
| options | 312 | An optional parameters field. |



Most of the DHCP messages sent by the DHCP server to the DHCP client are *unicast messages* (messages sent to a single IP address). This is because the DHCP server learns about the DHCP client's hardware address in messages sent by the DHCP client to the server. The DHCP client may request that the DHCP server respond with a broadcast address by setting the leftmost bit in the options field to 1. The DHCP client will do this if it does not know its IP address yet. The IP protocol module in DHCP client rejects a received datagram if the destination IP address in the datagram does not match the IP address of the DHCP client's network interface. If the IP address of the network interface is not known, the datagram still is rejected. However, the IP protocol module accepts any IP broadcast datagram. Therefore, to ensure that the IP protocol module accepts the DHCP server reply when the IP address is not yet configured, the DHCP client requests that the DHCP server reply use broadcast messages instead of unicast messages.

The options field is variable in length, with the minimum size extended to 312 octets, so that the total minimum size of a DHCP message is 576 octets, which is the minimum IP

datagram size a host must be prepared to accept. If a DHCP client needs to use larger size messages it can negotiate this using the Maximum DHCP message size option. Because the sname and file fields are quite large and may not always be used, DHCP options may be further extended into these fields by specifying the Option Overload option. If present, the usual meanings of the sname and file fields are ignored, and these fields are examined for options. Options are expressed using the Type, Length, Value (T-L-V) format.

Figure 8.9 shows that in DHCP, the option consists of a 1-octet Type field, followed by a 1-octet Length field. The value of the Length field contains the size of the Value field. The different DHCP messages themselves are expressed using a special Type value of 53. The option values that describe the DHCP messages are shown in Figure 8.10.

**FIGURE 8.9**
*Option format for DHCP messages.*

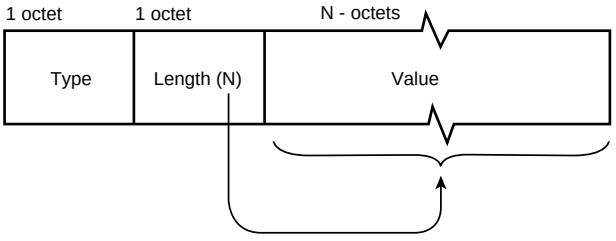
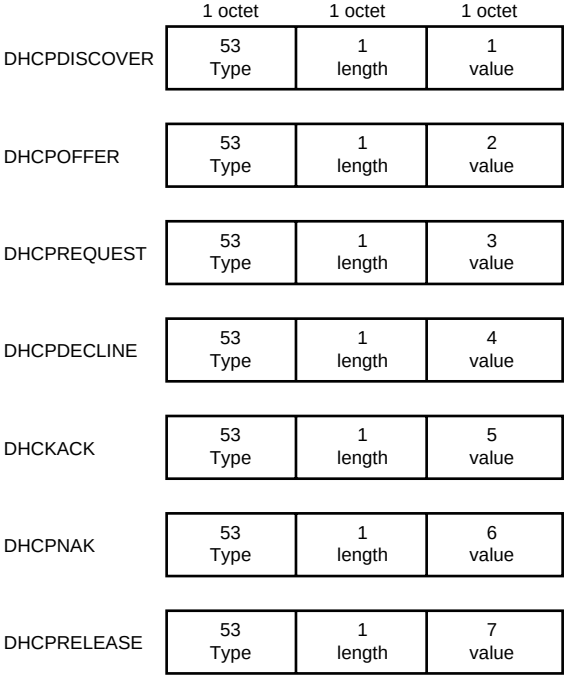



**FIGURE 8.10**
*Option values for the DHCP messages.*



# DHCP Protocol Trace

This section describes the DHCP message format for packets captured on a real-life network. Because the DHCP protocol and its packet format are a superset of BOOTP, only DHCP traces are discussed in this chapter. In fact, the DHCP client and servers use the same port number assignments for the BOOTP client and BOOTP servers. That is, DHCP client uses UDP port number 68, and DHCP server uses UDP port number 67. Most protocol analyzers will attempt to decode DHCP and BOOTP messages in the same format.

## DHCP Trace with Directed Reply from DHCP Server

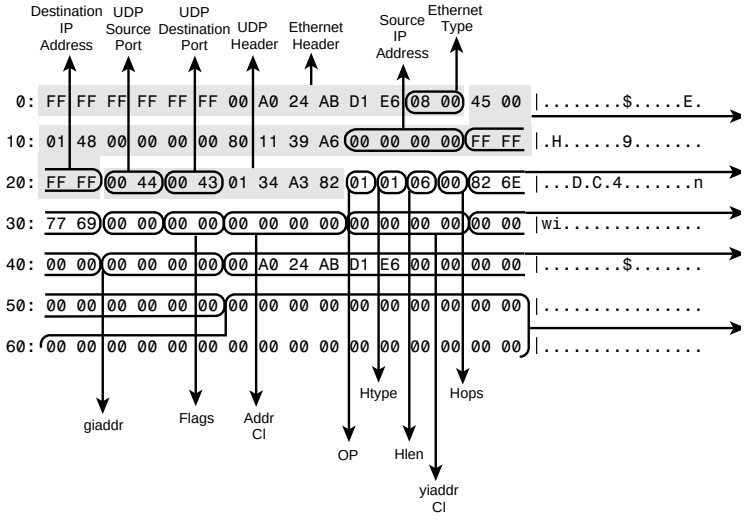
Figures 8.11 and 8.12 show a DHCP client broadcast request and a DHCP server reply. The DHCP server reply is a directed reply. A directed reply was possible because this implementation of the DHCP server made a direct entry in the ARP cache table at the server to add an entry for the DHCP client.

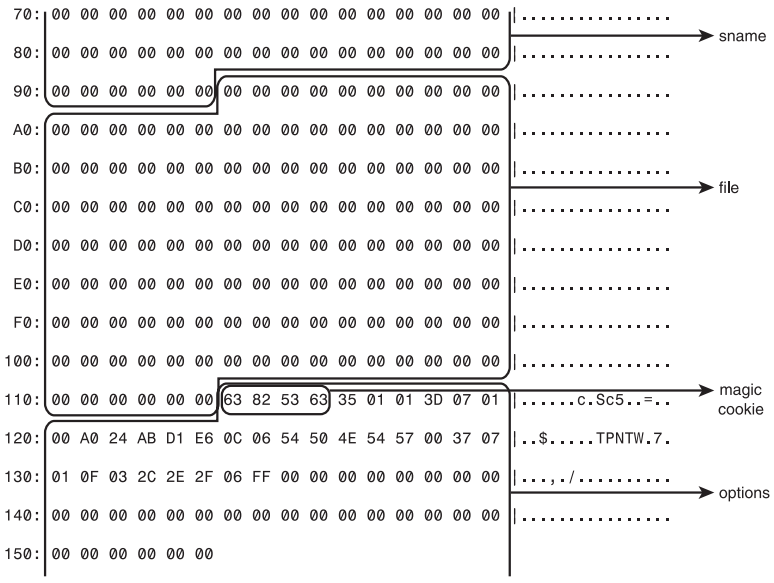
The DHCP broadcast request has the following parameters:

op = 1
htype = 1
hlen = 6
hops = 0
xid = 826E7769 (hex)
secs = 0
flags = 0
ciaddr = 0.0.0.0
yiaddr = 0.0.0.0
siaddr = 0.0.0.0
giaddr = 0.0.0.0
chaddr = 00A024ABD1E6 00000000000000000000
sname = 0 (64 octets)
file = 0 (128 octets)
options = 0 (312 octets)

**FIGURE 8.11**
*DHCP broadcast request.*

```
Packet Number : 4           7:28:11 PM
Length : 346 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-A0-24-AB-D1-E6 ----> FF-FF-FF-FF-FF-FF
       Type: 0x0800 (IP)
  ip: ======================= Internet Protocol =======================
       Station:0.0.0.0 ---->255.255.255.255
       Protocol: UDP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 328
       Identification:     0
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 128 seconds
       Checksum: 0x39A6(Valid)
 udp: ==================== User Datagram Protocol ====================
       Source Port: BOOTPC
       Destination Port: BOOTPS
       Length = 308
       Checksum: 0xA382(Valid)
Data:
   0: 01 01 06 00 82 6E 77 69 00 00 00 00 00 00 00 00 |.....nwi........
  10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A0 24 AB |..............$.
  20: D1 E6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  E0: 00 00 00 00 00 00 00 00 00 00 00 00 63 82 53 63 |............c.Sc
  F0: 35 01 01 3D 07 01 00 A0 24 AB D1 E6 0C 06 54 50 |5..=....$.....TP
 100: 4E 54 57 00 37 07 01 0F 03 2C 2E 2F 06 FF 00 00 |NTW.7....,./....
 110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 120: 00 00 00 00 00 00 00 00 00 00 00 00            |............
```





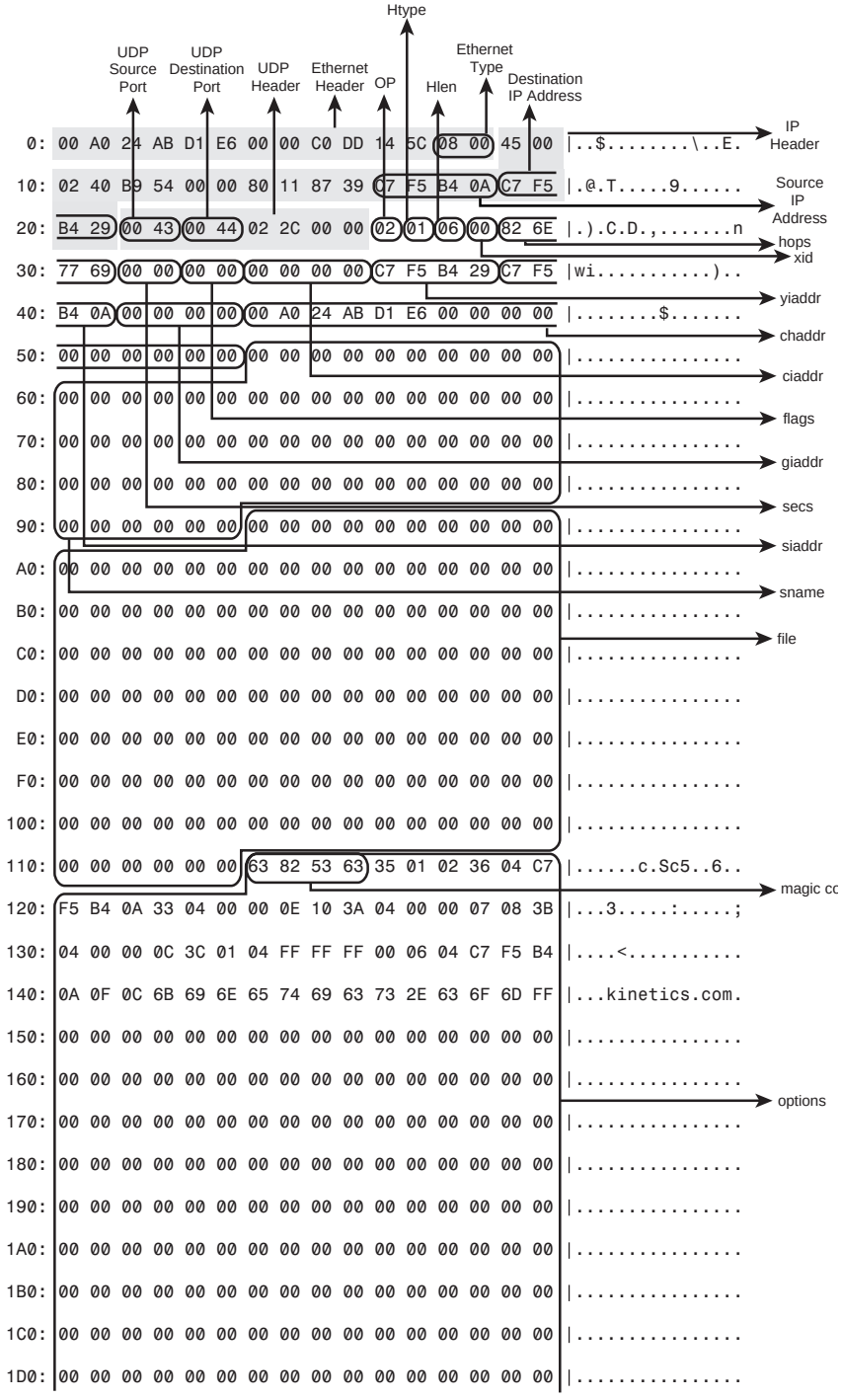


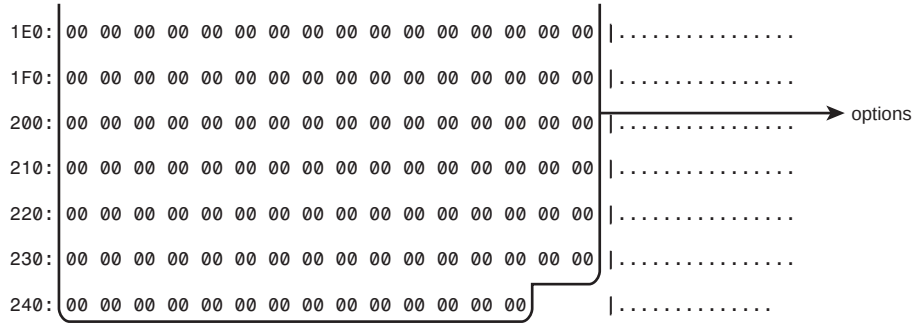

The DHCP directed reply has the following parameters:

op = 2
htype = 1
hlen = 6
hops = 0
xid = 826E7769 (hex)
secs = 0
flags = 0
ciaddr = 0.0.0.0
yiaddr = 199.245.180.41
siaddr = 199.245.180.10
giaddr = 0.0.0.0
chaddr = 00A024ABD1E6 00000000000000000000
sname = 0 (64 octets)
file = 0 (128 octets)
options: magic cookie = 63 82 53 63 (hex)

**FIGURE 8.12**
*DHCP directed reply.*

```
Packet Number : 5           7:28:11 PM
Length : 594 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-DD-14-5C ----> 00-A0-24-AB-D1-E6
       Type: 0x0800 (IP)
   ip: ======================= Internet Protocol =======================
       Station:199.245.180.10 ---->199.245.180.41
       Protocol: UDP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 576
       Identification: 47444
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 128 seconds
       Checksum: 0x8739(Valid)
  udp: ===================== User Datagram Protocol ====================
       Source Port: BOOTPS
       Destination Port: BOOTPC
       Length = 556
       Checksum: 0x0000(checksum not used)
Data:
   0: 02 01 06 00 82 6E 77 69 00 00 00 00 00 00 00 00 |.....nwi........
  10: C7 F5 B4 29 C7 F5 B4 0A 00 00 00 00 00 A0 24 AB |...).........$.
  20: D1 E6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  E0: 00 00 00 00 00 00 00 00 00 00 00 00 63 82 53 63 |............c.Sc
  F0: 35 01 02 36 04 C7 F5 B4 0A 33 04 00 00 0E 10 3A |5..6.....3.....:
 100: 04 00 00 07 08 3B 04 00 00 0C 3C 01 04 FF FF FF |.....;....<.....
 110: 00 06 04 C7 F5 B4 0A 0F 0C 6B 69 6E 65 74 69 63 |.........kinetic
 120: 73 2E 63 6F 6D FF 00 00 00 00 00 00 00 00 00 00 |s.com...........
 130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 220: 00 00 00 00
```


Htype
UDP Source Port
UDP Destination Port
UDP Header
Ethernet Header
OP
Hlen
Ethernet Type
Destination IP Address
0: 00 A0 24 AB D1 E6 00 00 C0 DD 14 5C 08 00 45 00 |..$.....\..E.
10: 02 40 B9 54 00 00 80 11 87 39 C7 F5 B4 0A C7 F5 |.@.T.....9.....
20: B4 29 00 43 00 44 02 2C 00 00 02 01 06 00 82 6E |.).C.D.,.....n
30: 77 69 00 00 00 00 00 00 00 00 C7 F5 B4 29 C7 F5 |wi..........)..
40: B4 0A 00 00 00 00 00 A0 24 AB D1 E6 00 00 00 00 |........$.....
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
110: 00 00 00 00 00 00 63 82 53 63 35 01 02 36 04 C7 |......c.Sc5..6..
120: F5 B4 0A 33 04 00 00 0E 10 3A 04 00 00 07 08 3B |...3.....:.....;
130: 04 00 00 0C 3C 01 04 FF FF FF 00 06 04 C7 F5 B4 |....<...........
140: 0A 0F 0C 6B 69 6E 65 74 69 63 73 2E 63 6F 6D FF |...kinetics.com.
150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
IP Header
Source IP Address
hops
xid
yiaddr
chaddr
ciaddr
flags
giaddr
secs
siaddr
sname
file
magic cc
options




## DHCP Trace with Broadcast Reply from DHCP Server

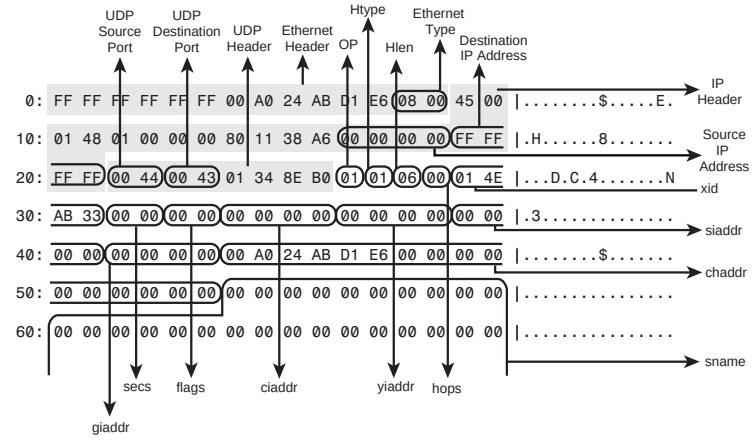
Figures 8.13 and 8.14 show a DHCP client broadcast request and a DHCP server reply. However, unlike the trace discussed in the preceding section, the DHCP server reply is sent as a broadcast reply. A directed reply was not possible because this implementation of the DHCP server does not make a direct entry in the ARP cache table.

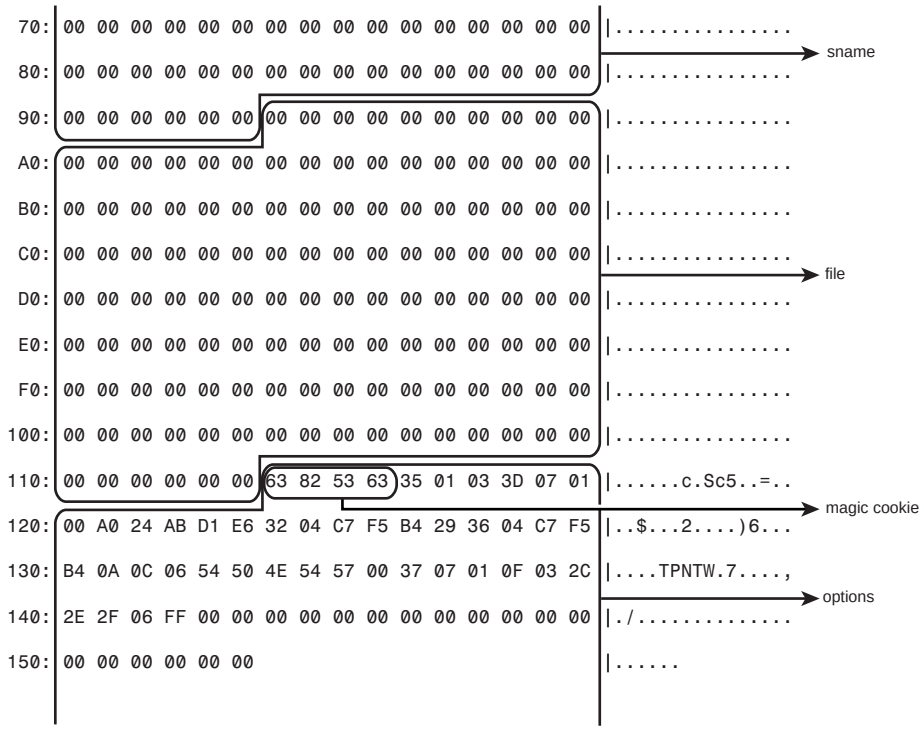
The DHCP broadcast request has the following parameters:

op = 1
htype = 1
hlen = 6
hops = 0
xid = 826E7769 (hex)
secs = 0
flags = 0
ciaddr = 0.0.0.0
yiaddr = 0.0.0.0
siaddr = 0.0.0.0
giaddr = 0.0.0.0
chaddr = 00A024ABD1E6 00000000000000000000
sname = 0 (64 octets)
file = 0 (128 octets)
options = 0 (312 octets)

**FIGURE 8.13**
*DHCP broadcast request.*

```
Packet Number : 6           7:28:11 PM
Length : 346 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-A0-24-AB-D1-E6 ----> FF-FF-FF-FF-FF-FF
       Type: 0x0800 (IP)
  ip: ======================= Internet Protocol =======================
       Station:0.0.0.0 ---->255.255.255.255
       Protocol: UDP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
             Normal Delay, Normal Throughput, Normal Reliability
       Total length: 328
       Identification:   256
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 128 seconds
       Checksum: 0x38A6(Valid)
 udp: ===================== User Datagram Protocol ====================
       Source Port: BOOTPC
       Destination Port: BOOTPS
       Length = 308
       Checksum: 0x8EB0(Valid)
Data:
   0: 01 01 06 00 01 4E AB 33 00 00 00 00 00 00 00 00 |.....N.3........
  10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A0 24 AB |...............$.
  20: D1 E6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  E0: 00 00 00 00 00 00 00 00 00 00 00 00 63 82 53 63 |............c.Sc
  F0: 35 01 03 3D 07 01 00 A0 24 AB D1 E6 32 04 C7 F5 |5..=....$...2...
 100: B4 29 36 04 C7 F5 B4 0A 0C 06 54 50 4E 54 57 00 |.)6.......TPNTW.
 110: 37 07 01 0F 03 2C 2E 2F 06 FF 00 00 00 00 00 00 |7....,./........
 120: 00 00 00 00 00 00 00 00 00 00 00 00             |............
```



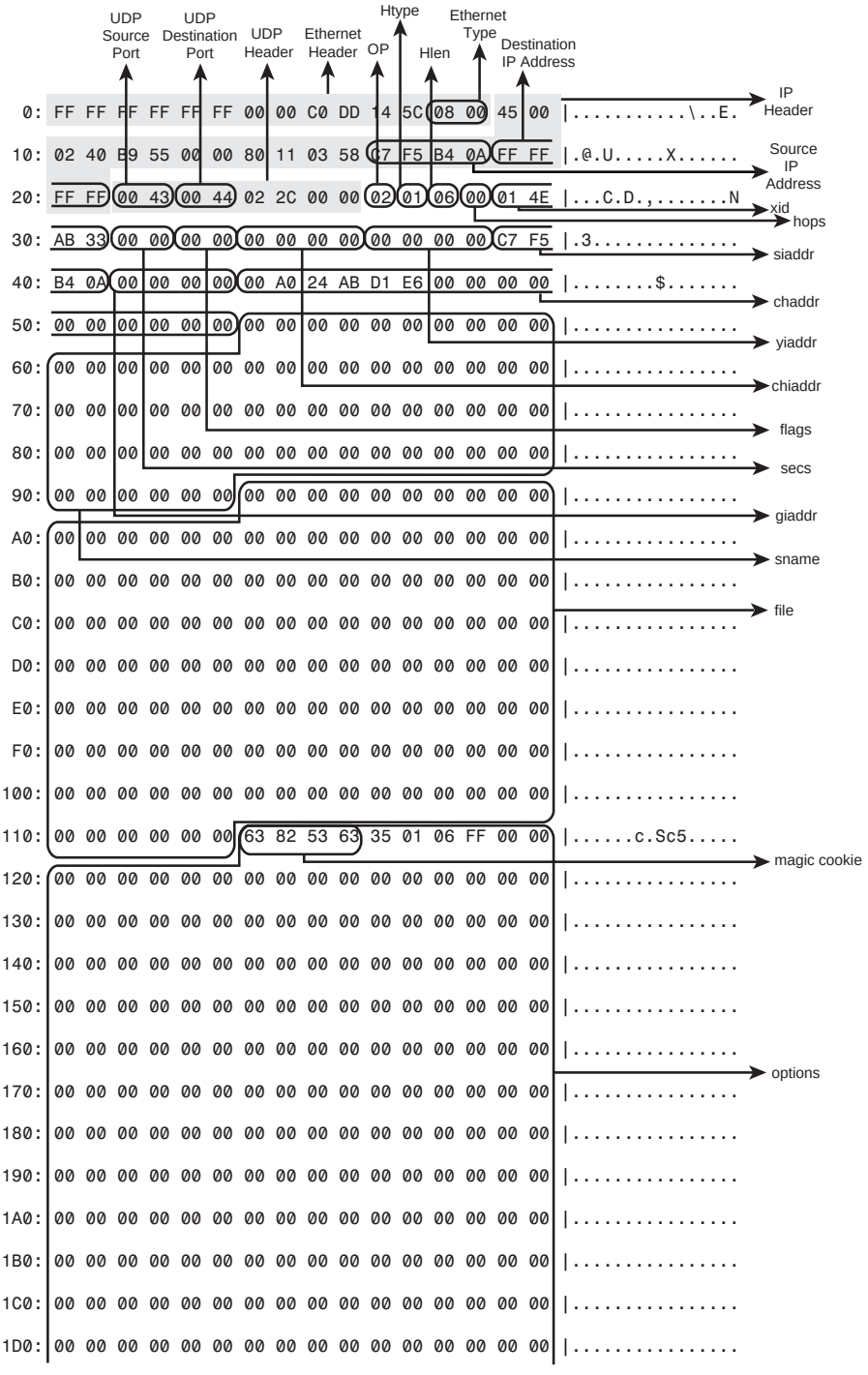


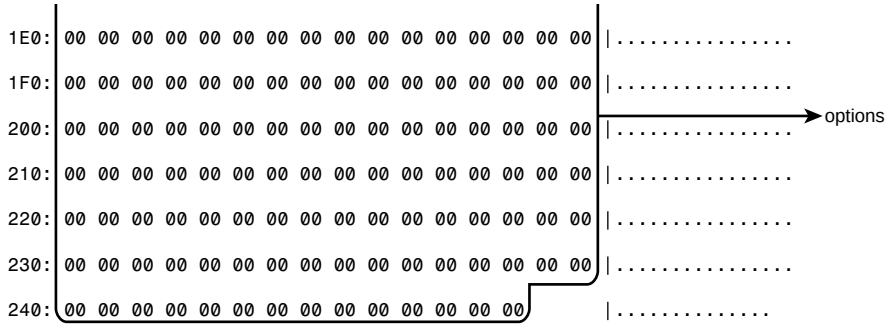

The DHCP broadcast reply has the following parameters:

op = 2
htype = 1
hlen = 6
hops = 0
xid = 826E7769 (hex)
secs = 0
flags = 0
ciaddr = 0.0.0.0
yiaddr = 0.0.0.0
siaddr = 199.245.180.10
giaddr = 0.0.0.0
chaddr = 00A024ABD1E6 00000000000000000000
sname = 0 (64 octets)
file = 0 (128 octets)
options: magic cookie = 63 82 53 63 (hex)

**FIGURE 8.14**
*DHCP broadcast reply.*

```
Packet Number : 7           7:28:11 PM
Length : 594 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-DD-14-5C ----> FF-FF-FF-FF-FF-FF
       Type: 0x0800 (IP)
   ip: ======================= Internet Protocol =======================
       Station:199.245.180.10 ---->255.255.255.255
       Protocol: UDP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 576
       Identification: 47445
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 128 seconds
       Checksum: 0x0358(Valid)
  udp: ===================== User Datagram Protocol =====================
       Source Port: BOOTPS
       Destination Port: BOOTPC
       Length = 556
       Checksum: 0x0000(checksum not used)
Data:
   0: 02 01 06 00 01 4E AB 33 00 00 00 00 00 00 00 00 |.....N.3........
  10: 00 00 00 00 C7 F5 B4 0A 00 00 00 00 00 A0 24 AB |..............$.
  20: D1 E6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  E0: 00 00 00 00 00 00 00 00 00 00 00 00 63 82 53 63 |............c.Sc
  F0: 35 01 06 FF 00 00 00 00 00 00 00 00 00 00 00 00 |5...............
 100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 1F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
 220: 00 00 00 00                                     |....
```


UDP Source Port
UDP Destination Port
UDP Header
Ethernet Header
OP
Htype
Hlen
Ethernet Type
Destination IP Address
0: FF FF FF FF FF FF 00 00 C0 DD 14 5C 08 00 45 00 |..........\..E.
10: 02 40 B9 55 00 00 80 11 03 58 C7 F5 B4 0A FF FF |.@.U.....X......
20: FF FF 00 43 00 44 02 2C 00 00 02 01 06 00 01 4E |...C.D.,.......N
30: AB 33 00 00 00 00 00 00 00 00 00 00 00 00 C7 F5 |.3..............
40: B4 0A 00 00 00 00 00 A0 24 AB D1 E6 00 00 00 00 |........$.......
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
110: 00 00 00 00 00 00 63 82 53 63 35 01 06 FF 00 00 |......c.Sc5.....
120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
IP Header
Source IP Address
xid
hops
siaddr
chaddr
yiaddr
chiaddr
flags
secs
giaddr
sname
file
magic cookie
options




# Summary

Both BOOTP and DHCP solve the important problem of automatic configuration of IP parameters, such as IP addresses and subnet masks, for individual devices on the network. Both protocols are client/server based and use the same UDP port numbers. The DHCP protocol is designed to replace the older BOOTP protocol and is a superset of the BOOTP message format.

A major advantage of DHCP is that IP addresses can be leased on a time basis. This enables a more flexible reuse of IP addresses where IP addresses do not have to be permanently allocated to a station based on their hardware addresses. The BOOTP IP address allocation is less flexible than that of DHCP because IP addresses are associated with the hardware address of a network interface.

# IP and Related Protocols

# PART III

## IN THIS PART

CHAPTER 9

# Overview of the IP Family of Protocols

*by Mark A. Sportack and Karanjit S. Siyan, Ph.D.*

## In This Chapter

TCP/IP has become a catch-all phrase that describes IP-based communications. Despite its popularity, few people realize that it is really a whole family of protocols, each with its own set of capabilities and limitations. This chapter examines the architecture, functionality, and uses of the various protocols within the IP family of protocols.

The purpose of this chapter is to establish the framework for discussing the details of TCP/IP. Later chapters describe the components of the TCP/IP family of protocols in more detail. Even though some of the information has been presented earlier, its review here will help you establish an overview of the TCP/IP protocol.

# The TCP/IP Model

When TCP/IP was developed, it was described in terms of its own Reference Model that describes the stratification of its functions. TCP/IP's model was developed long after the protocol suite itself was developed. Consequently, the model couldn't guide the development of the protocols, and anomalies were rampant!

The TCP/IP Reference Model is discussed in Chapter 1, "Introduction to Open Communications."

## The TCP/IP Suite of Protocols

Although they are frequently identified as just "TCP/IP," several different component protocols actually exist within the IP suite of protocols. These include:

- *IP*—The Internet Layer protocol.
- *TCP*—The reliable Host-to-Host Layer protocol.
- *UDP*—The best-effort Host-to-Host Layer protocol.
- *ICMP*—A multilayer protocol designed to facilitate control, testing, and management functions within an IP network. The various ICMP protocols span the Host-to-Host Layer and the Process/Application Layer.

The relationship between these protocols is illustrated in Figure 9.1.

**Note**

The applications that reside at the Process/Application Layer (such as Telnet, FTP, and many others) must also be considered a native component of the IP suite of protocols. However, because these are applications rather than protocols, they are not addressed in this chapter.

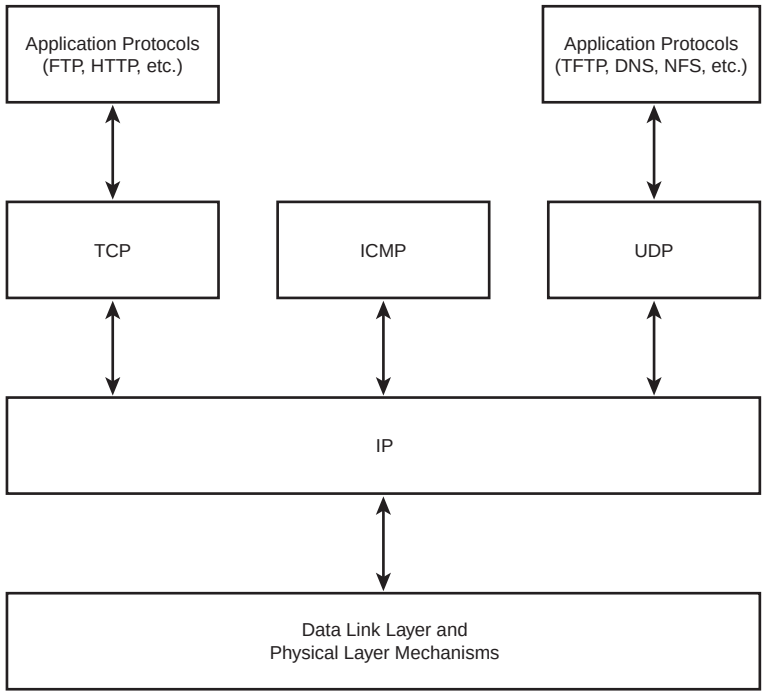



**FIGURE 9.1**
*TCP/IP is actually a suite of related protocols, rather than just a single protocol.*

# Understanding the Internet Protocol (IP)

The Internet Protocol has become the dominant internetworking protocol in the world. Other contenders, such as OSI, AppleTalk, and even IPX have ultimately lost out to IP due to its openness. Despite this success in the marketplace, IP remains a mostly misunderstood protocol. Its functionality is defined by the amount of data that resides in its header structure. The IP header structure, and its resulting capability set, were originally defined in a series of RFCs and other openly published documents that date back to the creation of the IETF. RFC 791, published in September of 1981, is generally accepted as the foundation document for today's version of IP.

IP is continuously evolving, thanks to the ceaseless efforts of the IETF. Many new features and capabilities have been added through subsequent RFCs. All, however, build upon the basis established in RFC 791. Architecturally speaking, the current version of IP is version 4 (IPv4). A new version, version 6, will eventually replace IPv4, but IPv4 is the current standard that is broadly supported. For additional information on IPv6, please see Chapter 12, "IP Version 6."

# The IPv4 Header

Figure 9.2 illustrates the structure of the IP header, as well as the sizes of its fields.

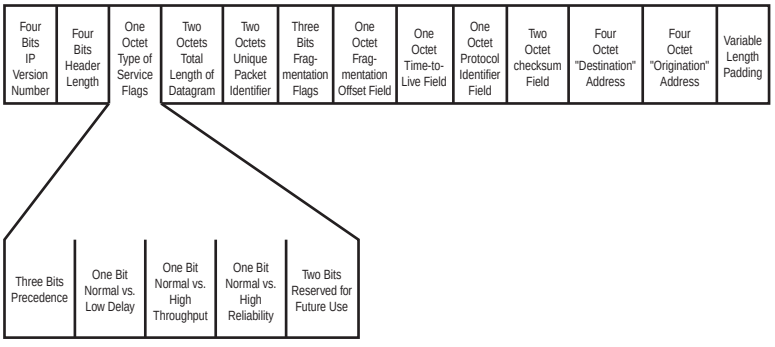



**FIGURE 9.2**
*The structure of an IP header reveals the many fields that support IP's capabilities.*

The IP header has the following sizes and fields:

- *Version*—The first 4 bits of the IP header identify the operating version of IP; for example, version 4 or version 6.
- *Internet Header Length*—The next 4 bits of the header contain the length of the header, expressed in multiples of 32.
- *Type of Service*—The next octet contains a series of flags that can be used to specify precedence (that is, absolute priority relative to other IP packets), delay, throughput, and reliability parameters for that packet of data. The precedence flag is 3 bits long whereas the delay, throughput, and reliability flags are each 1 bit in length. The remaining 2 bits are reserved for future use.
- *Total Length*—This 16-bit field contains the total length of the IP packet measured in octets. Valid values can range up to 65,535 octets.
- *Identifier*—Each IP packet is given a unique 16-bit identifier, which is used to identify the fragments of a datagram.
- *Fragmentation Flags*—The next field contains three 1-bit flags that indicate whether fragmentation of the packet is permitted, and if it is used. The first bit is reserved and always set equal to 0. The second bit indicates whether that packet's data can be fragmented. If this bit is equal to 0, the contents can be fragmented. If it is equal to 1, it cannot be fragmented. The third bit has significance only if the second bit was set to 0. If that bit was equal to 0 (and the data can be segmented across multiple packets), this bit indicates whether this particular packet is the last in the series of the fragment, or whether the receiving application can expect additional fragments. A 0 indicates that this packet is the last one.

- *Fragment Offset*—This 8-bit field measures the offset of the fragmented contents relative to the beginning of the entire packet. This value is measured in 64-bit increments.
- *Time-to-Live (TTL)*—The IP packet cannot be permitted to roam the WAN in perpetuity. It must be limited to a finite TTL. The 8-bit TTL field is incremented by one for each hop the packet makes. After reaching its maximum limit, the packet is assumed to be undeliverable. An ICMP message is generated and sent back to the source machine and the undeliverable packet is destroyed.
- *Protocol*—This 8-bit field identifies the protocol that follows the IP header, such as VINES, TCP, UDP, and so forth.
- *Checksum*—The Checksum field is a 16-bit error-checking field. The destination computer, and every gateway node in the network, will recompute the mathematical calculation on the packet's header as the source computer did. If the data survived the trip intact, the results of these two calculations are identical. This field also informs the destination host of the amount of incoming data.
- *Source IP Address*—The source address is the IP address of the source computer.
- *Destination IP Address*—The destination address is the IP address of the destination computer.
- *Padding*—Extra zeros are added to this field to ensure that the IP header is always a multiple of 32 bits.

These header fields reveal that IPv4's Internet Layer is inherently connectionless: The packet forwarding devices in the network are free to determine the ideal path for each packet to take through the network. It also doesn't provide any of the acknowledgments, flow control, or sequencing functions of higher-level protocols such as TCP. Nor can IP be used to direct the data contained in an IP datagram to the appropriate destination application. These functions are left to higher-level protocols, such as TCP and UDP.



## What Does IP Do?

The header information of an IP packet contains all the information necessary to enable some critical network functions. These functions include

- Addressing and routing
- Fragmentation and reassembly
- Detection and correction of data damaged in transit

## Addressing and Routing

One of the more obvious capabilities of IP is that it enables packets to be delivered to a specific destination. The destination IP address is used by routers and switches in the network that intervenes the source and destination machine pairs, to identify the optimal path through that network.

Similarly, IP packets also bear the IP address of the source machine. Thus, the destination machine may contact the source machine should the need arise.

## Fragmentation and Reassembly

Sometimes segments of application data don't fit cleanly inside a single IP packet constrained by the Maximum Transmission Unit (MTU) size of the underlying network. In this case the IP packet must be fragmented and split across two or more packets. When this occurs, IP must be able to reconstruct the original data segment, regardless of how many packets were required to get it to its destination.

It is important to note that the source and destination machine must understand, and adhere to, the exact procedure for fragmenting data segments. Otherwise, reassembling data that was fragmented into multiple packets for delivery through a network would be impossible. The delivered data has been successfully reassembled when it has been restored to the exact form it was in on the source machine, before it was fragmented. The fragmentation flags in the IP header are used to identify fragmented data segments.

**Note**

Reassembling fragmented data segments is quite different than resequencing data that has arrived out of order. Resequencing is a function of TCP.

## Compensating for Damaged Packets

The last major function of IP is to detect and compensate for any datagrams whose payload may have been damaged or lost in transit. There are many ways that a packet can become damaged: *Radio frequency interference (RFI)* and *electromagnetic interference (EMI)* are two of the more obvious. A packet is considered damaged when it arrives at its destination with a different bit pattern than the source machine created.

A packet may be lost for many reasons. For one thing, network congestion may result in a packet exceeding its time-to-live (TTL). The router that detects the expiration of the

TTL would simply discard that packet. Alternatively, a packet may become so corrupted in transit by either EMI or RFI that its header information becomes meaningless. In such cases, the packet would also likely be discarded.

Whenever an IP packet becomes undeliverable or unusable, the source machine must be notified. The IP header contains the source machine's IP address to facilitate this notification. Thus, although IP does not contain the mechanisms to coordinate a retransmission, it plays an integral role in the notification of a source machine about the possible need to retransmit.

### IP Conclusions

Despite these capabilities, IP must be acknowledged as just an internetworking protocol. To be useful, it must be accompanied by both a transport protocol (Layer 4 of the OSI Reference Model) and a Data Link Layer protocol (Layer 2 of the OSI Reference Model). Although Data Link Layer architectures are outside the scope of this book, the remainder of this chapter focuses on two of the transport protocols that rely upon IP for internetworking. These are TCP and UDP.

## Understanding the Transmission Control Protocol (TCP)

TCP is the transport layer (Layer 4 of the OSI Reference Model) protocol that uses IP to provide reliable delivery of application data. TCP establishes a connection-oriented session between two or more hosts. It can support multiple data streams, as well as coordinate flow and error control and even the reordering of packets that may have been received out of order. The Transmission Control Protocol was also developed through a series of open published IETF documents. The development of TCP culminated in RFC 793 in September of 1981. As with RFC 791 IP, RFC 793 TCP has been augmented over the past 18 years, but it has never been completely superceded. Thus, the contents of that RFC remain TCP's core.

### TCP Header Structure

As with IP, the functionality of TCP is limited by the amount of information that it carries in its header. Thus, understanding the mechanics and capabilities of TCP requires an appreciation for the contents of its header. Figure 9.3 illustrates the structure of the TCP header, as well as the sizes of its fields.

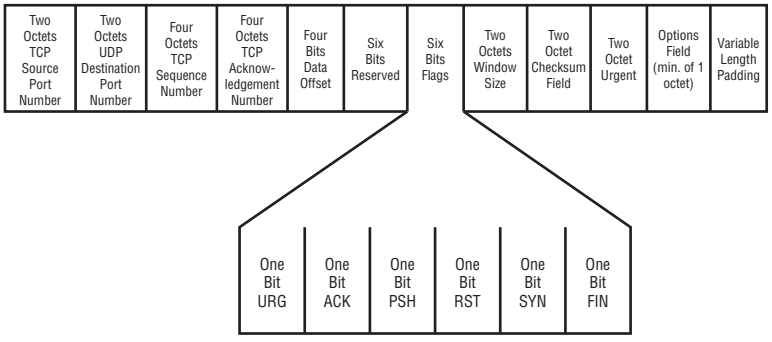



**FIGURE 9.3**
*The structure of a TCP header demonstrates that it relies upon several fields to function properly.*

The TCP protocol header is a minimum of 20 octets and contains the following fields:

- *TCP Source Port*—The 16-bit source port field contains the number of the port that initiates the communications session. The source port and source IP address function as the packet's return address.
- *TCP Destination Port*—The 16-bit destination port field is the address of the port for which the transmission is destined. This port contains the interface address of the application on the recipient's computer to which the packet's data will be passed.
- *TCP Sequence Number*—The 32-bit sequence number is used by the receiving computer to reconstruct the fragmented data back into its original form. In a dynamically routed network, it is quite possible for some of the packets to take different routes and, consequently, to arrive out of order. This sequencing field compensates for this inconsistency of delivery.
- *TCP Acknowledgment Number*—TCP uses a 32-bit acknowledgment (ACK) of the first octet of data contained in the next expected segment. It may seem counter-intuitive to acknowledge something that hasn't occurred yet, but a source TCP/IP machine that receives an ACK knows that all the data up to, but not including, that specified segment has been received. The number used to identify each ACK is the sequence number of the packet being acknowledged. This field is only valid if the ACK flag (see Flags, later in this list) is set.
- *Data Offset*—This 4-bit field contains the size of the TCP header, measured in a 32-bit data structure known as a "word."
- *Reserved*—This 6-bit field is always set to zero. It is reserved for an as-yet unspecified future use.
- *Flags*—The 6-bit flag field contains six 1-bit flags that enable the control functions of urgent field, acknowledgment of significant field, push, reset connection, synchronize sequence numbers, and finished sending data. The flags, in their order of appearance in the string, are *URG*, *ACK*, *PSH*, *RST*, *SYN*, and *FIN*. Given the

preceding description of their functions, these mnemonic abbreviations should be self-apparent.

- *Window Size*—This 16-bit field is used by the destination machine to tell the source host how much data it is willing to accept, per TCP segment.
- *Checksum*—The TCP header also contains a 16-bit error-checking field known as a "Checksum." The source host calculates a mathematical value, based upon the segment's contents. The destination host performs the same calculation. If the content remained intact, the result of the two calculations is identical, thereby proving the validity of the data.
- *Urgent*—The Urgent field is an optional 16-bit pointer that points to the last octet of urgent data within the segment. This field is only valid if the URG flag was set. If that flag is not set, the Urgent field is pre-empted with Padding. Segments of data that are identified as urgent are treated to expedited handling by all TCP/IP devices that lie in the network intervening the source and destination machines.
- *Options*—A variable length field of at least 1 octet identifies which options, if any, are valid for the TCP segment. If no options are set, this 1-octet field is set equal to 0, which indicates the end of the Options field. A value of 1 in this octet indicates that no operation is required. A value of 2 indicates that the next four octets contain the source machine's *Maximum Segment Size* (MSS). The MSS is the greatest number of octets that the data field can be, as agreed to by the source and destination machines.
- *Data*—Although not technically a part of the TCP header, it is important to recognize that segments of application data follow the Urgent and/or Options fields, but precede the Padding field. The field's size is the largest MSS that can be negotiated between the source and destination machines. Segments may be smaller than the MSS, but never larger.
- *Padding*—Contrary to any indication of superfluousity that its name might suggest, padding always serves a mathematical purpose in data communications. That purpose is to ensure predictability of spacing, timing, or sizing. Extra zeros are added to this field to ensure that the TCP header is always a multiple of 32 bits.



## What Does TCP Do?

TCP provides several important functions in a communications session. It can best be thought of as the liaison between multiple applications and the network. Its functions include

- Multiplexing data from multiple applications to and from the network
- Testing received data for integrity

- Resequencing application data that may have arrived out of order
- Acknowledging successful receipt of transmitted data
- Rate-adaptive flow control (via the TCP window size)
- Timing functions
- Coordinating the retransmission of data that was damaged or lost in transit

## Multiplexing Data Streams

TCP is the interface between a user's applications and the network's myriad communications protocols. Because it would be virtually unheard of for any user to be limited to just one application, TCP must be able to simultaneously accept data from multiple applications, bundle them into segments of data, and pass these segments off to IP. Similarly, TCP must be able to receive data for multiple applications at the same time.

TCP keeps track of which incoming packets must be forwarded to which applications through the use of *port numbers*. Thus, it is really useful if both the source and destination machines agree on a common set of port numbers for their application base. Unfortunately, there is such a tremendous amount of applications that can run on IP that it would be virtually impossible to get any sort of consistency of numbers on an ad hoc basis. Consequently, IANA, and now ICANN, have stepped up to regulating at least a portion of the available port numbers.

Simplifying ICANN's task is that many applications are so common that they are regarded as *well known*. As such, ICANN can assign port numbers to these applications, and anyone can reasonably expect any IP-capable host to recognize them. Examples of well-known port numbers include

- Port 80 (the Hypertext Transfer Protocol, or HTTP)
- Port 119 (Network News Transfer Protocol, or NNTP)
- Port 69 (Trivial File Transfer Protocol, or TFTP)

It would be impossible to list all the well-known port numbers here because there are 1,024 of them (ranging from 0 to 1,023). For a complete list of the well-known port numbers for both TCP and UDP, please refer to RFC 1700.

Because the port number field contains a 16-bit binary number, there are 65,535 mathematically possible port numbers. Whereas numbers 0 through 1023 are regarded as well known, anything greater than 1023 is generally regarded as a *high port number* or *temporary port number*, or *ephemeral port number*. The high port numbers are not regulated by ICANN. Thus, applications that are not well known (including homegrown applications or shrink-wrapped software) are not precluded from using IP for their communications needs. They can arbitrarily select one of the high port numbers that is not in use.

TCP maintains both the source and destination application's port number in each TCP segment. Another term that is frequently used is *socket*, although there is no discrete socket field in the TCP header. A socket is the concatenation of a host's IP address and the port number of a specific application that resides on that host. Thus, a socket describes a unique host and application pair. A ":" separates the two numbers. For example, socket 10.1.1.9:666 identifies the application port number 666 (the port number for DOOM) on host 10.1.1.9.

**Note**

The term *socket* also refers to a programming data structure that is used to keep track of the IP address and port number pair just described. The set of system calls that use the socket data structure are called the *sockets interface*. The sockets interface was popularized in BSD Unix, and is also sometimes called BSD Sockets interface. The implementation of BSD sockets for the Windows operating system is called Windows Sockets or WinSock. WinSock is usually implemented as a Dynamic Link Library (DLL).

## Testing for Data Integrity

TCP performs a mathematical operation on each segment of data that is encapsulated in a TCP segment, and places the result of that operation in the Checksum field of the TCP header. Upon arrival at its intended destination, performing the same mathematical operation on the received data should yield the same result as was stored in the TCP header. If it does, the data can reasonably be assumed to have arrived intact. Otherwise, a request is sent back to the source machine for another copy of that segment of data.

## Resequencing

It is not uncommon for segments of data to arrive out of sequence at their destination. There are many reasons for this. For example, in a highly used network, it is possible that the routing protocols selected different routes through the network. This could result in segments arriving out of sequence. Alternatively, packets could be lost or damaged in transit. Thus, the sequence of data that is needed by the recipient application would be thrown off. Regardless, the destination machine's TCP protocol would buffer the received segments of data until it could resequence them into the correct order.

This task is accomplished by examining the contents of the TCP Sequence Number field in the TCP header. Resequencing is simply a matter of mathematically sorting the received segments, based on this field.

## Flow Control

The source and destination machines in a TCP session are known as *peers*. Each peer has the capability to control the flow of data that is streaming into its physical input buffers. The mechanism that is used is the size of the TCP *window*. A window size is communicated between a source and destination machine via the TCP header. Any machine that is becoming backlogged with incoming data may throttle back the rate at which the transmitting machine can transmit, simply by informing that machine of its new window size. If a machine's buffers fill up completely, it will send an acknowledgment of the last received data segment with a new window size of 0. This effectively halts transmission until that congested machine can clear its buffers. Each segment that it processes must be acknowledged and, with this acknowledgment, comes another opportunity to restart transmissions by re-establishing a window size greater than 0.

Although this simple mechanism can effectively regulate the flow of data between two machines, it can only help ensure that these end systems aren't overwhelmed with incoming data. Window size, by itself, does not take into account any congestion that may exist on the network. Congestion on the network would mean that each transmitted segment takes longer than usual to actually reach its destination. Therefore, congestion management must be a function of time on the network. TCP implements congestion management through the use of timers.

## Timing Mechanisms

TCP uses timing mechanisms for several critical functions. Each time a segment is transmitted, a timer is set. If that timer expires (that is, decrements to 0) before an acknowledgment is received, the segment is assumed to be lost. Consequently, it is retransmitted. This timer can also be used to manage network congestion indirectly by slowing the rate of transmission whenever a timeout occurs. In theory, transmission of segments is throttled back until timeouts cease occurring. Thus, TCP can't quite manage congestion on a network, but it can decrease its own contribution to that congestion.

## Acknowledging Receipt

A destination TCP machine must acknowledge the receipt of a specific data segment (as identified by its sequence number), if the ACK flag is set. Given that TCP is almost always used in a reliable mode, it would be unusual for the ACK flag to not be set.

Data segments whose transmission is not acknowledged are assumed to be lost in transit, and are re-sent. The retransmission must be coordinated between the source and destination machines.

# Understanding the User Datagram Protocol (UDP)

The User Datagram Protocol is IP's other Host-to-Host Layer protocol (which corresponds to the Transport Layer of the OSI Reference Model). UDP provides a basic, low-overhead, data transmission known as "datagrams." To appreciate just how simple a mechanism UDP is, you need only compare RFC 768 (the original specification describing UDP's functionality, data structures, and mechanisms) to almost any other RFC. RFC 768 is svelte: a mere three pages in length. Many other RFCs require three pages just for their tables of content!

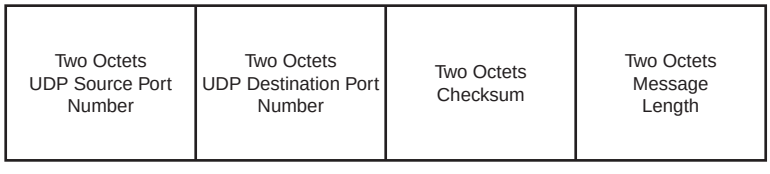
The simplicity of datagrams makes UDP inappropriate for some applications, but perfect for more sophisticated applications that can provide their own connection-oriented functionality. Other possible uses for UDP include exchanges of such data as forwarding routing table contents, system messages, network monitoring data, and so forth. These types of exchanges do not require flow control, acknowledgments, reordering, or any of the functionality that TCP provides.

Generally speaking, UDP is used in applications that are broadcast oriented, or message based, or that do not require the full reliability that TCP provides. More recently, UDP has also found use in real-time applications such as Voice over IP (VoIP) where the reliability and retransmissions of TCP for lost message segments is excessive overhead that is not needed and that would add to the overall delay in transmitting packets.

## UDP Header Structure

Figure 9.4 illustrates the structure of the IP header, as well as the sizes of its fields.

**FIGURE 9.4**
*The structure of a UDP header demonstrates its utter simplicity in form and function.*




The UDP protocol header has the following structure:

- *UDP Source Port Number*—The 16-bit source port is the connection number on the source computer. The source port and source IP address function as the packet's return address.

- *UDP Destination Port Number*—The 16-bit destination port is the connection number on the destination computer. The UDP Destination Port is used to forward the packet to the correct application after the packet arrives at the intended destination machine.
- *UDP Checksum*—The Checksum is a 16-bit error-checking field that is calculated based upon the contents of the segment. The destination computer performs the same mathematical function as the originating host. A discrepancy in the two calculated values indicates that an error has occurred during the transmission of the packet.
- *UDP Message Length*—The Message Length field is also 16-bits in length, and informs the destination computer of the size of the message. This provides another mechanism for the destination computer to use in determining the message's validity.

**Note**

It is important to note that though both TCP and UDP use 16-bit port numbers, they occupy different address spaces. That is, a TCP port number of 2000 is quite distinct from a UDP port number of 2000.

## What Does UDP Do?

Not much! UDP was deliberately designed to be an efficient and minimal transport protocol. This is directly reflected in its header structure. It contains just enough information to forward the datagram to the appropriate applications (that is, port numbers), and perform some error-checking.

UDP does not provide any of the more advanced functions that TCP supports. There are no timing mechanisms, flow control or congestion management mechanisms, acknowledgments, provisions for expedient delivery of urgent data, and so on. UDP, quite literally, makes a best-effort attempt to deliver a datagram. If that effort fails for any reason, the datagram is discarded and no attempt is made at retransmission.

## TCP Versus UDP

TCP and UDP are very different transport layer protocols that were designed to do different things. Their one commonality is that they both utilize IP as their Network Layer protocol. The major functional difference between TCP and UDP is reliability. TCP is reliable and UDP is a simple, best-effort datagram delivery mechanism. This fundamental difference implies that TCP is much more complex, and requires a substantial amount of overhead to function, whereas UDP is simple and efficient.

UDP is frequently criticized for being unreliable because it possesses none of TCP's reliability mechanisms. There is some truth to UDP being unreliable because UDP possesses none of TCP's mechanisms for acknowledging receipt of a datagram, resequencing datagrams received out of order, or even requesting a retransmission for a packet received damaged. In other words, there are no guarantees that a UDP datagram will ever reach its destination intact! Thus, UDP is best suited for small transmissions (that is, individual packets), whereas TCP is used to regulate a stream of transmitted data that spans multiple packets.

It is imperative to balance this description of UDP's unreliability with an explanation of its benefits. UDP is a minimal and frugal transport layer protocol. It is capable of operating much faster than TCP. Thus, it is well suited for the ever-emerging, time-sensitive applications such as Voice over IP (VoIP) and real-time video conferencing.

UDP is also well suited to many other functions in a network, such as transporting routing table updates between routers, or transporting network management/monitoring data. Such functions, although critical to the network's operability, could actually experience detrimental effects if forced to use the reliable delivery mechanisms of TCP. Thus, being an unreliable protocol does not mean that UDP is a useless protocol. It just means that it was designed to support different application types than TCP.

# Summary

The TCP/IP suite of protocols (including UDP and ICMP) have served the communications needs of a rapidly expanding base of users and applications for over 20 years. During that time, these protocols have been updated constantly to keep pace with technological innovation as well as the Internet's evolution from a semi-private research mechanism to a public, commercial infrastructure.

The commercialization of the Internet precipitated an unprecedented growth in the Internet's user population and a shift in its demographics. This, in turn, has created the need for more addresses and Internet Layer support for new types of service. Thus, IPv4's limitations have driven the development of a completely new version of the protocol. This new version is called IP Version 6 (IPv6) but is also commonly referred to as Internet Protocol: Next Generation (IPng). IPv6 is explored in more detail in Chapter 12.

CHAPTER 10

# The Internet Protocol

*by Karanjit S. Siyan, Ph.D.*

## In This Chapter

The Internet Protocol (IP) provides the first level of abstraction that provides a virtual view of the network where all nodes are treated as IP nodes. IP provides an abstract view of the network, a logical view wherein the network is viewed as an idealized network possessing properties that are described in this chapter.

Because of the abstraction layer that IP provides, protocols above the IP layers—such as the TCP and UDP protocols—can treat the network as an IP-only network. In reality, the network may not be an IP-only network; it may support other protocols besides TCP/IP. The IP node's network connections are identified by the 32-bit value known as the IP address. IP provides connectionless services to upper-layer services. The connectionless service is implemented using datagrams that contain the source and destination IP addresses and other parameters needed for IP operation. This chapter provides a strong conceptual understanding of how IP works in real-life networks.

# IP Abstraction

The IP layer relies on the underlying network hardware for its transmission. This means that IP datagram is encapsulated by the frames of the underlying network, such as Ethernet, Token Ring, or X.25.

The upper-layer protocols, such as TCP and UDP, need not be aware of the network hardware encapsulation and the underlying hardware. Upper-layer protocols may expect a certain quality of service, such as throughput and delay factors. These are called *Quality of Service* (QoS) parameters. The upper layers pass the QoS parameters along with the data to the IP layer. The IP layer may attempt to map the QoS parameters to services provided by the underlying network hardware. The underlying network hardware may or may not be able to supply this service.

Figure 10.1 shows that the IP host has three network connections. These connections are for Ethernet, Token Ring, and X.25 networks. In this example, IP runs on each of the three network connections. These network connections are identified by a unique 32-bit identifier called the *IP address*. The IP layer presents to the upper layers an abstraction for each of the three networks. This abstraction is independent of the physical attributes of the networks, such as address size, maximum transmission unit (MTU) size, network bandwidth, maximum data transmission rate, and so on. The MTU size is the maximum size of the data field for the underlying physical network. Because the data field contains IP datagrams for TCP/IP networks, the MTU size is also the maximum size of the IP datagram that can be carried on a physical network.

In the example network in Figure 10.1, the MTU size of the Ethernet network is 1500 bytes. The MTU size is 4440 for the IEEE 802.5, 4 Mbps Token Ring network; it is

17940 for the IEEE 802.5, 16 Mbps network. For the X.25 network, the MTU size is negotiated as an option and can range from 64 bytes to 4 KB. IP networks are required to process datagrams of at least 576 bytes in size. Because the smallest IP datagram that all routers must process is 576 bytes, the lower values are not used as MTU sizes in an X.25 network carrying TCP/IP data.

Regardless of the MTU size and speed differences of the networks, the IP layer translates these networks into a common logical IP network that is independent of physical differences.

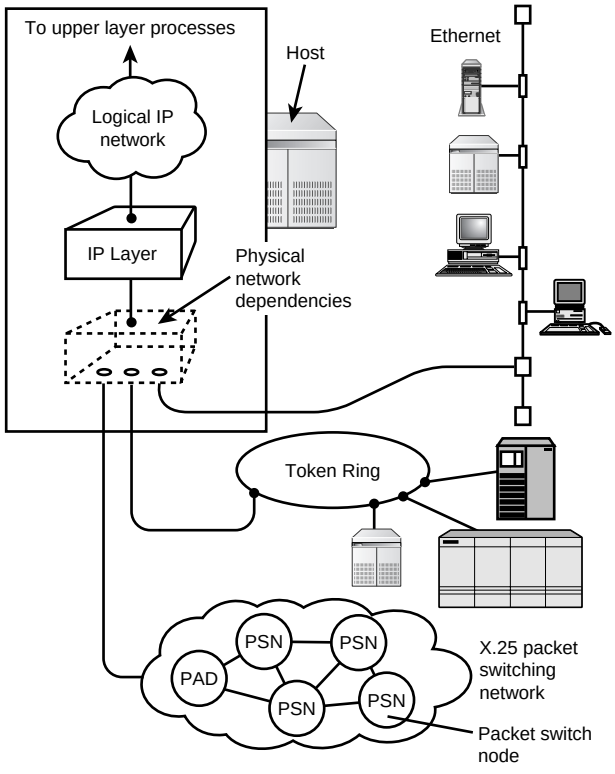



**FIGURE 10.1**
*IP abstraction. (Courtesy Learning Tree)*

The abstract IP network is connectionless. Each datagram is independently routed. It is possible, therefore, for successive datagrams to be transmitted along different paths (see Figure 10.2).

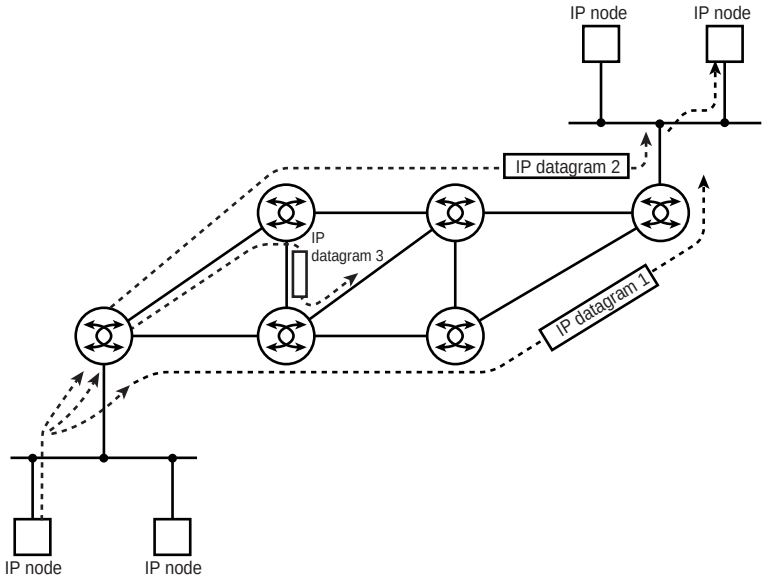



**FIGURE 10.2**
*Independent routing of IP datagrams.*

Because of the difference in time delays along the paths, datagrams arriving at the destination in a different sequence from that in which they were sent is possible. The IP layer makes no attempt to solve the problem of ensuring that datagrams are delivered to applications in the destination host in the correct order. Nor does it make any attempt to ensure that the datagrams are delivered reliably to the destination. Delivering datagrams in the proper order is called *sequencing*. The sequencing problem and the problem of reliable data delivery are solved by an upper-layer protocol such as TCP.

Because IP does not attempt to solve the sequencing and reliable data delivery problem, mapping IP onto a variety of network hardware is easy. Upper-layer protocols can add additional levels of reliability as needed by applications.

It is important to realize that the IP network in itself, without the assistance of any higher-level protocols, is inherently unreliable. The IP network uses a best-effort delivery method; that is, it tries to do the best job it can delivering the IP datagram, but it cannot guarantee delivery of the datagram. This is why IP is almost always used in conjunction with upper-layer protocols, which provides added functionality.

The IP service is connectionless, and each datagram is routed independently of others. Each datagram includes the sender and destination IP address. Intervening IP routers use the destination address to forward the IP datagram to the correct destination.

IP abstraction enables datagrams to be transmitted regardless of the MTU size of the network. IP nodes can fragment a large datagram when needed. The following sections explain this concept in greater detail.

## IP Datagram Size

IP was designed to accommodate a variety of network hardware types. As pointed out earlier, different networks have different restrictions on the maximum data size that can be transmitted by the Data Link Layer frame. Table 10.1 lists the MTU size of the different hardware types.

**TABLE 10.1** MTU Sizes

| *Network Type* | *MTU (octets)* |
|---|---|
| Ethernet | 1500 |
| IEEE 802.3 | 1492 |
| Token Ring | 4440 to 17940 (actual size depends on the Token Holding Time) |
| FDDI | 4352 |
| IEEE 802.4 | 8166 |
| SMDS | 9180 |
| X.25 | 1007 (ARPANET via DDN Standard X.25) 576 (for Public Data Networks, but can be increased by negotiation) |

Token Ring MTU sizes shown in Table 10.1 depend on the Token Holding Time (THT), which is approximately 8.58 ms (milliseconds). For a 4 Mbps transmission rate, the maximum Token Ring frame size is computed as follows:

Maximum Token Ring frame size (@ 4Mbps) $= 8.58 \text{ ms} \times 4 \text{ Mbps}$

$= 8.58 \times 4 \times 1024 \times 1024$

$= 36000$ bits

$= 36000 \div 8$ octets $= 4500$ octets

In the previous calculations 1 Mbps is actually 1 Kbps × 1 Kbps = 1024 × 1024 bps.

The largest Token Ring frame header is computed as follows:

| | |
|---|---|
| MAC Header | = 15 octets |
| Routing Information field | = 30 octets |
| LLC header | = 4 octets |
| SNAP header | = 5 octets |
| MAC trailer | = 6 octets |
| Total frame header | = 60 octets |
| MTU size for Token Ring @ 4 Mbps | = Maximum Frame size – Token Ring frame header size |
| | = 4500 – 60 = 4440 octets |

For Token Ring at 16 Mbps, the maximum Token Ring frame size is computed as follows:

| | |
|---|---|
| Maximum Token Ring frame size @ 16 Mbps | = 8.58 ms × 16 Mbps |
| | = 8.58 × 16 × 1024 × 1024 |
| | = 144000 bits |
| | = 144000 ÷ 8 octets = 18000 octets |
| MTU size for Token Ring @ 16 Mbps | = Maximum Frame size – Token Ring frame header size |
| | = 18000 – 60 = 17940 octets |

The FDDI MTU size in Table 10.1 is computed as follows:

| | |
|---|---|
| Maximum FDDI frame size using the 4B/5B encoding | = 4500 octets |

Largest FDDI frame header is computed as follows:

| | |
|---|---|
| MAC Header | = 16 octets |
| LLC header | = 4 octets |
| SNAP header | = 5 octets |
| MAC trailer | = 7 octets (approximately) |
| Total frame header | = 32 octets |

To accommodate future expansion, it is recommended that 148 octets be allocated for the FDDI frame header.

| | |
|---|---|
| MTU size for FDDI @ 100 Mbps | = Maximum Frame size – Adjusted FDDI frame header size |
| | = 4500 – 148 = 4352 octets |

The IP datagram can be up to 65536 bytes long. Most networks do not have an MTU size that is as big as 65536 bytes. The IP layer at the sender usually limits the size of the datagram to not exceed that of the MTU size of the local network. If an IP datagram must traverse intervening networks, the IP datagram cannot exceed the MTU size of the networks if the IP datagram is to be delivered in one whole piece. IP networks do not have any efficient mechanism to determine beforehand the MTU sizes of the intervening networks. If they did, the sender could send an IP datagram that does not exceed the MTU size of the intervening networks. Instead, IP networks depend on the fragmentation mechanism to deliver arbitrarily sized datagrams to any network.

## IP Fragmentation

If an IP datagram exceeds the MTU size of the network it must traverse, it cannot be sent in one whole piece. The IP datagram must be broken into smaller fragments that do not exceed the MTU size of the network. This process is called *fragmentation*. Each fragment of the original datagram is sent as an IP datagram. The IP header carries sufficient information to identify the fragment. This information is used by the destination host in assembling the fragments.

Figure 10.3 illustrates the process of sending an IP datagram across networks. In this example, network B's MTU size is smaller than the datagram size. Router R1 detects this fact and fragments the IP datagram into smaller IP datagrams that do not exceed the MTU size of network B. The fragmented datagrams are routed independently and arrive at host C. Though the MTU size of network C can accommodate the original datagram, the router R2 at the boundary between networks B and C makes no attempt at reassembling the fragments to form the original datagram. Reassembly of the IP datagram fragments is performed by the destination IP module and never by intervening routers.

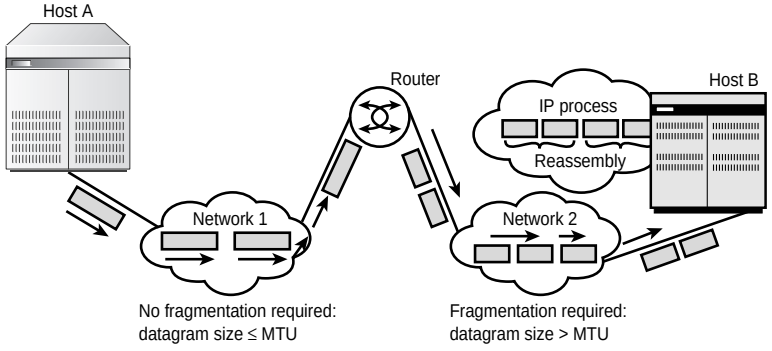



**FIGURE 10.3**
*IP datagram fragmentation. (Courtesy Learning Tree)*

What happens in Figure 10.3 when host C needs to reply to host A? Host C will adjust the datagram size to not exceed the MTU size of network C. The datagram from host C can be delivered to host A without any fragmentation. However, the intervening routers will not attempt to combine multiple datagrams into a larger datagram to increase the efficiency of transmission.

**Note**

In IP networks, only fragmented datagrams are assembled at the destination. Whole datagrams are never combined into a larger datagram to increase efficiency of transmission.

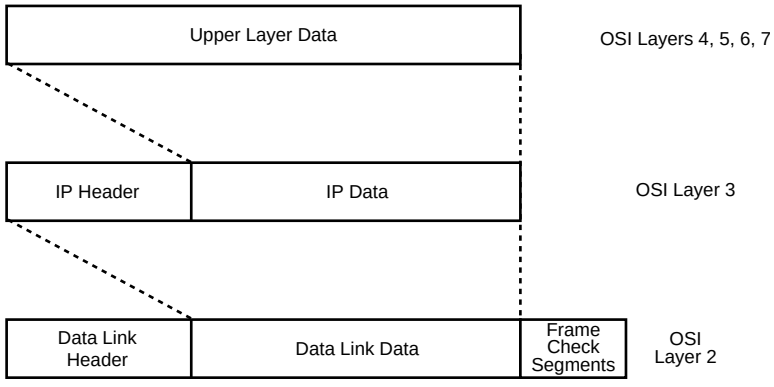
# IP Datagram Format

The IP datagram format contains an IP header and the IP data from upper-layer protocols (see Figure 10.4). The IP header is designed to accommodate the features of the IP layer.

From what you have learned about the IP layer in the previous sections, the IP header must contain at least the following information:

- IP address of source and destination. This is needed because IP is a connectionless protocol, and complete source and destination address information must be included with every datagram.
- Quality of service. This field is needed to specify the type of service expected from the underlying network.
- Fragmentation information. Because the IP datagram can get fragmented on a network boundary, there should be fields that help identify the fragment within the original datagram.
- Datagram size. IP datagrams can be of variable size. For this reason, there should be a field identifying the total length of the IP datagram.
- IP header size. IP datagrams can include optional fields that specify the IP options used for security, source routing, and so on. This causes the IP header size to be of variable length. For this reason the actual IP header size must be indicated in the IP header.

**FIGURE 10.4**
*IP datagram format.*




The following section discusses the details of the actual IP header format.

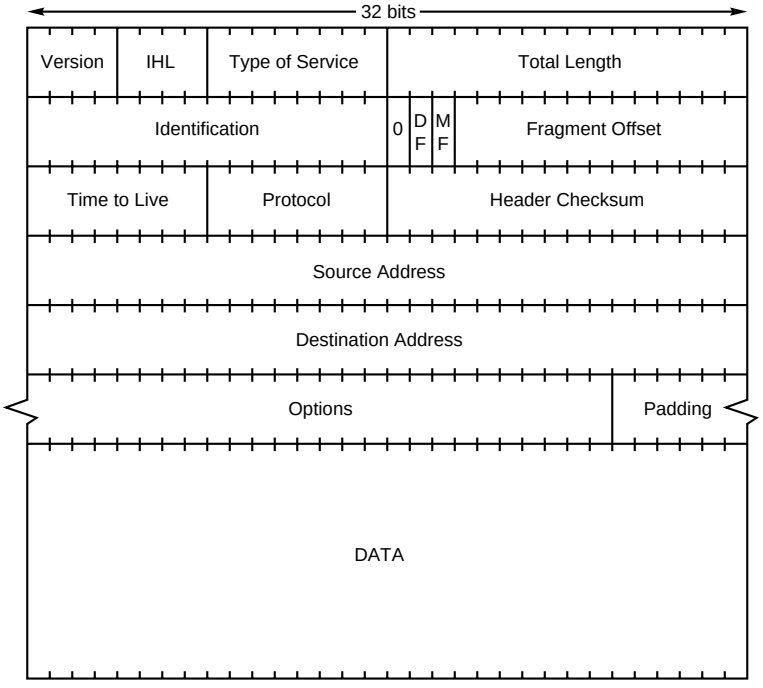
## IP Header Format

The IP header has a minimum length of 20 octets. If IP options are specified in the IP datagram, the header length may be longer. Figure 10.5 shows the fields and structure of the IP header.

**Note**

The IP header format is discussed in detail in RFC 791 and RFC 1122. RFC 1349 describes the use of the Type of Service (TOS) field in the IP header in detail.

**FIGURE 10.5**
*IP header.*




## The Version Field

In Figure 10.5, the *Version* field is four bits long and indicates the format of the IP header. This enables future IP packet structures to be defined. The current version number is 4, and this IP is called IPv4. The next generation of IP has a version number value of 6 and allows for 128-bit IP addresses. For this reason it is called IPv6. Chapter 12 contains more information about IPv6. Table 10.2 shows the other possible values of the version number field.

**Note**

Version number values of 7, 8, and 9 were assigned to experimental and proposed IP protocols to solve the limitation of the 32-bit IP address problem. These protocols have been superseded by Ipv6.

**TABLE 10.2** IP Version Number Values

| *IP Version* | *Meaning* |
|---|---|
| 0 | Reserved |
| 1–3 | Unassigned |
| 4 | IPv4 |
| 5 | Stream IP Datagram Mode (Experimental IP) |
| 6 | IPv6. Also called IPng. |
| 7 | TP/IX: The Next Internet |
| 8 | The "P" Internet Protocol |
| 9 | TUBA |
| 10–14 | Unassigned |
| 15 | Reserved |

The Version field is used by the sender, receiver, and any intervening routers to determine the format of the IP header. IP software is required to check the Version field to ensure that the IP header format is the one it expects. For example, if the IP software can only process version 4 datagrams, it will reject datagrams that have a different value than 4 in the Version field.

The following is an explanation of some of the entries that you see in Table 10.2:

- The Stream IP protocol is an experimental protocol used to provide end-to-end guaranteed service across an internet. Stream IP is described in RFC 1819.
- The TP/IX, "P" Internet protocol, and TUBA were all at one time contenders for the replacement of IPv4. They are no longer serious contenders and are relegated to historical status, because IPv6 is the approach that has been adopted as a replacement for IPv4.
- The "P" Internet protocol is a new protocol with advanced features and variable-length addresses. It was later merged with Simple Internet Protocol (SIP). SIP uses 64-bit addressing and was merged with IP Address Encapsulation (IPAE), which describes how the transition between IPv4 and longer addressing would be achieved. SIP is described in RFC 1710.
- The TP/IX protocol (refer to Table 10.2) is an older protocol that became the basis of the Common Architecture for the Internet (CATNIP) that integrates IPv4 with Internet Packet Exchange (IPX) and the OSI Connectionless Network Protocol (CLNP). The CATNIP protocol is primarily of historical interest. CATNIP is described in RFC 1707.

- TUBA stands for TCP and UDP with Bigger Addresses. TUBA is based on CLNP. TUBA is described in RFC 1347, RFC 1526, and RFC 1561.

## The Internet Header Length Field

The *Internet Header Length* (IHL) field is the length of the header in 32-bit words. This field is 4 bits long and is required because the IP header contains a variable-length options field. All other fields have a fixed length. The options field is padded, if necessary, to be a multiple of 32-bit words. Typically, most IP headers do not have any options listed. Excluding the options field, the IP header length is 20 octets. The typical IHL value is, therefore, five 32-bit words. The maximal internet header is 60 octets and specifies the use of IP options. In this case the IHL value will be fifteen 32-bit words.

## The Type of Service Field

The *Type of Service* (ToS) field informs the networks of the QoS desired, such as precedence, delay, throughput, and reliability. The meaning of this 8-bit field is shown in Figure 10.6.

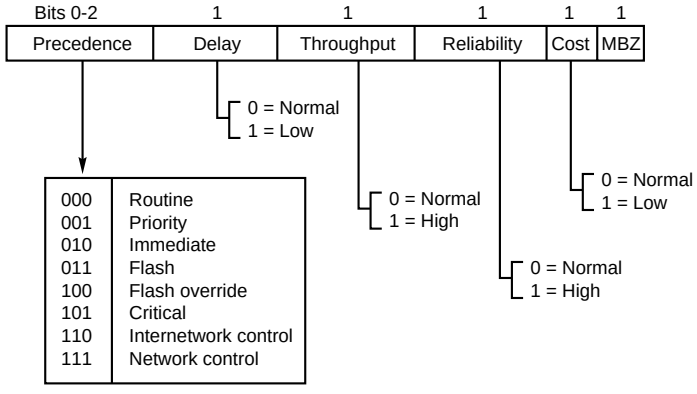



**FIGURE 10.6** *Type of Service field for IP packets.*

The first three bits deal with the Precedence value associated with the IP datagram. The *Precedence* field reflects the military origin of IP networks (the Department of Defense, which is discussed presently). The following are the meanings of some of the precedence values:

- Flash. ASAP (As Soon As Possible). Maximum priority on all circuits.
- Immediate. Within four hours.
- Priority. Same day.
- Routine. Within one day.

The Precedence values are 3 bits long. Table 10.3 lists the possible values and meanings of the precedence field.

**TABLE 10.3** Precedence Values

| *Precedence* | *Meaning* |
|---|---|
| 000 | Routine |
| 001 | Priority |
| 010 | Immediate |
| 011 | Flash |
| 100 | Flash-override |
| 101 | Critical |
| 110 | Internetwork control |
| 111 | Network control |

The Precedence field was intended for Department of Defense applications of the Internet protocols. The use of non-zero values in this field is outside the scope of the IP standard specification. The IP layer must provide a means for the transport layer to set the TOS field of every datagram that is sent; the default is all 0 bits. The IP layer should pass received TOS values up to the transport layer.

**Note**

Vendors should consult the Defense Information Systems Agency (DISA) for guidance on the IP Precedence field and its implications for other protocol layers. Vendors should note that the use of precedence most likely will require that its value be passed between protocol layers in much the same way as the TOS field is passed.

The 4 bits that follow the Precedence field are a code for types of service. These codes are interpreted as shown in Table 10.4.

**TABLE 10.4** Types of Service Codes

| *TOS code* | *Meaning* |
|---|---|
| 1000 | Minimum delay |
| 0100 | Maximum throughput |
| 0010 | Maximum reliability |
| 0001 | Minimum monetary cost |
| 0000 | Normal service |

The last bit of the TOS field must be zero. This 1-bit field is called the must be zero (MBZ) field and is reserved for future use. IP protocols participating in Internet experiments may make use of this field. Routers and recipients of datagrams ignore the value of this field. This field is copied on fragmentation, which results in all the IP datagram fragments having the same type of service.

Most IP implementations and routing protocols—RIP, HELLO, and so on—ignore the Type of Service field. Although the TOS field has been little used in the past, it is expected to play an increasing role with routing protocols such as OSPF that could make use of the TOS field. The TOS field is expected to be used to control two aspects of router operations: routing and queuing algorithms. The TOS field also may be mapped into link-layer sharing of serial lines by different classes of TCP traffic.

The TOS field can be used as a hint to the routing algorithms to select the most appropriate path. If a router knows of more than one possible route to a destination, it can use the TOS field to select the route that more closely matches the TOS field value. For example, a router may know of two paths to a destination: one with a high-delay and high-throughput connection such as a satellite link, and one with a low-delay and low-throughput connection such as a leased line. If the application traffic carries interactive user keystrokes, the requirement is one of low delay but not very high throughput. Consequently, the router should select the leased-line connection for routing this type of traffic. If the application traffic requires a high volume of file data transfer, throughput is important and the high delay is not so critical. In this case, the satellite link with high throughput is the best route.

**Note**

RFC 1349 describes the use of the TOS field in detail.

## The Total Length Field

The *Total Length field* contains the length of the IP header and data in bytes. This field is 16 bits long, which limits the datagram to a maximum size of 65535 octets. This number is the maximum value that can be represented by 16 bits and consists of 16 ones:

1111111111111111

The preceding number is equal to $2^{16}$–1, a decimal value of 65535.

Datagrams that are as long as 65535 octets are impractical for most hosts and networks. Most networks have an MTU size that is far less than 65535 octets. It also is not efficient to design the communication buffers where IP datagrams can be as large as 65535 octets. Typical datagram size for most networks and hosts seldom exceeds 16 KB.

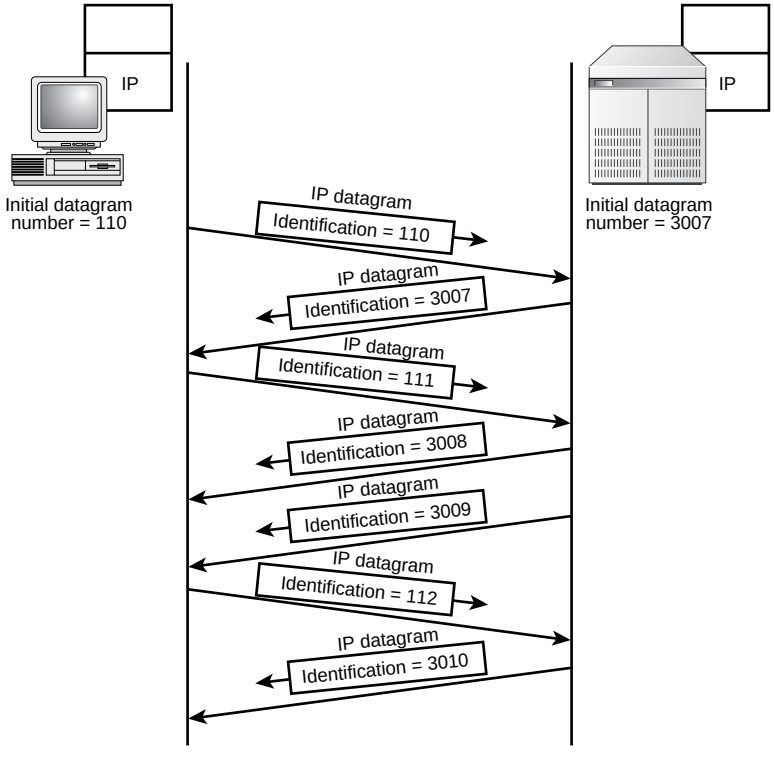
All IP nodes must be prepared to receive datagrams of at least 576 bytes whether they arrive whole or in fragments. The minimum size of 576 bytes is based on 512 bytes of data plus 64 bytes of protocol overhead.

Datagrams that are larger than the MTU size of the underlying network must be fragmented because they cannot be otherwise sent through the underlying network whose MTU size is too small to transmit the datagram.

## The Identification Field

The *Identification field* is set uniquely for each datagram, and is the datagram number. Each sending side numbers the datagram starting with some initial value (see Figure 10.7). Most IP implementations use a global memory counter that is incremented with each IP datagram that is sent. If you capture network traffic from hosts that have been up for some time, such as server computers, you will see Identification field values that start from a seemingly arbitrary initial value. This is because the hosts have been sending IP datagrams in previous network sessions, and the Identification field is incremented for each IP datagram that is sent.

**FIGURE 10.7**
*Identification field values.*



The identification field is 16 bits long. This allows for datagrams to be numbered from 0 to 65535.

The Identification field is used primarily to identify IP datagram fragments as belonging to a particular original IP datagram. The Identification field is used in conjunction with the fragment flags Don't Fragment (DF), More Fragments (MF), and Fragment Offset fields to reassemble the datagram. These flags are discussed in the sections that follow.

## Fragmentation Flags and Fragment Offset Fields

The DF flag set to 1 means that the datagram should not be fragmented. This may be done if the sender knows that the receiver may not have sufficient capability to reassemble the fragments into the original datagram. An example of this is a bootstrap program running in the ROM of a computer that downloads data from a server machine. If the entire data is designed to fit inside a datagram, the sender can set the DF flag to 1. Another reason for setting the DF flag is in the situation where the sender wants to eliminate the delays caused by reassembly of IP datagram fragments at the receiver. If the DF flag is set to 0, it indicates that a router or host may fragment the IP datagram.

The MF flag set to 1 indicates to the receiver that more fragments are to come. An MF set to 0 indicates that it is the last fragment. For complete IP datagrams, the MF flag will always be set to 0 to indicate that there are no more fragments to this datagram.

In the ideal situation, the MTU size of all the networks that an IP datagram needs to traverse should have an MTU size that is larger than the largest IP datagram to be transmitted by a sender. In this case, there is no need to perform any fragmentation. Even if the smallest MTU size along a path is known, you should realize that successive datagrams may follow a different path and, therefore, encounter a different smallest MTU size. (This is because each IP datagram is routed independently of other datagrams.)

How is an IP datagram fragmented? Consider the situation in Figure 10.8 where a host on network A is sending IP datagrams to a host on network C. There is an intervening network B that the IP datagrams must traverse. The MTU sizes of networks A, B, and C are 1500, 525, and 4470 octets, respectively. Router Ra joins networks A and B, and router Rc joins networks B and C. Typically host A will send IP datagrams that fit inside the MTU size of network A. In this example, assume that an IP datagram with an Identification value of 5 and a size of 1500 octets is sent by host A. When the router Ra encounters this IP datagram, it will fragment the IP datagram before it can forward the datagram to network B because the MTU size of network B is smaller than the datagram size. The fragment size is selected so that it will fit inside the MTU size of network B (525 bytes). The *offset*, the start of the data field relative to the original unfragmented data, is placed in the Fragment Offset field of each IP datagram fragment.

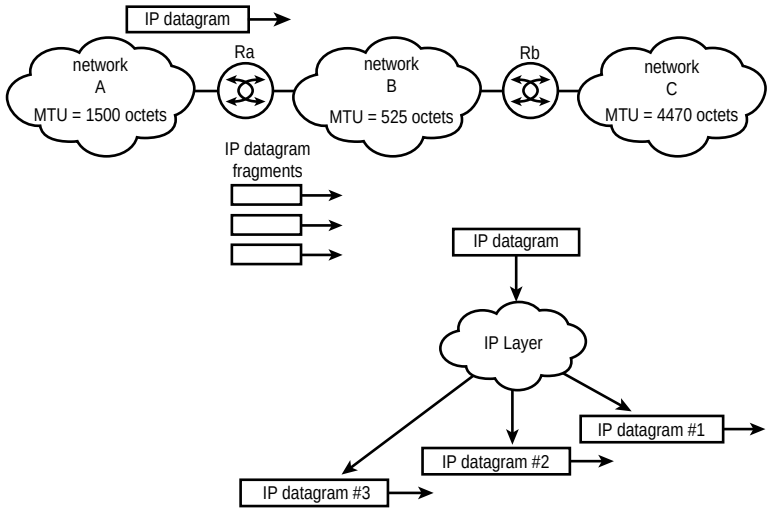



**FIGURE 10.8**
*Fragmentation example.*

The Fragment Offset field indicates the position of the fragment's data relative to the start of the original datagram. (The specifics of the value of this field are discussed shortly.) The Fragment Offset field is 13 bits long. If you study the IP header format in Figure 10.5, you will notice that only 13 bits are used because some of the other bits are used for the DF and MF flags.

Because the IP datagram can be up to 65535 octets long, this field is not sufficiently long to describe offset values greater than 8192 octets. For this reason, the Fragment Offset field is extended by appending three 0 bits:

fragment_offset_value000

The resulting fragment offset value is always a multiple of 8 bytes. Therefore, the fragment offset always describes fragments in units of 8-byte groups. The fragment offset value must be multiplied by 8 to get the actual offset of the fragment. This means that all fragments except the very last must be a multiple of 8. In addition the fragments must fit inside the MTU size of the network. In the example of the 1500-octet IP datagram, consider the situation where you decided to fragment as shown next because the MTU size of the network is too small:

| | |
|---|---|
| Fragment 1 size | = 525 octets |
| Fragment 2 size | = 525 octets |
| Fragment 3 size | = 450 octets |
| IP datagram size | = 1500 octets |

However, 525 bytes is not a multiple of 8. You also must take into account the size of the IP header (a minimum of 20 octets), which must also fit within the MTU size. With an MTU size of 525, subtracting the typical size of an IP header of 20 octets yields 505 octets for the fragment size. This number also is not a multiple of 8. The next smallest fragment size that is divisible by 8 is 504. Therefore, you can select the IP fragment size to be 504 octets. Figure 10.9 shows the breakdown of the original IP datagram into the following fragments:

| | |
|---|---|
| Fragment 1 | = 504 octets |
| Fragment 2 | = 504 octets |
| Fragment 3 | = 492 octets |
| IP datagram | = 1500 octets |

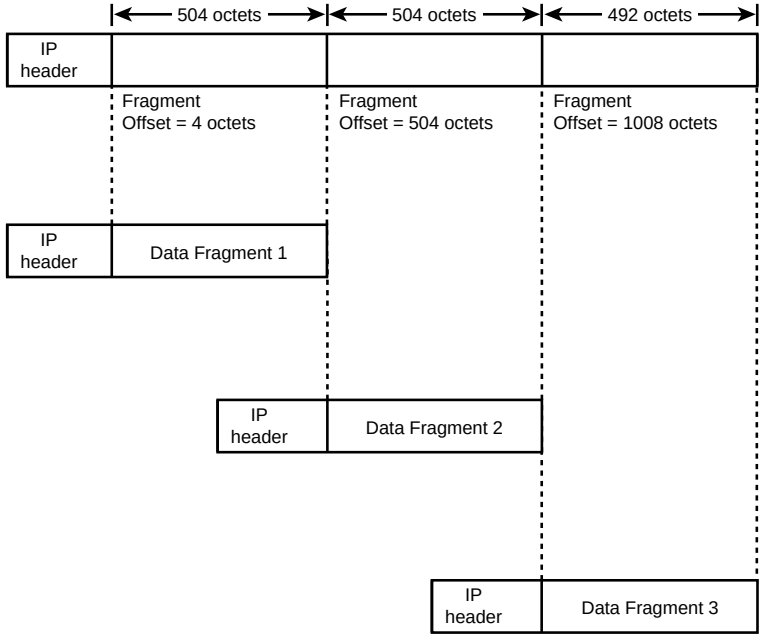



**FIGURE 10.9**
*Fragmentation of the original IP datagram.*

Each fragment must have its own IP header. The Identification field value in the fragmented IP datagrams have the value belonging to the original fragmented IP datagram. This identifies the IP fragment as belonging to a specific original IP datagram. The flag settings and fragment offset values of the fragmented IP datagrams (see Figure 10.10) are as follows:

**Fragment 1:**

| | |
|---|---|
| Total Length | = 504 octets |
| Identification | = 5 |
| DF flag | = 0 |
| MF | = 1 |
| Fragment Offset | = 0 |

**Fragment 2:**

| | |
|---|---|
| Total Length | = 504 octets |
| Identification | = 5 |
| DF flag | = 0 |
| MF | = 1 |
| Fragment Offset | = 63 8-octet words (504 octets) |

**Fragment 3:**

| | |
|---|---|
| Total Length | = 492 octets |
| Identification | = 5 |
| DF flag | = 0 |
| MF | = 0 |
| Fragment Offset | = 126 8-octet words (1008 octets) |

For the receiver to reassemble the IP datagram, it must obtain all fragments starting with the IP datagram fragment with a fragment offset value of 0, and going onto the highest fragment offset value. The receiver uses the size of each fragment (Total Length field of the IP datagram header), the fragment offset value, and the MF flag to determine whether it has received all the fragments. How does a receiver know that the last fragment has been received? The last fragment has an MF flag value set to 0. The first fragment is identified by an MF flag set to 1 and a fragment offset set to 0. All other intermediate fragments will have an MF flag set to 1 and a non-zero fragment offset value.

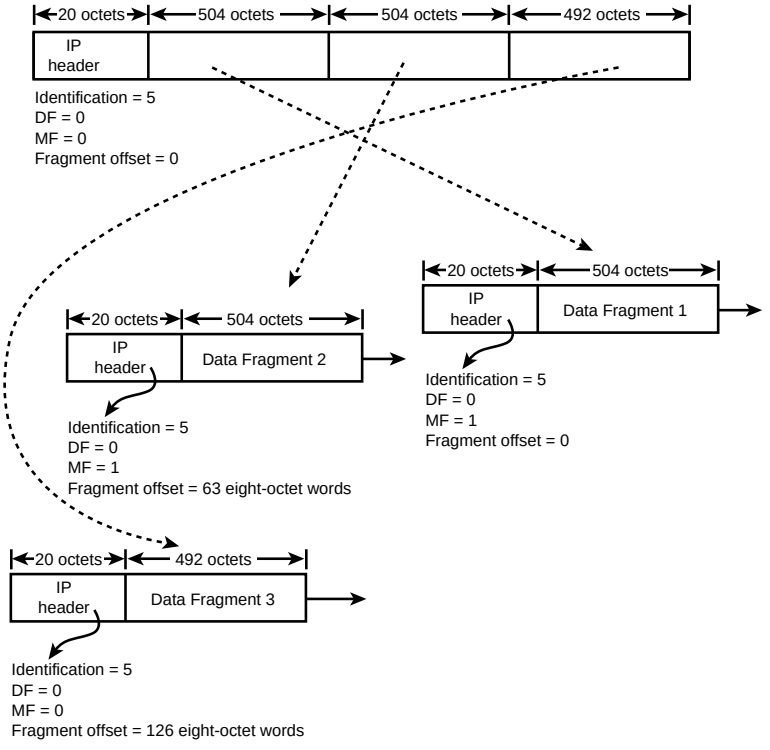



**FIGURE 10.10**
*IP datagram fragments.*

After an IP datagram has been fragmented, the individual pieces travel to the destination along potentially separate paths. The IP fragments are reassembled at the receiver and never at an intervening router. This fact leads to the following issues:

- IP fragments travel from their point of fragmentation to the destination.
- Despite intervening networks having a larger MTU size than the fragment size, the fragments will be carried in small size packets, which results in poor protocol efficiency.
- Excessive fragmentation leads to excessive network traffic.

If a fragment is lost, the original datagram cannot be reassembled. Despite successful transmission of the remaining fragments, the original datagram must be discarded. In this case, upper-layer protocols such as TCP ensure that the original datagram is present, but this datagram will be fragmented again at the point where the IP datagram exceeds the MTU size.

The receiving IP node starts a reassembly timer upon receiving the first IP fragment. If this timer expires before all the other fragments arrive, the IP node discards the remaining fragments even though they were received correctly. As the number of fragments increases, so does the possibility of losing an IP datagram. This is because a loss of a single fragment results in the loss of the entire datagram. When the reassembly timer expires, an Internet Control Message Protocol (ICMP) message is sent to the sender announcing that the reassembly timer has expired. A monitoring device such as a protocol analyzer can be used to alert you to this problem.

The decision to reassemble the IP datagram at the receiver, while creating the problems described previously, has the advantage of keeping the router implementation simple. The IP routers do not have to be concerned about storing and reassembling the IP fragments. If there are several networks with a small MTU size, reassembling the IP fragments at intervening routers would be somewhat wasteful because the reassembled IP datagram would have to be fragmented before traversing a network with a small MTU size.

## The TTL field

The *Time To Live* (TTL) is measured in seconds and represents the maximum time an IP datagram can live on the network. It should be decremented at each router by the amount of time taken to process the packet. When the TTL field becomes 0, the TTL timer expires. The intent is that TTL expiration causes a datagram to be discarded by a router, but not by the destination host. Hosts acting as routers by forwarding datagrams must follow the previously discussed behavior for TLL routing. The TTL field has two functions:

- Limits the lifetime of TCP segments
- Terminates Internet routing loops

When the TTL timer expires, an ICMP message announcing this fact is sent to the source.

Although TTL is measured in seconds, it is difficult for routers to estimate the exact transit time for IP datagrams on a network segment. Routers could note the time an IP datagram spends inside the router by noting the time when a datagram arrives and the time when it leaves the router. Because of the extra processing required to do so, many routers do not attempt to estimate the time an IP datagram spends inside a router and simply decrease the TTL value by 1. Because of this, the TTL field has some attributes of a hop count. Some earlier implementers mistakenly set the TTL value to 16 because 16 is infinity for RIP (Routing Information Protocol). TTL is independent of RIP metrics, however.

Other considerations for TTL fields are the following:

- A host must not send a datagram with a TTL value of 0, and a host must not discard a datagram just because it was received with a TTL less than 2.
- An upper-layer protocol may want to set the TTL to implement an expanding scope search for some Internet resource. This is used by some diagnostic tools and is expected to be useful for locating the "nearest" server of a given class using IP multicasting, for example. A particular transport protocol also may want to specify its own TTL boundary on maximum datagram lifetime.
- A fixed value must be at least big enough for the Internet *diameter*, the longest possible path. A reasonable value is about twice the diameter, which enables continued Internet growth.
- The IP layer must provide a means for the transport layer to set the TTL field of every datagram that is sent. When a fixed TTL value is used, that value must be configurable. Unfortunately, most implementations do not enable the initial TTL value to be set. A default value of 32 or 64 is very common.

## The Protocol Field

The *Protocol field* is used to indicate the upper-layer protocol that is to receive the IP data. The Protocol field is used for multiplexing/demultiplexing of data to upper-layer protocols. For example, TCP has a Protocol field value of 6, UDP has a value of 17, and ICMP has a value of 1. When IP sees a Protocol field value of 6, it knows that the IP header encapsulates TCP data that must be delivered to the TCP module. If IP sees a Protocol field value of 17, it knows that this must be delivered to the UDP module. Similarly, when IP sees a Protocol field value of 1, it knows that this must be delivered to the ICMP module.

Table 10.5 shows the Protocol field values for IP.

**TABLE 10.5** Protocol IDs

| *Protocol ID* | *Abbreviation* | *Meaning* |
|---|---|---|
| 0 | | (Reserved) |
| 1 | ICMP | Internet Control Message |
| 2 | IGMP | Internet Group Management |
| 3 | GGP | Gateway-to-Gateway |
| 4 | IP | IP in IP (encapsulation) |
| 5 | ST | Stream |
| 6 | TCP | Transmission Control |

**TABLE 10.5** Continued

| *Protocol ID* | *Abbreviation* | *Meaning* |
|---|---|---|
| 7 | UCL | UCL |
| 8 | EGP | Exterior Gateway Protocol |
| 9 | IGP | Any private interior gateway |
| 10 | BBN-RCC-MON | BBN RCC Monitoring |
| 11 | NVP-II | Network Voice Protocol |
| 12 | PUP | PUP |
| 13 | ARGUS | ARGUS |
| 14 | EMCON | EMCON |
| 15 | XNET | Cross Net Debugger |
| 16 | CHAOS | Chaos |
| 17 | UDP | User Datagram |
| 18 | MUX | Multiplexing |
| 19 | DCN-MEAS | DCN Measurement Subsystems |
| 20 | HMP | Host Monitoring |
| 21 | PRM | Packet Radio Measurement |
| 22 | XNS-IDP | XEROX NS IDP |
| 23 | TRUNK-1 | Trunk-1 |
| 24 | TRUNK-2 | Trunk-2 |
| 25 | LEAF-1 | Leaf-1 |
| 26 | LEAF-2 | Leaf-2 |
| 27 | RDP | Reliable Data Protocol |
| 28 | IRTP | Internet Reliable Transaction |
| 29 | ISO-TP4 | ISO Transport Protocol Class4 |
| 30 | NETBLT | Bulk Data Transfer Protocol |
| 31 | MFE-NSP | MFE Network Services Protocol |
| 32 | MERIT-INP | MERIT Internodal Protocol |
| 33 | SEP | Sequential Exchange Protocol |
| 34 | 3PC | Third Party Connect Protocol |
| 35 | IDPR | Inter-Domain Policy Routing Protocol |
| 36 | XTP | XTP |
| 37 | DDP | Datagram Delivery Protocol |

**TABLE 10.5** Continued

| *Protocol ID* | *Abbreviation* | *Meaning* |
|---|---|---|
| 38 | IDPR-CMTP | IDPR Control Message Transport Protocol |
| 39 | TP++ | TP++ Transport Protocol |
| 40 | IL | IL Transport Protocol |
| 41 | SIP | Simple Internet Protocol |
| 42 | SDRP | Source Demand Routing Protocol |
| 43 | SIP-SR | SIP Source Route |
| 44 | SIP-FRAG | SIP Fragment |
| 45 | IDRP | Inter-Domain Routing Protocol |
| 46 | RSVP | Reservation Protocol |
| 47 | GRE | General Routing Encapsulation |
| 48 | MHRP | Mobile Host Routing Protocol |
| 49 | BNA | BNA |
| 50 | SIPP-ESP | SIPP Encap Security Payload |
| 51 | SIPP-AH | SIPP Authentication Header |
| 52 | I-NLSP | Integrated Net Layer Security TUBA |
| 53 | SWIPE | IP with Encryption |
| 54 | NHRP | NBMA Next Hop Resolution Protocol |
| 55–60 | | (Unassigned) |
| 61 | | (Any host internal protocol) |
| 62 | CFTP | CFTP |
| 63 | | (Any local network) |
| 64 | SAT-EXPAK | SATNET and Backroom EXPAK |
| 65 | KRYPTOLAN | Kryptolan |
| 66 | RVD | MIT Remote Virtual Disk Protocol |
| 67 | IPPC | Internet Pluribus Packet Core |
| 68 | | (Any distributed file system) |
| 69 | SAT-MON | SATNET Monitoring |
| 70 | VISA | VISA Protocol |
| 71 | IPCV | Internet Packet Core Utility |

**TABLE 10.5** Continued

| *Protocol ID* | *Abbreviation* | *Meaning* |
|---|---|---|
| 72 | CPNX | Computer Protocol Network Executive |
| 73 | CPHB | Computer Protocol Heart Beat |
| 74 | WSN | Wang Span Network |
| 75 | PVP | Packet Video Protocol |
| 76 | BR-SAT-MON | Backroom SATNET Monitoring |
| 77 | SUN-ND | SUN ND PROTOCOL-Temporary |
| 78 | WB-MON | WIDEBAND Monitoring |
| 79 | WB-EXPAK | WIDEBAND EXPAK |
| 80 | ISO-IP | ISO Internet Protocol |
| 81 | VMTP | VMTP |
| 82 | SECURE-VMTP | SECURE-VMTP |
| 83 | VINES | VINES |
| 84 | TTP | TTP |
| 85 | NSFNET-IGP | NSFNET-IGP |
| 86 | DGP | Dissimilar Gateway Protocol |
| 87 | TCF | TCF |
| 88 | IGRP | IGRP |
| 89 | OSPFIGP | OSPFIGP |
| 90 | Sprite-RPC | Sprite RPC Protocol |
| 91 | LARP | Locus Address Resolution Protocol |
| 92 | MTP | Multicast Transport Protocol |
| 93 | AX.25 | AX.25 Frames |
| 94 | IPIP | IP-within-IP Encapsulation Protocol |
| 95 | MICP | Mobile Internetworking Control Pro. |
| 96 | SCC-SP | Semaphore Communications Sec. Pro. |
| 97 | ETHERIP | Ethernet-within-IP Encapsulation |
| 98 | ENCAP | Encapsulation Header |

**TABLE 10.5** Continued

| *Protocol ID* | *Abbreviation* | *Meaning* |
|---|---|---|
| 99 | | (Any private encryption scheme) |
| 100 | GMTP | GMTP |
| 101–254 | | (Unassigned) |
| 255 | | (Reserved) |

## Header Checksum Field

The *Header Checksum* is used for the IP header only. The 1's complement of each 16-bit value making up the header is added (excluding the Header Checksum field). Then the 1's complement of the sum is taken. This field is recomputed at each router because the TTL field is decremented at the router, and this results in the modification of the IP header.

It is quite simple to compute the checksum, and experimental evidence indicates that it is adequate in detecting errors in transmitting the datagram.

## Source and Destination IP Addresses

The *Source Address* and *Destination Address* are the 32-bit IP addresses of the source and destination nodes. These are sent in every IP datagram because the IP network is a connectionless network, and each IP datagram must include the sender and destination IP addresses.

The routers use the destination IP address value to perform the routing for each IP datagram.

# IP Options

IP implements facilities for indicating the security of a datagram, source routing information, and timestamp information. Because these facilities are infrequently used, they are implemented as optional fields called *IP options*.

The IP options are as follows:

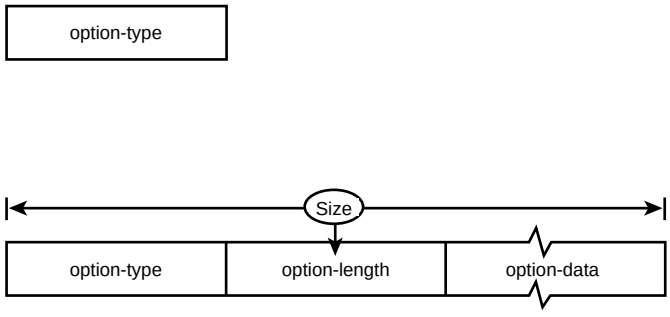
- Security
- Record Route
- Strict Source Routing
- Loose Source Routing
- Internet Timestamp

The IP options must be handled by all IP nodes, and may optionally appear in IP datagrams near the end of the IP header. Their transmission in any particular datagram is optional, but their implementation is not. For example, in environments that need a high level of security, the Security option may be required in all IP datagrams.

The option field is variable in length, and values must be multiples of 32-bit words. Zero pad values may be added to make the options field size multiples of 32-bit words. The two formats for IP option values (see Figure 10.11) are

- Option format 1. A single octet of option-type.
- Option format 2. An option-type octet, an option-length octet, and the actual option-data octets. The option-length octet contains the overall size of this option including the octet-type, option-length, and option-data fields.

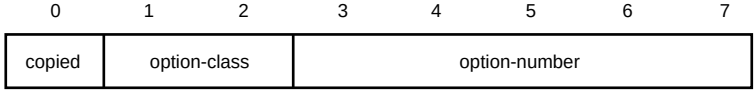
**FIGURE 10.11**
*Option formats.*




The option-type octet is the first octet that occurs in an option specification. In option format 1 the option-type octet is the only octet, and in option format 2 it is followed by two other fields. The option-type consists of the following three fields (see Figure 10.12):

- Copied flag (1 bit)
- Option-class (2 bits)
- Option-number (5 bits)

**FIGURE 10.12**
*Option octet.*




The Copied flag controls how routers process the options field during fragmentation. When set to 1, the Copied flag indicates that the options should be copied into all IP datagram fragments. When set to 0, the Copied flag indicates that the options should be copied into the first IP fragment and not into all IP datagram fragments.

The Option-class field is 2 bits long and can have a value from 0 to 3. The meanings of these values are listed in Table 10.6.

**TABLE 10.6** Option-Class Field Values in Option-Type Octet

| *Code* | *Meaning* |
|---|---|
| 0 | Network control |
| 1 | Reserved for future use |
| 2 | Debugging and measurement |
| 3 | Reserved for future use |

The class field is used to determine the general category of the option. The categories are for network control and debugging. The Option-number field is 5 bits long and enables the specification of up to 32 options for a given option category. Table 10.7 lists the different options that have been defined.

**TABLE 10.7** IP Options

| *Option-class* | *Option-number* | *Length* | *Meaning* |
|---|---|---|---|
| 0 | 0 | - | End of Option list. This option occupies only 1 octet and has no length octet. It is used if options do not end at an end of header. |
| 0 | 1 | - | No Operation. This option occupies only 1 octet; it has no length octet. It is used to align octets in a list of options. |
| 0 | 2 | 11 | Basic Security. This is used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DoD requirements. |
| 0 | 3 | variable | Loose Source Routing. Used to route the IP datagram based on information supplied by the source. |
| 0 | 5 | variable | Extended Security. Used for specifying additional security. |
| 0 | 7 | variable | Record Route. Used to trace the route an IP datagram takes. |
| 0 | 8 | 4 | Stream ID. Used to carry the 16-bit SATNET stream identifier through networks that do not support the stream concept. It is now obsolete. |

**TABLE 10.7** Continued

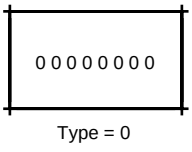
| *Option-class* | *Option-number* | *Length* | *Meaning* |
|---|---|---|---|
| 0 | 9 | variable | Strict Source Routing. Used to route the IP datagram based on information supplied by the source. |
| 2 | 4 | variable | Internet Timestamp. Used to record time-stamps along the route. This is currently the only option defined for the debugging and measurement category of option-class. |

The following sections explain the different IP option types in greater detail.

## End of Option List and the No Operation Options

The End of Option List and the No Operation options are the only single octet options. The End of Option list (refer to Table 10.7) is a 1-octet option consisting of all zeros (see Figure 10.13).

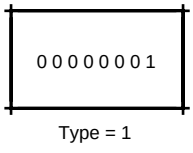
**FIGURE 10.13**
*The End of Option list.*




This option indicates the end of the option list, which may not coincide with the end of the internet header according to the IHL field. This option is used at the end of all options, and not at the end of each option. This option need only be used if the end of the options would not otherwise fall on a 32-bit boundary.

The No Option list (refer to Table 10.7) is a 1-octet option consisting of the following bit pattern (see Figure 10.14).

**FIGURE 10.14**
*The No Option list.*



The No Operation option may be used between options to align the beginning of a subsequent option on a 32-bit boundary.

## The Security Option

The Security option provides a way for hosts to send security, compartmentation, handling restrictions, and closed user group parameters. The two types of Security options are

- Basic Security. Used to indicate security-level designations Confidential, Unclassified, Secret, and so on.
- Extended Security. Used to specify additional security per the needs of a military organization.

These are discussed in the following sections.

### Basic Security Option

Figure 10.15 shows the Basic Security option format. It is characterized by an octet-type code of 130. The codes define protection authorities that have various rules concerning the treatment of the IP datagram. These protection authorities include the U.S. National Security Agency (NSA), Central Intelligence Agency (CIA), and Department of Energy (DOE).

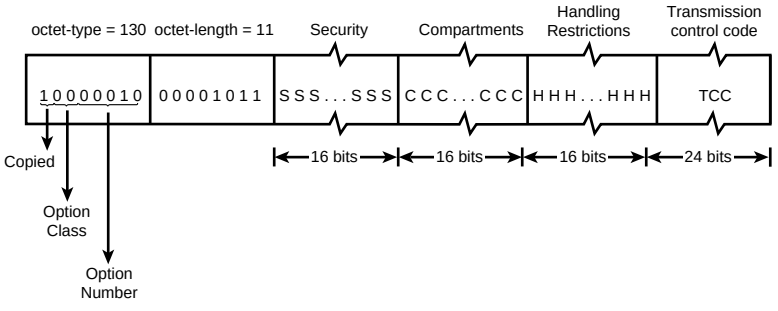



**FIGURE 10.15**
*Basic Security Option format.*

The Security field is 16 bits long and specifies one of 16 levels of security, 8 of which are reserved for future use. The security code values are described in Table 10.8. Some of the code values have meanings that are internal to the various protection authorities.

**TABLE 10.8** Security Code Values for IP Options

| *Code* | *Meaning* |
|---|---|
| 00000000 00000000 | Unclassified |
| 11110001 00110101 | Confidential |
| 01111000 10011010 | EFTO |
| 10111100 01001101 | MMMM |
| 01011110 00100110 | PROG |
| 10101111 00010011 | Restricted |
| 11010111 10001000 | Secret |
| 01101011 11000101 | Top Secret |
| 00110101 11100010 | Reserved for future use |
| 10011010 11110001 | Reserved for future use |
| 01001101 01111000 | Reserved for future use |
| 00100100 10111101 | Reserved for future use |
| 00010011 01011110 | Reserved for future use |
| 10001001 10101111 | Reserved for future use |
| 11000100 11010110 | Reserved for future use |
| 11100010 01101011 | Reserved for future use |

The Compartments field is 16 bits long, and an all-zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

The Handling Restrictions field is 16 bits long. The values represent control and release markings and are alphanumeric digraphs that are defined in the Defense Intelligence Agency Manual (DIAM) 65-19, "Standard Security Markings."

The Transmission Control Code (TCC) field is 24 bits long and is used to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs and are available from the Defense Communication Agency (DCA).

## Extended Security Option

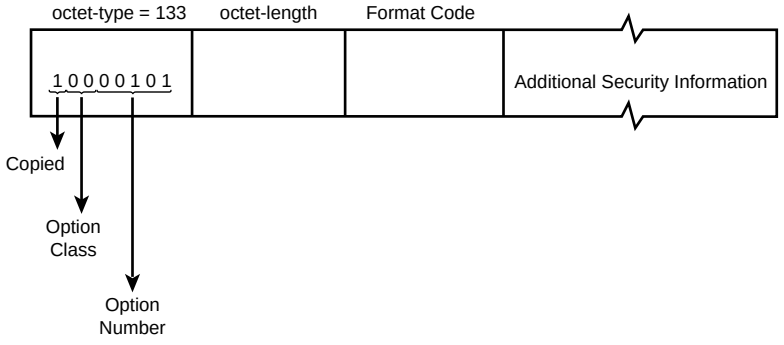
The Extended Security option is used for specifying additional security requirements (see Figure 10.16). It is characterized by an octet-type code of 133.

Commercial routers may not support IP security options.

## Record Route Option

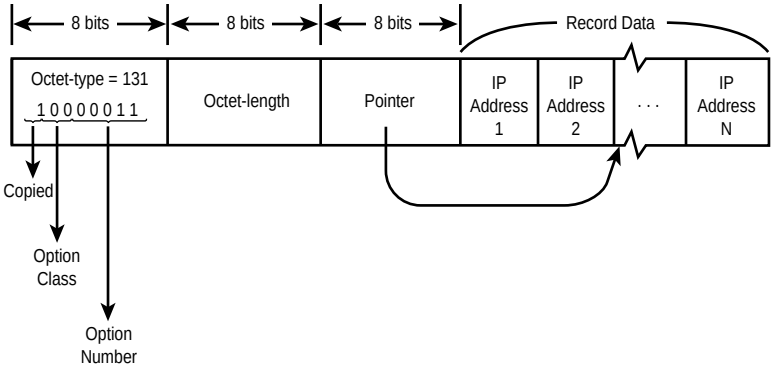
The Record Route option is used by the sender to record the IP addresses of the routers that forward the IP datagram to the destination. The list of routers actually visited is recorded.

**FIGURE 10.16**
*Extended Security option.*




The Record Route option begins with the Option-type code of value 131 (see Figure 10.17). The second octet is the option length (the length of the entire option). The Record Data field contains slots for a list of IP address values. The third octet is the pointer into the Route Data field and indicates the start of the next slot to be processed. The pointer is relative to this option, and the smallest value for the pointer is 4. If the pointer value is greater than the Octet-length, there are no empty slots in the Record Data field; that is, the Record Data field is full. If the Record Data list is full, a router forwards the datagram without inserting itself in the list.

**FIGURE 10.17**
*Record Route option.*




If the pointer is not greater than the Octet-length, the Record Data list is not full and empty slots are available. The router inserts its IP address at the position specified by the

pointer field and increments the pointer field value by 4, the size of the IP address. The recorded address is the IP address of the network interface of the router through which this datagram is being forwarded.

Both sender and receiver have to agree that record route information will be used and processed. The sender adds the Record Route option and sufficient slots for the IP addresses to the IP datagram. The receiver agrees to process the list of IP addresses recorded in the Record Data field. If the receiver has not agreed to process the record route data, then this information is ignored by the receiver.

## Strict and Loose Source Routing

In source routing, the sender supplies the path to reach the destination. The path consists of a list of IP addresses of the routers that must be visited. Normally, the path to reach the destination is decided by the routers based on the optimal path determined by routing protocols. In source routing, the sender dictates the path that a datagram must follow in arriving at the destination. The source routing facility is useful in testing a particular path even when you know that the routers would not normally select that path. This also gives you the flexibility of routing datagrams through networks that are known to work reliably. The disadvantage of source routing is the presumption that you know the topology of the network and the list of routers that are used for forwarding a datagram. In a complex network, the topology and the actual path an IP datagram must follow may not be easy to determine.

The two types of source routing are

- Strict source routing
- Loose source routing

In the Strict Source Routing option (see Figure 10.18), the sender includes a sequence of IP addresses that must be followed exactly in forwarding the datagram. The path between two successive IP addresses in the source route list can only be a single physical network segment. A single physical network segment can consist only of layers 1 (repeater) and 2 (bridge) devices, but not layer 3 (router) devices. If routers cannot follow the strict sequence of IP addresses, the datagram is rejected.

In the Loose Source Routing option (see Figure 10.19), the sender also includes a sequence of IP addresses that must be followed in forwarding the datagram. However, the path between two successive IP addresses may include any number of routers. Figure 10.20 shows the comparison between strict and loose source routing.

**FIGURE 10.18**
*Strict source routing.*

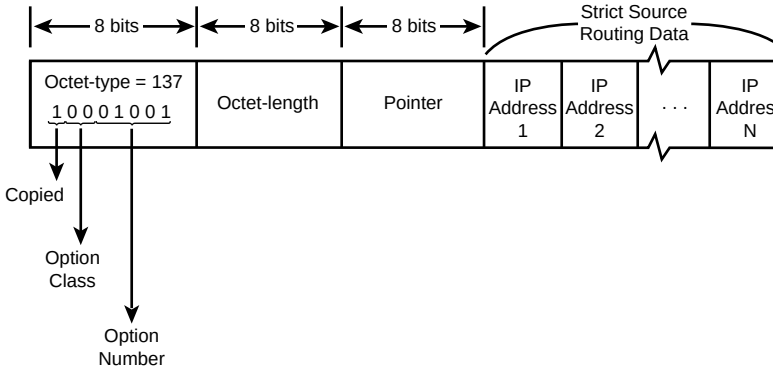



**FIGURE 10.19**
*Loose source routing.*

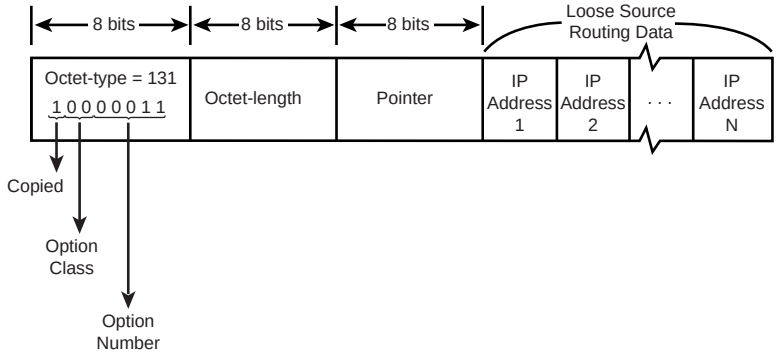
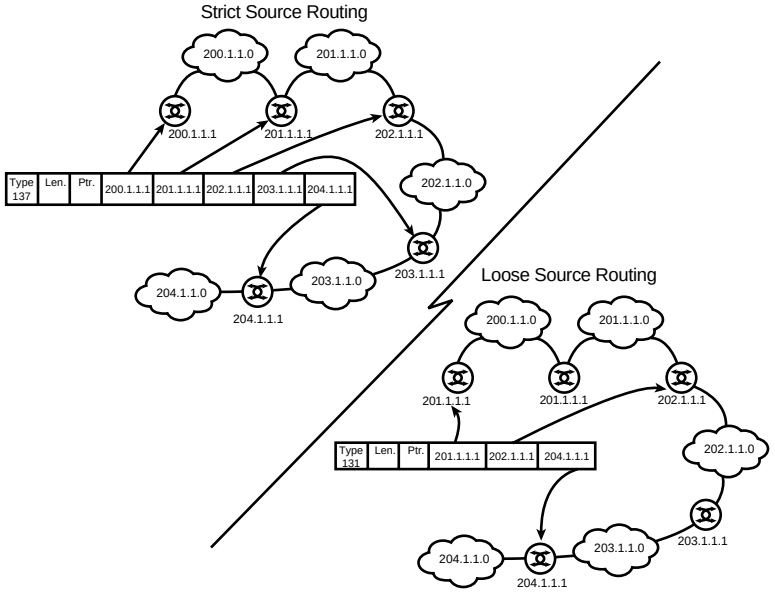



**FIGURE 10.20**
*Strict versus loose source routing.*

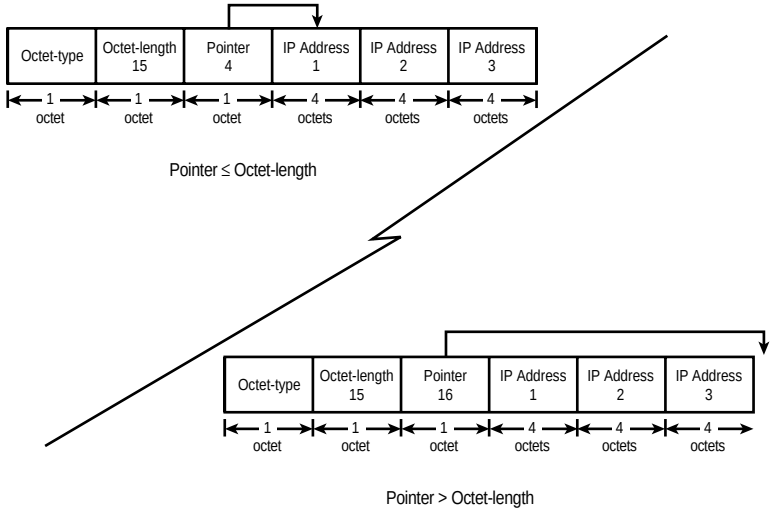


In both the Strict and Loose Source Routing options, the routers overwrite the source address in the option with their local IP addresses. When a datagram arrives at its destination, the IP address list contains the actual route that was taken. This list is like the one produced by the Record Route option. For this reason the Loose Source Routing option is also called the *Loose Source and Record Route* (LSRR) option, and the Strict Source Routing option is called the *Strict Source and Record Route* (SSRR) option.

As you examine the formats of the record route, strict source routing, and loose source routing in Figures 10.17, 10.18, and 10.19, notice that the option format is very similar. The Pointer field acts like an index into the IP address list. The Octet-length field indicates the number of IP address slots. When a route finishes processing an IP address, it advances the Pointer index to the next available IP address slot. Because IP addresses are 4 octets long, the Pointer field value is incremented by 4 to advance it to the next IP address slot.

When the Pointer field value is greater than the Octet-length field value, the IP address list has been processed or exhausted. In this case, the IP datagram is forwarded as usual by information in the router's routing table, without the help of the information in the IP option fields. If the Pointer is not greater than the Octet-length field value, the IP address list has not been fully processed (see Figure 10.21). In this case, the router uses the Pointer field index to obtain the value of the next router address to which it must forward the datagram. Additionally, the router then places the IP address of the network interface through which the datagram is forwarded in the IP address slot pointed to by the Pointer field index. This is done for the purpose of recording the actual route that was taken. The Pointer field index is advanced to the next IP address slot by incrementing it by 4.

**FIGURE 10.21**
*Use of Pointer and Octet-length field.*



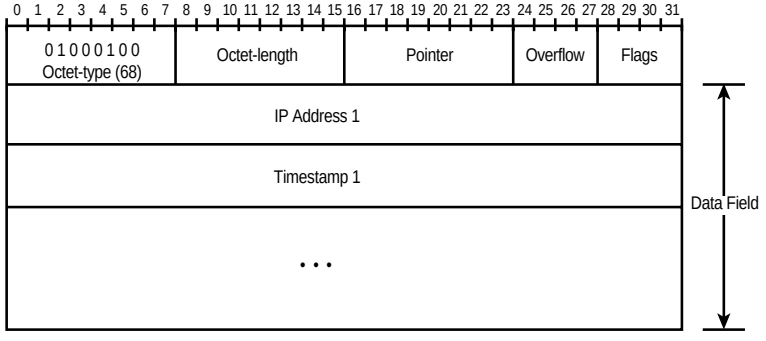
The Route Data field consists of a series of IP addresses. Each IP address is 4 octets long. If the Pointer value is greater than the Octet-length field, the source route is empty and the recorded route is full. The routing is based on the destination address field of the IP datagram.

## Internet Timestamp

The Internet Timestamp option is used to record the timestamps at which the IP datagram is received at each router. The router's IP address and time at which the IP datagram is received by a router is recorded. The timestamps are measured as the number of milliseconds since midnight, Universal Time Coordinated (m). UTC was formerly called Greenwich Mean Time (GMT) and is the time at 0 degrees longitude.

In addition to timestamps, the IP addresses of the router can also be recorded using the Internet Timestamp option. Figure 10.22 shows the format of the Internet Timestamp option.

**FIGURE 10.22**
*Internet Timestamp option.*




The Option-type is 68, and the Option-length is the number of octets in the option counting all the fields. Each entry in the data field of the Timestamp option contains two 32-bit values: the IP address of the router and the timestamp that it recorded.

The Octet-length specifies the amount of space reserved in the data field for the IP addresses and timestamps. The Pointer field is an index into the data field and points to the next unused slot. The index value in the Pointer field is measured from the beginning of the option. The smallest legal value of the Pointer field is 5. When the Pointer value is greater than the Octet-length field value, the timestamp area is full. The sender must create the Timestamp option with a large enough data area to hold all the timestamp information expected. The size of this option is not changed when the IP datagram is in transit. The sender must initialize the data area with zero values.

The Overflow field is 4 bits long and contains the number of routers that could not supply the timestamp because the data field was too small. The maximum value in the Overflow field is 15 routers. If there is insufficient room for a full timestamp to be inserted in the data area, or the overflow count exceeds 15, then the IP datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message is sent to the sender.

The Flags field controls the format of the information in the data field. It can have three values whose codes are described in Table 10.9.

**TABLE 10.9** IP Timestamp Option Flag Values

| *Flag Value* | *Meaning* |
|---|---|
| 0 | Timestamps only, stored in consecutive 32-bit words. IP addresses are omitted. |
| 1 | Each timestamp is preceded with the IP address of the registering entity (router). This format is shown in Figure 10.20. |
| 3 | The IP address fields are specified by the sender. An IP module (router) only registers its timestamp if its IP address matches the next specified IP address. |

The Timestamp values are right-justified, with 32-bit values measured in milliseconds since midnight UTC. If the time is not available in milliseconds or cannot be provided with respect to midnight UTC, then any time value, such as local time, may be inserted in the Timestamp field provided the high-order bit of the Timestamp field is set to 1. Setting the high-order bit of the Timestamp field to 1 indicates the use of a nonstandard value.

Routers may be set to the local time, and their clocks may not be synchronized. Because of differences in the routers' clock values, the Timestamp values should be treated as approximate values.

The Timestamp option is not copied upon fragmentation and is carried only in the first IP datagram fragment.

## Network Byte Order

The Physical Layer transmits the bits in the order in which they are received from the upper layers. The bits that constitute the transmitted data may be stored in a different way on each computer. For example, not all computers store 32-bit integer values (such as IP addresses) in the same way. Some computers store the low-order bytes of the 32-bit integer in lower-memory addresses. This storage method is called *little endian*, and

the computers are called little endian computers. The other method is the *big endian* method in which the high-order bytes are stored in lower-memory addresses.

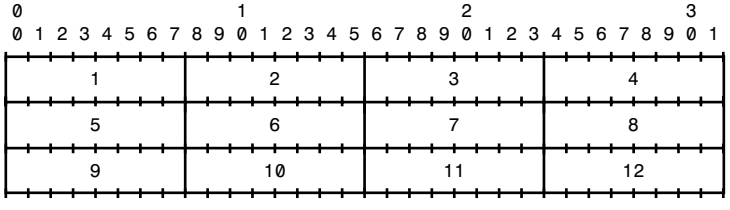
Some computers, especially the older 16-bit computers, store data in units of 16 bits with lower-order bytes stored in lower-memory addresses, but with the bytes swapped in the 16-bit word.

Because of differences in storage of data bytes, you cannot directly copy the data from one IP node to another. The TCP/IP protocols define a *network standard byte order* that is used for storing binary fields in internet packets. All IP nodes must represent the data in packets using this network standard byte order. Before sending a packet, an IP node converts its local data representation to the network standard byte order. On receiving the IP packet, an IP node converts the network standard byte order to the local data representation for that IP node. In most TCP/IP implementations, conversion between the network standard byte order to the node-specific representation is handled by the TCP/IP application programming interfaces and support routines.

The network standard byte order specifies that integers with the most significant byte are sent first; this is the *big endian* style. This means that if you examine an internet packet header field in transit, you see integer values with the most significant byte near the beginning of the packet and the least significant byte near the end of the packet.

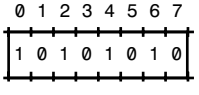
TCP/IP protocol headers are often laid out as 32-bit words (refer to Figure 10.5). The order of transmission of the octets in the header is the normal order in which they are read in English (from left to right). For example, in Figure 10.23 the octets are transmitted in the order in which they are numbered.

**FIGURE 10.23**
*Transmission order of bytes.*




Whenever an octet represents a numeric value, the left-most bit in the diagram is the high-order or most significant bit (MSB). That is, the bit labeled 0 is the most significant bit. For example, Figure 10.24 represents the value 170 (decimal).

**FIGURE 10.24**
*Significance of bits.*



Whenever a multi-octet field represents a numeric quantity, the left-most bit of the entire field is the most significant bit. When a multi-octet quantity is transmitted, the most significant octet is transmitted first.

# IP Trace

The following sections present you with protocol decodes of IP packets captured on a real-life network. You should study this protocol decode after briefly reviewing the previous sections that describe the meaning of the IP fields. Figure 10.25 shows the summary protocol trace of the first 50 packets captured in a Telnet session. Because this chapter discusses only the IP protocol, only selected packets in this Telnet trace will be discussed.

**FIGURE 10.25**
*Telnet trace.*

| No. | Source | Destination | Layer | Size | Summary |
|---|---|---|---|---|---|
| 1 | 0000C024282D | FFFFFFFFFFFF | arp | 0064 | Req by 144.19.74.44 for 144.19.74 |
| 2 | 0000C024282D | FFFFFFFFFFFF | arp | 0064 | Req by 144.19.74.44 for 144.19.74 |
| 3 | 0000C0A20F8E | 0000C024282D | arp | 0064 | Reply 144.19.74.201=0000C0A20F8E |
| 4 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET SYN |
| 5 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK SYN |
| 6 | 0000C024282D | 0000C0A20F8E | telnt | 0069 | Cmd=Do; Code=Echo; Cmd=Do; Code=S |
| 7 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 8 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Cmd=Do; Code=; |
| 9 | 0000C024282D | 0000C0A20F8E | telnt | 0064 | Cmd=Won't; Code=; |
| 10 | 0000C0A20F8E | 0000C024282D | telnt | 0067 | Cmd=Will; Code=Echo; Cmd=Will; Co |
| 11 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 12 | 0000C024282D | 0000C0A20F8E | telnt | 0072 | Cmd=Do; Code=Suppress Go Ahead; C |
| 13 | 0000C0A20F8E | 0000C024282D | telnt | 0070 | Cmd=Do; Code=Terminal Type; Cmd=D |
| 14 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 15 | 0000C024282D | 0000C0A20F8E | telnt | 0072 | Cmd=Will; Code=Terminal Type; Cmd |
| 16 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 17 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Cmd=Subnegotiation Begin; Code=Te |
| 18 | 0000C024282D | 0000C0A20F8E | telnt | 0071 | Cmd=Subnegotiation Begin; Code=Te |
| 19 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 20 | 0000C0A20F8E | 0000C024282D | telnt | 0067 | Cmd=Do; Code=Echo; Cmd=Will; Code |
| 21 | 0000C024282D | 0000C0A20F8E | telnt | 0069 | Cmd=Won't; Code=Echo; Cmd=Don't; |
| 22 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 23 | 0000C0A20F8E | 0000C024282D | telnt | 0104 | Data=..Linux 1.3.50 (ltreec1.ltre |
| 24 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 25 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Data=.. |
| 26 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 27 | 0000C0A20F8E | 0000C024282D | telnt | 0073 | Data=ltreec1 login: |
| 28 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 29 | 0000C024282D | 0000C0A20F8E | telnt | 0064 | Data=u |
| 30 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 31 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Data=u |
| 32 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 33 | 0000C024282D | 0000C0A20F8E | telnt | 0064 | Data=s |
| 34 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 35 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Data=s |
| 36 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 37 | 0000C024282D | 0000C0A20F8E | telnt | 0064 | Data=e |
| 38 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 39 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Data=e |
| 40 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 41 | 0000C024282D | 0000C0A20F8E | telnt | 0064 | Data=r |
| 42 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 43 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Data=r |
| 44 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 45 | 0000C024282D | 0000C0A20F8E | telnt | 0064 | Data=1 |
| 46 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |
| 47 | 0000C0A20F8E | 0000C024282D | telnt | 0064 | Data=1 |
| 48 | 0000C024282D | 0000C0A20F8E | tcp | 0064 | Port:6295 ---> TELNET ACK |
| 49 | 0000C024282D | 0000C0A20F8E | telnt | 0064 | Data=.. |
| 50 | 0000C0A20F8E | 0000C024282D | tcp | 0064 | Port:TELNET ---> 6295 ACK |

Notice that the first three packets of this Telnet trace are ARP packets. The use of ARP to resolve hardware addresses is discussed in Chapter 5, "ARP and RARP."

The Ethernet header for all the IP datagrams contains an EtherType value of 800 hex. This EtherType value indicates that the Ethernet frame encapsulates an IP datagram.

The Protocol field in the IP headers of all the datagrams has a value of 6. This indicates that the IP datagram encapsulates a TCP packet.

**IP Datagram #1: IP Packet with Flags = 0, MF = 0, DF = 0**

Figure 10.26 shows the first IP datagram (packet # 4 in Figure 10.25) sent from the source at 199.245.180.44 to the Telnet host at 199.245.180.201. As you examine the IP header, you will notice the following parameter values:

Version: 4

Header Length (32 bit words): 5

TOS: 0 (Precedence: Routine, Normal Delay, Normal Throughput, Normal Reliability)

Total length: 44

Identification: 1

Flags: 0, DF = 0, MF = 0 (Fragmentation allowed, Last fragment)

Fragment Offset: 0

Time to Live: 100 seconds

Protocol: 6 (TCP)

Checksum: 0xA1AF(Valid)

Source IP address: 144.19.74.44

Destination IP address: 144.19.74.201

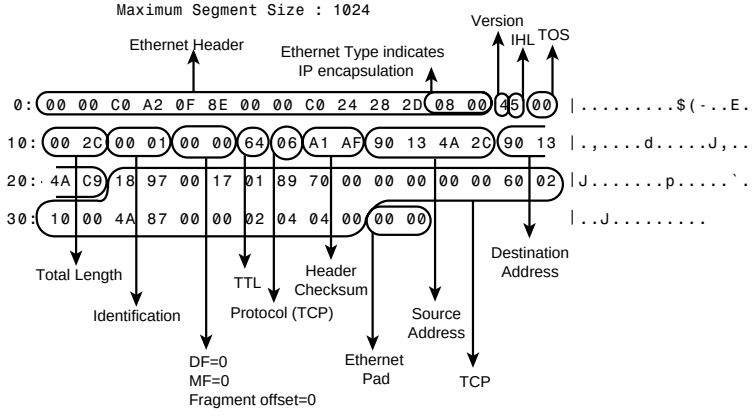
The Version field value is 4, so the format of the IP datagram is IPv4.

The Header Length is 5 words (20 octets). This is the minimum size of the IP datagram, so there are no IP options.

The Total Length of this IP datagram is 44 octets. The Ethernet header and CRC checksum is 18 octets long. Adding these two gives an Ethernet packet size of 62 octets. An additional 2 octets of pad are added to bring the total Ethernet frame size to 64 octets.

**FIGURE 10.26**
*IP datagram 1.*

```
Packet Number : 4           6:38:38 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-24-28-2D ----> 00-00-C0-A2-0F-8E
       Type: 0x0800 (IP)
  ip: ====================== Internet Protocol ======================
       Station:144.19.74.44 ---->144.19.74.201
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 44
       Identification:     1
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 100 seconds
       Checksum: 0xA1AF(Valid)
 tcp: ================ Transmission Control Protocol ================
       Source Port: 6295
       Destination Port: TELNET
       Sequence Number: 25784320
       Acknowledgement Number: 0
       Data Offset (32-bit words): 6
       Window: 4096
       Control Bits: Synchronize Sequence Numbers (SYN)
       Checksum: 0x4A87(Valid)
       Urgent Pointer: 0
       Option:MAXIMUM SEGMENT SIZE
              Option Length: 4
              Maximum Segment Size : 1024
```




This IP datagram has an Identification field value of 1 and a TTL value of 100 seconds. The TTL value of 100 seconds is the default TTL value that is used by the host at 199.245.180.44 for all IP datagrams that it sends.

The TOS value is 0, so this IP datagram should be treated with routine precedence, normal delay, normal throughput, and normal reliability.

The fragmentation Flags field has a value of 0, so the flags DF = 0, and MF = 0. This datagram could be fragmented, and this is the last fragment. The Fragment Offset value of 0 indicates that this is also the first fragment. A datagram that is the first and last fragment has no other fragments—this is a complete unfragmented datagram.

The Protocol field value has a code of 6, indicating that the IP datagram is carrying a TCP message.

The Checksum value is A1AF (hex) and is valid.

**IP Datagram #2: Response IP Packet with Flags = 0, MF = 0, DF = 0**

Figure 10.27 shows the second IP datagram (packet #5 in Figure 10.25) that is sent in response to the IP datagram discussed in the preceding section. This datagram is sent from the Telnet host at 199.245.180.201 to the Telnet client at 199.245.180.44. As you examine the IP header, notice the following parameter values:

Version: 4

Header Length (32 bit words): 5

TOS: 0 (Precedence: Routine, Normal Delay, Normal Throughput, Normal Reliability)

Total length: 44

Identification: 21410

Flags: 0, DF = 0, MF = 0 (Fragmentation allowed, Last fragment)

Fragment Offset: 0

Time to Live: 64 seconds

Protocol: 6 (TCP)

Checksum: 0x720E (Valid)

Source IP address: 144.19.74.201

Destination IP address: 144.19.74.44

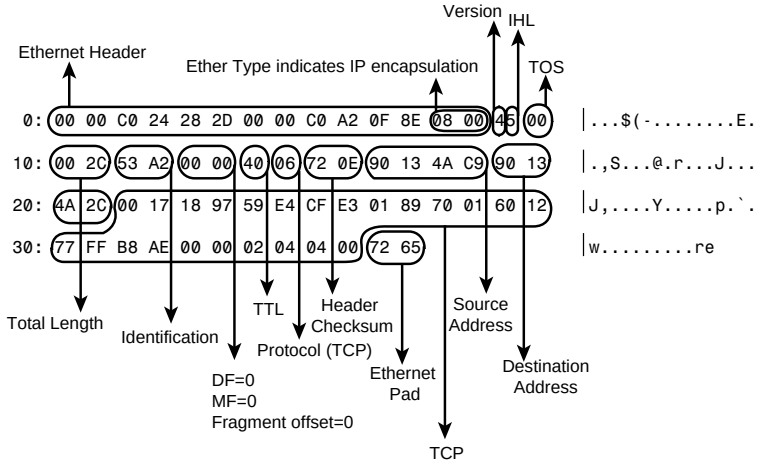
The Version field value is 4, so the format of the IP datagram is IPv4.

The Header Length is 5 words (20 octets). This is the minimum size of the IP datagram, so there are no IP options.

The Total Length of this IP datagram is 44 octets. The Ethernet header and CRC checksum is 18 octets long. Adding these two gives an Ethernet packet size of 62 octets. An additional 2 octets of pad are added to bring the total Ethernet frame size to 64 octets.

**FIGURE 10.27**
*IP datagram 2.*

```
Packet Number : 5           6:38:38 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
   ip: ======================= Internet Protocol =======================
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 44
       Identification: 21410
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x720E(Valid)
  tcp: ================= Transmission Control Protocol =================
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508167651
       Acknowledgement Number: 25784321
       Data Offset (32-bit words): 6
       Window: 30719
       Control Bits: Acknowledgement Field is Valid (ACK)
              Synchronize Sequence Numbers (SYN)
       Checksum: 0xB8AE(Valid)
       Urgent Pointer: 0
       Option:MAXIMUM SEGMENT SIZE
              Option Length: 4
              Maximum Segment Size : 1024
```



This IP datagram has an Identification field value of 21410. Notice that this Identification value is different from that sent by the Telnet client. The Identification value starts from a large number because the Telnet host has sent many IP datagrams prior to this session.

Another interesting item to note is that the TTL value is 64 seconds. This is the default TTL value that is used by the Telnet host at 199.245.180.201 for all IP datagrams that it sends. Notice that this TTL value is different from that sent by 199.245.180.44. Different TCP/IP implementations use different initial TTL values.

The TOS value is 0, which indicates that this IP datagram should be treated with routine precedence, normal delay, normal throughput, and normal reliability.

The fragmentation Flags field has a value of 0, so the flags DF = 0, and MF = 0. This datagram could be fragmented, and this is the last fragment. The Fragment Offset value of 0 indicates that this is also the first fragment. A datagram that is the first and last fragment has no other fragments—this is a complete unfragmented datagram.

The Protocol field value has a code of 6, so the IP datagram is carrying a TCP message.

The Checksum value is 720E (hex) and is valid.

**IP Datagrams #3 and #4: Observing the Identification Field**

Figures 10.28 and 10.29 show the next two IP datagrams that are exchanged between the Telnet client at 199.245.180.44 and the Telnet host at 199.245.180.201.

It is instructive to examine the Identification field value in these datagrams. The datagram in Figure 10.28 is the second datagram sent from the Telnet client to the Telnet host. The identification field in this datagram is 2, which is one more than that of the datagram sent in Figure 10.26. The datagram in Figure 10.28 is the second datagram sent from the Telnet host to the Telnet client. The identification field in this datagram is 21411, which is one more than that of the previous datagram sent in Figure 10.27. This example shows that the Identification fields are maintained separately by each sending side.

**FIGURE 10.28**
*IP datagram 3.*

```
Packet Number : 6           6:38:38 PM
Length : 69 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-24-28-2D ----> 00-00-C0-A2-0F-8E
       Type: 0x0800 (IP)
   ip: ======================= Internet Protocol =======================
       Station:144.19.74.44 ---->144.19.74.201
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
             Normal Delay, Normal Throughput, Normal Reliability
       Total length: 49
       Identification:     2
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 100 seconds
       Checksum: 0xA1A9(Valid)
  tcp: ================= Transmission Control Protocol =================
       Source Port: 6295
       Destination Port: TELNET
       Sequence Number: 25784321
       Acknowledgement Number: 1508167652
       Data Offset (32-bit words): 5
       Window: 4096
       Control Bits: Acknowledgement Field is Valid (ACK)
             Push Function Requested (PSH)
       Checksum: 0x18A9(Valid)
       Urgent Pointer: 0
telnt: ======================== Telnet Protocol ========================
       Command: Do
       Option Code: Echo
       Command: Do
       Option Code: Suppress Go Ahead
       Command: Will
       Option Code:
```

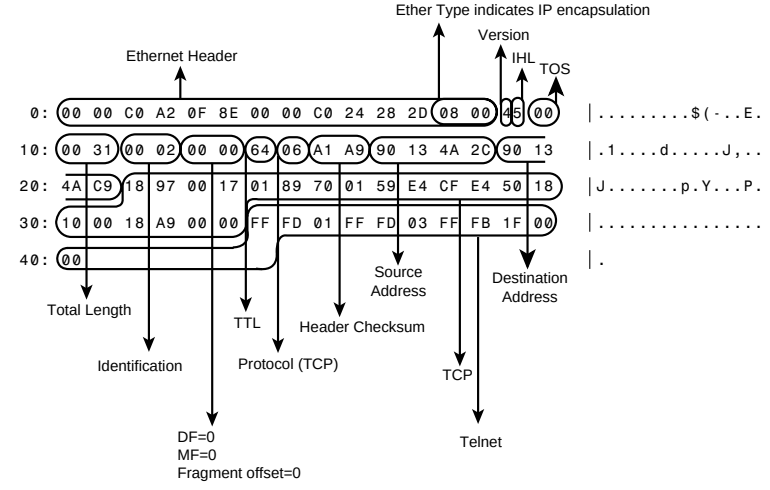
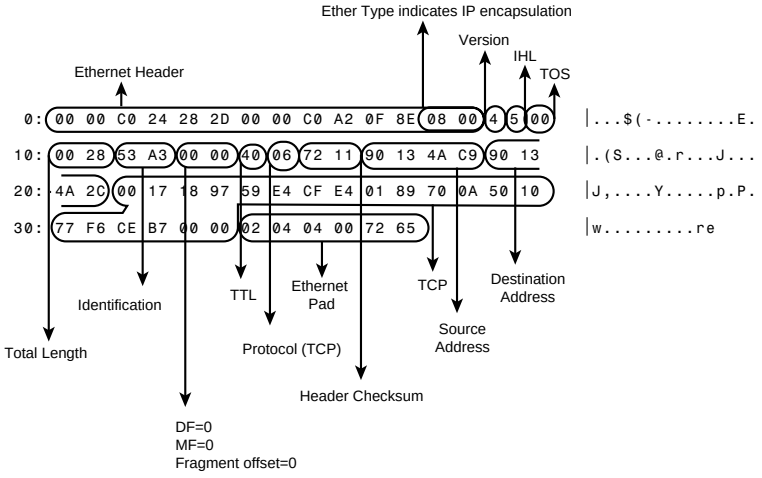


**FIGURE 10.29**
*IP datagram 4.*

```
Packet Number : 7           6:38:38 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
   ip: ======================= Internet Protocol =======================
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 40
       Identification: 21411
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x7211(Valid)
  tcp: ================ Transmission Control Protocol ================
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508167652
       Acknowledgement Number: 25784330
       Data Offset (32-bit words): 5
       Window: 30710
       Control Bits: Acknowledgement Field is Valid (ACK)
       Checksum: 0xCEB7(Valid)
       Urgent Pointer: 0
```



**IP Datagram #5: Datagram with DF = 1**

Figure 10.30 shows an IP datagram that is sent from the Telnet host at 199.245.180.201 to the Telnet client at 199.245.180.44. As you examine the IP header, notice the following parameter values:

Version: 4

Header Length (32-bit words): 5

TOS: 0 (Precedence: Routine, Normal Delay, Normal Throughput, Normal Reliability)

Total length: 43

Identification: 21412

Flags: 0, DF = 1, MF = 0 (Fragmentation not allowed, Last fragment)

Fragment Offset: 0

Time to Live: 64 seconds

Protocol: 6 (TCP)

Checksum: 0x320D (Valid)

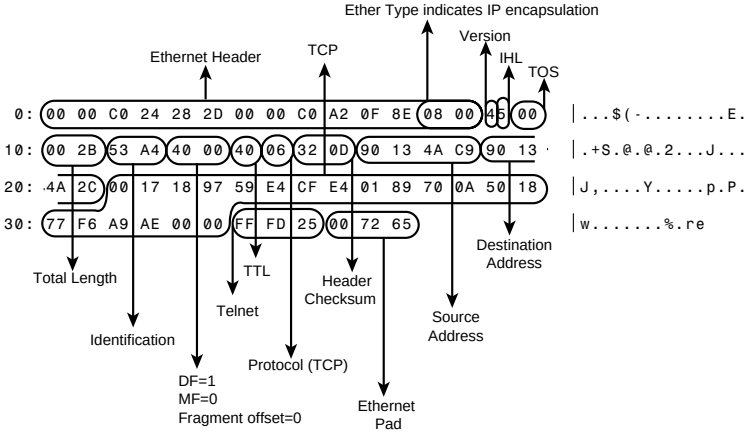
Source IP address: 144.19.74.201

Destination IP address: 144.19.74.44

The field values are similar to those discussed in earlier IP datagrams. The difference is in the fragmentation Flags field, which has the DF flag set to 1. This means that the Telnet host is requesting that this datagram should not be fragmented.

**FIGURE 10.30**
*IP datagram 5.*

```
Packet Number : 8           6:38:39 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
   ip: ======================= Internet Protocol =======================
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 43
       Identification: 21412
       Fragmentation not allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x320D(Valid)
  tcp: ================ Transmission Control Protocol =================
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508167652
       Acknowledgement Number: 25784330
       Data Offset (32-bit words): 5
       Window: 30710
       Control Bits: Acknowledgement Field is Valid (ACK)
              Push Function Requested (PSH)
       Checksum: 0xA9AE(Valid)
       Urgent Pointer: 0
telnt: ======================= Telnet Protocol =======================
       Command: Do
       Option Code:
```



# Summary

The Internet Protocol provides a logical view of the network regardless of the different network technologies used to build the physical network. The logical network is independent of Physical Layer addresses, MTU size, and any other differences. Upper-layer protocols, such as the TCP and UDP, treat the network as an IP-only network. The IP network provides connectionless services to upper-layer services. The connectionless service is implemented using datagrams that contain the source and destination IP addresses and other parameters needed for IP operation. An important aspect of IP operation is that datagrams in transit are fragmented as needed by routers to ensure that the datagrams can be transmitted in the packet size of the underlying networks. When IP fragmentation takes place, the receiver is responsible for assembling the IP datagram fragments into the original datagram.

CHAPTER 11

# The Transport Protocols

*by Karanjit S. Siyan, Ph.D.*

## In This Chapter

The TCP/IP transport protocols correspond to the Transport Layer of the OSI model. The TCP/IP protocol suite defines two standard transport protocols: TCP and UDP. TCP (Transmission Control protocol) implements a reliable data-stream protocol, whereas UDP (User Datagram protocol) implements an unreliable data-stream protocol.

Both TCP and UDP run on top of the Internet protocol and build on the services provided by IP (see Figure 11.1). IP provides a connectionless datagram service between two computers. By using TCP and UDP, you can deliver data not just to a remote computer, but to an application process running on the remote computer. These application processes are identified by port numbers. TCP can ensure that data is delivered reliably to the destination by providing a connection-oriented service. UDP, on the other hand, is connectionless and cannot guarantee delivery of data. However, UDP is useful in many applications, such as those where data needs to be sent to a particular application running on a machine, or in situations where application data needs to be broadcasted or multicasted.

Although Chapter 3, "Overview of TCP/IP" gave you some overview details about TCP and UDP, this chapter presents this information in considerably more detail.

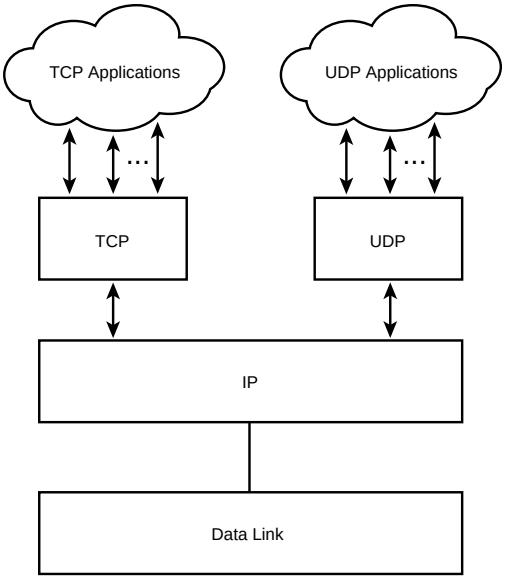



**FIGURE 11.1**
*TCP and UDP.*

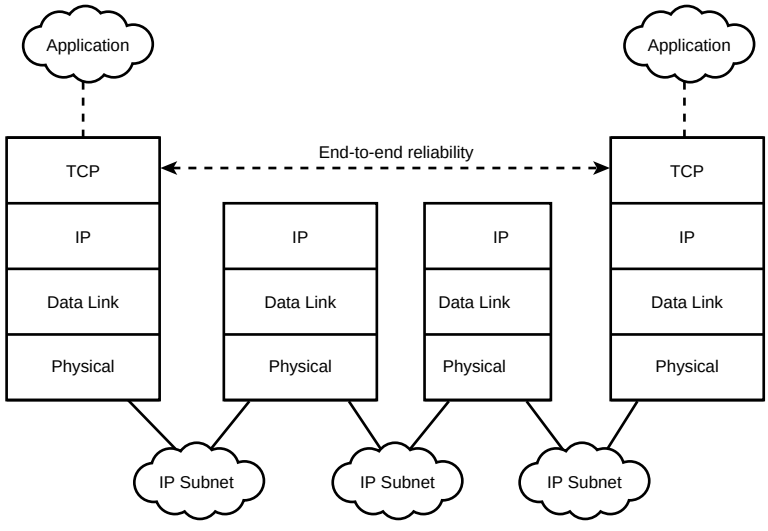
# The Transmission Control Protocol (TCP)

The TCP protocol provides a very important service in the TCP/IP protocol suite because it provides a standard general-purpose method for reliable delivery of data. Rather than inventing their own transport protocol, applications typically use TCP to provide reliable delivery of data because the TCP protocol has reached considerable maturity and many improvements have been made to the protocol to improve its performance and reliability.

TCP provides end-to-end reliability between an application process running on one computer system to another application process running on another computer system (see Figure 11.2). TCP provides this reliability by adding services on top of IP. IP is connectionless and does not guarantee delivery of packets. TCP assumes that IP is inherently unreliable, so TCP adds services to ensure end-to-end delivery of data. Because TCP has very few expectations on the services provided by the network, TCP can run across a large variety of hardware. All that TCP asks for is that some kind of simple, unreliable datagram service be provided at the lower layer. Sometimes networks such as X.25 provide sophisticated connection-oriented services at the Network Layer. In this case, the IP datagrams are transmitted on an X.25 virtual circuit, and the TCP protocol packets are encapsulated inside the IP datagrams.

TCP is the primary transport protocol used to provide reliable, full-duplex, virtual-circuit connections. The most common use of TCP is to run it over IPv4 or IPv6, although several experimental projects have been done to run TCP on other Network Layer protocols.

**FIGURE 11.2**
*End-to-end TCP reliability.*



Although IP is implemented on hosts and routers, TCP is typically implemented on hosts only to provide reliable end-to-end data delivery. Today, many routers are not just routers—they also provide other services that make them easy to configure and manage. For example, many commercial routers can also implement TCP or UDP to provide remote login and network-management facilities. Even though TCP and UDP are implemented in routers, the transport protocols are not used by routing services and messages.

The TCP protocol is described in RFC 792 (STD 7), "Transmission Control Protocol." The UDP protocol is described in RFC 768 (STD 6), "User Datagram Protocol." RFC 1122 (STD 3), "Requirements for Internet Hosts—Communication Layers," contains important additions. Both the TCP and UDP protocol status is recommended. But in actual practice, all TCP/IP devices—unless they are pure routers—will implement TCP and UDP. In most implementations, UDP logic is embedded in the TCP module and is not separate from the TCP implementation.

The following section discusses the features of TCP.

# TCP Features

TCP has the following noteworthy features:

- Basic data transfer
- Reliability
- Flow control
- Multiplexing
- Connections
- Precedence and security

## Basic Data Transfer

*Basic data transfer* is the capability of TCP to transfer a continuous stream of octets in each direction. The octets are sent among application processes running on remote systems that use TCP. The application processes then group a set of bytes that need to be sent/received into a message *segment*. Message segments can be of arbitrary length. Ultimately, the messages have to be sent in IP datagrams that are limited by the MTU size of a network interface. However, at the TCP level, there is no real restriction on message size because the details of accommodating the message segments in IP datagrams is the task of the IP Layer. For reasons of efficiency in managing messages, TCP connections typically negotiate a maximum segment size.

Messages sent by TCP have an octet stream orientation (see Figure 11.3). TCP keeps track of each octet that is sent/received. The TCP has no inherent notion of a block of data, unlike other transport protocols, which typically keep track of the Transport Protocol Data Unit (TPDU) number and not the octet number. TCP can be used to provide multiple virtual-circuit connections between two TCP hosts.

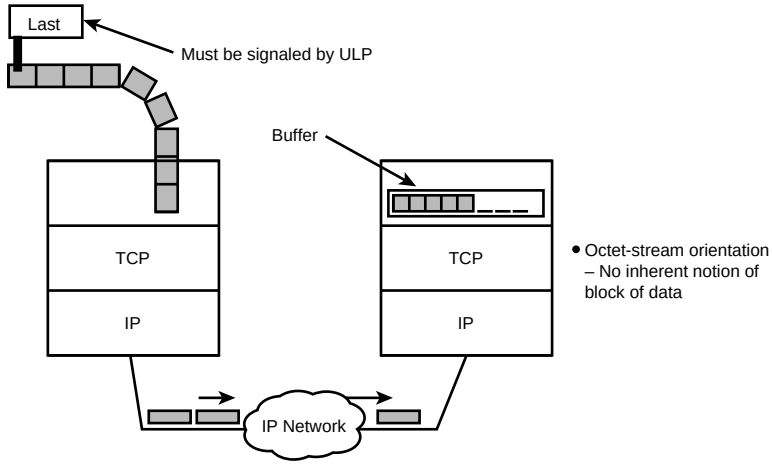



**FIGURE 11.3**
*TCP octet stream orientation. (Courtesy Learning Tree)*

Application processes that use TCP send data in whatever size is convenient for sending. For example, an application can send data that is as little as one octet or as big as several kilo-octets. TCP numbers each octet that it sends. The octets are delivered to the application processes at the receiving end in the order in which they are sent. This process is called *sequencing* of octets.

An application can send data to TCP a few octets at a time. TCP buffers this data and sends these octets either as a single message or as several smaller message segments. All that TCP guarantees is that data arrives at the receiver in the order in which it was sent. For example, if an application sends 1,024 octets of data over a period of ten seconds, the data can be sent across the network in a single TCP packet of 1,024 octets, or in four TCP packets of 256 octets, or in any combination of octets.

Because TCP sends data as a stream of octets, there is no real end-of-message marker in the data stream. To ensure that all the data submitted to the TCP module has been transmitted, a *push* function is required to be implemented by TCP. The push causes the TCP promptly to send any data that it has received from an application up to that point.

The actual data that is sent by TCP is treated as an unstructured stream of octets. TCP does not contain any facility to superimpose an application-dependent structure on the

data. For example, you cannot tell TCP to treat the data as a set of records in a database and to send one record at a time. Any such structuring must be handled by the application processes that communicate by using TCP.

## Reliability

One of the most important features of TCP is reliable end-to-end data delivery. In order to provide reliability, TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the Network Layer. TCP uses the Positive Acknowledgment Retransmission (PAR) scheme for achieving reliability. Figure 11.4 shows the PAR scheme, where all data and acknowledgments are received as expected. This figure shows that data is sent only after previously sent data has been acknowledged. Figure 11.5 shows how the PAR scheme recovers from lost data.

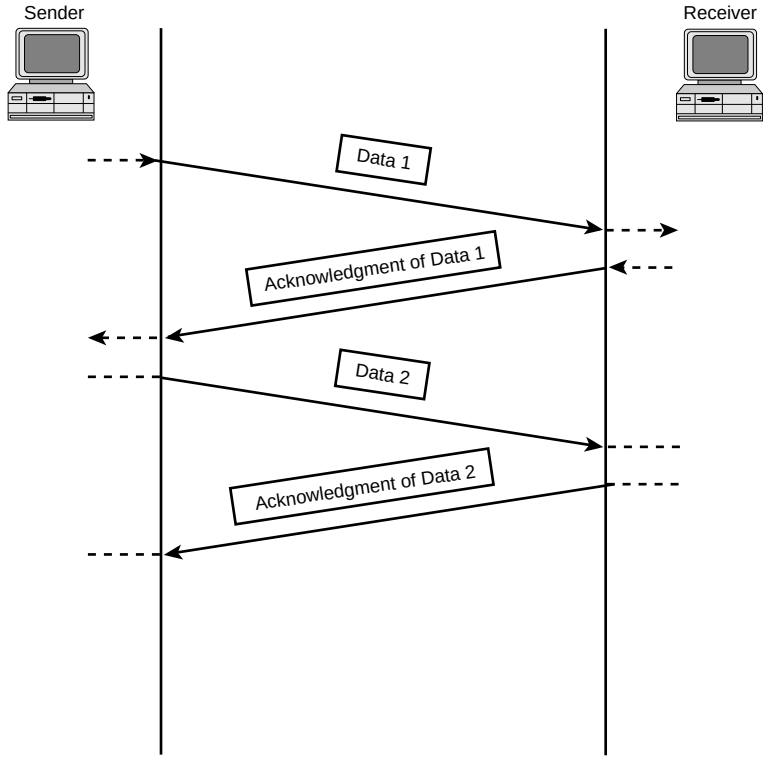



**FIGURE 11.4**
*PAR scheme under normal operation.*

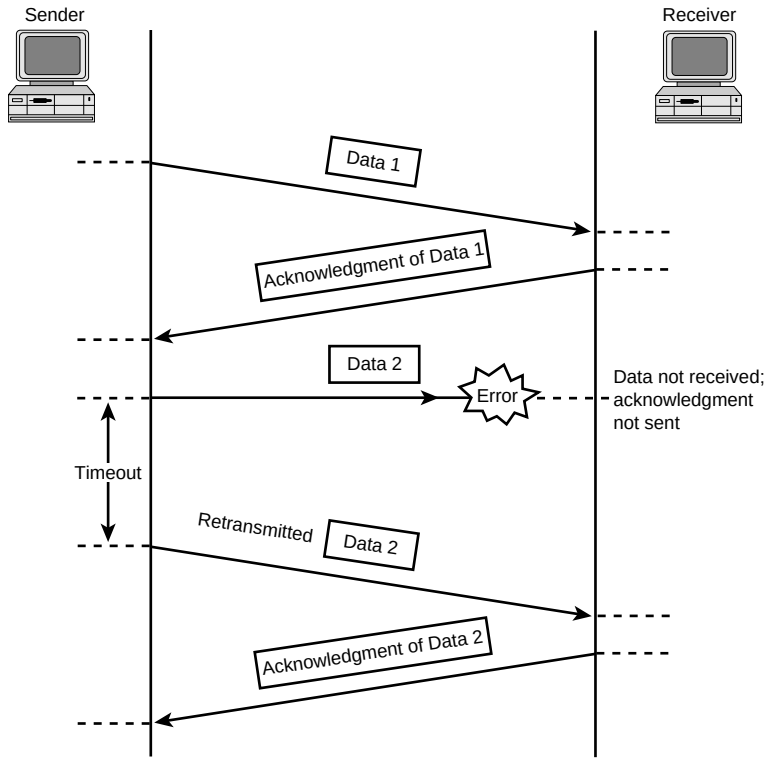



**FIGURE 11.5**
*PAR scheme illustrating recovery from lost data.*

TCP implements PAR by assigning a sequence number to each octet that is transmitted and by requiring a positive acknowledgment (ACK) from the receiving TCP module. If the ACK is not received within a time-out interval, the data is retransmitted. At the receiver TCP module, the sequence numbers are used to correctly order segments that may have arrived out of order and to eliminate duplicates. Corruption of data is detected by using a checksum field filed in the TCP packet header. Data segments that are received with a bad checksum field are discarded. Unless a physical break in the link causes physical partitioning of the network, TCP recovers from most Internet communications system errors.

## Flow Control

Computers that send and receive TCP data segments can operate at different data rates because of differences in CPU and network bandwidth. As a result, it is quite possible for a sender to send data at a much faster rate than the receiver can handle. TCP implements a flow control mechanism that controls the amount of data sent by the sender. TCP uses a sliding window mechanism for implementing flow control. Figure 11.6 illustrates the sliding window flow control used in TCP. The stream of data in TCP is numbered at the octet level. The number assigned to an octet is called the *sequence number*.

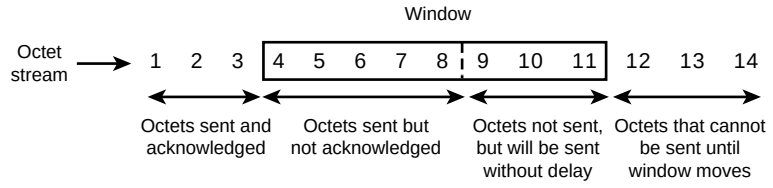



**FIGURE 11.6**
*TCP Sliding Window flow control. (Courtesy Learning Tree)*

The receiver TCP module sends back to the sender an acknowledgment that indicates a range of acceptable sequence numbers beyond the last segment successfully received. This range of acceptable sequence numbers is called a *window*. The window, therefore, indicates the number of octets that the sender can transmit before receiving further permission.

In Figure 11.6, the window range is from sequence numbers 4 through 11. In this example, octets 1 through 3 have been sent and acknowledged. Octets 4 through 11, which are in the window range, may have been sent or are waiting to be sent, but they have not been acknowledged yet. The sender TCP is not required to send a TCP segment that is equal to the window size. The window size indicates the maximum number of unacknowledged octets that TCP can send. In Figure 11.6, octets 4 through 8 have been sent; octets 9 through 11, which are within the window range, have not been sent, but they can be sent without delay. Octets 12 and beyond cannot be sent because they are to the right of the window range. To summarize, the TCP flow control mechanism exhibits the following properties:

- Octets that are to the left of the window range have already been sent and acknowledged.
- Octets in the window range can be sent without any delay. Some of the octets in the window range may already have been sent, but they have not been acknowledged. Other octets may be waiting to be sent.
- Octets that are to the right of the window range have not been sent. These octets can be sent only when they fall in the window range.

The left edge of the window is the lowest numbered octet that has not been acknowledged. The window can advance; that is, the left edge of the window can move to the right when an acknowledgment is received for data that has been sent. The TCP packet containing the acknowledgment contains information about the window size that the sender should use.

The window size reflects the amount of buffer space available for new data at the receiver. If this buffer space size shrinks because the receiver is being overrun, the receiver will send back a smaller window size. In the extreme case, it is possible for the receiver to send a window size of only one octet, which means that only one octet can be sent. This situation is referred to as the *silly window syndrome*, and most TCP implemen-

tations take special measures to avoid it. See "Avoiding the Silly Window Syndrome," later in this chapter.

A TCP module sending back a window size of zero indicates to the sender that its buffers are full and no additional data should be sent. TCP includes mechanisms to shrink window size when the receiver experiences congestion of data and to expand window size as the congestion problem clears.

The goal of the sliding window mechanism is to keep the channel full of data and to reduce to a minimum the delays experienced by waiting for acknowledgments.

## Multiplexing

TCP enables many processes within a single computer to use the TCP communications services simultaneously; this is called TCP *multiplexing.* Because these processes may be communicating over the same network interface, they are identified by the IP address of the network interface. However, you need more than the IP address of the network interface to identify a process because all processes that are using the same network interface on a computer have a common IP address.

TCP associates a port number value for applications that use TCP. This association enables several connections to exist between application processes on remote computers because each connection uses a different pair of port numbers. Figure 11.7 shows several connections being multiplexed over TCP.

The binding of ports to application processes is handled independently by each computer. In many computer systems, a *logger* or *super daemon* process watches over the port numbers that are identified or well known to other computer systems.

## Connections

Before application processes can send data by using TCP, they must establish a connection. The connections are made between the port numbers of the sender and the receiver nodes. A TCP connection identifies the end points involved in the connection. The end point (see Figure 11.8) is formally defined as a pair that includes the IP address and port number:

(IP address, port numbers)

The *IP address* is the internetwork address of the network interface over which the TCP/IP application communicates. The *port* number is the TCP port number that identifies the application. The *end point* contains both the IP address and port numbers because port identifiers are selected independently by each TCP, and they may not be unique. By concatenating the unique IP address with port numbers, a unique value for the end point is created.

**FIGURE 11.7**
*TCP Multiplexing.*

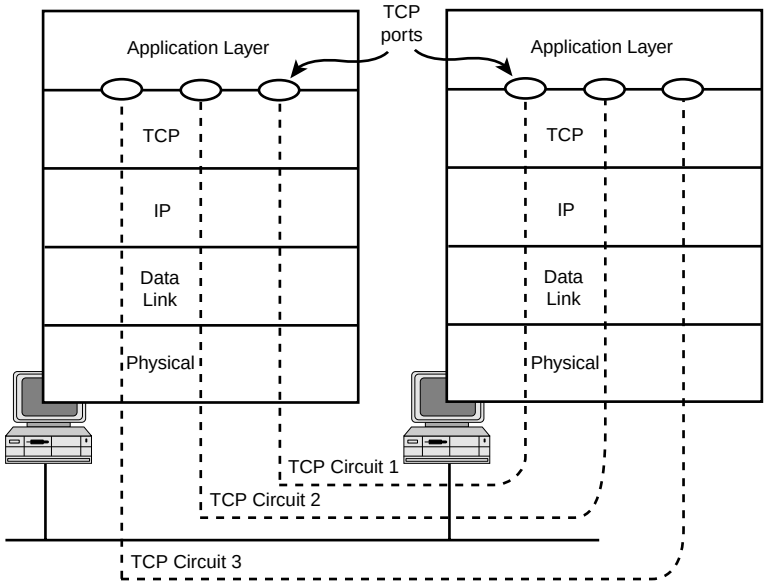
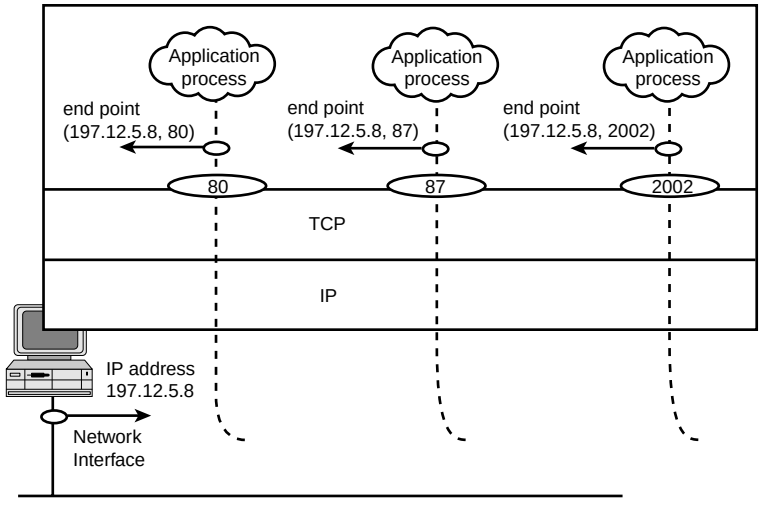



**FIGURE 11.8**
*TCP end point.*




A TCP connection is established between two end points (see Figure 11.9). The TCP connection is identified by the parameters of both end points, as follows:

(IP address1, port number1, IP address2, port number2)

These parameters make it possible to have several application processes connect to the same remote end point. Figure 11.9 shows that several application processes connect to

the same remote end point (199.11.23.1, 2001). The TCP module at 199.11.23.1 can keep the different TCP connections separate because TCP uses both the local and remote end points to identify the connection. In Figure 11.9, the end point (199.11.23.1, 2001) is the same, but the end points at the other end of the connection are different. This difference enables TCP to keep these connections separate.

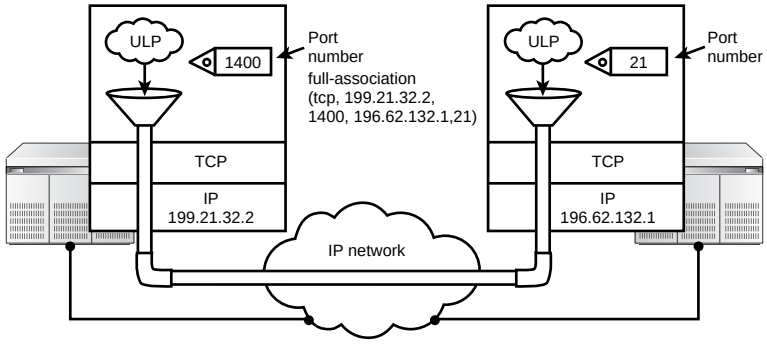



**FIGURE 11.9**
*TCP connection between end points. (Courtesy Learning Tree)*

Figure 11.9 also illustrates that TCP can support multiple connections concurrently. These connections are multiplexed over the same network interface.

A connection is fully specified by the pair of end points. A local end point can participate in many connections to different foreign end points. A TCP connection can carry data in both directions; that is, it is *full duplex*.

In relationship to TCP connections, it is also helpful to define the notion of a half association and a full association. A *half association* is an end point that also includes the transport protocol name, as follows:

(TransportProtocol, IP address, port number)

The half associations in Figure 11.9 are, therefore, the following:

(tcp, 199.21.32.2, 1400)

(tcp, 196.62.132.1, 21)

A *full association* consists of two half associations and is expressed by the following ordered pair:

(TransportProtocol, IP address1, port number1, IP address2, port number2)

The *TransportProtocol* is listed only once because it has to be the same on both parts of the half association. The concept of half and full associations is useful when dealing with

different transport protocols. As an example, the full association in Figure 11.9 is listed as follows:

(tcp, 199.21.32.2, 1400, 196.62.132.1, 21)

A full association consisting of source and destination IP addresses, and source and destination port numbers uniquely identifies a TCP connection.

## TCP Host Environment

TCP is implemented as a protocol module that interacts with the computer's operating system. In many operating systems, the TCP module is accessed like the file system of the operating system. The BSD Sockets interface that is implemented on Unix systems and the more recent Winsock interface implemented by Microsoft's proprietary operating systems are based on this file system model.

The TCP module depends on other operating system functions to manage its data structures and services. The actual interface to the network is typically controlled by a device driver module. TCP does not interact directly with the device driver module. Instead, TCP calls the IP module, which in turn calls on the device driver module (see Figure 11.10).

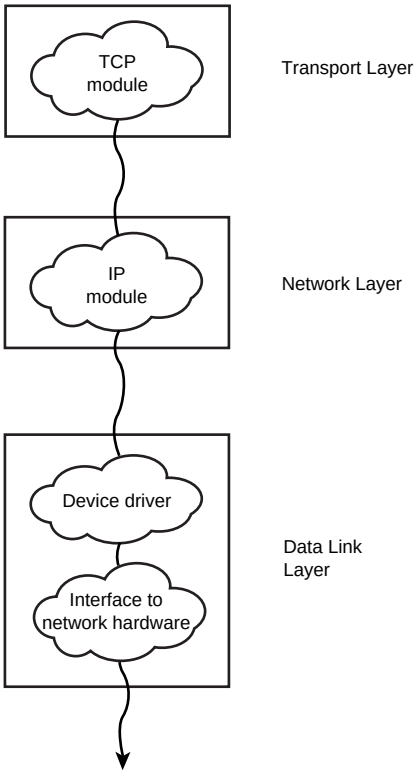
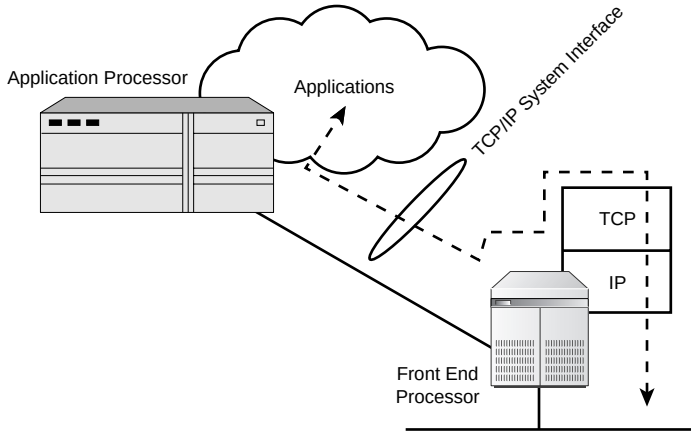



**FIGURE 11.10**
*TCP module interaction with network.*

Offloading the processing of the TCP/IP protocol stack modules to a front-end processor is also possible. The front-end processor acts as a dedicated protocol communications device that executes the protocol modules (see Figure 11.11). There has to be a mechanism for communication between the host and its front-end processor. Typically, this mechanism is provided by a file system interface.

**FIGURE 11.11**
*Offloading TCP processing.*




From an abstract viewpoint, applications will interface with the TCP module with the following system calls:

- OPEN to open a connection
- CLOSE to close a connection
- SEND to send data on an open connection
- RECEIVE to receive data from an open connection
- STATUS to find information about a connection

These system calls are implemented just like calls from user programs on the operating system. For example, the calls to open and close a connection are like the calls to open and close a file. The call to send or receive data is like the call to write or read from a file.

The TCP system calls can interact with other TCP modules anywhere on an internet system. The TCP system calls must be passed parameters for executing the system call. Examples of such parameters are IP addresses, type of service, precedence, security, application port number, and so on.

## TCP Connection Opening and Closing

A connection is specified in the OPEN call on the local port. The parameters supplied are the destination end point. The return from the system call contains a short integer value called the *handle*, by which the user refers to the connection in subsequent calls. Information about the connection is stored in a data structure called a *Transmission Control Block* (TCB), and the handle is used to access the information in the TCB.

TCP identifies two types of OPEN calls:

- Active OPEN
- Passive OPEN

In an active OPEN call, the connection establishment is to be actively initiated. An active OPEN call translates to a TCP message that is generated to contact another end point.

A passive OPEN call signals an intent to receive an active OPEN connection; it does not generate any TCP message segment. A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN call will accept a connection request from any caller. Alternatively, a passive OPEN call can specify that it can accept a connection only from a specific end point.

Processes that issue passive OPENs and wait for matching active OPENs from other processes can be informed by the TCP when connections have been established (see Figure 11.12).

If two processes issue active OPENs to each other at the same time, they will be correctly connected (see Figure 11.13). Using two active OPENs to set up a TCP connection is useful in distributed computing, where components can act asynchronously with respect to each other.

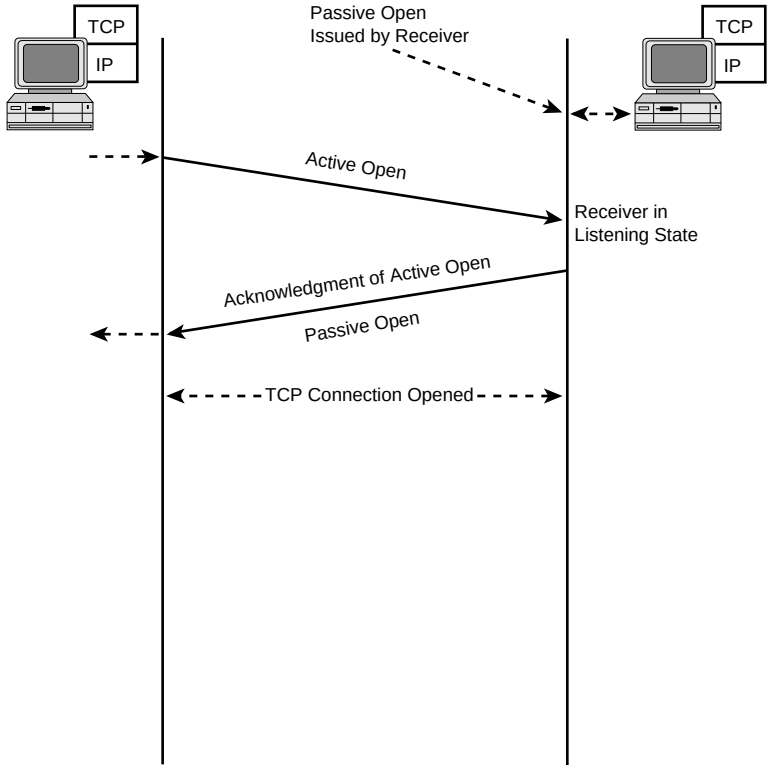



**FIGURE 11.12**
*Active and passive OPENs for setting up TCP connections.*

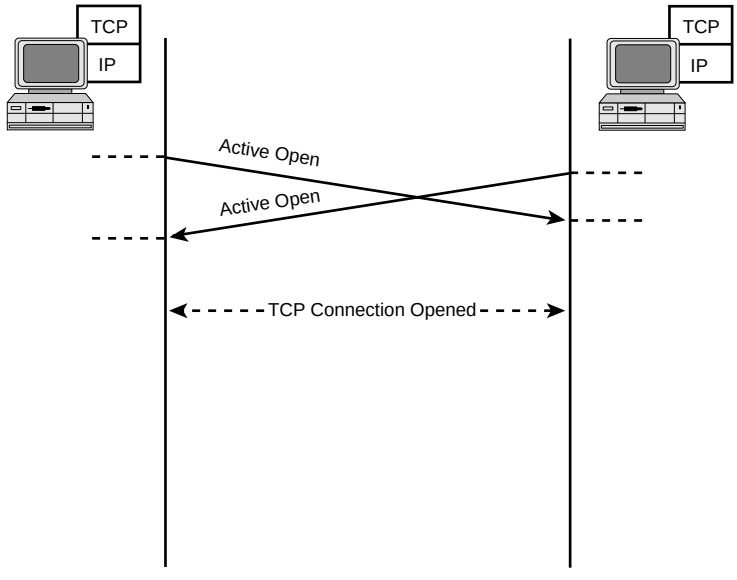
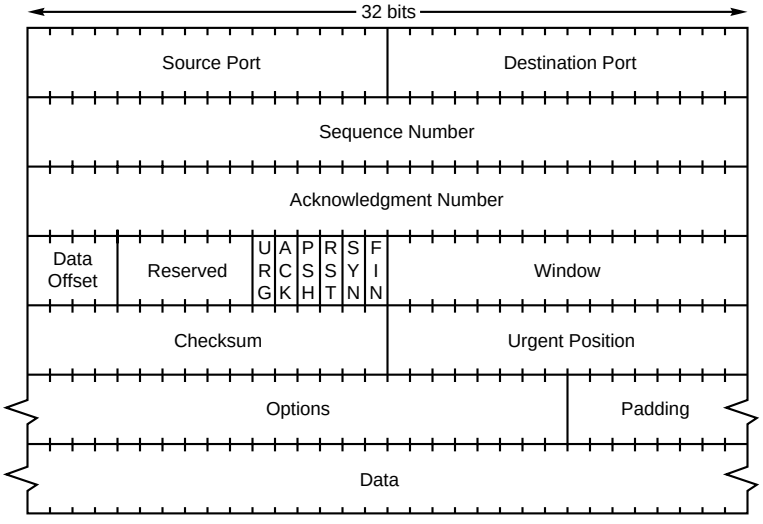



**FIGURE 11.13**
*Using only active OPENs for setting up TCP connections.*

# TCP Message Format

Figure 11.14 shows the TCP packet structure. Like most TCP/IP protocols, the TCP header is using a 32-bit word format.

**FIGURE 11.14**
*TCP packet structure.*




The following sections define the individual fields in the TCP header.

## Source and Destination Port Number Fields

The Source Port and Destination Port number fields (refer to Figure 11.14) are used to identify the end point processes in the TCP virtual circuit. Source and Destination Port numbers are needed in defining associations between processes. Some port numbers are well-known port numbers, others have been registered, and still others are dynamically assigned.

The Assigned Number RFC, which is issued periodically as a different numbered RFC, contains a description of some of the well-known port numbers.

**Note**

Port numbers in the range of 0–1,023 are called *well-known port numbers* and are administered by IANA. Port numbers greater than 1,023 can be registered but are not controlled by IANA. These port numbers are, therefore, called

*registered port numbers*. Not all port numbers at 1,024 and greater are registered. Some are assigned on an as-needed basis and are called *transient port numbers*. Transient port numbers are allocated within a TCP implementation. The actual algorithm used to allocate transient port numbers is implementation-dependent.

TCP modules are free to associate any port number with application processes, with the restriction that well-known ports are reserved for special applications. In many operating systems, application processes can "own" ports. These ports are allocated on a random basis as long as they are greater than 1,023. Some implementations use a hashing function that takes into account the process name when allocating port numbers. Typically, the high-order bits of the port number are associated with the process name.

## Sequence and Acknowledgment Number Fields

Transmission by TCP is made reliable via the use of sequence numbers and acknowledgment numbers. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a message segment is sent in the TCP header for that segment and is called the *segment sequence number*.

When a message segment is sent by the receiver, it also carries an acknowledgment number, which is the sequence number of the next expected data octet of transmission. TCP transmissions are full duplex. At any time, a TCP module is both a sender for the data that it is sending and a receiver for data from other senders.

When the TCP needs to transmit a segment, it puts a copy of the segment into its transmission queue and starts a timer. When the acknowledgment for that data is received before the timer expires, the segment is deleted from the queue. If the acknowledgment is not received before the timer expires, the segment is retransmitted from its copy in the transmission queue.

The 32-bit Sequence Number field (refer to Figure 11.14) is the number of the first byte of data in the current message. If the SYN flag field is set to 1, this field defines the initial sequence number (ISN) to be used for that session, and the first data offset is ISN+1. A 32-bit value is used to avoid using old sequence numbers that already may have been assigned to data that is in transit on the network.

The Acknowledgment Number field is used to indicate the sequence number of the next byte expected by the receiver. TCP acknowledgments are cumulative. That is, a single acknowledgment can be used to acknowledge a number of prior TCP message segments. After a connection is established, the acknowledgment number is always sent.

An acknowledgment by TCP guarantees only that the TCP module has received the data. The acknowledgment does not guarantee that the data has been delivered to the application running on top of TCP. An acknowledgment indicates that the receiving TCP has taken the responsibility to deliver the data to the application. It is possible that other system errors in the TCP module and the application interface may prevent the data from being delivered to the application.

## Data Offset Field

The Data Offset field is the number of 32-bit words in the TCP header (refer to Figure 11.14). This field is needed because the TCP options field could be variable in length. Without TCP options, the Data Offset field is five words (20 octets). The TCP header, even if it includes options, is an integral number of 32 bits.

The only TCP option that has been defined is the Maximum Segment Size (MSS) option. This option is used at the start of a TCP connection. When this option has been specified, the Data Offset field value is six words (24 octets).

The Reserved fields that follow the Data Offset field must be set to 0.

## The Flags Field

The Flags field in the TCP header (refer to Figure 11.14) have the following meanings:

- The ACK flag being set indicates that the Acknowledgment Number field is valid.
- The SYN flag is used to indicate the opening of a virtual-circuit connection.
- The FIN flag is used to terminate the connection.
- The RST bit is used to reset the virtual circuit due to unrecoverable errors. When an RST is received in a TCP segment, the receiver must respond by immediately terminating the connection. A reset causes both sides immediately to release the connection and all its resources. As a result, transfer of data ceases in both directions, which can result in loss of data that is in transit. A TCP RST is not the normal way to close a TCP connection; it indicates an abnormal condition. To close a TCP connection normally, use the FIN flag. The reason for a reset can be a host crash or delayed duplicate SYN packets.
- When the PSH flag is set, it tells TCP immediately to deliver data for this message to the upper-layer process.
- The URG flag is used to send out-of-band data without waiting for the receiver to process octets already in the stream.

TCP connections are opened by using the three-way-handshake procedure. The SYN and the ACK flags are used to indicate the following packets:

| | |
|---|---|
| SYN = 1 and ACK = 0 | Open connection packet |
| SYN = 1 and ACK = 1 | Open connection acknowledgment |
| SYN = 0 and ACK = 1 | Data packet or ACK packet |

The procedures to establish connections utilize the synchronize (SYN) control flag and involves an exchange of three messages termed the three-way handshake (see Figure 11.15).

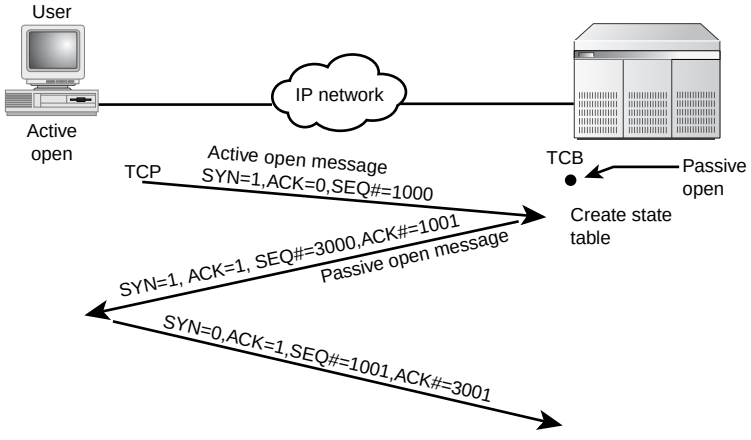



**FIGURE 11.15**
*TCP SYN flag use. (Courtesy Learning Tree)*

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry. The arriving TCP segment is generated by an active OPEN, and the waiting TCB entry is created by a passive OPEN. The matching of an active and a passive OPEN results in the creation of a connection. The SYN flag is set to 1 to indicate that sequence numbers have been synchronized in both directions. In Figure 11.15, the active OPEN generates a TCP segment with the following flags:

ULSYN = 1, ACK = 0

When the receiving TCP that has issued a passive OPEN receives an active OPEN, it acknowledges this active OPEN with the following TCP flags:

SYN = 1, ACK = 1

This acknowledgment of the active OPEN packet is sometimes called the passive OPEN packet. The ACK = 1 flag indicates that the Acknowledgment Number field contains valid data.

Figure 11.16 illustrates the general mechanism of the three-way handshake.

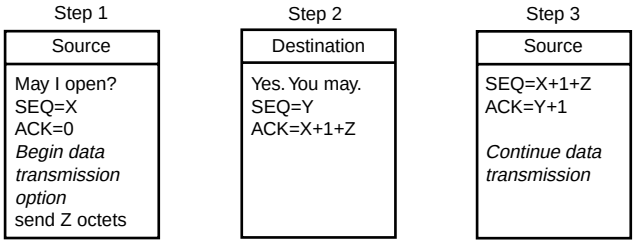



**FIGURE 11.16**
*Three-way handshake. (Courtesy Learning Tree)*

In Step 1 (see Figure 11.16), the active OPEN is sent with a Sequence Number value of *X*. As an option, additional data can be sent with the active OPEN packet. This additional data (Z octets) may contain authentication information, such as user name and password, for the service being requested. In most TCP/IP applications, authentication information is sent after the connection is established. But in some networks, such as those in which TCP/IP is run on X.25 networks, authentication information may be sent as part of the OPEN connection request. The Sequence and Acknowledgment Number fields are set as follows:

Sequence Number = X

Acknowledgment Number = 0

In Step 2, the receiver responds with an Acknowledgment Number set to X+1+Z, the next expected octet number. The initial Sequence Number (Y), which is used by the destination, is also sent. The Sequence and Acknowledgment Number fields are set as follows:

Sequence Number = Y

Acknowledgment Number = X + 1 + Z

In Step 3, the source acknowledges that it has received the Sequence Number value from the receiver. The Sequence and Acknowledgment Number fields are set as follows:

Sequence Number = X + 1 + Z

Acknowledgment Number = Y + 1

Why is the third step necessary in the three-way handshake? This step is often a source of puzzlement and confusion. You use the third step to complete the handshake to inform the receiver that its initial sequence number has been received by the source that opens the TCP connection. Without this acknowledgment, there is no way to ensure that the sender knows about the initial receiver sequence number. The following summarizes the three steps involved in the three-way handshake:

- Step 1: Source sends its initial SEND sequence number (ISS).
- Step 2: Receiver acknowledges reception of ISS by sending an acknowledgment number that is 1 greater than ISS or the amount of data sent by the source. Receiver sends its initial RECEIVER sequence number (IRS).
- Step 3: Source sends an acknowledgment number to acknowledge that it has received the receiver ISN.

To ensure that TCP does not send a segment containing a sequence number that is the duplicate of an old segment on the network, TCP has the notion of a *maximum segment lifetime (MSL).* MSL is the time that must elapse before assigning any sequence numbers upon starting up or recovering from a crash (in which memory of sequence numbers previously used was lost). In the TCP specification, MSL is taken to be two minutes. If a TCP implementation can retain memory of sequence numbers in use, then it need not wait for the MSL duration because it can start using sequence numbers larger than those recently used.

In TCP, because every octet is numbered, sequence numbers are consumed rapidly. If message octets are sent at the rate of 2 Mbps, it takes about 4.5 hours to use up $2^{32}$ octets of sequence numbers. Because the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this amount of time is adequate protection for rates that escalate to l0s of Mbps. At 100 megabits/sec, the MSL time is 5.4 minutes, which is still greater than the two minutes assumed by the TCP specification in RFC 793.

How is the value of an ISN selected? RFC 793 suggests the use of an initial sequence number (ISN) generator to select a new 32-bit ISN. One possibility is that this generator is bound to a 32-bit time clock whose low-order bit is incremented roughly every four microseconds. If this is the case, the ISN cycles approximately every 4.55 hours. It is unlikely that segments will stay in the network more than 4.55 hours; therefore, you can reasonably assume that ISNs will be unique.

After a TCP connection has been established, the ACK flag is always set to 1 to indicate that the Acknowledgment Number field is valid. Data transmission then takes place with messages being exchanged by both sides in a full-duplex fashion until the TCP connection is ready to be closed.

When a TCP connection is ready to close, the FIN control flag is issued. In TCP, a connection is not automatically closed when TCP sends a FIN flag. Both sides of the connection must send a FIN flag and agree to close the connection. This kind of FIN close is called a *graceful close* because it ensures that the connections are not abruptly closed. Consider what would happen if one side of the connection closed the connection without informing the other side. The other side may still continue sending data that is never

acknowledged because the connection is closed. This scenario would result in the loss of data. The TCP graceful close mechanism illustrated in Figure 11.17 prevents the loss of data due to premature closing of a TCP connection.

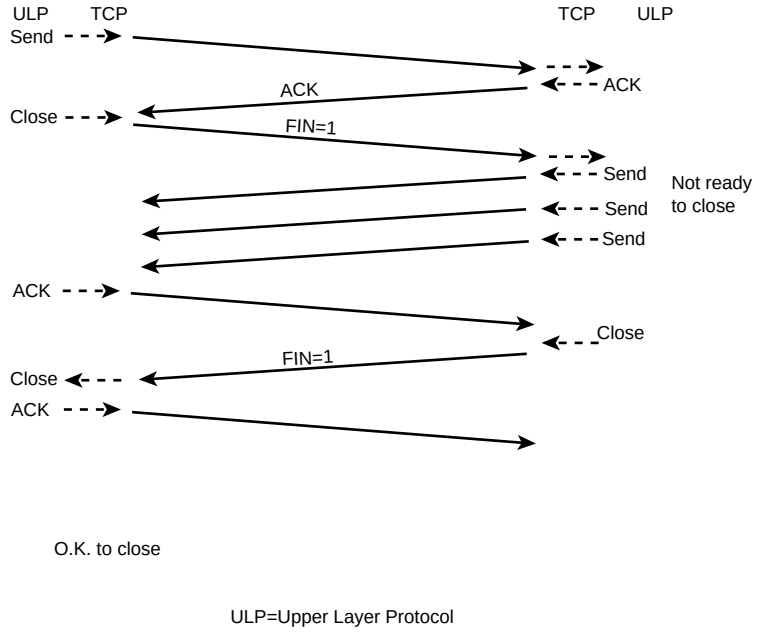



**FIGURE 11.17**
*TCP FIN flag use (the graceful close). (Courtesy Learning Tree)*

Another point to note about the use of TCP flags in sending messages is how the PSH flag is used. The PSH flag was initially designed to inform TCP that it must act upon the data that it has received so far. When an application issues a series of SEND calls without setting the PSH flag, the TCP can aggregate the data internally without sending it. Similarly, when a series of segments is received without the PSH bit, a TCP can queue the data internally without passing it to the receiving application. The PSH bit is not a record marker and is independent of segment boundaries. Some implementations incorrectly think of the PSH as a record marker, however. The transmitter should collapse successive PSH bits when it packetizes data to send the largest possible segment.

A TCP can implement PSH flags on SEND calls. If PSH flags are not implemented, then the sending TCP must do the following:

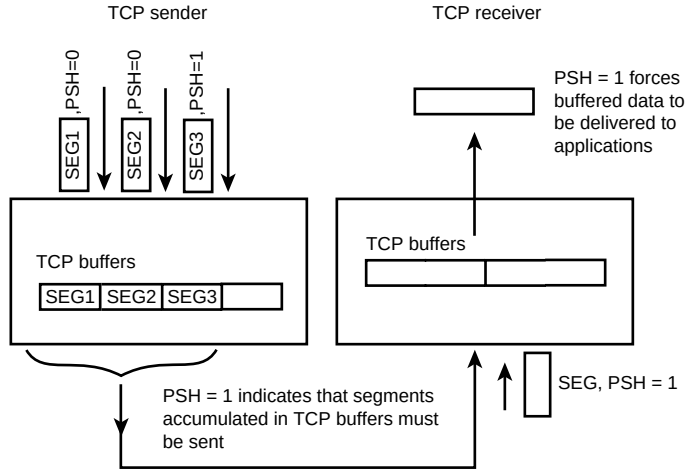
1. It must not buffer data indefinitely.
2. It must set the PSH bit in the last buffered segment (for example, when there is no more queued data to be sent).

RFC-793 erroneously implies that a received PSH flag must be passed to the Application Layer. Passing a received PSH flag to the Application Layer is now optional.

An application program is logically required to set the PSH flag in a SEND call whenever the program needs to force delivery of the data to avoid a communications deadlock. A TCP should send a maximum-size segment whenever possible to improve performance, however. This means that on the sender side, a PSH may not result in the segment's being immediately transmitted.

When the PSH flag is not implemented on SEND TCP calls (or when the application/TCP interface uses a pure streaming model), responsibility for aggregating any tiny data fragments to form reasonable-size segments is partially borne by the Application Layer. Generally, an interactive application protocol must set the PSH flag, at least in the last SEND call in each command or response sequence. A bulk transfer protocol such as FTP should set the PSH flag on the last segment of a file or when necessary to prevent buffer deadlock. Figure 11.18 illustrates the use of the PSH flag.

**FIGURE 11.18**
*TCP PSH flag use. (Courtesy Learning Tree)*



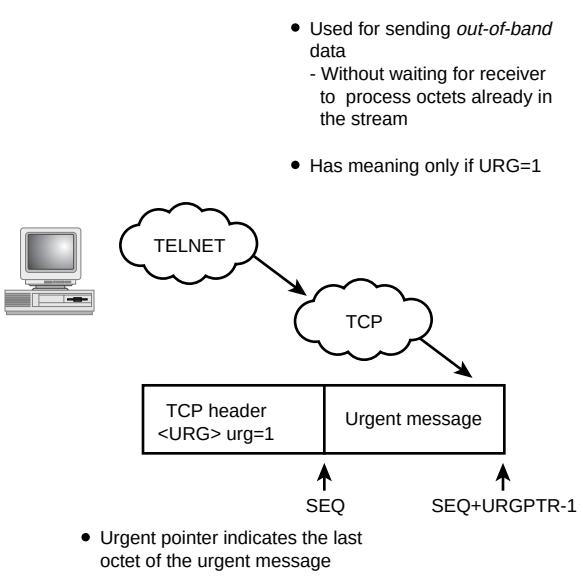

At the receiver, the PSH bit forces buffered data to be delivered to the application (even if less than a full buffer has been received). Conversely, the lack of a PSH avoids unnecessary wake-up calls to the application process. Avoiding these calls can be an important performance optimization for large time-sharing hosts.

TCP contains a mechanism to send out-of-band data by using the URG flag. The URG flag is used in conjunction with the Urgent Pointer field to "jump the queue" for sending data and to communicate an important message to the receiver. This feature enables the sender to send interrupt signals to the receiver without these signals ending up in the normal data queue at the receiver.

Consider what would happen if such a mechanism did not exist. Suppose that you want to interrupt the processing of user data that is sent. If the urgent message ends up in the normal data queue for TCP at the receiver, it may never be read because data that is ahead of the urgent message in the queue will have to be processed before the urgent message is detected. In many cases, the urgent message may arrive too late. Also, the urgent message may never be read if the application process that is supposed to read the message is hung up.

When the URG flag is set, the Urgent Pointer field is valid (see Figure 11.19). RFC 1122 states that the urgent pointer points to the sequence number of the LAST octet (not LAST+1) in a sequence of urgent data, and that RFC 793 describes it incorrectly as LAST+1.

**FIGURE 11.19**
*TCP URG flag. (Courtesy Learning Tree)*



A TCP implementation must support a sequence of urgent data of any length. A TCP layer must inform the Application Layer asynchronously whenever the TCP layer receives an urgent pointer with no previous pending urgent data, or whenever the urgent pointer advances in the data stream. There must be a way for the application to learn how much urgent data remains to be read from the connection or at least to determine whether more urgent data remains to be read. Although the urgent mechanism may be used for any application, it is normally used to send interrupt-type commands to a Telnet program. The asynchronous, or out-of-band, notification enables the application to go into urgent mode by reading data from the TCP connection. This allows control commands to be sent to an application whose normal input buffers are full of unprocessed data.

## Window Field

The Window field is used to implement flow control and is used by the receiver. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets—starting with the acknowledgment number—that the receiving TCP is currently prepared to receive.

## Checksum Field

The checksum field is 1's complement of the 1's complement sum of all the 16-bit words in the TCP packet. A 96-bit pseudoheader (see Figure 11.20) is prepended to the TCP header for checksum computation. The pseudoheader is used to identify whether the packet has arrived at the correct destination. The pseudoheader gives the TCP protection against misrouted segments. The pseudoheader has the protocol ID (6 for TCP), source, and destination IP address. Because the TCP header contains the source and destination port number, this describes the connection between the end points. Between the IP and TCP headers, there is sufficient information to identify completely the *full association* formed by the TCP connection.

The TCP Length field in the pseudoheader is the TCP header length plus the data length in octets. (This field is not an explicitly transmitted quantity; it is computed.) This field does not count the 12 octets of the pseudoheader.

## Options Field

The TCP Options field may occupy space at the end of the TCP header and is a multiple of eight bits in length. The Options field is included in the checksum. An option may begin on any octet boundary. The two formats for an option are

- Case 1: A single octet of option-kind.
- Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

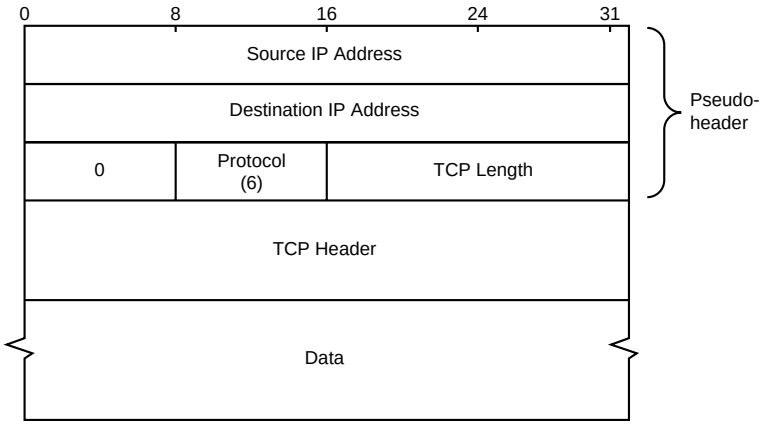



**FIGURE 11.20**
*Pseudoheader in TCP checksum.*

In Case 2, the option-length counts the two octets of option-kind and option-length as well as the option-data octets. The actual list of options may be shorter than the Data Offset field implies. The content of the header beyond the End-of-Option list must be zero-padded.

Currently, the defined options include the following:

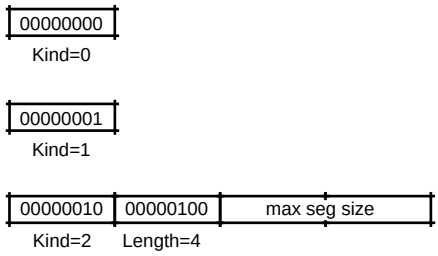
| *Kind* | *Length* | *Meaning* |
|---|---|---|
| 0 | - | End-of-Option list |
| 1 | - | No-Operation |
| 2 | 4 | Maximum Segment Size |

Options with the option-kind value of 0 (End-of-Option list) and 1 (No-Operation) are examples of the Case 1 format option. The TCP options are shown in Figure 11.21.

The End-of-Option list indicates the end of the option list that may not coincide with the end of the TCP header according to the Data Offset field. The End-of-Option list is used at the end of all options, but not at the end of each option. You need to use this option only if the end of the options would not otherwise coincide with the end of the TCP header.

The No-Operation option code may be used between options to align the beginning of a subsequent option on a word boundary. Because there is no guarantee that TCP senders will use this option, TCP receivers must be prepared to process options even if they do not begin on a word boundary.

**FIGURE 11.21**
*TCP options.*




Currently, the only specific option carrying any real information is the Maximum Segment Size (MSS). The MSS can be negotiated only at the start of the TCP connection when the sequence numbers are being synchronized. This synchronization occurs when the SYN flag is set to 1. Typically, each TCP sends a MSS option informing the remote TCP about its MSS. If the MSS option is not used, any segment size is allowed.

## Cumulative ACKs in TCP

Compared to transport protocols in other protocol suites, TCP is unusual as far as the meaning of the sequence numbers is concerned. In most transport protocols, the sequence number refers to the number of a packet. Not so in TCP. In TCP, sequence numbers refer to octets. Every octet is numbered in TCP. TCP sends octets in segments of variable length. When an acknowledgment is not received, segments are retransmitted. There is no guarantee that a retransmitted segment will be exactly the same as the original transmission. A retransmitted segment may contain additional data because the sending application may have generated additional data since the original transmission. The acknowledgment number cannot, therefore, refer to the segment that was sent.

In TCP, the acknowledgment number indicates the position in the data stream up to which data has been received and acknowledged by the remote TCP module. More specifically, an acknowledgment number value refers to the next octet number that needs to be sent. The acknowledgment number corresponds to the left edge of the TCP window.

TCP acknowledgment numbers are cumulative in the sense that an acknowledgment number indicates how much of the data stream has been accumulated so far. It is possible for a single acknowledgment number to acknowledge octets received in multiple data segments. Figure 11.22 shows that the three segments sent by the sender are acknowledged by a single acknowledgment number.

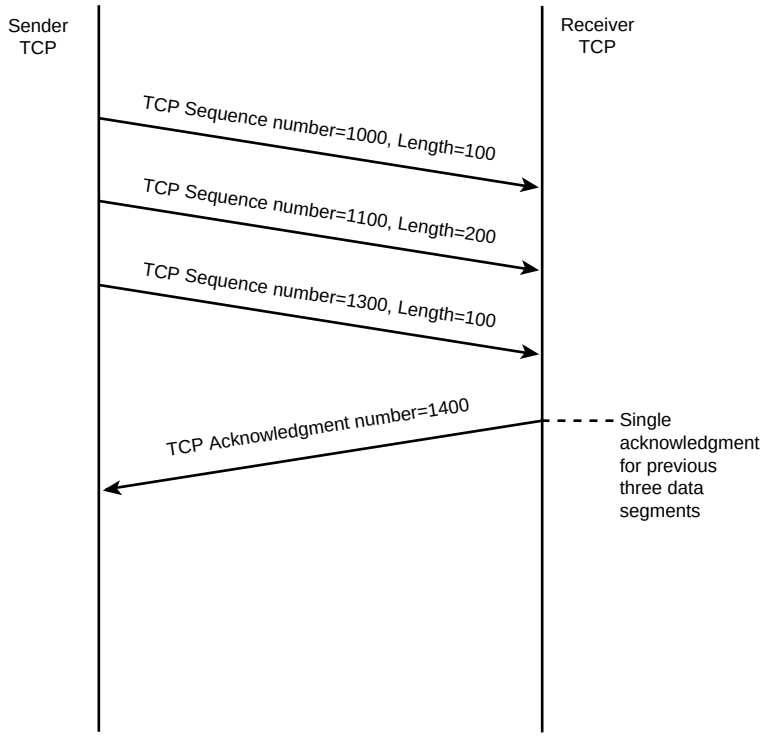



**FIGURE 11.22**
*Single acknowledgment for multiple data segments.*

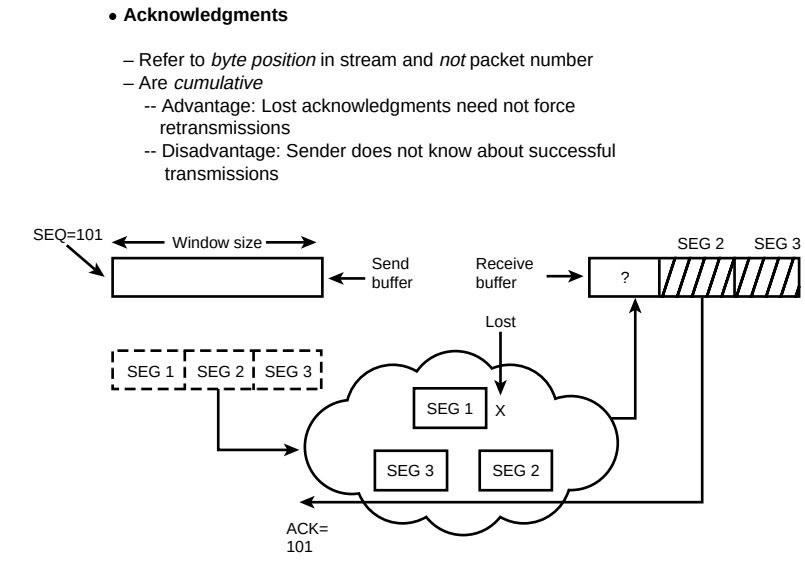
Many TCP implementations try to minimize the number of separate acknowledgment segments that need to be sent. For example, a TCP implementation may try to send a single acknowledgment for every two TCP data segments that are received.

Because TCP segments are ultimately encapsulated in IP datagrams, there is no guarantee that they will arrive in order at the destination. The receiver TCP uses the sequence numbers of the octets in the segments to construct a continuous stream of octets. The receiver acknowledges the longest continuous prefix octet stream that it has received so far. If there is a gap in the octet stream that represents data not received, the received TCP can acknowledge only data up to the gap.

Cumulative acknowledgments have the advantage in that lost acknowledgments do not force retransmission. Therefore, even if one acknowledgment is lost in transmission, subsequent acknowledgments will acknowledge all the data that has been received so far. However, cumulative acknowledgments are not efficient when there is a gap representing lost data and when data after the gap has been received. Data received after the gap cannot be acknowledged unless the missing data in the gap is received. The sender does not receive information about the successful transmission of data after the gap because the

receiver can acknowledge only data prior to the gap. This situation is illustrated in Figure 11.23. In this figure, segments SEG2 and SEG3 have been successfully received. However, segment SEG1 has been lost in transmission.

**FIGURE 11.23**
*A TCP sender does not know about successful transmissions. (Courtesy Learning Tree)*




Even though segments SEG2 and SEG3 have been successfully received, the loss of segment SEG1 represents a gap in the data stream at the receiver. The sender cannot advance the window because it does not receive acknowledgment about SEG1. There is no mechanism for the receiver to tell the sender that it has received segments SEG2 and SEG3 successfully. At this point, the following two situations can arise:

1. The sender will time-out and send segments SEG1, SEG2, and SEG3. In this case, segments SEG2 and SEG3 are sent needlessly because they were previously sent successfully. Retransmission of segments SEG2 and SEG3 is inefficient and adds to additional network traffic.
2. The sender will time-out and send only segment SEG1 and wait for acknowledgment to come back before deciding on sending additional data. When the receiver receives SEG1, it sends a cumulative acknowledgment packet that acknowledges correct reception of segment SEG1 and the previously sent segments SEG2 and SEG3. Although this scenario may appear to be more efficient, you should take into account that the sender has to wait for an acknowledgment of SEG1 before sending SEG2 and SEG3. This wait means that the sender is not taking advantage of its large window size and is sending only a segment at a time.

## Adaptive Time-Outs in TCP

TCP was originally designed to run over WAN links; later on, TCP adapted for LANs also. Therefore, TCP has an advantage over those protocols that have their origins in LANs only. This advantage means that TCP has sophisticated algorithms that adjust to the wide variance in time delays that are encountered in sending messages over WAN links. LANs, on the other hand, typically do not exhibit such wide variances in time delays.

An interesting fact to note about IP is that even if the IP layers send the same message over the same path at two different times, the time delay to reach the destination is probably different because of the following factors:

- WAN links typically send messages serially, one at a time. If the WAN link is busy sending other messages, the message has to wait in a queue. This process leads to uncertainties in time delay.
- Routers connecting the WAN links may be flooded with bursts of traffic. The IP datagrams that a router receives may have to wait in a queue until the time the router can resolve the route for the datagram. Even after its route is determined, the datagram may have to queue up for a network interface over which it is to be routed to become available. In the worst case, the datagrams arrive at a rate that fills the available router memory, in which case the router will have to drop datagrams that it cannot store in its memory. This scenario can happen because there is no pre-allocation of memory for individual TCP connections in an IP datagram network.

The IP Layer is so designed that each IP datagram is routed independently. Successive TCP segments in a connection may not be sent on the same path and may experience different delays as they traverse different sets of routers and links.

When TCP sends a message, it waits for acknowledgments to come back before it can advance its window. How long should it wait? The time it should wait is called the *time-out interval*. Because of the wide variance in time delays between successive transmissions in TCP, it uses an adaptive time-out. The algorithm for this adaptive time-out is called the *adaptive retransmission algorithm*. This algorithm is described in the following steps:

1. Measure the elapsed time between sending a data octet with a particular sequence number and receiving an acknowledgment that covers that sequence number. This measured elapsed time is the round-trip time (RTT). Based on the RTT, compute a smoothed round-trip time (SRTT) as

SRTT = ($\alpha \times$ SRTT) + (($1 - \alpha \times$ RTT)

$\alpha$ is the weighting factor whose value is between 0 and 1 ($0 \leq \alpha < 1$). The SRTT is an estimated time-out. The SRTT computation, therefore, takes into account its old SRTT value and the new RTT value and takes a weighted average of these two. When $\alpha = 0.5$, SRTT is just the simple average of old SRTT and the following RTT values:

SRTT = (SRTT + RTT) ÷ 2, when $\alpha = 0.5$

When $\alpha$ is less than 0.5, a greater weight is given to the actual round-trip time (RTT). When $\alpha$ is greater than 0.5, a greater weight is given to the preceding estimated smoothed round-trip time (SRTT).

When $\alpha = 0$ or almost 1, you have the two extremes. When $\alpha = 0$, SRTT is always set to RTT; and when $\alpha$ is almost 1, a fixed SRTT is used with little deference to the actual RTT. Needless to say, the two extremes are seldom used in actual practice, but they are useful for understanding the nature of the weighted average.

2. The SRTT is then used to compute the retransmission time-out (RTO) by using the following:

   RTO = min[UBOUND,max[LBOUND,($\beta \times$ SRTT)]]

   UBOUND is an upper boundary on the time-out (for example, 1 minute), and LBOUND is a lower boundary on the time-out (for example 1 second). These values ensure that the RTO will fall between LBOUND and UBOUND. $\beta$ is a delay variance factor that changes the sensitivity of the delay. If $\beta$ is set to 1, RTO will be set to SRTT unless the SRTT value is below LBOUND or above UBOUND. This value ($\beta = 1$)will make TCP sensitive to packet loss and will not cause TCP to wait for a long time before retransmitting. However, small delays may cause unnecessary retransmissions. To make TCP less susceptible to small delays, a larger $\beta$ value can be used. RFC 793 suggests using a value from 1.3 to 2.0.

In 1987, Karn and Partridge published a paper, "Improving Round-Trip Estimates in Reliable Transport Protocols," in the Proceedings of ACM SIGCOMM '87. Although not issued as an RFC, this paper influenced several TCP implementations, particularly the BSD UNIX implementation of TCP. This algorithm has since become known as *Karn's algorithm*. RFC 1122 recommends that Karn's algorithm be used in conjunction with Van Jacobson's slow start algorithm for congested networks. Van Jacobson's slow start algorithm is discussed in the section "Minimizing Impact of Congestion in TCP" in this chapter. Karn's algorithm takes into account the fact of retransmitted segments and is described next.

When retransmissions take place in Karn's algorithm, the SRTT value is not adjusted. Instead, a backoff strategy is used to adjust the time-out by a factor. The idea is that if a retransmission takes place, something drastic happened on the network, and the time-out should be increased sharply to avoid further retransmissions. The timer backoff strategy is implemented by using a simple equation such as the following:

$$\text{Backoff TIMEOUT} = \Gamma \times \text{TIMEOUT}$$

This equation is used until some previously decided upon upper boundary is reached. Typically, $\Gamma$ is set to 2 so that this algorithm behaves like a binary exponential backoff algorithm. The Backoff TIMEOUT does not affect the SRTT that is used in normal computation. When an acknowledgment is received for a segment that does not require retransmission, TCP measures the RTT value and uses this value to compute the SRTT and the RTO values.

## Minimizing Impact of Congestion in TCP

When congestion occurs in a network, TCP can respond with adjusting the time-out and retransmitting when acknowledgments are not received within the time-out interval. However, retransmissions can add to the problems of an already congested network by increasing the amount of data already being processed by the network. If retransmissions increase to a point that the retransmitted data being added to the network approaches the storage capacity of the network, then a condition called *congestion collapse* is imminent. When a network is experiencing congestion, a method is needed to stop or slow down additional data from being added to the network.

In 1988, Van Jacobson published a paper on "Congestion Avoidance and Control" in Proceedings ACM SIGCOMM'88. This paper outlines a technique for responding to congestion by reducing the amount of data sent by the sender. Routers that detect congestion can send the ICMP Source Quench message to the sender. The sender passes this message to the TCP module, which is alerted to a congestion condition.

**Note**

Techniques such as the Jacobson's slow start algorithm can deal successfully with congestion by adjusting the TCP window size dynamically.

Jacobson's slow start algorithm tries to avoid congestion by automatically adjusting the TCP window size, and it is implemented in all modern TCP implementations. In fact, Jacobson's algorithm, along with Karn's algorithm, is required to be implemented as per RFC 1122.

In Jacobson's slow start algorithm, TCP keeps track of a second limit called the *congestion window size*. The actual window size that is used is the smaller of the congestion window size and the TCP receiver window size.

Actual window size = min {congestion window size, TCP window size}

Under normal network conditions when there is no congestion, the congestion window size is the same as the TCP window size, which means that the actual window size is the same as the TCP window size. When congestion is detected, such as when there is a loss of segment, Jacobson's algorithm reduces the congestion window size by half. At every successive loss of a segment, the congestion window is again reduced by half. This process results in a rapid exponential decrease in window size. If the loss continues, the TCP window size is reduced to one segment, which means that only one datagram is transmitted at a time. At the same time, Karn's algorithm is used to double the time-out value before retransmitting. This dramatic reduction is called *exponential backoff* or *multiplicative decrease*.

Jacobson's algorithm uses a slow start to recover from congestion. If an acknowledgment is received for a segment, the algorithm increases the congestion window size by one segment. Because the initial window size for a severely congested network is 1, when an acknowledgment is received, the congestion window size becomes 2. This process means that TCP can now send two segments. If the acknowledgments for these segments come back, the congestion window size becomes 4, which enables TCP to send four segments. When the acknowledgments for these segments come back, the congestion window size becomes 8. This process continues until the congestion window size is equal to the receiver window size. Because of the exponential increase in congestion window size, it takes only $\log_2(N)$ successful transmissions before TCP can send $N$ segments.

If the congestion window size increases too rapidly, it can add to network traffic before the network has completely recovered from congestion. If congestion is experienced again, the congestion window size will shrink rapidly. This alternating increase and decrease in congestion window size causes the congestion window size to oscillate rapidly. Jacobson's algorithm prevents too rapid an increase in congestion window size by using the *congestion avoidance* technique. In this technique, when the congestion window size becomes one-half the original TCP window size, the congestion window size is increased by one segment only if all the segments sent so far have been acknowledged.

## Avoiding the Silly Window Syndrome

Early TCP implementations exhibited a pathological condition called the *Silly Window Syndrome (SWS)*. This condition was so named because the TCP transmission took place

one octet at a time under certain conditions. Therefore, a TCP segment consists of one octet only, and each octet is individually acknowledged. The SWS can occur when any of the following is true:

- The application at the receiver reads data one octet at a time from a buffer that is full, and the receiver TCP sends a window size of 1 in an acknowledgment to the sender (see Figure 11.24).
- The application at the sender generates data one octet at a time, and the sender TCP sends this data immediately (see Figure 11.25).

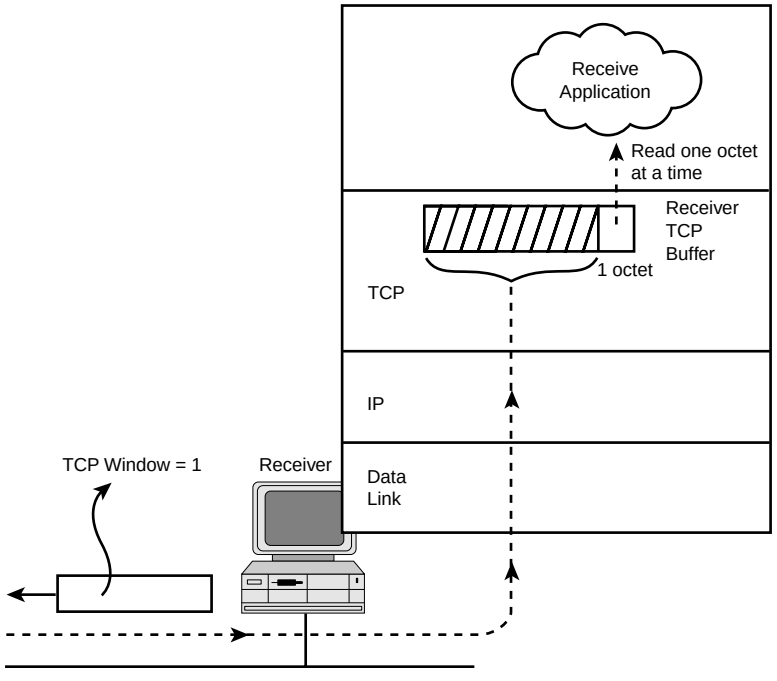



**FIGURE 11.24**
*Silly Window Syndrome caused by receiver.*

In Figure 11.24, because the receiver sends a window size of 1 octet, the sender responds by sending only one octet of data. As soon as this octet is received, the receiver application reads just one octet of data, and the receiver TCP again advertises a window size of 1. And the cycle continues.

In Figure 11.25, the sender application sends only one octet of data. The sender TCP does not wait to see whether other data is accumulated but sends this one octet immediately. As the sender application continues to generate data one octet at a time, each octet is sent immediately.

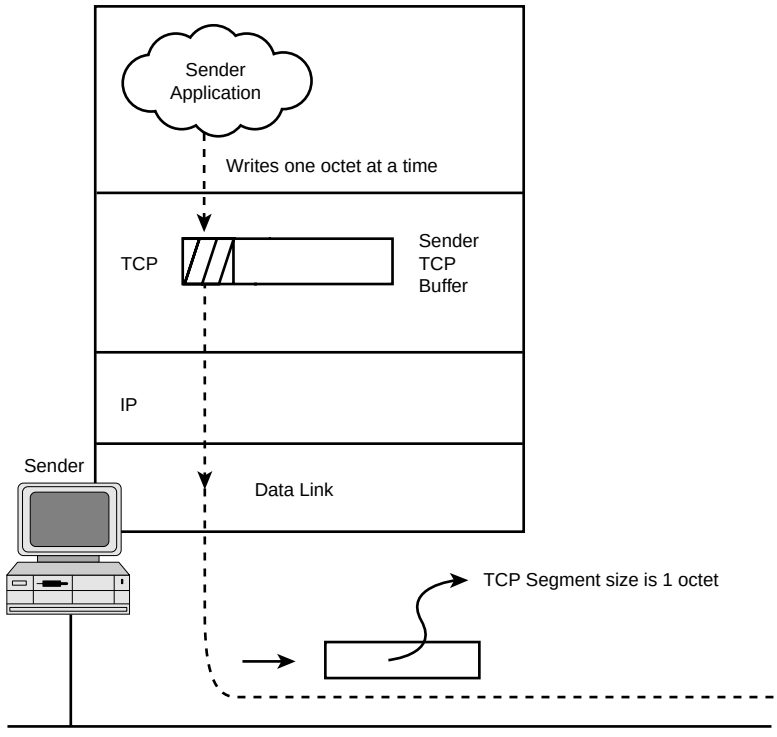



**FIGURE 11.25**
*Silly Window Syndrome caused by sender.*

In both situations, the overhead of sending the data is enormous because of the following reasons:

- A large number of small packets is generated.
- Each packet is processed separately by each of the Data Link, IP, and TCP Layers at the sender and the receiver. Intervening routers must process the packets at the Data Link and IP Layers.
- There is a large computational overhead in each of the protocol layers because each packet must be individually routed, checksummed, and encapsulated/decapsulated. Each packet must also have its sequence and acknowledgment numbers computed, and must be handled by send/receive buffers.
- The ratio of the protocol headers to actual payload (data delivered) is high; that is, protocol efficiency is low. As a result, data throughput is low.
- As each octet segment is individually acknowledged, the delays for each octet accumulate to a significant value, and the data throughput is low. For 1,000 octets sent in 1,000 packets over a link with 50 ms delay, total delay is 1,000 × 50 ms = 5 seconds. Assuming that the time to transmit the packet to the link is small compared to the delay of the link, the overall throughput is only 1,000 octets × 8 bits/octet ÷ 5 seconds = 1,600 bps.

To avoid the SWS on the receiver side, TCP implementations use the following mechanisms:

- Delay advertising. If the receive window size is smaller than a given size, the receiver delays advertising the window size until the receiver window size is at least equal to the following:

  min (0.5 × Receiver buffer size, Maximum Segment Size)

- Delay acknowledgments. Acknowledgments that contain the receiver window size computed in the previous bullet are delayed. This delay reduces network traffic by decreasing the number of acknowledgments. Recall that TCP acknowledgments are cumulative, so a later acknowledgment can acknowledge all data received so far. The risk here is that the delaying of acknowledgments can cause the sender to time-out and retransmit segments. To prevent this occurrence, ACKs are delayed to a maximum of 500 milliseconds, after which they must be sent. Many implementations delay acknowledgments by no more than 200 milliseconds. Additionally, the receiver should acknowledge every other data segment to ensure that the sender has a sufficient number of round-trip time estimates to implement its adaptive time-out algorithm.

Rather than delaying the acknowledgment, you can use another approach. In this approach, the acknowledgment is not delayed, but it is sent immediately without advertising an increase in window size until the prespecified limit has been reached. This approach is not recommended by the TCP standards, however.

To avoid the SWS on the sender side, TCP implementations use the Nagle avoidance algorithm. The basic ideas behind Nagle's SWS avoidance algorithm include the following:

- The sender sends the first data segment. After the first data segment, the sender does not immediately send short size segments even if the sender application sets the PSH flag.
- Data is sent only if the following is true:
  1. There is sufficient data to send to fill a maximum-size segment.
  2. An acknowledgment arrives while waiting for data to send.
  3. A preset time-out occurs.

If an application rapidly generates data, it will rapidly fill up the sender TCP buffer to the maximum segment size. TCP will send the maximum segment size without delay. Also, when an acknowledgment is received, the segment data in the buffer is sent without delay even if it is smaller than the maximum segment size.

If the application slowly generates data, segments will be sent when successive acknowledgments are received. Meanwhile, the sender TCP buffer will accumulate data.

The Nagle avoidance algorithm can be disabled for real-time applications that require minimum delay and send large bursts of data at a time.

## Dealing with Dead TCP Connections

When a remote computer crashes or the network links are down, the TCP connections will break. How does TCP detect a connection that is dead and release its resources? An ICMP message may arrive informing TCP that the destination is unreachable. TCP retries a few times by retransmitting segments before reporting the condition to the application. What if TCP does not receive any ICMP messages because they are lost in transmission? TCP then retries for a first threshold number of retransmissions, after which TCP notifies IP to check whether a dead router or route is the problem. IP tries to recompute the route to the destination. Meanwhile, TCP retries for a second threshold number of transmissions, after which TCP declares that the connection is dead. On detecting a dead connection, TCP must release all resources such as TCB (transmission control buffers) associated with the connection.

When neither end point has sent data for a long time, the connection is maintained in the idle state. In the idle state, the connection consumes resources to maintain the connection even if the network or one of the other end points is down. Some TCP implementations send a keep-alive TCP message, which periodically checks to see whether the connection is alive. When a keep-alive TCP message is sent, an acknowledgment is expected from the receiver. The suggested default for the keep-alive timer is two hours. In some TCP implementations, the default is much smaller (for example, 10–15 minutes). The keep-alive timers should be set to what makes sense for the majority of applications running on a TCP host. Keeping this timer too short will generate excessive traffic, and keeping the timer value too long will not quickly alert TCP that the connection is dead.

## TCP Finite State Machine

Figure 11.26 shows the behavior of the TCP as a finite state machine. A *finite state machine* is a logical model for the behavior of a system whose internal state changes with the occurrence of specified events. The boxes represent the state of the machine, and the arrows represent the transition from one state to another. The labels on the arrows represent the event that caused the transition between the states and the response of TCP to that event in the following format:

$$\frac{\textit{event}}{\textit{response}}$$

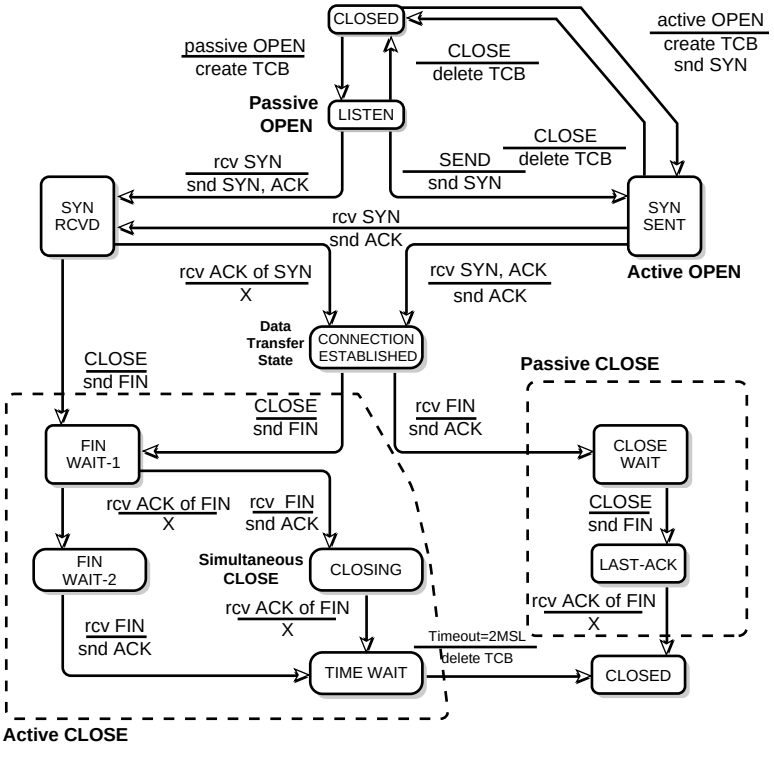
In Figure 11.26, an *x* for the *response* indicates that no special response occurs.

Each end point of a TCP connection starts from the closed state. Arrows from this closed state are shown to indicate that a connection can occur by an active OPEN or a passive OPEN.

An active OPEN causes TCP to create a TCB and send a SYN flag segment and enter into the SYN SENT state. When a SYN flag and ACK flag are sent by the other end point, TCP sends an acknowledgment to complete the three-way handshake and enters the CONNECTION ESTABLISHED state.

When a passive OPEN is issued by an end point, it creates a TCB and enters into the LISTEN state. When TCP receives a SYN flag in this state (active OPEN request), it sends the flag's sequence number and the SYN and ACK flags in a TCP segment and enters the SYN RCVD state. When TCP receives an ACK or SYN flag from the other end point, the connection moves to the CONNECTION ESTABLISHED state.

**FIGURE 11.26**
*TCP finite state machine.*




**MSL**= Maxium Segment Lifetime (2 minutes per RFC 793)

**TCB** = Transmission Control Buffer (contains state information on the TCP end point)

The TIME WAIT state is entered upon closing the connection to ensure that there is a delay of twice the MSL (maximum segment lifetime) time interval. This state is used to avoid duplicate segment numbers. See the earlier discussion in the section "The Flags Field" on avoiding duplicate sequence numbers.

The FIN WAIT-1 and FIN WAIT-2 states show the graceful close mechanism of TCP. Both sides have to agree on closing the connection before a connection is closed.

## TCP Traces

The Telnet protocol uses TCP, and it is a good candidate for studying its protocol trace to examine the behavior of TCP. Figure 11.27 shows a complete Telnet trace of a user who performs the following activities:

1. Logs on to a Telnet server
2. Executes various commands at a Telnet server
3. Logs off from the Telnet server

**FIGURE 11.27**
*Telnet trace.*

```
No.   Source        Destination   Layer  Size  Summary
     1 0000C024282D  FFFFFFFFFFFF  arp    0064  Req by 144.19.74.44 for 144.19.74
     2 0000C024282D  FFFFFFFFFFFF  arp    0064  Req by 144.19.74.44 for 144.19.74
     3 0000C0A20F8E  0000C024282D  arp    0064  Reply 144.19.74.201=0000C0A20F8E
     4 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET SYN
     5 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK SYN
     6 0000C024282D  0000C0A20F8E  telnt  0069  Cmd=Do; Code=Echo; Cmd=Do; Code=S
     7 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
     8 0000C0A20F8E  0000C024282D  telnt  0064  Cmd=Do; Code=;
     9 0000C024282D  0000C0A20F8E  telnt  0064  Cmd=Won't; Code=;
    10 0000C0A20F8E  0000C024282D  telnt  0067  Cmd=Will; Code=Echo; Cmd=Will; Co
    11 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    12 0000C024282D  0000C0A20F8E  telnt  0072  Cmd=Do; Code=Suppress Go Ahead; C
    13 0000C0A20F8E  0000C024282D  telnt  0070  Cmd=Do; Code=Terminal Type; Cmd=D
    14 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    15 0000C024282D  0000C0A20F8E  telnt  0072  Cmd=Will; Code=Terminal Type; Cmd
    16 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    17 0000C0A20F8E  0000C024282D  telnt  0064  Cmd=Subnegotiation Begin; Code=Te
    18 0000C024282D  0000C0A20F8E  telnt  0071  Cmd=Subnegotiation Begin; Code=Te
    19 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    20 0000C0A20F8E  0000C024282D  telnt  0067  Cmd=Do; Code=Echo; Cmd=Will; Code
    21 0000C024282D  0000C0A20F8E  telnt  0069  Cmd=Won't; Code=Echo; Cmd=Don't;
    22 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    23 0000C0A20F8E  0000C024282D  telnt  0104  Data=..Linux 1.3.50 (ltreec1.ltre
    24 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
    25 0000C0A20F8E  0000C024282D  telnt  0064  Data=..
    26 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
    27 0000C0A20F8E  0000C024282D  telnt  0073  Data=ltreec1 login:
    28 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
    29 0000C024282D  0000C0A20F8E  telnt  0064  Data=u
    30 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    31 0000C0A20F8E  0000C024282D  telnt  0064  Data=u
    32 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
    33 0000C024282D  0000C0A20F8E  telnt  0064  Data=s
    34 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    35 0000C0A20F8E  0000C024282D  telnt  0064  Data=s
    36 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
    37 0000C024282D  0000C0A20F8E  telnt  0064  Data=e
    38 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
    39 0000C0A20F8E  0000C024282D  telnt  0064  Data=e
    40 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
    41 0000C024282D  0000C0A20F8E  telnt  0064  Data=r
```

```
 42 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 43 0000C0A20F8E  0000C024282D  telnt  0064  Data=r
 44 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 45 0000C024282D  0000C0A20F8E  telnt  0064  Data=1
 46 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 47 0000C0A20F8E  0000C024282D  telnt  0064  Data=1
 48 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 49 0000C024282D  0000C0A20F8E  telnt  0064  Data=..
 50 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 51 0000C0A20F8E  0000C024282D  telnt  0064  Data=..
 52 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 53 0000C0A20F8E  0000C024282D  telnt  0068  Data=Password:
 54 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 55 0000C024282D  0000C0A20F8E  telnt  0064  Data=u
 56 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 57 0000C024282D  0000C0A20F8E  telnt  0064  Data=s
 58 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 59 0000C024282D  0000C0A20F8E  telnt  0064  Data=e
 60 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 61 0000C024282D  0000C0A20F8E  telnt  0064  Data=r
 62 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 63 0000C024282D  0000C0A20F8E  telnt  0064  Data=1
 64 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 65 0000C024282D  0000C0A20F8E  telnt  0064  Data=p
 66 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 67 0000C024282D  0000C0A20F8E  telnt  0064  Data=w
 68 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 69 0000C024282D  0000C0A20F8E  telnt  0064  Data=..
 70 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 71 0000C0A20F8E  0000C024282D  telnt  0064  Data=....
 72 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 73 0000C0A20F8E  0000C024282D  telnt  0113  Data=Last login: Wed May 28 18:33
 74 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 75 0000C0A20F8E  0000C024282D  telnt  0073  Data=Linux 1.3.50...
 76 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 77 0000C0A20F8E  0000C024282D  telnt  0064  Data=..
 78 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 79 0000C0A20F8E  0000C024282D  telnt  0069  Data=Power, n:..
 80 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 81 0000C0A20F8E  0000C024282D  telnt  0119  Data=.The only narcotic regulated
 82 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 83 0000C0A20F8E  0000C024282D  telnt  0064  Data=..
 84 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 85 0000C0A20F8E  0000C024282D  telnt  0069  Data=ltreec1:~$
 86 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 87 0000C024282D  0000C0A20F8E  telnt  0064  Data=l
 88 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 89 0000C0A20F8E  0000C024282D  telnt  0064  Data=l
 90 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 91 0000C024282D  0000C0A20F8E  telnt  0064  Data=s
 92 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 93 0000C0A20F8E  0000C024282D  telnt  0064  Data=s
 94 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 95 0000C024282D  0000C0A20F8E  telnt  0064  Data=
 96 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
 97 0000C0A20F8E  0000C024282D  telnt  0064  Data=
 98 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
 99 0000C024282D  0000C0A20F8E  telnt  0064  Data=-
100 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
101 0000C0A20F8E  0000C024282D  telnt  0064  Data=-
102 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
103 0000C024282D  0000C0A20F8E  telnt  0064  Data=a
104 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
105 0000C0A20F8E  0000C024282D  telnt  0064  Data=a
106 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
107 0000C024282D  0000C0A20F8E  telnt  0064  Data=l
108 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
109 0000C0A20F8E  0000C024282D  telnt  0064  Data=l
110 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
111 0000C024282D  0000C0A20F8E  telnt  0064  Data=..
112 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
113 0000C0A20F8E  0000C024282D  telnt  0064  Data=..
114 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
115 0000C0A20F8E  0000C024282D  telnt  0067  Data=total 7..
116 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
```

```
117 0000C0A20F8E  0000C024282D  telnt  0130  Data=drwxr-xr-x   3 user1    user
118 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
119 0000C0A20F8E  0000C024282D  telnt  0131  Data=drwxr-xr-x  19 root     root
120 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
121 0000C0A20F8E  0000C024282D  telnt  0138  Data=-rw-r--r--   1 user1    user
122 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
123 0000C0A20F8E  0000C024282D  telnt  0132  Data=-rw-r--r--   1 user1    user
124 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
125 0000C0A20F8E  0000C024282D  telnt  0130  Data=-rw-r--r--   1 user1    user
126 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
127 0000C0A20F8E  0000C024282D  telnt  0132  Data=-rw-r--r--   1 user1    user
128 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
129 0000C0A20F8E  0000C024282D  telnt  0134  Data=drwxr-xr-x   2 user1    user
130 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
131 0000C0A20F8E  0000C024282D  telnt  0064  Data=l
132 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
133 0000C0A20F8E  0000C024282D  telnt  0068  Data=treec1:~$
134 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
135 0000C024282D  0000C0A20F8E  telnt  0064  Data=p
136 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
137 0000C0A20F8E  0000C024282D  telnt  0064  Data=p
138 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
139 0000C024282D  0000C0A20F8E  telnt  0064  Data=s
140 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
141 0000C0A20F8E  0000C024282D  telnt  0064  Data=s
142 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
143 0000C024282D  0000C0A20F8E  telnt  0064  Data=
144 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
145 0000C0A20F8E  0000C024282D  telnt  0064  Data=
146 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
147 0000C024282D  0000C0A20F8E  telnt  0064  Data=-
148 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
149 0000C0A20F8E  0000C024282D  telnt  0064  Data=-
150 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
151 0000C024282D  0000C0A20F8E  telnt  0064  Data=a
152 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
153 0000C0A20F8E  0000C024282D  telnt  0064  Data=a
154 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
155 0000C024282D  0000C0A20F8E  telnt  0064  Data=l
156 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
157 0000C0A20F8E  0000C024282D  telnt  0064  Data=l
158 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
159 0000C024282D  0000C0A20F8E  telnt  0064  Data=x
160 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
161 0000C0A20F8E  0000C024282D  telnt  0064  Data=x
162 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
163 0000C024282D  0000C0A20F8E  telnt  0064  Data=..
164 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
165 0000C0A20F8E  0000C024282D  telnt  0064  Data=..
166 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
167 0000C0A20F8E  0000C024282D  telnt  0133  Data= F    UID   PID  PPID PRI NI
168 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
169 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0     1     0  30 15
170 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
171 0000C0A20F8E  0000C024282D  telnt  0143  Data=  0     0     2     1  30 15
172 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
173 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0     8     1  30 15
174 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
175 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    24     1  30 15
176 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
177 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    38     1  30 15
178 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
179 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    40     1  30 15
180 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
181 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     1    42     1  30 15
182 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
183 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    44     1  30 15
184 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
185 0000C0A20F8E  0000C024282D  telnt  0139  Data=  0     0    46     1  30 15
```

```
186 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
187 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    49     1  30 15
188 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
189 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    51     1  30 15
190 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
191 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    53     1  30 15
192 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
193 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    57     1  30 15
194 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
195 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    61     1  30 15
196 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
197 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    62     1  30 15
198 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
199 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    63     1  30 15
200 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
201 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    64     1  30 15
202 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
203 0000C0A20F8E  0000C024282D  telnt  0140  Data=  0     0    65     1  30 15
204 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
205 0000C0A20F8E  0000C024282D  telnt  0131  Data=  0     0   145     1  30 15
206 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
207 0000C024282D  0000C0A20F8E  telnt  0064  Data=.
208 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
209 0000C0A20F8E  0000C024282D  telnt  0136  Data=  0     0   258    44  29 15
210 0000C0A20F8E  0000C024282D  telnt  0064  Cmd=;
211 0000C024282D  0000C0A20F8E  telnt  0064  Data=.
212 0000C0A20F8E  0000C024282D  telnt  0064  Data=.
213 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
214 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
215 0000C0A20F8E  0000C024282D  telnt  0071  Data=..ltreec1:~$
216 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
217 0000C024282D  0000C0A20F8E  telnt  0064  Data=l
218 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
219 0000C0A20F8E  0000C024282D  telnt  0064  Data=l
220 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
221 0000C024282D  0000C0A20F8E  telnt  0064  Data=o
222 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
223 0000C0A20F8E  0000C024282D  telnt  0064  Data=o
224 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
225 0000C024282D  0000C0A20F8E  telnt  0064  Data=g
226 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
227 0000C0A20F8E  0000C024282D  telnt  0064  Data=g
228 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
229 0000C024282D  0000C0A20F8E  telnt  0064  Data=o
230 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
231 0000C0A20F8E  0000C024282D  telnt  0064  Data=o
232 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
233 0000C024282D  0000C0A20F8E  telnt  0064  Data=u
234 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
235 0000C0A20F8E  0000C024282D  telnt  0064  Data=u
236 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
237 0000C024282D  0000C0A20F8E  telnt  0064  Data=t
238 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
239 0000C0A20F8E  0000C024282D  telnt  0064  Data=t
240 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK
241 0000C024282D  0000C0A20F8E  telnt  0064  Data=..
242 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
243 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK FIN
244 0000C024282D  0000C0A20F8E  tcp    0064  Port:6295 ---> TELNET ACK FIN
245 0000C0A20F8E  0000C024282D  tcp    0064  Port:TELNET ---> 6295 ACK
```

Packets 1 through 3 in Figure 11.27 deal with ARP resolution. ARP resolution is discussed in Chapter 5, "ARP and RARP." The actual Telnet trace begins from packet 4.

The following sections examine only the TCP aspect of the Telnet trace in Figure 11.27 and not the Telnet application data. The packet numbers in the following traces correspond to the packet numbers in Figure 11.27. For this reason, the entire trace of the

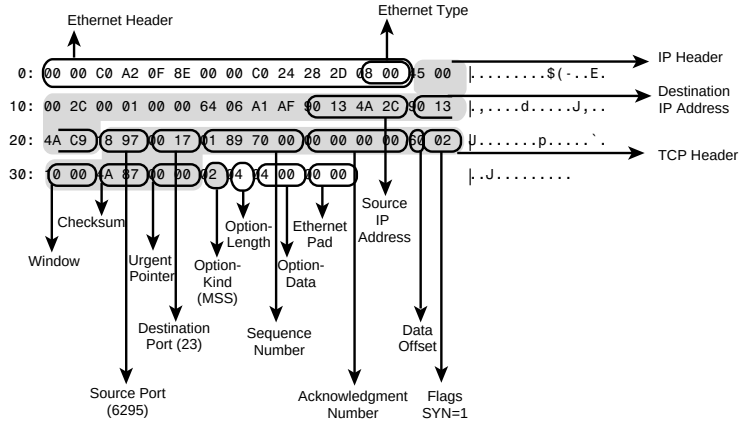
Telnet session is reproduced in Figure 11.27 for your reference, even though the trace itself is somewhat long.

## TCP Open Connection Three-Way Handshake Trace

Figures 11.28 through 11.30 show the three TCP packets involved in the three-way handshake that is used to open a TCP connection.

**FIGURE 11.28**
*OPEN TCP connection request: sending the initial sender sequence number (ISS).*

```
Packet Number : 4           6:38:38 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-24-28-2D ----> 00-00-C0-A2-0F-8E
       Type: 0x0800 (IP)
  ip: ====================== Internet Protocol ========================
       Station:144.19.74.44 ---->144.19.74.201
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 44
       Identification:     1
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 100 seconds
       Checksum: 0xA1AF(Valid)
 tcp: ================= Transmission Control Protocol =================
       Source Port: 6295
       Destination Port: TELNET
       Sequence Number: 25784320
       Acknowledgement Number: 0
       Data Offset (32-bit words): 6
       Window: 4096
       Control Bits: Synchronize Sequence Numbers (SYN)
       Checksum: 0x4A87(Valid)
       Urgent Pointer: 0
       Option:MAXIMUM SEGMENT SIZE
              Option Length: 4
              Maximum Segment Size : 1024
```



The OPEN connection request TCP packet has the following flags and sequence number and represents Step 1 in the three-way handshake:

e number = 25784320 (ISS)

Acknowledgment Number = 0 (not valid because ACK = 0)

Another interesting thing to note about the OPEN connection option is that the Data Offset field is 6, which indicates the existence of a TCP option. This option is the MSS option (option-kind = 2) with the following hex codes:

02 04 04 00

These codes translate to the following TCP option values:

| | |
|---|---|
| Option: | MAXIMUM SEGMENT SIZE |
| Option Length: | 4 |
| Maximum Segment Size: | 1,024 |

The source port, destination port, and window size of this initial packet are as follows:

| | |
|---|---|
| Source Port: | 6295 |
| Destination Port: | 23 (standard Telnet server port) |
| Window: | 4,096 |

The source port represents the port on which the Telnet client is listening to receive data from the Telnet server. The destination port is the port on the Telnet server that is being contacted for opening a connection. The window is the number of octets that the Telnet client can receive; that is, it is the size of the Telnet client computer's TCP receive buffer.

Figure 11.29 shows the acknowledgment of the OPEN connection request TCP packet by the receiver. This is Step 2 in the three-way handshake:

SYN = 1

ACK = 1

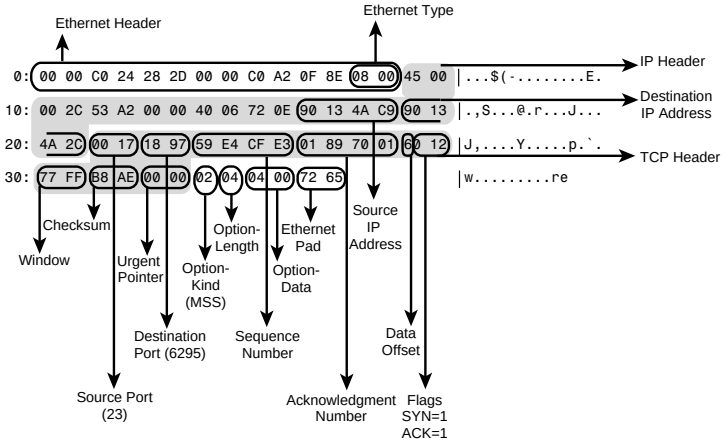
*All other TCP flags set to 0*

Sequence Number = 1508167651 (IRS)

Acknowledgment Number = 25784321 (ISS + 1)

**FIGURE 11.29**
*Acknowledgment of OPEN connection request: sending the initial receiver sequence number (IRS).*

```
Packet Number : 5          6:38:38 PM
Length : 64 bytes
ether: ===================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
  ip: ======================= Internet Protocol ========================
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 44
       Identification: 21410
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x720E(Valid)
 tcp: ================= Transmission Control Protocol =================
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508167651
       Acknowledgement Number: 25784321
       Data Offset (32-bit words): 6
       Window: 30719
       Control Bits: Acknowledgement Field is Valid (ACK)
              Synchronize Sequence Numbers (SYN)
       Checksum: 0xB8AE(Valid)
       Urgent Pointer: 0
       Option:MAXIMUM SEGMENT SIZE
              Option Length: 4
              Maximum Segment Size : 1024
```




The packet in Figure 11.29 is sent by the Telnet server to synchronize its sequence number and contains the server's initial sequence number. Another interesting thing to note about the OPEN connection option is that the Data Offset field is 6, which indicates the existence of a TCP option. This option is the MSS option (option–kind = 2) with the following hex codes:

02 04 04 00

These codes translate to the following TCP option values:

| | |
|---|---|
| Option: | MAXIMUM SEGMENT SIZE |
| Option Length: | 4 |
| Maximum Segment Size: | 1,024 |

In this example, both the Telnet client and server use the same MSS size, but this situation does not always have to exist. The source port, destination port, and window size of the OPEN connection acknowledgment packet are as follows:

| | |
|---|---|
| Source Port: | 23 (standard Telnet server port) |
| Destination Port: | 6295 |
| Window: | 30,719 |

The source port represents the port on which the Telnet server is listening to receive data from the Telnet client. The destination port is the port on the Telnet client that made the request for opening a connection. The window is the number of octets that the Telnet server can receive; that is, it is the size of the Telnet server computer's TCP receive buffer. Note that this number is larger than that of the Telnet client.

Figure 11.30 shows the last step in the three-way handshake. In this step the Telnet server's sequence number received by the Telnet client is being acknowledged. This packet has the following flags and sequence numbers:

ACK = 1

PSH = 1

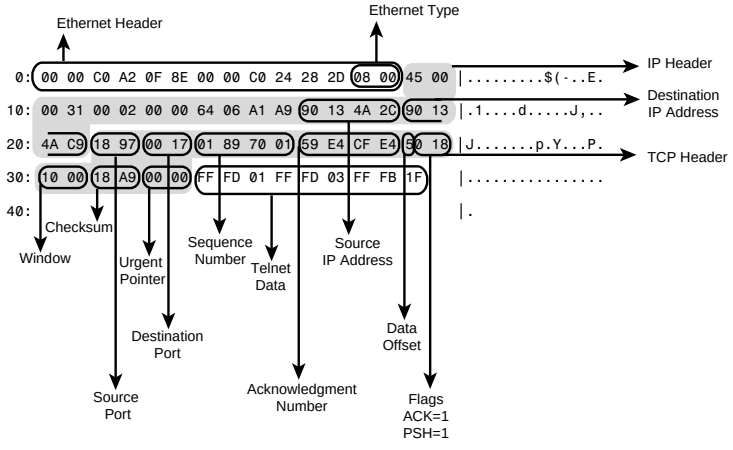
*All other TCP flags set to 0*

Sequence Number = 25784321 (ISS + 1)

Acknowledgment Number = 1508167652 (IRS + 1)

**FIGURE 11.30**
*Acknowledgment of initial receiver sequence number (IRS).*

```
Packet Number : 6           6:38:38 PM
Length : 69 bytes
ether: ========== Ethernet Datalink Layer ==========
       Station: 00-00-C0-24-28-2D ----> 00-00-C0-A2-0F-8E
       Type: 0x0800 (IP)
  ip: ========== Internet Protocol ==========
       Station:144.19.74.44 ---->144.19.74.201
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 49
       Identification:     2
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 100 seconds
       Checksum: 0xA1A9(Valid)
 tcp: ========== Transmission Control Protocol ==========
       Source Port: 6295
       Destination Port: TELNET
       Sequence Number: 25784321
       Acknowledgement Number: 1508167652
       Data Offset (32-bit words): 5
       Window: 4096
       Control Bits: Acknowledgement Field is Valid (ACK)
              Push Function Requested (PSH)
       Checksum: 0x18A9(Valid)
       Urgent Pointer: 0
telnt: ========== Telnet Protocol ==========
       Command: Do
       Option Code: Echo
       Command: Do
       Option Code: Suppress Go Ahead
       Command: Will
       Option Code:
```




The PSH flag is set to suggest to TCP that it should send the data. Nagle's avoidance technique is used to prevent the Silly Window Syndrome (SWS). See the earlier section "Avoiding the Silly Window Syndrome" for details on Nagle's algorithm to avoid SWS.

Another interesting thing to note about the OPEN connection option is that the Data Offset field is 5, which indicates no TCP options.

The source port, destination port, and window size of this initial packet are as follows:

| | |
|---|---|
| Source port: | 6295 |
| Destination port: | 23 (standard Telnet server port) |
| Window: | 4,096 |

The source port represents the end point of the TCP connection on the Telnet client. The destination port represents the end point of the TCP connection on the Telnet server. The window is the number of octets that the Telnet client can receive; that is, it is the size of the Telnet client computer's TCP receive buffer.

## TCP Normal Data Mode Trace

In packet 6 (refer to Figure 11.30), which concludes the three-way handshake and acknowledges the TCP receiver's IRS, nine octets of Telnet data are sent:

FF FD 01 FF FD 03 FF FB 1F

How do you know that nine octets of Telnet data have been sent? Examine the IP header field to determine the size of the IP datagram (refer to Figure 11.30). The IP datagram is 49 octets. Of these 49 octets, 20 octets (Header Length = 5 words) are taken up by the IP header, and 20 octets (Data Offset = 5 words) are taken up by the TCP header. What remains is 49 – 20 – 20 = 9 octets for the Telnet data.

Packet 6 represents the start of data transmission mode in the TCP trace. Packet 7 (see Figure 11.31) contains an acknowledgment of this data from the Telnet server. The Telnet server at this point does not have any data to send and, therefore, sends only an acknowledgment.

Note that this packet has the following flags and sequence numbers:

ACK = 1
*All other TCP flags set to 0*
Sequence Number: 1508167652
Acknowledgment Number: 25784330

Note that the acknowledgment number that is sent back contains the acknowledgment for the nine octets of Telnet client data:

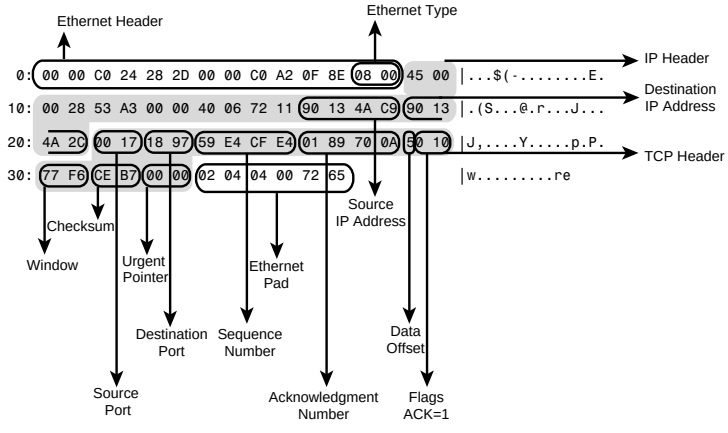
ISN from Telnet client = 25784321

Telnet data = 9

Acknowledgment Number data = ISN from Telnet client + Telnet
= 25784321 + 9
= 25784330

**FIGURE 11.31**
*Acknowledgment packet.*

```
Packet Number : 7          6:38:38 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
  ip: ======================= Internet Protocol =======================
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 40
       Identification: 21411
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x7211(Valid)
 tcp: ================ Transmission Control Protocol =================
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508167652
       Acknowledgement Number: 25784330
       Data Offset (32-bit words): 5
       Window: 30710
       Control Bits: Acknowledgement Field is Valid (ACK)
       Checksum: 0xCEB7(Valid)
       Urgent Pointer: 0
```




One second after the acknowledgment packet is sent, the Telnet server sends a response to the Telnet client data in packet 8 (see Figure 11.32). If the Telnet server had been ready to send this response right away, it could have done so in packet 7. TCP will delay sending acknowledgments to avoid sending small packets, but the maximum delay

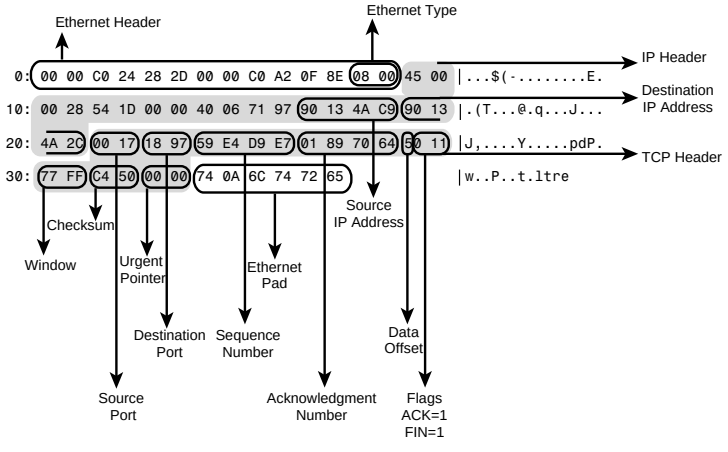
according to the TCP standard is 500 milliseconds (see the discussion in the previous section "Avoiding the Silly Window Syndrome"). You know that one second has transpired between packets 7 and 8 by examining the timestamps that are displayed on the first line in the protocol decodes. These timestamps indicate the time that a packet was captured by using a protocol analyzer.

The amount of application data that is sent in packet 8 is computed as before:

IP Total Length – IP Header Length – TCP Data Offset
= 43 – 20 – 20 = 3 octets

**FIGURE 11.32**
*TCP data transmission.*

```
Packet Number : 243         6:39:12 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
   ip: ====================== Internet Protocol ======================
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 40
       Identification: 21533
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x7197(Valid)
  tcp: ================= Transmission Control Protocol =================
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508170215
       Acknowledgement Number: 25784420
       Data Offset (32-bit words): 5
       Window: 30719
       Control Bits: Acknowledgement Field is Valid (ACK)
              No More Data from Sender (FIN)
       Checksum: 0xC450(Valid)
       Urgent Pointer: 0
```

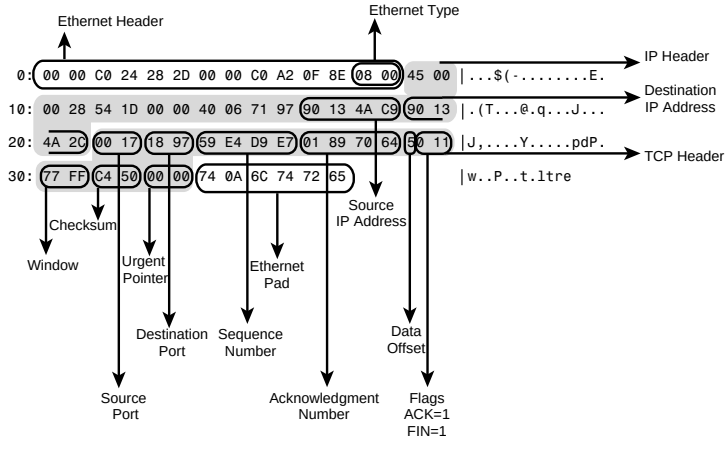


## TCP Graceful Close Trace

Packets 217 through 242 (refer to Figure 11.27) show the Telnet client transmitting the logout command one character at a time to the Telnet server. Packets 243 through 245 illustrate the graceful close mechanism. In packet 243 (see Figure 11.33), the Telnet server that receives the logout command requests a close of the TCP connection by setting the FIN flag to 1. This flag indicates that the Telnet server has no more data to send.

**FIGURE 11.33**
*Graceful close request issued by Telnet server.*

```
Packet Number : 243         6:39:12 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
  ip: ====================== Internet Protocol ======================
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 40
       Identification: 21533
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x7197(Valid)
 tcp: ================= Transmission Control Protocol =================
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508170215
       Acknowledgement Number: 25784420
       Data Offset (32-bit words): 5
       Window: 30719
       Control Bits: Acknowledgement Field is Valid (ACK)
              No More Data from Sender (FIN)
       Checksum: 0xC450(Valid)
       Urgent Pointer: 0
```

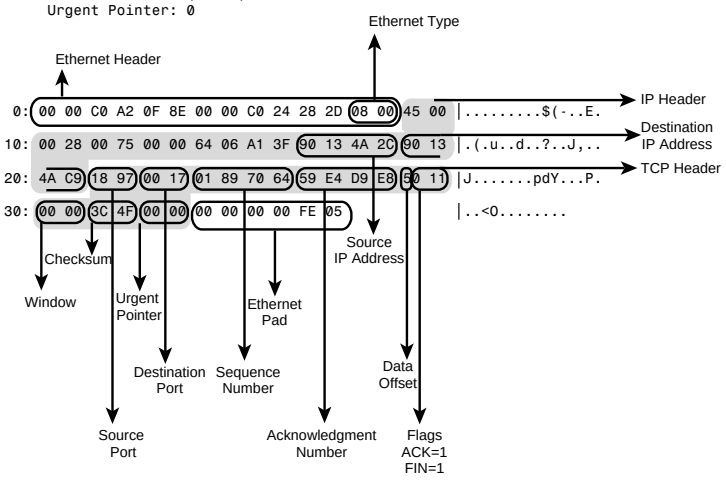



In packet 244 (see Figure 11.34), the Telnet client responds by setting the FIN flag to 1, thereby indicating that the client is ready to close the connection and that it has no more

data to send. Note that in this implementation, the window size is set to 0 to indicate that TCP at the Telnet client will not accept any more data.

**FIGURE 11.34**
*Graceful close request agreed upon by Telnet client.*

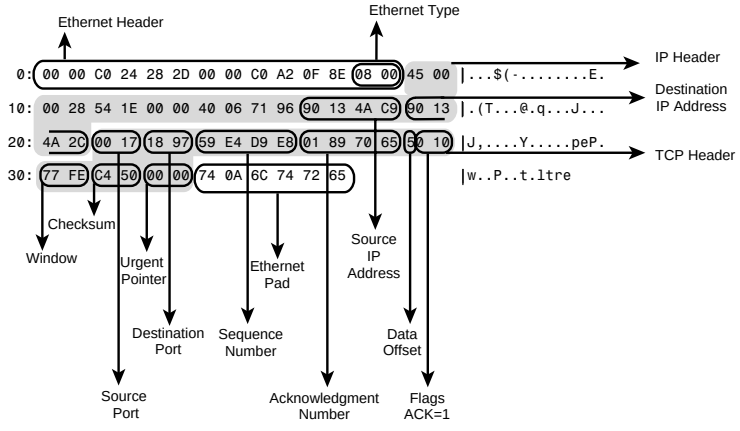
```
Packet Number : 244         6:39:12 PM
Length : 64 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-24-28-2D ----> 00-00-C0-A2-0F-8E
       Type: 0x0800 (IP)
  ip: ======================= Internet Protocol =======================
       Station:144.19.74.44 ---->144.19.74.201
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
             Normal Delay, Normal Throughput, Normal Reliability
       Total length: 40
       Identification:   117
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 100 seconds
       Checksum: 0xA13F(Valid)
 tcp: ================= Transmission Control Protocol =================
       Source Port: 6295
       Destination Port: TELNET
       Sequence Number: 25784420
       Acknowledgement Number: 1508170216
       Data Offset (32-bit words): 5
       Window: 0
       Control Bits: Acknowledgement Field is Valid (ACK)
             No More Data from Sender (FIN)
       Checksum: 0x3C4F(Valid)
       Urgent Pointer: 0
```




In Figure 11.35, the Telnet server acknowledges the reception of the FIN flag from the Telnet client. At this point, the TCP modules at the Telnet client and the server will release the resources associated with the TCP connection.

**FIGURE 11.35**
*Acknowledgment of graceful close request from Telnet client.*

```
Packet Number : 245         6:39:12 PM
Length : 64 bytes
ether: ========== Ethernet Datalink Layer ==========
       Station: 00-00-C0-A2-0F-8E ----> 00-00-C0-24-28-2D
       Type: 0x0800 (IP)
  ip: ========== Internet Protocol ==========
       Station:144.19.74.201 ---->144.19.74.44
       Protocol: TCP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 40
       Identification: 21534
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x7196(Valid)
 tcp: ========== Transmission Control Protocol ==========
       Source Port: TELNET
       Destination Port: 6295
       Sequence Number: 1508170216
       Acknowledgement Number: 25784421
       Data Offset (32-bit words): 5
       Window: 30718
       Control Bits: Acknowledgement Field is Valid (ACK)
       Checksum: 0xC450(Valid)
       Urgent Pointer: 0
```




# The User Datagram Protocol (UDP)

There are a number of applications for which the robustness of TCP is not required. Instead, what is needed is a transport protocol that can identify the applications on the computers and provide a rudimentary error check. The User Datagram Protocol (UDP) provides these capabilities.

Unlike TCP, which is connection oriented, UDP (as its name suggests) operates in the datagram mode. UDP makes no attempt to create a connection. Data is sent by encapsulating it in a UDP header and passing the data to the IP Layer. The IP Layer sends the UDP packet in a single IP datagram unless fragmentation is required. UDP does not attempt to provide sequencing of data; therefore, it is possible for data to arrive in a different order from which it was sent. Applications that need sequencing services must

either build their own sequencing mechanism as part of the application or use TCP instead of UDP. In many LAN environments, the chance of data being received out of sequence is small because of small predictable delays and simple network topology.

UDP is useful in applications that are command/response oriented and in which the commands and responses can be sent in a single datagram. There is no overhead involved in opening and then closing a connection just to send a small amount of data. Another advantage of UDP is for applications that require broadcast/multicast. In TCP, if a broadcast has to be sent to 1,000 stations, the sender has to open 1,000 connections, send data on each connection, and then close the 1,000 connections. The overhead of opening these connections, maintaining them (resource utilization), and then closing them is high. If UDP is used, the sender can send the data to the IP module requesting a broadcast/multicast distribution. The underlying network's broadcast/multicast capability may be used in sending the data.

UDP provides a rudimentary form of optional error checking over the integrity of the data. If the underlying network is known to be reliable, the UDP checksums can be disabled, which will speed up UDP processing.

The features of UDP are summarized as follows:

- UDP has a provision for identifying application processes by using UDP port numbers.
- UDP is datagram oriented with no overhead for opening, maintaining, and closing a connection.
- UDP is efficient for broadcast/multicast applications.
- There is no sequencing of data. Data cannot be correctly guaranteed for delivery.
- UDP uses optional error checksum of data only.
- UDP is faster, simpler and more efficient than TCP; however, it is also less robust then TCP.

UDP provides unreliable, connectionless delivery services over TCP. UDP is regarded as a Transport Layer protocol. As a Transport Layer protocol, UDP is somewhat of a paradox because the function of a Transport Layer is to provide end-to-end data integrity, which is something UDP does not do. Despite this drawback, a large number of practical applications are built around UDP. These applications include the following:

- TFTP (Trivial File Transfer Protocol)
- DNS (Domain Name System)
- NFS (Network File System) version 2
- SNMP (Simple Network Management Protocol)
- RIP (Routing Information Protocol)

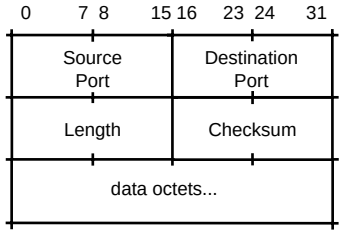
- Many services that broadcast data, such as WHOD (Who daemon on Unix servers)

The upcoming sections explore UDP in greater detail.

## UDP Header Format

The UDP header has a fixed length of eight octets. The format of the UDP header is shown in Figure 11.36.

**FIGURE 11.36**
*UDP header format.*




The Source Port is a 16-bit field. This field is optional. When meaningful, it indicates the following:

- The port number of the sending process.
- The source port number is the port to which a reply should be addressed in the absence of any other information.
- When set to 0, the field indicates that the source port number is not used.

The Destination Port field identifies the process at the destination IP address. This process is to receive the UDP data being sent. Like TCP, UDP performs demultiplexing of data to the destination process by using port number values. If UDP receives a datagram with a port number that is not in use (no UDP application associated with the port), UDP generates an ICMP port unreachable error message and rejects the datagram.

The Length field is the length of this UDP packet in octets. This length includes the UDP header and its data. The minimum value of the Length field is 8 and indicates a zero-size data field.

The checksum field is the 16-bit 1's complement of the 1's complement sum of a pseudoheader of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets. The checksum procedure is the same as is used in TCP.

The pseudoheader, which is prefixed conceptually to the UDP header, contains the source address, the destination address, the protocol (UDP protocol number is 17), and

the UDP length (see Figure 11.37). This information provides protection against misrouted datagrams. The pseudoheader has a fixed size of 12 octets. The information in the UDP pseudoheader completely specifies the half association of the destination, even though no connection is being set up.

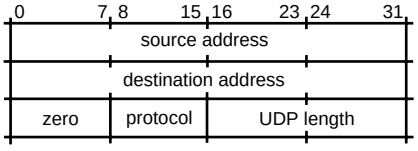



**FIGURE 11.37** *UDP pseudoheader used in checksum calculations.*

If the computed checksum is 0 (zero), it is transmitted as all 1s (the equivalent in 1 in complement arithmetic). Therefore, an all-0 checksum will never be generated by the preceding calculation, and it is used to indicate a special condition: An all 0 transmitted checksum value means that checksums are not being used. Another way of looking at this situation is that 0 has two representations if you use 1's complement arithmetic: all 0s, and all 1s. All 1s is used for calculated checksum, and all 0s indicates that checksumming is disabled. An application running on a reliable IP network, such as a LAN, may want to disable the extra overhead of generating checksums on transmission and verifying them on receiving the UDP datagram.

The UDP protocol runs on top of IP. The IP checksum is done only over the header, whereas the UDP checksum is done over the UDP header and data. The UDP checksum is needed for guaranteeing the integrity of the data field.

## UDP Layering and Encapsulation

UDP runs on top of IP, as you can see in Figure 11.38. Applications that run on top of UDP are associated with a UDP port number. The UDP port number is in a different address space than the TCP port number address space. Thus it is possible to have a TCP port number of 1017 and a UDP port number of 1017 to refer to separate application processes.

Figure 11.39 shows how UDP data is encapsulated in an IP header, which in turn is encapsulated by the Data Link layer. The protocol number assigned to UDP is 17.

There is a strong degree of coupling between UDP and IP because of the necessity of the UDP layer to know the IP address for the pseudoheader. Recall that the pseudoheader is used as part of the UDP checksum. The implication here is that the UDP module must be able to determine the source and destination IP addresses and the Protocol field from the IP header. One possible UDP/IP interface would return the whole IP datagram, including all of the IP header, in response to a RECEIVE operation issued by a UDP application. This interface then would enable UDP to pass a full IP datagram, complete with header,

to the IP to send. The IP Layer then would verify certain fields for consistency and would compute the IP header checksum.

**FIGURE 11.38**
*UDP protocol layering.*

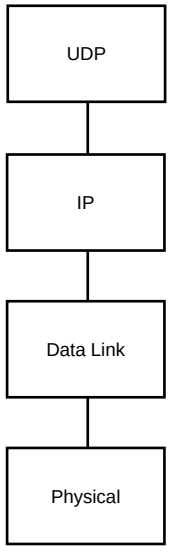
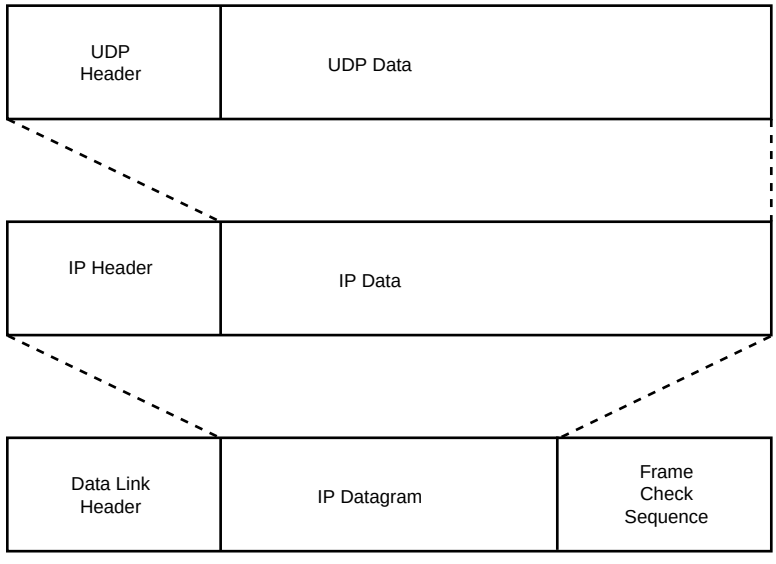



**FIGURE 11.39**
*UDP encapsulation.*



# UDP Trace

Figures 11.40 and 11.41 show two UDP packets: one generated by the Unix WHOD (Who daemon) and another generated by the RIP protocol.

The WHOD UDP source and destination port numbers are 513. The RIP UDP source and destination port numbers are 520. Both UDP protocol decodes show a nonzero checksum, which means that checksums are in use.

**FIGURE 11.40**
*UDP packet for the WHOD process.*

```
Packet Number : 2           10:57:22 PM
Length : 130 bytes
ether: ===================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-BB-E8-73 ----> FF-FF-FF-FF-FF-FF
       Type: 0x0800 (IP)
  ip: ======================= Internet Protocol ========================
       Station:199.245.180.15 ---->199.245.180.255
       Protocol: UDP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 112
       Identification:   296
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x805B(Valid)
 udp: ===================== User Datagram Protocol ====================
       Source Port: WHO
       Destination Port: WHO
       Length = 92
       Checksum: 0x086A(Valid)
Data:
   0: 01 01 00 00 31 EE A4 F5 00 00 00 00 67 61 74 65 |....1.......gate
  10: 6B 65 65 70 65 72 00 00 00 00 00 00 00 00 00 00 |keeper..........
  20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
  30: 00 00 00 00 00 00 00 00 31 EE 7F 85 63 6F 6E 73 |........1...cons
  40: 6F 6C 65 00 72 6F 6F 74 00 00 00 00 31 EE 80 10 |ole.root....1...
  50: 00 00 24 36                                     |..$6
```

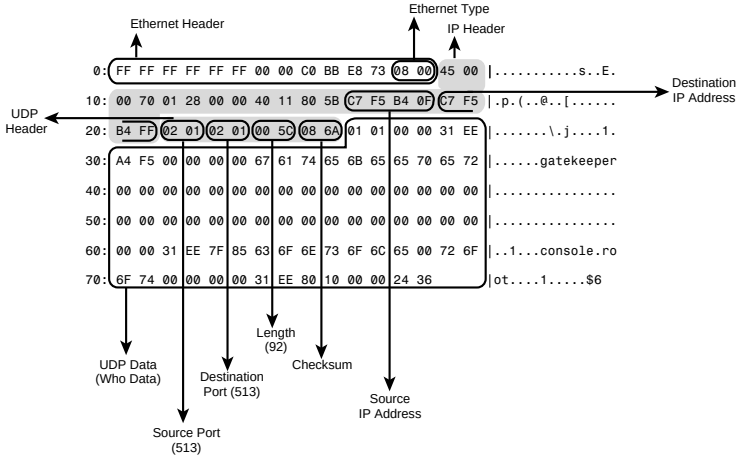
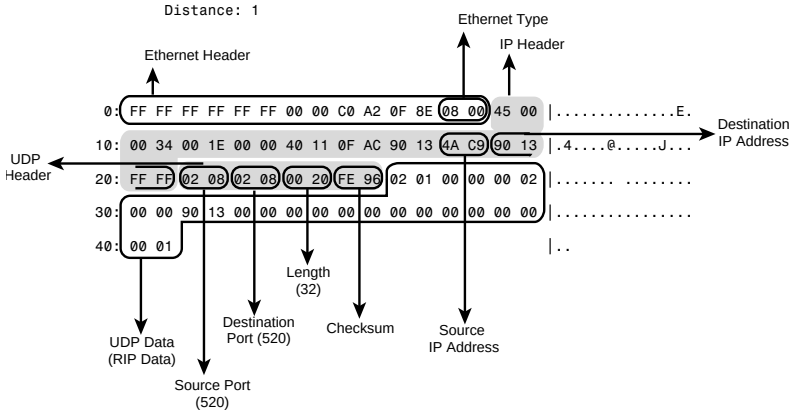


**FIGURE 11.41**
*UDP packet for the RIP process.*

```
Packet Number : 4          10:57:57 PM
Length : 70 bytes
ether: ==================== Ethernet Datalink Layer ====================
       Station: 00-00-C0-A2-0F-8E ----> FF-FF-FF-FF-FF-FF
       Type: 0x0800 (IP)
  ip: ======================= Internet Protocol =======================
       Station:144.19.74.201 ---->144.19.255.255
       Protocol: UDP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
             Normal Delay, Normal Throughput, Normal Reliability
       Total length: 52
       Identification:    30
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 64  seconds
       Checksum: 0x0FAC(Valid)
 udp: ===================== User Datagram Protocol ====================
       Source Port: ROUTER
       Destination Port: ROUTER
       Length = 32
       Checksum: 0xFE96(Valid)
 rip: ================== Routing Information Protocol =================
       Command: Response
       Version: 1
       Family ID: IP
          IP Address: 144.19.0.0
          Distance: 1
```




Also, note that the IP Protocol field is set to 17 to indicate that UDP is being encapsulated.

# Summary

TCP and UDP are standard Transport Layer protocols that are implemented for IP networks. All nonrouter devices that implement the TCP/IP protocol suite are used to implement TCP. Pure router devices do not implement TCP or UDP. However, most commercial routers provide network management and remote logon capability. These services require the use of the Transport protocols TCP and UDP. In many TCP implementations, UDP is implemented as part of the TCP module, even though UDP is separate from TCP conceptually.

By using TCP and UDP, you can deliver data not just to a remote computer, but to an application process running on the remote computer. These application processes are identified by port numbers. TCP can ensure that data is delivered reliably to the destination by providing a connection-oriented service. UDP, on the other hand, is connectionless and cannot guarantee delivery of data. However, UDP is useful in many applications such as those in which data needs to be sent to a particular application running on a machine, or in situations in which application data needs to be broadcasted or multicasted.

CHAPTER 12

# IP Version 6

*by Tim Parker*

## In This Chapter

When IP version 4 (the current release) was developed, the use of a 32-bit IP address seemed more than adequate to handle projected use of the Internet. With the accelerated growth rate of the Internet, though, that 32-bit IP address has proven to be a problem. To counter this limit, IP Next Generation, also known as IP version 6 (IPv6), is under development.

Before the finalization of the IP version 6 (IPv6) protocol several proposals were studied. The most popular of these proposals were TUBA (TCP and UDP with Bigger Addresses), CATNIP (Common Architecture for the Internet), and SIPP (Simple Internet Protocol Plus). None of these three meet all the proposed changes for version 6, but a compromise or modification based on several of these proposals was made.

What does IP Next Generation have to offer? This list of changes tells you the main features of IPv6 in a nutshell:

- 128-bit network address instead of 32-bit
- More efficient IP header with extensions for applications and options
- Optional fields added as optional headers after a fixed base header
- A flow label for quality-of-service requirements
- Prevention of intermediate fragmentation of datagrams
- Better support for mobile stations
- Automatic assignment of IPv6 addresses based on MAC addresses
- Built-in security for authentication and encryption

The next sections look at IPv6 in a little more detail to highlight the changes, as well as network programmers and network administrators. In the following section, the IPv6 header format is examined.

# IPv6 Datagram

As already mentioned, the header for IP datagrams with version 6 has been modified. The changes are mostly to provide support for the new, longer 128-bit IP addresses and to move optional fields to extension headers. The basic layout of the IPv6 header is shown in Figure 12.1. For comparison, the older IP version 4 header layout is shown in Figure 12.2.

The version number in the IP datagram header is four bits long and holds the release number, which is 6 with IPv6. The Priority field is four bits long and holds a value indicating the datagram's priority. The priority, which is used to define the transmission order, is set first with a broad classification, then with a narrower identifier within each class. (See the upcoming "Priority Classification" section for more information.)

**FIGURE 12.1**
*The IPv6 header layout.*

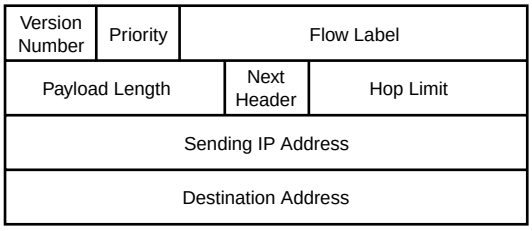
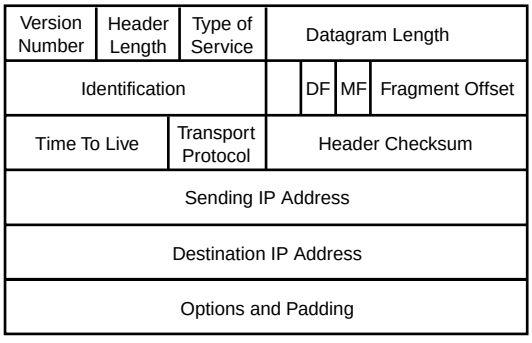



**FIGURE 12.2**
*The IPv4 header layout.*






The Flow Label field is 24 bits long. It is combined with the source machine IP address to provide flow identification for the network. For example, if you are using a Unix workstation on the network, the flow will be different from that of another machine such as a Windows machine. This field can be used to identify flow characteristics and provide some adjustment capabilities. The field can also be used to help identify target machines for large transfers, in which case a cache system becomes more efficient at routing between source and destination. Flow labels are discussed in more detail later in this chapter in the section "Flow Labels."

The Payload Length field is a 16-bit field used to specify the total length of the IP datagram, given in bytes. The total length is exclusive of the IP header itself. The use of a 16-bit field limits the maximum value in this field to 65,535, but there is a provision to send large datagrams using an extension header. (See "IP Extension Headers," later in this chapter.)

The Next Header field is used to indicate which header follows the IP header when other applications want to piggyback on the IP header. Several values have been defined for the Next Header field, as shown in Table 12.1. The Next Header is an 8-bit field that identifies the type of header immediately following the IPv6 basic header. The header that immediately follows can be the protocol type value identifying a transport protocol such as TCP (6) or UDP (17), or it can be a value identifying an IPv6 extension header. IPv6 extension headers are used to specify IPv6 options and other information.

Figure 12.3 shows the use of the Next Header field to specify a TCP message that follows immediately, and extension headers. A Next Header value of 59 indicates no next header; that is, no encapsulation of any other protocol or data.

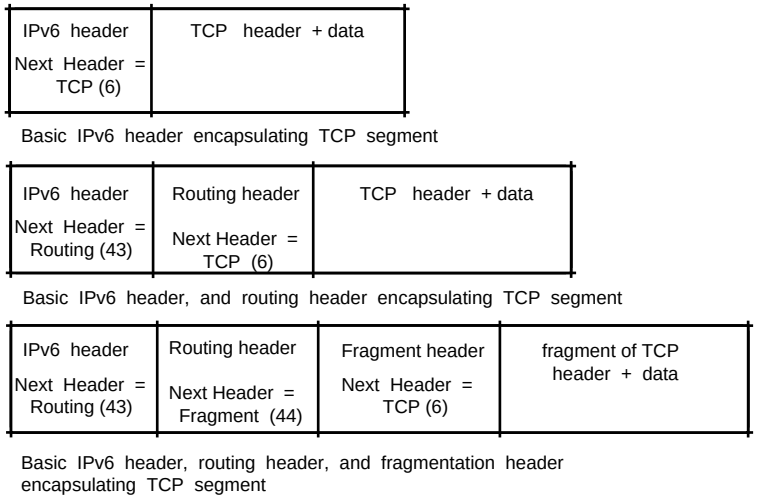



**FIGURE 12.3** *Use of Next Header field.*

**TABLE 12.1** IP Next Header Field Values

| *Value* | *Description* |
|---|---|
| 0 | Hop-by-hop options |
| 4 | IP |
| 6 | TCP |
| 17 | UDP |
| 43 | Routing |
| 44 | Fragment |
| 45 | Interdomain Routine |
| 46 | Resource Reservation |
| 50 | Encapsulating Security |
| 51 | Authentication |
| 58 | ICMP |
| 59 | No next header |
| 60 | Destination options |

The Hop Limit field determines the number of hops that the datagram will travel. With each forwarding, the number is decremented by one. When the Hop Limit field reaches zero, the datagram is discarded. The Hop Limit field prevents old datagrams from circulating around the network endlessly, which could occur if a routing loop is created. The initial value of the Hop Limit is set by the sender. The Hop Limit is similar to the TTL field in the IPv4 header. The difference is that in IPv4, the TTL field is measured in seconds and in IPv6, the Hop Limit is measured in hops. Actually, even in IPv4, the TTL for all practical considerations has the property of a hop count because routers decrease the TTL value by 1 instead of the time it takes to process the datagram. In reality, protocols such as TCP use large sequence number fields (see Chapter 11, "The Transport Protocols") to guard against the resurgence of old packets.

Finally, the sender and destination IP addresses in 128-bit format are placed in the header. The new IP address format is discussed in more detail later in this chapter in the section "128-Bit IP Addresses."



## Priority Classification

The Priority Classification field in the IPv6 header first divides the datagram into one of two categories: congestion-controlled and non–congestion-controlled. Non–congestion-controlled datagrams are always routed as a priority over congestion-controlled datagrams. Subclassifications of non–congestion-controlled datagram priorities (priority 8 through 15) are available for use.

If the datagram is congestion-controlled, it is sensitive to congestion problems on the network. If congestion occurs, the datagram can be slowed down and held temporarily in caches until the problem is alleviated. Within the broad congestion-controlled category are several subclasses that further refine the priority of the datagram. The subcategories of congestion-controlled priorities are listed in Table 12.2.

**TABLE 12.2** Priorities for Congestion-Controlled Datagrams

| *Priority* | *Description* |
|---|---|
| 0 | No priority specified |
| 1 | Background traffic |
| 2 | Unattended data transfer |
| 3 | Unassigned |
| 4 | Attended bulk transfer |
| 5 | Unassigned |
| 6 | Interactive traffic |
| 7 | Control traffic |

Non–congestion-controlled traffic has priorities 8 through 15. Priority values 8 to 15 are used to specify the priority of traffic that does not back off in response to congestion, such as real-time packets being sent at a constant rate.

Examples of each of the primary subcategories may help you see how the datagrams are prioritized. Routing and network management traffic that is considered highest priority is assigned category 7. Interactive applications such as Telnet and Remote X sessions are assigned as interactive traffic (category 6). Transfers that are not time-critical, such as Telnet sessions, but are still controlled by an interactive application like FTP, are assigned as category 4. Email is usually assigned as category 2, whereas low-priority material such as news is set to category 1.

## Flow Labels

As mentioned earlier, the Flow Label field new to the IPv6 header can be used to help identify the sender and destination of a number of IP datagrams. By employing caches to handle flows, the datagrams can be routed more efficiently. Not all applications will be able to handle flow labels, in which case the field is set to a value of zero.

A simple example may help show the usefulness of the Flow Label field. Suppose a PC running Windows is connected to a Unix server on another network and sending a large number of datagrams. By setting a specific value of the flow label for all the datagrams in the transmission, the routers along the way to the server can assemble an entry in their routing caches that indicates which way to route each datagram with the same flow label. When subsequent datagrams with the same flow label arrive, the router doesn't have to recalculate the route; it can simply check the cache and extract the saved routing information. This speeds up the passage of the datagrams through each router.

To prevent caches from growing too large or holding stale information, IPv6 stipulates that the cache maintained in a routing device cannot be kept for more than six seconds. If a new datagram with the same flow label is not received within six seconds, the cache entry is removed. To prevent repeated values from the sending machine, the sender must wait six seconds before using the same flow label value for another destination.

IPv6 allows flow labels to be used to reserve a route for time-critical applications. For example, a real-time application that has to send a number of datagrams along the same route and needs as rapid a transmission as possible (such as is needed for video or audio) can establish the route by sending datagrams ahead of time, being careful not to exceed the six-second timeout on the interim routers.

## 128-Bit IP Addresses

Probably the most important aspect of IPv6 is the ability to provide for longer IP addresses. Version 6 increases the IP address from 32 bits to 128 bits. This will enable an exponential increase in the number of addresses to be assembled.

The new IP addresses support three kinds of addresses: unicast, multicast, and anycast. These are described here:

- *Unicast addresses* are meant to identify a particular machine's interface. This will make it possible for a PC, for example, to have several different protocols in use, each with its own address. Thus, you could send messages specifically to a machine's IP interface address and not the NetBEUI interface address.
- A *multicast address* identifies a group of interfaces, enabling all machines in a group to receive the same packet. This will be similar to multicasts in version 4 IP. Multicast addresses provide more flexibility for defining groups. Your machine's interfaces could belong to several multicast groups.
- An *anycast address* will identify a group of interfaces on a single multicast address. In other words, more than one interface can receive the datagram on the same machine.

We will look at the three types of addresses in a little more detail later in this chapter in the section "Unicast, Multicast, and Anycast Headers."

The IP header also changes considerably with version 6, providing lots more information and flexibility. In addition, the handling of fragmentation and reassembly is changed, to provide more capabilities for IP. Also proposed for IPv6 is an authentication and encryption scheme that can ensure that the data has not been corrupted between sender and receiver, as well as that the sending and receiving machines are who they claim they are.

**Note**

The authentication and encryption scheme is part of the IPSec (IP Security) proposal.

# IP Extension Headers

IPv6 has the provision to enable additional optional headers to be added to the IP header. This may be necessary when a simple routing to the destination is not possible, or when special services, such as authentication, are required for the datagram. The additional information required is packaged into an extension header and appended to the IP header.

IPv6 defines several types of extension headers identified by a number that is placed in the Next Header field of the IP header. The currently accepted values and their meanings were shown in Table 12.1. Several extensions can be appended onto one IP header, with each extension's Next Header field indicating the next extension. Extension headers are appended in ascending order, as show in the following:

1. IPv6 header
2. Hop-by-Hop Options header
3. Destination Options header
4. Routing header
5. Fragment header
6. Authentication header
7. Encapsulating Security Payload header
8. Destination Options header
9. Upper-Layer header

The ordering of the extension headers makes it easier for routers to analyze the extensions, stopping the examination when it gets past router-specific extensions.

## Hop-by-Hop Headers

Extension type 0 is hop-by-hop, which is used to provide IP options to every machine the datagram passes through. The options included in the hop-by-hop extension have a standard format of a type value, a length, and a value (except for the Pad1 option, which has a single value set to zero and no Length or Value field). Both the Type and Length fields are a single byte in length, whereas the Value field's length is variable and indicated by the length byte.

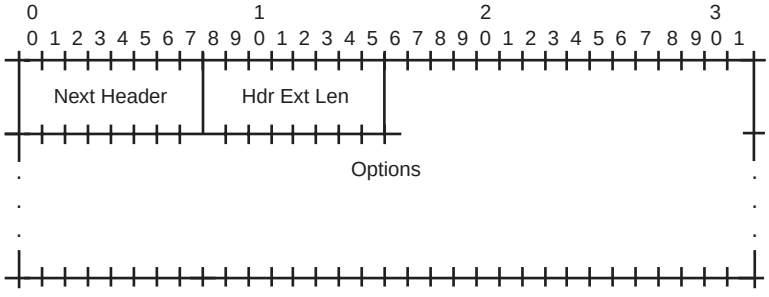
The three types of hop-by-hop extensions defined so far are Pad1, PadN, and Jumbo Payload. The Pad1 option is a single byte with a value of zero, no length, and no value. It is used to alter the order and position of other options in the header when necessary, dictated usually by an application. The PadN option is similar except there are N zeros placed in the value field, and a calculated value for the length.

The Jumbo Payload extension option is used to handle datagram sizes in excess of 65,535 bytes. The Length field in the IP header is limited to 16 bits, hence the limit of 65,535 for the datagram size. To handle larger datagram lengths, the IP header's Length field is set to zero, which redirects the routers to the extension to pick up a correct length value. The Length field can be defined in the extension header using 32 bits, which yields datagrams in excess of 4 billion bytes!

The Hop-by-Hop Options header is identified by a Next Header value of 0 in the IPv6 header, and has the format shown in Figure 12.4.



**FIGURE 12.4**
*IPv6 Hop-by-Hop Options header.*

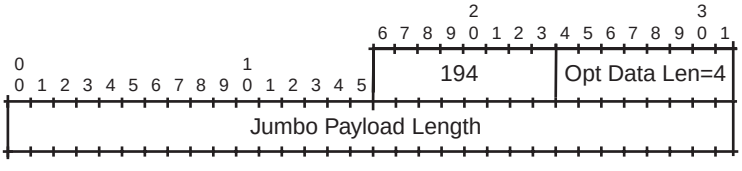



The fields in the Hop-by-Hop Options header have the following meaning:

- Next Header. An 8–bit selector that identifies the type of header immediately following the routing header.
- Hdr Ext Len. An 8–bit unsigned integer. This is the length of the routing header in 8–octet units, excluding the first 8 octets.
- Options. This is a variable–length field, such that the complete Destination Options header length is an integer multiple of 8 octets. The option values are encodes using the Type-Length-Value (TLV) format.

In addition to the Pad1 and PadN options, the Jumbo Payload option is defined for use with the hop-by-hop option (see Figure 12.5).

**FIGURE 12.5**
*IPv6 Jumbo Payload option.*



The alignment requirement for the Jumbo Payload option is *4n + 2*. This means that the start of the option must be a multiple of 4 octets plus 2 from the start of the header.

The Jumbo Payload option is used to send IPv6 packets with payloads longer than 65,535 octets. The Jumbo Payload Length field is 32 bits wide and represents the length of the packet in octets, excluding the IPv6 header but including the Hop-by-Hop Options header; it must be greater than 65,535. If a packet is received with a Jumbo Payload option containing a Jumbo Payload Length less than or equal to 65,535, an ICMP Parameter Problem message, Code 0, is sent to the packet's source, pointing to the high-order octet of the invalid Jumbo Payload Length field.

In addition, the Payload Length field in the IPv6 header must be set to zero in every packet that carries the Jumbo Payload option. If a packet is received with a valid Jumbo Payload option present and a non-zero IPv6 Payload Length field, an ICMP Parameter Problem message, Code 0, is sent to the packet's source, pointing to the Option Type field of the Jumbo Payload option.

## Routing Headers

A routing extension can be appended onto the IP header when the sending machine wants to control the routing of the datagram instead of leaving it to the routers along the path. This type of routing is called *source routing*. The routing extension, which includes fields for each IP address along the desired route, can be used to give routes to the destination.

The IPv6 routing extension header (see Figure 12.6) is used to specify one or more intermediate routers that an IPv6 datagram must visit on its way to the destination. This header is used to implement the idea of the source routing option that is also specified in IPv4. The IPv6 routing extension header is identified by the Next Header value of 43 in the IPv6 basic header.

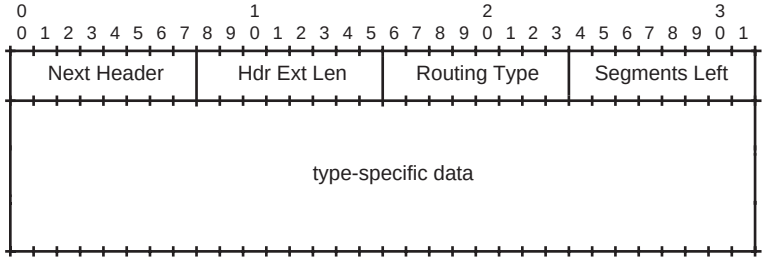



**FIGURE 12.6**
*IPv6 routing extension header.*

The fields in the IPv6 routing extension header have the following meaning:

- Next Header. This is an 8-bit selector that identifies the type of header that immediately follows the routing header. It can be another IPv6 extension header as part of a chain of IPv6 extension headers, or an upper-layer transport protocol.
- Hdr Ext Len. This is an 8-bit unsigned integer that indicates the length of the routing header in 8-octet units, excluding the first 8 octets.
- Routing Type. This field is an 8-bit identifier of a particular routing header variant. Currently a routing type of 0 is defined in the IPv6 specification (RFC 1883). Other values can be used to identify alternative routing header formats.
- Segments Left. This field is an 8-bit unsigned integer and specifies the number of route segments remaining. The number of remaining route segments is the number of explicitly listed intermediate nodes that still need to be visited before reaching the final destination.
- Type-specific data. This is a variable-length field whose format is determined by the Routing Type. The length of the field is such that the complete routing header length is an integer multiple of 8 octets.



If a node encounters a routing header with an unrecognized Routing Type value, it processes the packet based on the value of the Segments Left field. If Segments Left is zero, the node ignores the routing header and proceeds to process the next header in the packet. The next header is identified by the Next Header field in the routing header. If Segments Left is non-zero, the node must discard the packet and send an ICMP Parameter Problem, Code 0, message to the packet's destination indicated by the Source Address field, and pointing to the unrecognized Routing Type.

Figure 12.7 shows a routing header for Routing Type 0. Note that the first 32 bits of this header are the same as shown in Figure 12.6. The Type-specific data field contains a list of router addresses that allow the source to override the route that might have been otherwise followed by the IPv6 datagram as determined by the routing protocol running at the router.

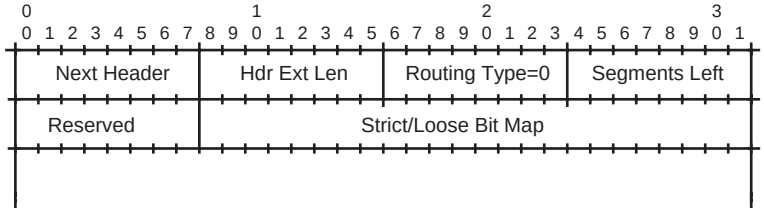



**FIGURE 12.7**
*Routing Type 0, IPv6 routing header.*

The fields in the IPv6 type 0 routing extension header have the following meaning:

- Next Header. An 8-bit selector that identifies the type of header immediately following the routing header.
- Hdr Ext Len. An 8-bit unsigned integer. This is the length of the routing header in 8-octet units, excluding the first 8 octets. For the Type 0 routing header, the Hdr Ext Len is equal to two times the number of addresses in the header, and must be an even number less than or equal to 46.
- Segments Left. An 8-bit unsigned integer. This is the number of route segments remaining. It's maximum legal value is 23 for Route Type 0.
- Reserved. An 8-bit reserved field that is initialized to zero for transmission and ignored on reception.
- Strict/Loose Bit Map. A 24-bit bitmap, numbered 0 to 23, left-to-right. Each bit corresponds to a segment of the route, and indicates whether or not the next destination address must be a neighbor of the preceding address. A bit value of 1 means strict source routing; that is, the next destination address must be a neighbor. A bit value of 0 means loose source routing; that is, the next destination address need not be a neighbor.
- Address[1..n]. This is a vector of 128-bit addresses, numbered 1 to n.

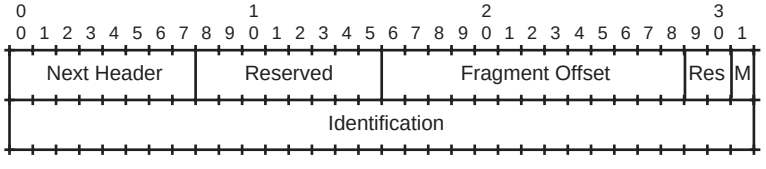
## Fragment Headers

The fragment header can be appended to an IP datagram to allow a machine to fragment a large datagram into smaller parts. Part of the design of IPv6 was to prevent subsequent fragmentation, but in some cases fragmentation must be allowed in order to pass the datagram along the network.

Unlike IPv4, fragmentation in IPv6 is performed only by source nodes, and not by routers along a packet's delivery path. An IPv6 source node knows the path MTU size to the destination. IPv6 nodes implement Path MTU Discovery as defined in RFC 1191 on "Path MTU Discovery" to discover the path MTU to the destination.

An IPv6 source will insert the fragment header if it has to send packets that are larger than the path MTU to the destination. The fragment header is identified by a Next Header value of 44 in the immediately preceding header. Figure 12.8 shows the format of the fragment header.

**FIGURE 12.8**
*IPv6 fragment extension header.*



The fields in the fragmentation header have the following meanings:

- Next Header. This is an 8-bit selector that identifies the type of header that immediately follows the routing header. It can be another IPv6 extension header as part of a chain of IPv6 extension headers, or an upper-layer transport protocol.
- Reserved. An 8-bit reserved field that is initialized to zero for transmission and ignored on reception.
- Fragment Offset. A 13-bit unsigned integer that indicates the offset, in 8-octet units, of the data following this header. Offsets are measured relative to the start of the Fragmentable Part of the original packet.
- Res. A 2-bit reserved field that is initialized to zero for transmission and ignored on reception.
- M flag. This field is set to 1 for more fragments, and 0 for the last fragment.
- Identification. This is a 32-bit field. For every packet that is to be fragmented, the source node generates an Identification value. The Identification must be different than that of any other fragmented packet sent recently with the same Source Address and Destination Address.



## Authentication Headers

The authentication header is used to ensure that no alteration was made to the contents of the datagram, and that the datagram originated at the machine shown in the IP header. By default, IPv6 uses an authentication scheme called Message Digest 5 (MD5). Other authentication schemes can be used as long as both ends of the connection agree on the same scheme.

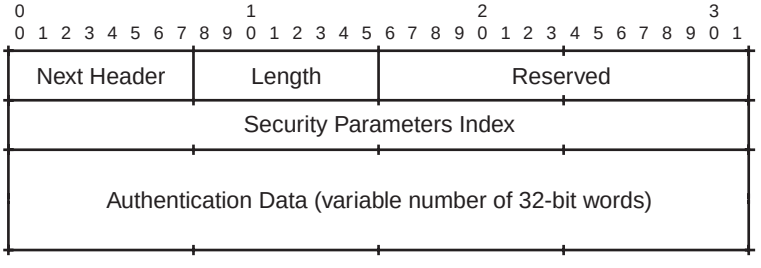
The authentication header consists of a *security parameters index* (SPI), which, when combined with the destination IP address, defines the authentication scheme. The SPI is followed by authentication data, which with MD5 is 16 bytes long. MD5 starts with a key (padded to 128 bits if it is shorter), then appends the entire datagram. The key is then tagged at the end, and the MD5 algorithm is run on the whole. To prevent problems with hop counters and the authentication header itself altering the values, they are zeroed for the purposes of calculating the authentication value. The MD5 algorithm generates a 128-bit value that is placed in the authentication header. The steps are repeated on the receiving end. Both ends must have the same key value, of course, for the scheme to work.

The datagram contents can be encrypted prior to generating authentication values using the default IPv6 encryption scheme, called *Cipher Block Chaining* (CBC), part of the *Data Encryption Standard* (DES).

The fields of the authentication header (see Figure 12.9) have the following meaning:

- Next Header. An 8-bit selector that identifies the type of header immediately following the routing header.
- Length. This field is 8 bits wide and specifies the length of the Authentication Data field in 32-bit words. It's minimum value is 0 words, which is used only in the special case when no authentication algorithm is used.
- Reserved. This field is 16 bits wide and is reserved for future use. It must be set to all zeros by the sender. The value is included in the Authentication Data calculation, but is otherwise ignored by the recipient.
- Security Parameters Index (SPI). This is a 32-bit pseudo-random value identifying the security association for this datagram. The Security Parameters Index value 0 is reserved to indicate that no security association exists. The set of Security Parameters Index values in the range 1 through 255 are reserved to the Internet Assigned Numbers Authority (IANA) for future use. A reserved SPI value will not normally be assigned by IANA unless the use of that particular assigned SPI value is openly specified in an RFC.
- Authentication Data. This is a variable-length field, but is always an integral number of 32-bit words and contains the calculated authentication data for this packet.

**FIGURE 12.9**
*IPv6 authentication header.*




## IPv6 Encrypted Security Payload Header

The IPv6 Encrypted Security Payload (ESP) header is used for providing integrity and confidentiality of IP datagrams. It may be used to provide authentication, depending on which is used. Non-repudiation and protection from traffic analysis are not provided by ESP. The IPv6 authentication header can provide non-repudiation if used with certain authentication algorithms.

The IP authentication header can be used in conjunction with ESP to provide authentication. An application requiring integrity and authentication without confidentiality should use the authentication header instead of ESP.

ESP encrypts data to be protected and places the encrypted data in the data portion of the Encapsulating Security Payload. This mechanism may be used to encrypt either a transport-layer segment or an entire IP datagram.

ESP has two modes of operation:

- Tunnel-mode ESP
- Transport-mode ESP

In Tunnel-mode ESP, the original IP datagram is placed in the encrypted portion of the Encapsulating Security Payload and that entire ESP frame is placed within a datagram having unencrypted IP headers. The information in the unencrypted IP headers is used to route the secure datagram from origin to destination. An unencrypted IP routing header might be included between the IP header and the Encapsulating Security Payload.



In Transport-mode ESP, the ESP header is inserted into the IP datagram immediately prior to the transport-layer protocol header. In this mode bandwidth is conserved because there are no encrypted IP headers or IP options.

ESP may appear anywhere after the IP header and before the final transport-layer protocol. The header immediately preceding an ESP header must contain the value 50 in its Next Header field. ESP consists of an unencrypted header followed by encrypted data. The encrypted data includes both the protected ESP header fields and the protected user data, which is either an entire IP datagram or a transport-layer protocol. A high-level diagram of a secure IP datagram is shown in Figure 12.10.

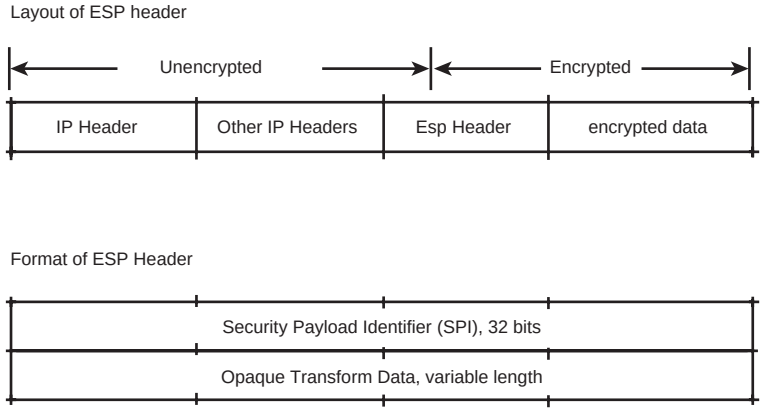



**FIGURE 12.10**
*IPv6 Encrypted Security Payload header.*

Encryption and authentication algorithms, and the precise format of the Opaque Transform Data associated with them are known as *transforms*. The ESP format is designed to support new transforms in the future to support new or additional cryptographic algorithms.

The SPI field (refer to Figure 12.10) is a 32-bit pseudo-random value identifying the security association for this datagram. If no security association has been established, the value of the SPI field is set to 0. An SPI is similar to the Security Association Identifier (SAID) used in other security protocols. SPI values from 0 through 255 are reserved by the Internet Assigned Numbers Authority (IANA) for future use. The SPI is the only mandatory transform-independent field.

# Multiple IP Addresses per Host

Apart from the potential problem of running out of IP addresses, why are we bothering to develop a radically different IP structure? As you can probably determine from the makeup of the IP headers, a lot of features are embedded in this new version. One of the more important ideas with IPv6 has to be the potential for more than one IP address per host machine.

With today's TCP/IP systems, almost every host has a single IP address. The exceptions tend to be machines that act as gateways, routers, firewalls, proxy servers, and Network Address Translators (NATs). Single-homed machines allow a number of advantages: By counting the number of IP addresses on a network, we know the number of machines there are. With single IP addresses per host, configuring networks is easier than it would be with multiple hosts. For users, remembering a single IP address for FTP and Telnet purposes is easier than remembering a bunch of different addresses. DNS and other services require a single IP address for mapping purposes, and these services will have to be changed with IPv6. But we are moving to a multiple-address per host model with IPv6 for a number of good reasons.

One of the most useful advantages for multiple addresses is on multiuser machines. If you share a workstation with four other users (who may be on diskless workstations or other devices), for example, having a separate IP address for each user would be handy. This would make connecting to their filesystem easier, as well as providing better tracking and billing. Having a single IP address per user allows some of the encryption technologies that are enabled by IPv6 to be used with a different encryption key for each user, enhancing security.

Encryption is an important advantage of multiple host addresses, as well. Consider that most servers today can be reached with a variation of their domain name (`ftp.microsoft.com` for FTP daemons; `http://www.miscrosoft.com` for the World Wide Web, for example), and all these services are run from a single host with the same

security built into IP. With IPv6, you could have a different IP address for each service (although the names could map to the same alphabetical names for convenience) but could involve different encryption or authentication methods based on the IP address. For example, one address could lead to `http://www.microsoft.com` and involve little encryption and authentication, but another address that maps to `ftp.microsoft.com` could require strict authentication to ensure only valid systems are allowed to transfer files. Although this type of handling based on the service is possible with IP today, IPv6 adds a lot more capability. On the downside, there will need to be many more addresses mapped to names.



TCP port numbers can be added more easily with IPv6 than with the current IP. For example, if you want to reach TCP port 14 on a server, you need to address the system with the IP address and port number (such as 205.150.89.1:14). As a user, you need to know the port number to specify it. The TCP port could easily be resolved with multiple addresses in IPv6. One address could lead to one FTP port with wider protection than a second address leading to another FTP port, for example.

Multiple addresses work well when subnetworks are merged. If your company has two LANs, one for research and development and another for management, and you happen to manage the R&D group, your machine needs an IP address on both networks. Your machine may act as a type of router in some cases, and you often have to specify which LAN you want to work with. With IPv6, IP addresses can be assigned so that you can send information on both LANs at once, or can move data between the two without much hassle (essentially eliminating a router).

# Unicast, Multicast, and Anycast Headers

Unicast addresses are supported in a number of variations, including a global unicast address that goes to all providers, a site-specific unicast address for a particular network, and a version of unicast that is compatible with IPv4 machines. The specifications for IPv6 allow for other types of unicast addresses in the future, as they are needed.

The global unicast address is used for connecting to every provider on the Internet. The format of this global, or provider-based, unicast address is shown in Figure 12.11.

In Figure 12.11, the first three bits are set to 010 and identify the unicast as a provider-based type. The REG ID (Registry ID) field is the Internet address registry that assigns the Provider ID (PROV ID), which subsequently grants IP addresses to customers

(PROV ID is the ISP in most cases). SUBSC ID allows multiple subscribers (customers) to be identified in the provider's network, and SUBNET ID allows a specific address to be used. Finally, INTF ID is the interface ID, which can be used to identify a particular subscriber interface.

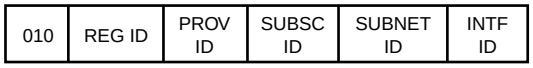



**FIGURE 12.11**
*The provider-based unicast header layout.*

A site-specific, or local-user, unicast address is used only within a network or subnetwork, so it needs less information. The header for a local-user unicast is shown in Figure 12.12. The INTF ID is an interface ID on the network or subnetwork. A slight variation of the local-user unicast address adds a Subnet ID to the header, removing space from the INT ID field. When used, the Subnet ID field can be used to specify a particular subnet on a network.

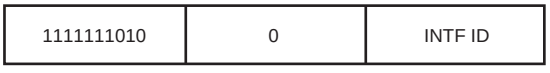



**FIGURE 12.12**
*The local-user unicast header layout.*

Finally, the unicast address with embedded IPv4 addresses is shown in Figure 12.13. This has the IPv4 address appended to the end of the unicast header and can be used by the older version of IP.

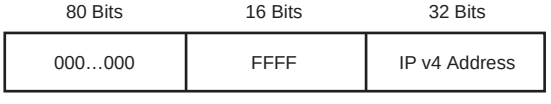



**FIGURE 12.13**
*The embedded IPv4 unicast header layout.*

Anycast addresses use the same layout as unicast headers, and in most cases cannot be distinguished from unicast broadcasts.

Multicast headers have the layout shown in Figure 12.14.

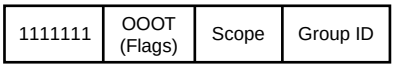



**FIGURE 12.14**
*The multicast header layout.*

The leading eight bits of 1s identify the header as a multicast. The Flags field has four bits that lead with three zeros (reserved bits for future use) and a trailing bit that can be either a zero for a permanently assigned multicast address or a one for a non-permanent multicast address. The Scope field is four bits that can be used to limit the multicast reach. Legal values for the Scope field are shown in Table 12.3. Finally, the Group ID identifies a particular multicast group.

**TABLE 12.3** Legal Values for the Multicast Scope Field

| *Value* | *Description* |
|---|---|
| 0 | Reserved |
| 1 | Node-local |
| 2 | Link-local |
| 3,4 | Unassigned |
| 5 | Site-local |
| 6,7 | Unassigned |
| 8 | Organization-local |
| 9,A,B,C,D | Unassigned |
| E | Global |
| F | Reserved |



# Transition from IPv4 to IPv6

Although IPv6 is technically superior to the current version of IP (IPv4), potential problems exist with implementing IPv6 worldwide. Simply switching from the old IP version to the newer on a single date is impossible, so the rollout must be managed in such a manner as to provide compatibility for both versions for a while. Because the establishment of IPv6 worldwide is expected to take several years, transition becomes a very important issue.

One problem with changing from IPv4 to IPv6 is the simple fact that IPv4 is embedded in many layers of the TCP/IP suite and within many applications as well. To switch over to IPv6, every application, driver, and TCP stack that uses IP has to be converted. This amounts to hundreds of thousands of changes, involving millions of lines of code. Spread over thousands of vendors, it is unlikely that all will change their code within a specific timespan. Again, this means that IPv4 and IPv6 must coexist for quite a while.

All current machines (hosts computers, routers, bridges, and so on) use IPv4. As machines are converted to run IPv6 (either through a software or hardware update), all these machines will likely need two sets of IP software, one for the old version and one for the new release. In some cases, that's going to be very difficult to implement because of memory or performance problems, so some devices may have to be one version of IP or the other (or be replaced with more powerful devices).

Conversion software must be developed for applications that can't or won't be updated to IPv6. For example, some devices and applications that communicate using IPv4 and yet need to talk to IPv6 systems will need to have a conversion or translation application sitting between the two parts. This adds more overhead to the system and slows down performance, but it may be the only way to handle some legacy software and hardware.

Translation between IPv4 and IPv6 may not seem like a big problem, but it could be. The primary issue is header translation, where even the slightest problem causes loss of data. IPv6 is based on IPv4, but the headers are very different. Any information in an IPv6 header that is not supported in IPv4 (such as priority classification) will be lost when converted. Conversely, a packet leaving an IPv4 host that is converted to IPv6 will have a lot of information missing from the header, some of it potentially important.

Address mapping (converting IPv4 IP addresses to IPv6 IP addresses and vice versa) will require some special handling. If you have a host that has several IP addresses under IPv6 but only one IP address under IPv4, a converter, router, or other forwarding device that converts from one version of IP to another must have a large table of mappings. This will become impractical in large organizations, and when converting from IPv4 to IPv6, may result in incorrect destinations. An IPv4 IP address can be embedded in an IPv6 header, but this may cause routing problems for IPv6-based systems.

Some TCP/IP services are going to need drastic overhauls for the change to IPv6. DNS, for example, holds a mapping of common names to IP addresses. When IPv6 rolls out, DNS is going to have to handle both versions of IP, as well as resolve multiple IP addresses for each host. You may have a PC with 10 IP addresses under IPv6 (different IP addresses for different services on your machine, for example). Broadcasting is a problem with IPv4 at the moment because there are many times when a LAN- or WAN-wide broadcast is sent with IPv4 (for example, ARP uses data-link layer broadcast). IPv6 reduces broadcasting because of the multicast feature, which is designed to permit a multicast to travel on a LAN or WAN only once. Resolving the two broadcast systems during a conversion is again going to be a problem.

There's a lot more involved in converting an entire network infrastructure from IPv4 to IPv6. A lot of technical issues need to be worked on to provide maximum flexibility as companies and networks convert from one version of IP to another. The process is not easy, and is expected to take years, but the end result should be a network based entirely on IPv6 (although it seems likely that many older devices will not be able to upgrade to IPv6, and hence will need a translator of some sort). For most people, the change from IPv4 to IPv6 will be transparent: The network administrators will take care of the conversion for you. For those involved in network management, though, the conversion from IPv4 to IPv6 promises to be an experience.



# Summary

Although IPv6 is going to take a number of years to gain widespread usage, it will eventually happen. Whether you make a change all at once or employ a gradual changeover, eventually you will need to implement IPv6. For most users, especially those connecting through an ISP, the actual time of changeover is quite a long way away. For companies, though, the advantages IPv6 offers will tend to make the changeover faster. Compatibility between IPv6 and IPv4 is built in, so there will be support for the older IP system for quite a while.

# Internetworking with IP

# PART IV

## IN THIS PART

CHAPTER 13

# Routing in IP Networks

*by Mark A. Sportack*

## IN THIS CHAPTER

One of the most critical functions in an interconnected IP network is routing. *Routing* is the process of discovering, comparing, and selecting paths through the network to any destination IP address. Typically, the routing function is embodied in special-purpose devices called *routers*. Technological advances, however, are rapidly blurring the distinctions between traditional routers, LAN switches, and even network-attached hosts. Today, all three types of devices are capable of discovering, comparing, and selecting routes. Thus, routing must be regarded as a function rather than as a physical device.

The essence of routing exists in the form of highly specialized network protocols that enable routers (regardless of which type of physical device that may be) to perform their vital functions. These functions include

- Sharing information about locally connected hosts and networks
- Comparing potentially redundant paths
- Converging upon an agreement of a network's topology

Examining the mechanics of these basic functions in an IP network will provide the context for a more in-depth examination of three of the more commonly encountered dynamic IP routing protocols, which is provided in Chapters 14 through 16.

# The Fundamentals of Routing

Routers can route in two basic ways. They can use preprogrammed *static* routes, or they can dynamically calculate routes using any one of a number of dynamic routing protocols. Dynamic routing protocols are used by routers to discover routes. Routers then forward packets (or datagrams) over those routes. Routers that are statically programmed are incapable of discovering routes; they lack any mechanism for communicating routing information with other routers. Statically programmed routers can only forward packets using routes defined by a network administrator.

In addition to static programming of routes, there are two basic categories of dynamic routing protocols:

- Distance-vector
- Link-state

The primary differences between these types of dynamic routing protocols lie in the way they discover and calculate new routes to destinations.

### Two Functional Classes of Dynamic Routing Protocols

Many ways exist to classify routing protocols, including by operational characteristics such as their field of use, the number of redundant routes to each destination supported, and so on. This book classifies routing protocols by the way they discover and calculate routes. However, referencing them by their field of use is still sometimes useful; that is, categorizing them by the role they perform in an internetwork. For example, there are two functional classes of dynamic routing protocols: Interior Gateway Protocols (IGPs) and Exterior Gateway Protocols (EGPs).

Perhaps the easiest way to explain this is that IGPs are used *within* autonomous systems, such as intranets, whereas EGPs are used *between* autonomous systems. An autonomous system is a collection of networks that is self-contained, usually administered by a single administrator or group of administrators. *Border Gateway Protocol* (BGP—an EGP) is the protocol used to calculate routes across the Internet. The Internet, from a routing perspective, is nothing more than a backbone transport for a global collection of privately owned and operated autonomous systems.

The specific protocols covered in this book are identified as either IGPs or EGPs.



## Static Routing

Static, or preprogrammed, routes are the simplest form of routing. The tasks of discovering routes and propagating them throughout a network are left to the internetwork's administrator(s).

A router that is programmed for static routing forwards packets out of predetermined ports. After the relationship between a destination address and a router port is configured, there is no longer any need for routers to attempt route discovery or even communicate information about routes to that destination. It is possible, however, for a router to use static routes for certain destinations and dynamically route to others.

There are many benefits to using static routes. For instance, statically programmed routes can make for a more secure network: There can be only a single path into, and out of, a network connected with a statically defined route. That is, of course, unless multiple static routes are defined.

Another benefit is that static routing is much more resource-efficient. It uses far less bandwidth across the transmission facilities, doesn't waste any router CPU cycles trying to calculate routes, and requires less memory. In some networks, you might even be able to use smaller, less expensive routers by using static routes. Despite these benefits, there are some inherent limitations to static routing of which you must be aware.

Static routes are sometimes used for small networks whose topology seldom changes. An example of such a network is two sites connected by a point-to-point link.

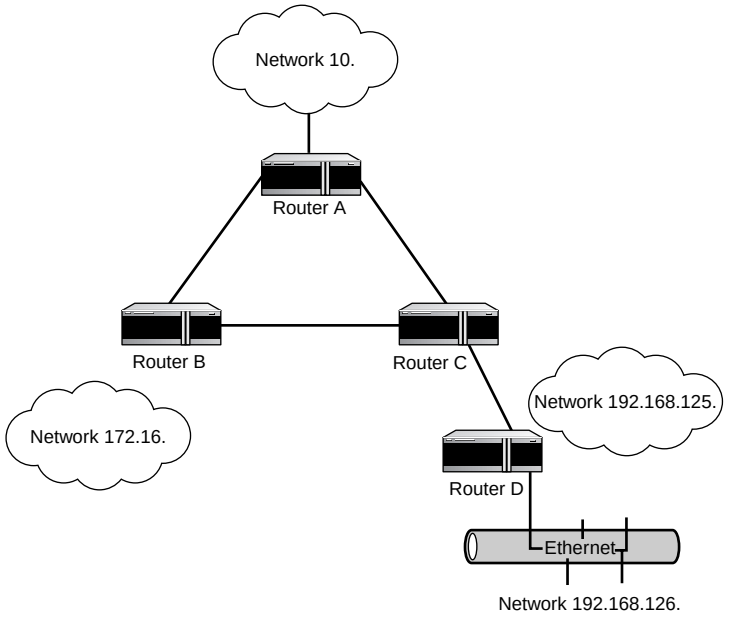
Static routes can also be used for troubleshooting and temporary fixes to routing problems. For instance, if you know that the routing entries in a table are corrupt, overriding these incorrect entries with static routes that contain the correct route paths might be useful.

Generally speaking, mixing static routing with dynamic routing is not a good idea. Doing so can cause static routes to be in conflict with dynamically computed routes. In this case, static routes will be used even if the dynamic routes are more efficient.

## Drawbacks to Static Routing

In the event of a network failure, or other source of topology change, the onus is on the network administrator to manually accommodate the change. Figure 13.1 illustrates this point.

**FIGURE 13.1**
*A simple internetwork with static routes.*



In this simple example, the networks' administrators have collaborated on a scheme for distributing routing information that they believe will minimize their workload as well as network traffic loads. The internetwork is relatively small, consisting of three different networks, one of which supports a small, isolated network. Such networks are called *stub* networks. Each network uses its own address space and a different dynamic routing protocol. Given the innate incompatibility of the three different routing protocols, the administrators chose not to redistribute routing information between their networks. Rather, they aggregated the routes into network numbers, and statically defined paths for them. The routing tables of the three gateway routers are summarized in Table 13.1. Router D connects a small, stub network to the other networks. As such, this router uses its serial port as a default gateway for all packets addressed to any IP address that does not belong to 192.168.126.

**TABLE 13.1** Statically Defined Routes

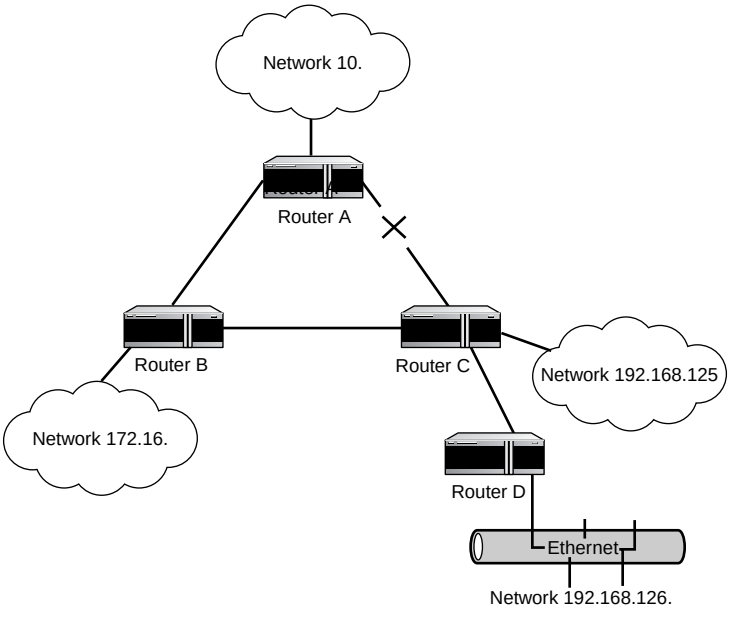
| *Router* | *Destination* | *Next Hop* |
|---|---|---|
| A | 172.16.0.0 | B |
| A | 192.168.125.0 | C |
| A | 192.168.126.0 | C |
| B | 10.0.0.0 | A |
| B | 192.168.125.0 | C |
| B | 192.168.126.0 | C |
| C | 10.0.0.0 | A |
| C | 172.16.0.0 | B |
| C | 192.168.126.0 | D |



In this scenario, Router A will forward all packets addressed to any hosts within the 172.16 network address space to Router B. Router A will also forward all packets addressed to hosts within networks 192.168.125 and 192.168.126 to Router C. Router B will forward all packets addressed to any hosts within the 192.168.125 and 192.168.126 address spaces to Router C. Router B will forward packets addressed to hosts within network 10 to Router A. Router C forwards all packets destined for network 10 to Router A, and 172.16 to Router B. Additionally, Router C forwards packets addressed to 192.168.126 to Router D—its stub network. This network is a stub because it is literally a dead-end in the network. There is only one way in and one way out. This small network is completely dependent upon its link to Router C, and Router C itself, for connectivity to all the internetworked hosts.

In this example, a failure will result in unreachable destinations despite the fact that an alternate path is available for use. In Figure 13.2, the transmission facility between Gateway Routers A and C has failed.

**FIGURE 13.2**
*A link failure in a statically programmed internetwork can disrupt communications.*




The effect of this failure is that end systems in networks 10 and 192.168 are unable to communicate with each other, even though a valid route exists through Router B! The effects of this type of failure on the routing tables are summarized in Table 13.2.

**TABLE 13.2** Static Routes with a Failed Link

| *Router* | *Destination* | *Next Hop* |
|---|---|---|
| A | 172.16.0.0 | B |
| A | 192.168.125.0 | C—unreachable |
| A | 192.168.126.0 | C—unreachable |
| B | 10.0.0.0 | A |
| B | 192.168.125.0 | C |
| B | 192.168.126.0 | C |
| C | 10.0.0.0 | A—unreachable |
| C | 172.16.0.0 | B |

The lack of any dynamic mechanism prevents Routers A and C from recognizing the link failure. They are not using a routing protocol that would otherwise discover and test the qualities of the links to known destinations. Consequently, Routers A and C cannot discover the alternate path through Router B. Although this route is valid and usable, their programming prevents them from discovering, or using, it. This situation will remain constant until the network administrator(s) takes corrective action manually.

## Benefits of Static Routing

At this point, you might be wondering what possible benefit there is in statically defined routes. Static routing is good only for very small networks that have a single path to any given destination. In such cases, static routing can be the most efficient routing mechanism because it doesn't consume bandwidth trying to discover routes or communicate with other routers.

As networks grow larger, and add redundant paths to destinations, static routing becomes a labor-intensive liability. Any changes in the availability of routers or transmission facilities in the WAN must be manually discovered and programmed in. WANs that feature more complex topologies that offer multiple potential paths absolutely require dynamic routing. Attempts to use static routing in complex, multipath WANs will defeat the purpose of having that route redundancy.

There are instances where statically defined routes would be desirable, even in large or complex networks. For example, static routes can be configured to enhance security. Your company's connection to the Internet could have a statically defined route to a security server. No ingress would be possible without having first passed whatever authentication mechanisms the security server provides.

Alternatively, statically defined routes might be extremely useful in building extranet connections using IP to other companies with which your employer does a lot of business. Lastly, static routes might be the best way to connect small locations with stub networks to your WAN. The point is that static routes can be quite useful. You just need to understand what they can and can't do.

# Distance-Vector Routing

In routing based on distance-vector algorithms, also sometimes called Bellman-Ford algorithms, the algorithms periodically pass copies of their routing tables to their immediate network neighbors. Each recipient adds a *distance vector*, or its own distance "value," to the table and forwards it on to their immediate neighbors. This process occurs omnidirectionally between immediately neighboring routers. This step-by-step process results in each router learning about other routers, and developing a cumulative perspective of network "distances."

The cumulative table is then used to update each router's routing tables. When completed, each router has learned approximate information about the "distances" to networked resources. It does not learn anything specific about other routers, or the network's actual topology.

## Drawbacks to Distance-Vector Routing

Distance-vector routing can, under certain circumstances, actually create routing problems for distance-vector protocols. For example, a failure or other change in the network requires some time for the routers to *converge* on a new understanding of the network's topology. During the convergence process, the network may be vulnerable to inconsistent routing, and even infinite loops. Safeguards exist to contain many of these risks, but the fact remains that the network's performance is at risk during the convergence process. Thus, older distance-vector protocols that are slow to converge may not be appropriate for large, complex WANs.

Even in smaller networks, distance-vector routing protocols may be problematic at worst, or suboptimal, at best. This is because the simplicity that is this genre's strength can also be a source of weakness. For example, Figure 13.3 presents an internetwork with specific geographic locations.

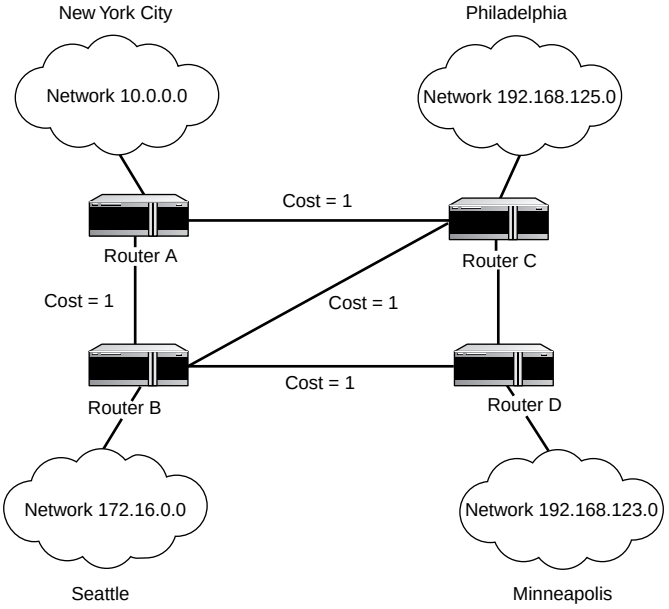



**FIGURE 13.3**
*An internetwork using a distance-vector routing protocol.*

In this example, network 1 is located in New York, 2 is in Seattle, 3 is in Philadelphia, and 4 is in Minneapolis. The distance-vector routing protocol uses a statically assigned cost of 1 for each hop, regardless of the distance of the link or even its bandwidth. Table 13.3 summarizes the number of hops to each of the destination network numbers. In this table, you will notice that routers do not have to create separate entries in their routing tables for every known end-system. Your network can perform such host-based routing, but it will likely perform much better if you perform only network-based routing. Network-based routes are those routes based on just the network portion of an IP address; the host identifier is truncated.

In theory, the same path can be used to get to all the hosts or end-systems on any given network. Thus, nothing is gained by creating separate entries for each host address.

**TABLE 13.3** The Number of Hops with the Distance-Vector Protocol

| *Router* | *Destination* | *Next Hop* | *Number of Hops to Destination* |
|---|---|---|---|
| A | 172.16.0.0 | B | 1 |
| A | 192.168.125.0 | C | 1 |
| A | 192.168.253.0 | B or C | 2 |
| B | 10.0.0.0 | A | 1 |
| B | 192.168.125.0 | C | 1 |
| B | 192.168.253.0 | D | 1 |
| C | 10.0.0.0 | A | 1 |
| C | 172.16.0.0 | B | 1 |
| C | 192.168.253.0 | D | 1 |
| D | 10.0.0.0 | B or C | 2 |
| D | 172.16.0.0 | B | 1 |
| D | 192.168.125.0 | C | 1 |



In any internetwork with redundant routes, using a distance-vector protocol rather than static routes is better. This is because distance-vector routing protocols can automatically detect and correct most failures in the network. Unfortunately, they aren't perfect. For instance, consider the routing table entries for Gateway Router A. This is the New York Gateway. From its perspective, the Minneapolis Gateway is two hops away, regardless of whether it goes through Philadelphia or Seattle. This router would be indifferent to accessing Minneapolis through either Philadelphia or Seattle.

If all the variables in the network were held constant, (including such things as traffic levels, the bandwidth of each link, and even transmission technology) the geographically shortest path would incur the least amount of propagation delay. Thus, logic dictates taking the shorter route, through Philadelphia. In reality, such logic is beyond the abilities of simple distance-vector protocols. Distance-vector protocols aren't exactly limited by this because propagation delay is often the least significant of the factors driving the performance of a route. Bandwidth and traffic levels can both have much more noticeable effects on the performance of a network.

### Benefits of Distance-Vector Routing

Distance-vector protocols are generally very simple protocols that are easy to understand, configure, maintain, and use. Consequently, they are quite useful in very small networks that have few, if any, redundant paths and no stringent network performance requirements. The epitome of the distance-vector routing protocol is *Routing Information Protocol* (RIP). RIP uses a single distance metric to determine the next best path to take for any given packet: cost. RIP has been widely used for decades, and has only recently warranted updating. For more information on RIP, see Chapter 15, "Routing Information Protocol (RIP)."

## Link-State Routing

Link-state routing algorithms, known cumulatively as *Shortest Path First* (SPF) protocols, maintain a complex database of the network's topology. Unlike distance-vector protocols, link-state protocols develop and maintain a full knowledge of the network's routers, as well as how they interconnect. This is achieved via the exchange of *Link-State Advertisements* (LSAs) with other routers in a network.

Each router that has exchanged LSAs then constructs a topological database using all received LSAs. An SPF algorithm is then used to compute reachability to networked destinations. This information is used to update the routing table. This process is capable of discovering changes in the network topology that were caused by component failure or network growth.

In fact, the LSA exchange is triggered by an event in the network, rather than running periodically. This can greatly expedite the convergence process because there is no need to wait for a series of arbitrary timers to expire before the networked routers can begin to converge!

If the internetwork depicted in Figure 13.3 were to use a link-state routing protocol, the concerns about connectivity between New York and Minneapolis would be rendered moot. Depending on the actual protocol employed, and the metrics selected, it is highly likely that the routing protocol would be able to discriminate between the two paths and try to use the best one. The summarized contents of the gateways' routing tables are presented in Table 13.4.

**TABLE 13.4** Hop Counts in a Link-State Network

| *Router* | *Destination* | *Next Hop* | *Number of Hops to Destination* |
|---|---|---|---|
| A | 172.16.0.0 | B | 1 |
| A | 192.168.125.0 | C | 1 |
| A | 192.168.253.0 | B | 2 |
| A | 192.168.253.0 | C | 2 |
| B | 10.0.0.0 | A | 1 |
| B | 192.168.125.0 | C | 1 |
| B | 192.168.253.0 | D | 1 |
| C | 10.0.0.0 | A | 1 |
| C | 172.16.0.0 | B | 1 |
| C | 192.168.253.0 | D | 1 |
| D | 10.0.0.0 | B | 2 |
| D | 10.0.0.0 | C | 2 |
| D | 172.16.0.0 | B | 1 |
| D | 192.168.125.0 | C | 1 |



As is evident in this table's routing entries for the New York-to-Minneapolis routes, a link-state protocol would remember both routes. Some link-state protocols may even provide a means to assess the performance capabilities of these two routes, and bias toward the better performing one. If the better performing path, for example the route through Philadelphia, were to experience operational difficulties of any kind (including congestion or component failure), the link-state routing protocol would detect this change and begin forwarding packets through Seattle.

## Drawbacks to Link-State Routing

Despite all of its features and flexibility, link-state routing raises two potential concerns:

- During the initial discovery process, link-state routing protocols can flood the network's transmission facilities, thereby significantly decreasing the network's ability to transport data. This performance degradation is temporary, but can be very noticeable. Whether or not this flooding process will impede a network's performance noticeably will depend on two things: the amount of available bandwidth, and the number of routers that must exchange routing information. Flooding in large networks with relatively small links (such as low-bandwidth DLCIs on a Frame Relay network) will be much more noticeable than a similar exercise on a small network with large-sized links (such as T3s).
- Link-state routing is both memory and processor intensive. Consequently, more fully configured routers are required to support link-state routing than distance-vector routing. This can tend to increase the cost of the routers configured for link-state routing.

These flaws are hardly fatal in the link-state approach to routing. The potential performance impacts of both can be addressed, and resolved, through foresight, planning, and engineering.

## Benefits of Link-State Routing

The link-state approach to dynamic routing can be quite useful in networks of any size. In a well-designed network, a link-state routing protocol will enable your network to gracefully weather the effects of unexpected topological change. Using events, such as changes, to drive updates (rather than fixed-interval timers) enables convergence to begin that much quicker after a topological change.

The overheads of the frequent, time-driven updates of a distance-vector routing protocol are also avoided. This allows more bandwidth to be used for routing traffic instead of network maintenance, provided you design your network properly.

A side benefit of the bandwidth efficiency of link-state routing protocols is that they facilitate network scalability better than either static routes or distance-vector protocols. Taking into account the limitations of static routes and distance-vector protocols, it is easy to see that link-state routing is best in larger, more complicated networks or in networks that must be highly scalable. It may be challenging to initially configure a link-state protocol in a large network, but it is well worth the effort in the long run. For more information on link-state routing, see Chapter 16, "Open Shortest Path First (OSPF)."

# Convergence in an IP Network

One of the most fascinating aspects of routing is a concept known as *convergence*. Quite simply, whenever there is a change in a network's topology, or shape, all the routers in that network must develop a new understanding of the changed network topology. This process is both collaborative and independent; the routers share information with each other, but must independently calculate the impacts of the topology change on their own routes. Because they must mutually develop an agreement of the new topology independently from different perspectives, they are said to *converge* upon this consensus.

Convergence is necessary because routers are intelligent devices capable of making their own routing decisions. This is simultaneously a source of strength and vulnerability. Under normal operating conditions, this independent and distributed intelligence is a tremendous advantage. During changes in the network's topology, the process of converging on a new consensus of the network's shape may actually introduce instability and routing problems.

## Accommodating Topological Changes

Unfortunately, the independent nature of routers can also be a source of vulnerability whenever a change occurs in the network's topology. Such changes, by their very nature, change a network's topology. Figure 13.4 illustrates how a change in the network is, de facto, a change in its topology.



**FIGURE 13.4**
*A four-gateway internetwork.*

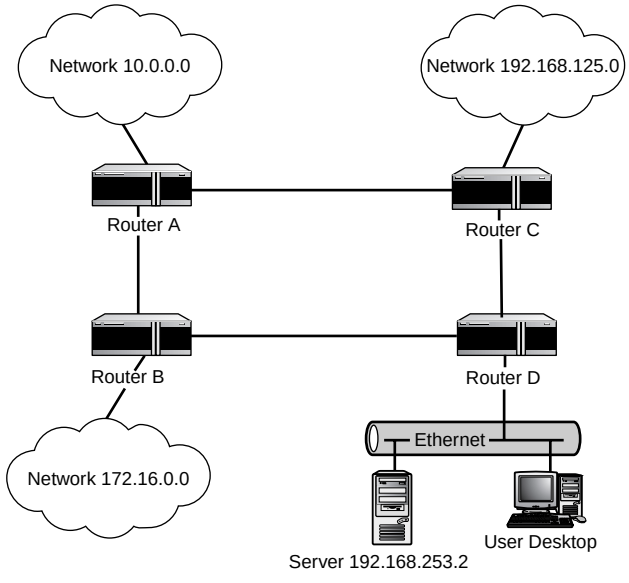


Figure 13.4 features another fairly simple, four-node internetwork with some route redundancy. The routing tables of the four routers are summarized in Table 13.5. For the sake of this example, consider this table to be preconvergence routing table information.

**TABLE 13.5** Preconvergence Routing Table Contents

| *Router* | *Destination* | *Next Hop* | *Number of Hops to Destination* |
|---|---|---|---|
| A | 172.16.0.0 | B | 1 |
| A | 192.168.125.0 | C | 1 |
| A | 192.168.253.0 | B or C | 2 |
| B | 10.0.0.0 | A | 1 |
| B | 192.168.125.0 | A or D | 2 |
| B | 192.168.253.0 | D | 1 |
| C | 10.0.0.0 | A | 1 |
| C | 172.16.0.0 | A or D | 2 |
| C | 192.168.253.0 | D | 1 |
| D | 10.0.0.0 | B or C | 2 |
| D | 172.16.0.0 | B | 1 |
| D | 192.168.125.0 | C | 1 |

If packets sent by Router C to server 192.168.253.2 suddenly become undeliverable, it is likely that an error occurred somewhere in the network. This could have been caused by a seemingly infinite number of different, specific failures. Some of the more common suspects include

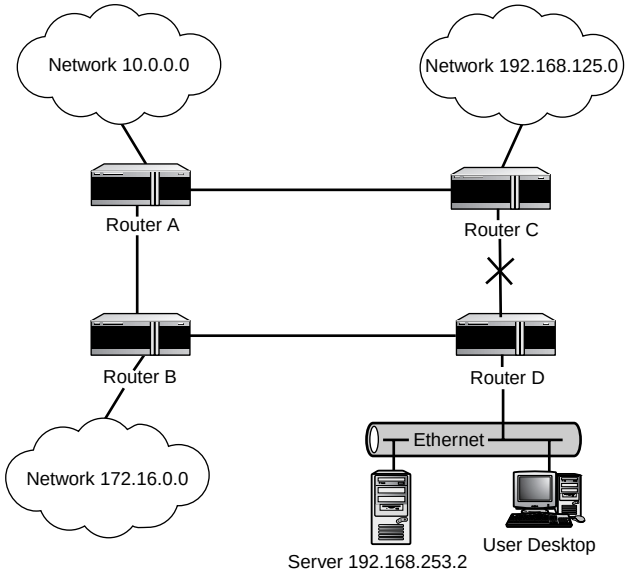
- The server has failed completely (due to a hardware, software, or electrical failure).
- The LAN connection to the server has failed.
- Router D has experienced a total failure.
- Router D's serial interface port to Router C has failed.
- The transmission facility between Gateway Routers C and D has failed.
- Router C's serial interface port to Router D has failed.

Obviously, the new network topology can't be determined until the exact location of the failure has been identified. Similarly, the routers can't attempt to route around the problem until the failure location has been isolated. If any of the first two scenarios occurred, server 192.168.253.2 would be completely unavailable to all the users of the internetwork, regardless of any route redundancy that may have been built in to the network.

Similarly, if router D had failed completely, all the LAN-attached resources at that location would be isolated from the rest of the network. If, however, the failure was either a partial failure of that router, or elsewhere in the network, there might still be a way to reach hosts located within network 192.168.255. Finding a new route to network 192.168.255 requires the network's routers to recognize, and agree, on what piece of the network failed. Subtracting this component from the network, in effect, changes the network's topology.

To continue with the example, assume that Router D's serial interface port to Router C has failed. This renders the link between C and D unusable. The new network topology is illustrated in Figure 13.5.

**FIGURE 13.5**
*The link between routers C and D is unusable.*




Routers using a dynamic routing protocol would quickly determine that server 192.168.253.2 was unreachable through their current, preferred route. Individually, none of the routers would be able to determine where the actual failure occurred, nor could they determine whether any viable alternate routes still existed. However, by sharing information with each other, a new composite picture of the network can be developed.

**Note**

For the purposes of this chapter, this example uses an intentionally generic method of convergence. More specific details about each routing protocol's convergence characteristics are presented in Chapters 14 through 16.

The routing protocol used in this internetwork is relatively simple. It limits each router to just exchanging routing information with its immediate neighbors, although it supports the recording of multiple routes per destination. Table 13.6 summarizes the pairs of immediately adjacent routers illustrated in Figure 13.5.

**TABLE 13.6** Routers That Share Routing Information with Immediate Neighbors

| *Router* | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| **A** | — | yes | yes | no |
| **B** | yes | — | no | yes |
| **C** | yes | no | — | yes |
| **D** | no | yes | yes | — |

The entries in Table 13.6 that contain the word *yes* indicate a physically adjacent pair of routers that would exchange routing information. The entries that contain a dash denote the same router: A router cannot be adjacent to itself. Lastly, those entries that contain the word *no* indicate nonadjacent routers that cannot directly exchange routing information. Such routers must rely upon their adjacent neighbors for updates about destinations on nonadjacent routers.

From this table, it is apparent that, because they are not directly connected to each other, Routers A and D must rely on Routers B and C for information about each other's destinations. Similarly, Routers B and C must rely on Routers A and D for information about each other's destinations.

Figure 13.6 pictorially demonstrates this sharing of routing information between immediate neighbors.

The important implication in this scenario is that, because not every router is immediately adjacent to every other router, more than one routing update may be required to fully propagate new routing information that accommodates the failed link. Thus, accommodating topological change is an iterative and communal process.

For the sake of simplicity, assume that convergence occurs within two routing table updates in this example. During the first iteration, the routers are starting to converge on a new understanding of their topology. Routers C and D, due to the unusable link between them, cannot exchange routing information. Consequently, they invalidate this route, and all destinations that use it. The contents of the four routers' routing tables *during* the convergence process are summarized in Table 13.7. Please note that the contents of some routing tables may reflect the mistaken belief that the link between Routers C and D is still valid.

**FIGURE 13.6**
*Immediate neighbors sharing routing data.*

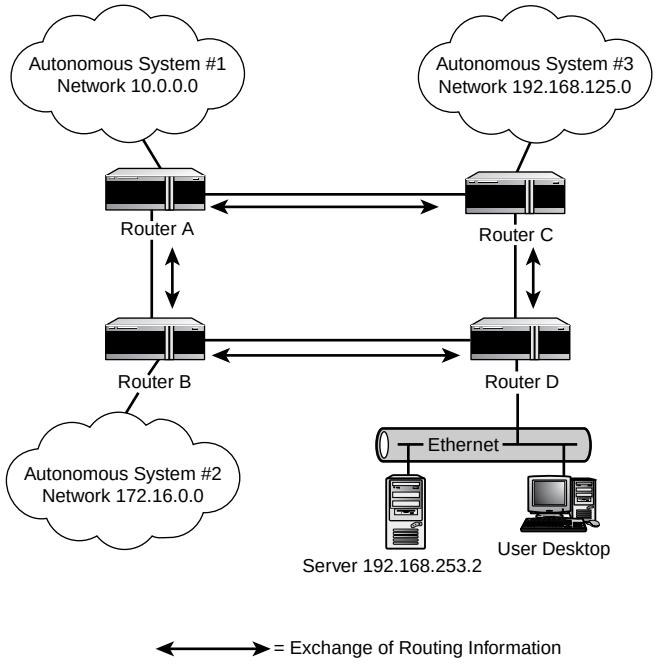





**TABLE 13.7** Midconvergence Routing Table Contents

| *Gateway Router* | *Destination* | *Next Hop* | *Number of Hops to Destination* |
|---|---|---|---|
| A | 172.16.0.0 | B | 1 |
| A | 192.168.125.0 | C | 1 |
| A | 192.168.253.0 | B or C | 2 |
| B | 10.0.0.0 | A | 1 |
| B | 192.168.125.0 | A or D | 2 |
| B | 192.168.253.0 | D | 1 |
| C | 10.0.0.0 | A | 1 |
| C | 172.16.0.0 | A only (D failed) | 2 |
| C | 192.168.253.0 | D—invalid route | not reachable |
| D | 10.0.0.0 | B or C | 2 |
| D | 172.16.0.0 | B | 1 |
| D | 192.168.125.0 | C—invalid route | not reachable |

In Table 13.7, Routers C and D have invalidated the route between them. Routers A and B, however, still believe that their routes through this link are viable. They must await a routing update from either Router C or D before they can recognize the change in the internetwork's topology.

Table 13.8 contains the contents of the four routers' routing tables *after* they have converged on a new topology. Remember, this depiction of the convergence process is intentionally generic, and not indicative of any particular routing protocol's mechanics.

**TABLE 13.8** Postconvergence Routing Table Contents

| *Router* | *Destination* | *Next Hop* | *Number of Hops to Destination* |
|---|---|---|---|
| A | 172.16.0.0 | B | 1 |
| A | 192.168.125.0 | C | 1 |
| A | 192.168.253.0 | B only | 2 |
| B | 10.0.0.0 | A | 1 |
| B | 192.168.125.0 | A only | 2 |
| B | 192.168.253.0 | D | 1 |
| C | 10.0.0.0 | A | 1 |
| C | 172.16.0.0 | A only | 2 |
| C | 192.168.253.0 | A | 3 |
| D | 10.0.0.0 | B only | 2 |
| D | 172.16.0.0 | B | 1 |
| D | 192.168.125.0 | B only | 3 |

As evident in Table 13.8, all the routers in the internetwork eventually agree that the link between C and D is unusable, but that destinations in each autonomous system are still reachable via an alternate route.

## Convergence Time

It is virtually impossible for all routers in a network to simultaneously detect a topology change. In fact, depending on the routing protocol in use, as well as numerous other factors, there can be a considerable time delay before all the routers in that network reach a consensus, or agreement, on what the new topology is. This delay is referred to as *convergence time*. The important thing to remember is that convergence is not immediate. The only uncertainty is how much time is required for convergence to occur.

Some factors that can exacerbate the time delay inherent in convergence include

- A router's distance (in hops) from the point of change.
- The number of routers in the network that use dynamic routing protocols.
- Bandwidth and traffic load on communications links.
- A router's load.
- Traffic patterns vis-à-vis the topological change.
- The routing protocol used.

The effects of some of these factors can be minimized through careful network engineering. For example, a network can be engineered to minimize the load on any given router or communications link. Other factors, such as the number of routers in the network, must be accepted as risks inherent in a network's design. However, it may be possible to engineer the network such that fewer routers need to converge. By using static routes to interconnect stubs to the network, you reduce the number of routers that must converge. This directly reduces convergence times. Given these factors, clearly the two keys to minimizing convergence times are

- Selection of a routing protocol that can calculate routes efficiently.
- Designing the network properly.



# Calculating Routes in IP Networks

As demonstrated through the examples in the preceding section, convergence is absolutely critical to a network's ability to respond to operational fluctuations. The key factor in convergence is communications between the routers in the network. Routing protocols are responsible for providing this function. Specifically, these protocols are designed to enable routers to share information about routes to the various destinations within the network.

Unfortunately, all routing protocols are not created equal. In fact, one of the best ways to assess the suitability of a routing protocol is to evaluate its abilities to calculate routes and converge relative to other routing protocols. It should be obvious from the preceding list of factors that convergence times may be difficult for you to calculate with any degree of certainty. Your router vendor may be able to assist you with this process, even if it only provides you with general estimates.

A routing protocol's convergence ability is a function of its ability to calculate routes. The efficacy of a routing protocol's route calculation is based upon several factors:

- Whether or not the protocol calculates, and stores, multiple routes to each destination.
- The manner in which routing updates are initiated.
- The metrics used to calculate distances or costs.

Each of these specific functions, and how they contribute to the overall operating efficiency of a routing protocol, is examined in the following sections.

# Storing Multiple Routes

Some routing protocols attempt to improve their operational efficiency by recording only a single route (ideally, the best route) to each known destination. The drawback to this approach is that, when a topology change occurs, each router must calculate a new route through the network for the impacted destinations.

Other protocols accept the processing overheads that accompany larger routing table sizes, and store multiple routes to each destination. Under normal operating conditions, maintaining multiple routes enables the router to balance traffic loads across multiple links. If, or when, a topology change occurs, the routers already have alternate routes to the impacted destinations in their routing tables. Having an alternate route already mapped out does not necessarily accelerate the convergence process. It does, however, enable networks to more gracefully sustain topology changes.

# Initiating Updates

Some routing protocols use the passing of time to initiate routing updates. Others are event-driven. That is, they are initiated whenever a topological change is detected. Holding all other variables constant, event-driven updates will result in shorter convergence times than will timed updates.

## Timed Updates

A timed update is a very simple mechanism. Time is decremented in a counter as it elapses. When a specified period of time has elapsed, an update is performed regardless of whether a topological change has occurred. This has two implications:

- Many updates will be performed unnecessarily, which wastes bandwidth and router resources.
- Convergence times can be needlessly inflated if route calculations are driven by the passing of time.

Distance Vector routing protocols generally use timed updates.

## Event-Driven Updates

Event-driven updates are a much more sophisticated means of initiating routing updates. Ostensibly, an update is only initiated when a change in the network's topology has been detected. Given that a topology change is what creates the need for convergence, this approach is obviously the more efficient one.

You can select an update initiator simply by selecting a routing protocol; each protocol implements either one or the other. Thus, this is one factor that must be considered when selecting a routing protocol.

Link-state routing protocols generally use event-driven updates.

# Routing Metrics

Another important mechanism that is determined by the routing protocol is metric(s). A wide disparity exists in terms of both the number and the type of metrics used.

## Quantity of Metrics

Simple routing protocols support as few as one or two routing metrics. More sophisticated protocols can support five or more metrics. It is safe to assume that the more metrics exist, the more varied and specific they are. Thus, the greater the variety of available metrics, the greater your ability to tailor the network's operation to your particular needs. For example, the simple distance-vector protocols use a euphemistic metric: distance. In reality, that distance is not at all related to geographic mileage, much less the physical cable mileage that separates source and destination machines. Instead, it usually just counts the number of hops between those two points.

Link-state protocols may afford the ability to calculate routes based on several factors:

- Traffic load
- Available bandwidth
- Propagation delay
- The network cost of a connection (although this metric tends to be more of an estimate than an actual value)

Most of these factors are highly dynamic in a network; they vary by time of day, day of week, and so on. The important thing to remember is that as they vary, so does the network's performance. Therefore, the intent of dynamic-routing metrics is to allow optimal routing decisions to be made using the most current information available.

## Static Versus Dynamic Metrics

Some metrics are simplistic and static, whereas others are highly sophisticated and dynamic. Static metrics usually offer the ability to customize their values when they are configured. After this is done, each value remains a constant until it is manually changed.

Dynamic protocols enable routing decisions to be made based upon real-time information about the state of the network. These protocols are only supported by the more sophisticated link-state or hybridized routing protocols.

# Summary

Routing is the function that enables users and end-systems to communicate within, and across, IP networks. However, there are many ways to calculate and compare routes. The differences are more than academic or philosophic—they can directly impact the performance and functionality of your IP network. This chapter provided a generic overview of the various ways that routing can be performed, as well as the benefits and limitations of each. The next three chapters build on this overview of IP routing, with specific details about how the RIP, OSPF, and BGP operate.

CHAPTER 14

# Gateway Protocols

*by Tim Parker*

## In This Chapter

TCP/IP was developed primarily to support internetwork traffic on the network that eventually became the Internet. To that end, TCP/IP was designed with a layered architecture, which specifically works well across networks. As a datagram passes from network to network along the internetwork, it passes through machines that act as routers into each network.

The routers determine whether the datagram is for the local network to which the gateway leads, and if so, removes it from the internetwork backbone and routes it through the local network. If the datagram is to be passed on to other routers further down the internetwork, the router performs that function. In order to correctly forward datagrams on to other routers, each route has to have an up-to-date table of destinations that are used by the routing software. This chapter looks at how internetwork routers handle the routing of information between themselves. Special protocols have been developed specifically for different kinds of routers.

# Gateways, Bridges, and Routers

When a router receives a datagram from an internetwork, it performs a simple check of the message's destination address, which is contained in the TCP Protocol Data Unit. If the network portion of the IP address for the destination machine matches the network's IP address, the router knows the datagram is for a machine on its directly connected network and passes the datagram into the network for delivery. If the datagram's IP address reveals that the datagram is not for the directly connected network, the datagram is passed on to the next router on the internetwork.

Moving messages from machine to machine on a small network is easy because each machine can be aware of the IP addresses of every other machine on the network. With a large network or several networks tied together into an internetwork, the complexity increases enormously. For very large internetworks, such as the Internet, it would be impossible for a single router to hold all the valid IP addresses of every machine on the Net. For this reason, several special devices were developed to simplify the routing of datagrams from network to network, across an internetwork, or through a wide area network. These devices are called gateways, bridges, and routers. They vary in purpose, as the following definitions show:

- A *gateway* is a machine that performs protocol translations. Gateways operate at OSI layers 4 to 7. Typically, they operate at Layer 7, such as email gateways.
- A *bridge* is a machine that connects two or more networks that use the same protocol. A bridge operates at Layer 2 of the OSI model. Lately, a new class of devices called Layer 2 switches have become available that can be used as replacements for bridges. Bridges are aware of Data Link Layer addresses but not IP addresses of the Network Layer.

- A *router* is a machine that forwards datagrams around a network. Routers operate at Layer 3 of the OSI model. Commercial routers may perform other non-Layer 3 functions. Examples are routers that combine routing functions with a bridge, routers that perform network address translation (NAT) functions, and routers that perform security-related functions such as packet filtering.

**Note**

In the earlier Internet literature, the term *gateways* was commonly used to refer to devices that are now called routers. This usage is unfortunate because it causes some confusion. The modern terminology for the older gateway devices is routers.

## Gateway

A gateway is a device that can convert protocols. This is necessary if the gateway is acting as an interface between the Internet (using TCP/IP) and a local area network (using Novell NetWare, for example). The gateway has to convert the NetWare IPX/SPX packets to TCP/IP datagrams, and vice versa, for the two networks to be able to exchange data. Gateways can perform translations between many different protocols, often servicing more than two protocols at the same time, depending on the network connections. Gateways may also have to perform conversion of file formats or handle encryption and decryption, depending on the network systems.

## Bridge

Bridges are easily thought of as a link between two or more networks. Often, a leased high-speed line is used to connect one LAN to another, as would be the case of a multi-national company with offices in separate locations. Both networks may use the same protocol (such as TCP/IP), but need a fast routing system between the two over a high-speed telecommunications line. A bridge handles the routing of datagrams from one LAN to another. Bridges can handle many LANs at the same time, but they all must use the same protocol.

## Router

Routers operate more or less at the network level. Their function is to forward datagrams to their destination. Some routers can perform protocol conversions, like a gateway, when there are optional routes to a destination.

## Autonomous System

The term *autonomous system* is often used when talking about networks attached to the Internet and other internetworks. An autonomous system is one in which the structure of the local area network to which it is attached is not visible to the rest of the internetwork. Usually, a border router leads into the local area network and all traffic for that network goes through the border router. This hides the internal structure of the local area network from the rest of the internetwork, which both simplifies handling of datagrams and adds security.

# Gateway Protocols: The Basics

As mentioned earlier in this chapter, having a single router hold the entire Internet routing table is practically impossible, so most routers handle only a specific section of the internetwork and rely on neighboring gateways to know more about their own attached networks. This results in a common problem, though, when a lack of information results in incomplete routing decisions. For this reason, default routes are used.

Routing protocols exchange routing and status information between gateways. Several router protocols are designed for fast, reliable data transfer with a minimum of overhead. Before looking at the protocols, distinguishing between two types of gateways used on the Internet (and most other internetworks, too) is necessary. The gateway types are called core and non-core.

*Core gateways* are machines administered by the *Internet Network Operations Center* (INOC) and form part of the backbone of the Internet. As noted earlier, even though these machines are called gateways, they are actually routers. Core gateways were first developed for ARPAnet, where they were called *stub gateways*. *Non-core gateways* are administered by groups outside the Internet organization that are connected to the Internet but administered by the owning company or organization. Typically, corporations and educational institutions that reside on the Internet use non-core gateways. Back in ARPAnet days, any gateway not under direct control of ARPAnet (any non-core gateway in current terminology) was called a *nonrouting gateway*.

The change to the Internet structure and its growing number of core gateways required the development of a protocol to enable the core gateways to communicate with each other. This is the *Gateway-to-Gateway Protocol* (GGP), which is usually used only between core gateways. GGP is used primarily to spread information about the non-core gateways attached to each core gateway, allowing each core gateway to update its routing tables.

Some local area or wide area networks have more than one router within them. For example, you may have a large network that has so much Internet-bound traffic that two routers are used to handle the shared load. On the other hand, if you have two distinct LANs that are part of a larger corporate-wide area network, you may set up the local area networks so each has its own router. If two routers are used in a LAN or WAN and they can talk to each other, they are considered interior neighbors. If the routers don't talk to each other directly (they belong to different autonomous systems), they are called *exterior gateways* or *border routers*. When default routes are required, it is up to the exterior gateways to route messages between autonomous systems.

Within a single local or wide area network, routing information is usually transferred between interior gateways through the *Routing Information Protocol* (RIP). Some systems use a less common protocol called HELLO. Both HELLO and RIP are *Interior Gateway Protocols* (IGPs) designed specifically for interior neighbors to communicate with each other. Messages between two exterior gateways are usually handled through the *Exterior Gateway Protocol* (EGP).

The RIP, HELLO, and EGP protocols all rely on a frequent transfer of a status datagram between gateways to update routing tables. The three gateway protocols are not independent but share a relationship. EGP is used between gateways of autonomous systems, whereas RIP and HELLO (both IGPs) are used within the network itself. GGP is used between core gateways on the Internet.

# Interior and Exterior Gateway Protocols

The details of the protocols used in router communications are not really important to users or developers because few applications use the routing protocols directly.

## Gateway-to-Gateway Protocol

Core gateways need to know what is happening to the rest of the Internet in order to route datagrams properly and efficiently. This includes routing information and the characteristics of attached subnetworks. A common example of using this type of information occurs when one gateway is particularly slow processing a heavy load and it is the only access method to a subnetwork; other gateways on the network can tailor the traffic to better offload the gateway.

GGP is used primarily to exchange routing information. It is important not to confuse routing information (containing addresses, topology, and details on routing delays) with algorithms used to make routing decisions. Routing algorithms are usually fixed within a gateway and not modified by GGP. A core gateway talks to other core gateways by sending out GGP messages, waiting for replies, and then updating routing tables if the reply has specific information in it.

GGP is a called a *vector-distance protocol*, meaning that messages tend to specify a destination (vector) and the distance to that destination. For a vector-distance protocol to be effective, a gateway must have complete information about all the gateways on the internetwork; otherwise, computing an efficient route to a destination is impossible. For this reason, all core gateways maintain a table of all the other core gateways on the Internet. This is a fairly small list and can easily be handled by the gateways.

**Note**

The routing protocol used on the Internet to communicate reachability information amongst the core Internet routers is the Border Gateway Protocol (BGP). The current version of BGP is version 4. BGP4 is required for supporting CIDR (Classless Internet Domain Routing), which summarizes routing entries.

## The Exterior Gateway Protocol

The Exterior Gateway Protocol is used to transfer information between non-core neighboring gateways. Non-core gateways contain all routing information about their immediate neighbors on the internetwork and the machines attached to them, but they lack information about the rest of the Internet.

For the most part, EGP is restricted to information about the LAN or WAN the gateway serves. This prevents too much routing information from passing through the local or wide area networks. EGP imposes restrictions on the non-core gateways about the machines it communicates with about routing information.

Because core gateways use GGP and non-core gateways use EGP, but they both reside on the Internet, there must be some method for the two to communicate with each other. The Internet enables any autonomous (non-core) gateway to send "reachability" information to other systems, which must also go to at least one core gateway. If there is a larger autonomous network, one gateway usually assumes the responsibility for handling this reachability information.

As with GGP, EGP uses a polling process to keep gateways aware of their neighbors and to continually exchange routing and status information with all of their neighbors. EGP is a *state-driven protocol*, meaning it depends on a state table containing values that reflect gateway conditions and a set of operations that must be performed when a state table entry changes.

## Interior Gateway Protocols

Several interior gateway protocols are in use, the most popular of which are RIP and HELLO (mentioned earlier in this chapter), and a third protocol called *Open Shortest Path First* (OSPF). No single protocol has proven dominant, although RIP and OSPF are probably the most common IGP protocols. The specific choice of an IGP is made on the basis of network architecture.

Both the RIP and HELLO protocols calculate distances to a destination, and their messages contain both a machine identifier and the distance to that machine. In general, messages tend to be long because they contain many entries for a routing table. Both RIP and HELLO are constantly connecting between neighboring gateways to ensure the machines are active.

The Routing Information Protocol version 1 uses a broadcast technology. This means that the gateways broadcast their routing tables to other gateways on the network at intervals. This is also one of RIP's problems because the increased network traffic and inefficient messaging can slow down networks. RIP version 2 uses multicast for sending routing table updates. Multicast is more efficient than broadcast because only the RIP routers that are part of the multicast group will process the message.

The HELLO protocol is different from RIP in that HELLO uses time instead of distance as a routing factor. This requires the gateway to have reasonably accurate timing information for each route. For this reason, the HELLO protocol depends on clock synchronization messages. HELLO was used in some university environments but is not widely used in corporate networks.

The Open Shortest Path First protocol was developed by the Internet Engineering Task Force to provide a more efficient IGP protocol. The name "shortest path" is inaccurate in describing this protocol's routing process. A better name would be "optimum path," in which a number of criteria are evaluated to determine the best route to a destination.



# Summary

This chapter took a brief look at gateway protocols. Routers are a critical component for forwarding information from one network to another. There are several important gateway protocols, all of which have been mentioned. The details of how some of these protocols work, such as RIP and OSPF, are covered in later chapters.

CHAPTER 15

# Routing Information Protocol (RIP)

*by Mark A. Sportack*

## IN THIS CHAPTER

The Routing Information Protocol, or RIP, as it is commonly called, is one of the most enduring of all routing protocols. RIP is one of a class of protocols based on distance-vector routing algorithms that predate ARPANET. These algorithms were academically described between 1957 and 1962. Throughout the 1960s, these algorithms were widely implemented by different companies and marketed under different names. The resulting products were highly related but, due to proprietary enhancements, couldn't offer complete interoperability.

This chapter delves into the details, mechanics, and uses of today's open standard RIP. RIP version 2 (RIPv2) is the latest version and adds subnet support to RIP. In addition RIPv2 uses multicasting instead of broadcasting that is used in original RIP. The original RIP is also called RIP version 1 (RIPv1).

# Understanding RFC 1058

The implementation of RIP predates the standard RIP described in RFC 1038. TCP/IP's implementation of RIP is based on the Xerox Networking System's (XNS) version of RIP. XNS was a suite of protocols developed by Xerox, and widely adopted in various forms by different organizations in the early days of the network industry. TCP/IP's RIP was first implemented in BSD Unix before there was a standard describing its operation. For a number of years, the BSD Unix source code was considered to be the "standard" RIP.

In June 1988, Request for Comments (RFC) 1058 was released that described RIP. Pre-RFC 1058 versions of RIP may, therefore, not be compatible.

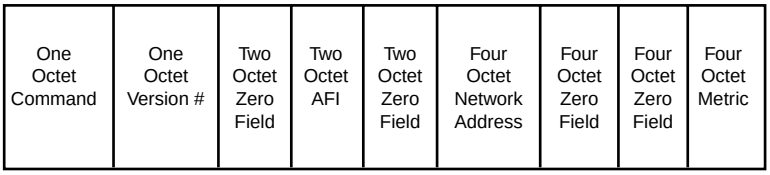
RFC 1058 described a new, and truly open, form of distance-vector routing protocol: an open standard RIP. This RIP, like its proprietary ancestors, was a simple distance-vector routing protocol that was specifically designed for use as an Interior Gateway Protocol (IGP) in small, simple networks.

Each device that uses RIP is assumed to have at least one network interface. Assuming that this network is one of the LAN architectures (such as Ethernet, Token Ring, and FDDI), RIP would only need to calculate routes to devices that are not directly connected to the same LAN. Depending on the application being used, devices that reside on the same LAN may communicate using just that LAN's mechanisms.

## RIP Packet Format

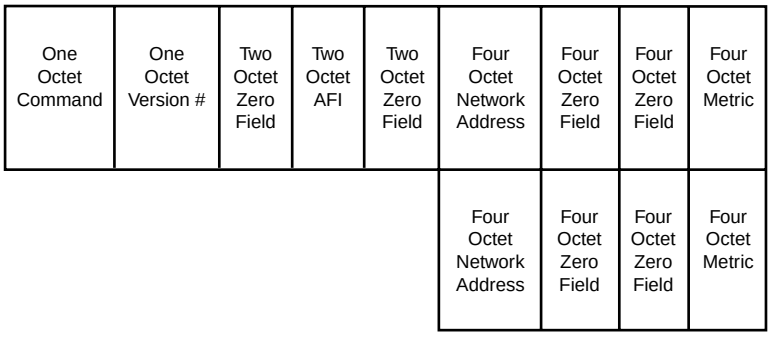
RIP uses a special packet to collect and share information about distances to known internetworked destinations. Figure 15.1 illustrates a RIP packet with routing information fields for just a single destination.

**FIGURE 15.1**
*The structure of a RIP packet.*




RIP packets can support up to 25 occurrences of the AFI, Internetwork Address, and Metric fields within a single packet. This enables one RIP packet to be used to update multiple entries in other routers' routing tables. RIP packets that contain multiple routing entries simply repeat the packet structure from the AFI through the Metric field, including all Zero fields. The repeated structures are appended to the end of the structure depicted in Figure 15.1. A RIP packet with two table entries is illustrated in Figure 15.2.

**FIGURE 15.2**
*The RIP packet format with two table entries.*




The Address field can contain either the address of its originator or a series of IP addresses that the originator has in its routing table. Request packets contain a single entry and include the originator's address. Response packets can include up to 25 entries of a RIP router's routing table.

The overall size limitation of a RIP packet is 512 octets. Thus, in larger RIP networks, a request for a full routing table update may require the transmission of several RIP packets. No provisions were made for the resequencing of the packets upon arrival at their destination; individual routing table entries are not split between RIP packets. Thus, the contents of any given RIP packet are complete unto themselves, even though they may only be a subset of a complete routing table. The recipient node is free to process the updates as the packets are received, without having to resequence them.

For example, a RIP router may contain 100 entries in its routing table. Sharing its routing information with other RIP nodes would require four RIP packets, each one containing 25 entries. If a receiving node received packet number four first (containing entries

numbered 76 through 100), it could simply update that portion of its routing table first. There are no sequential dependencies. This allows RIP packets to be forwarded without the overheads of a fully featured transport protocol, such as TCP.

**Note**

RIP uses UDP as its transport protocol. RIP messages are encapsulated in UDP/IP packets. The RIP processes on routers use UDP port 520.

## The Command Field

The Command field indicates whether the RIP packet was generated as a request or as a response to a request. The same frame structure is used for both occurrences:

- A request packet asks a router to send all, or part, of its routing table.
- A response packet contains routing table entries that are to be shared with other RIP nodes in the network. A response packet can be generated either in response to a request or as an unsolicited update.

## The Version Number Field

The Version Number field contains the version of RIP that was used to generate the RIP packet. Although RIP is an open standard routing protocol, it is not frozen in time. RIP has been treated to updates over the years, and these updates are reflected in a version number. Despite the many RIP-like routing protocols that have emerged, there are only two versions of RIP: version 1 and version 2. RIP version 1 uses broadcast to send RIP messages, whereas RIP version 2 uses multicast. Multicast is more efficient than broadcasts and in the case of RIP version 2, only RIPv2 routers process the multicast messages. RIPv2 uses the class D multicast address 224.0.0.9.

## The Zero Fields

The numerous Zero fields embedded in each RIP packet are silent testimony to the proliferation of RIP-like protocols before RFC 1058. Most of the Zero fields were contrived as a means of providing backward compatibility with older RIP-like protocols, without supporting all of their proprietary features.

For example, two such obsolete mechanisms are *traceon* and *traceoff*. These mechanisms were abandoned by RFC 1058, yet the open standard RIP needed to be backward compatible with the proprietary RIP-like protocols that did support them. Thus, RFC 1058 preserved their space in the packet, but requires this space to always be set to zeroes. Packets received with these fields set to something other than zeroes are simply discarded.

Not all the Zero fields originated in this manner. At least one of the Zero fields was reserved for an unspecified future use.

In RIPv2, the first zero field is the routing tag, and the second zero field is the routing domain. The zero routing entries in RIPv1 packet format are replaced with subnet mask and next hop router information in RIPv2. RIPv1 does not support subnet mask information in routing table entries. When RIPv1 was created, the subnet mask mechanism did not exist.

## The AFI Field

The Address Family Identifier (AFI) field specifies the address family that is represented by the Internetwork Address field. Although the RFC 1058 RIP was created by the IETF, which would imply the use of the Internet Protocol (IP), it was explicitly designed to provide backward compatibility with previous versions of RIP. This meant it had to provide for the transport of routing information of a wide variety of internetworking address architectures or families. Consequently, the open standard RIP needed a mechanism for determining which type of address was being carried in its packets. In the case of RIP, an AFI field value of 2 is used.

## The Internetwork Address Field

The four-octet Internetwork Address field contains an internetwork address. This address can be a host, a network, or even a default gateway address code. Two examples of how this field's contents can vary are:

- In a single-entry request packet, this field would contain the address of the packet's originator.
- In a multiple-entry response packet, these fields would contain the IP addresses stored in the originator's routing table.

## The Metric Field

The last field in the RIP packet, the Metric field, contains the packet's metric counter. This value is incremented as it passes through a router. The valid range of metrics for this field is between 1 and 15. The metric can actually be incremented to 16, but this value is associated with invalid routes. Consequently, 16 is an error value for the Metric field, and not part of the valid range. Thus, a metric value of 16 means that the destination is unreachable.

# The RIP Routing Table

RIP hosts communicate  described in the previous section. This information is stored in a routing table. The routing table contains one entry for each known, reachable destination. The one entry per destination is the lowest cost route to that destination.

**Note**

The number of entries stored per destination may vary by router vendor. Vendors may elect to follow the published specifications, or "enhance" those specifications as they see fit. Consequently, it is quite likely you will find particular brands of routers that store up to four routes of equal cost for any given destination in the network.

Each routing table entry contains the following fields:

- The Destination IP Address field
- The distance-vector Metric field
- The Next Hop IP Address field
- The Route Change Flag field
- Route timers

**Note**

Although RFC 1058 RIP is an open standard that can support a variety of internetwork address architectures, it was designed by the IETF for use in autonomous systems within the Internet. As such, it was tacitly assumed that IP would be the internetworking protocol used by this form of RIP.

## The Destination IP Address Field

The most important piece of information contained in any routing table is the IP address of the known destinations. Whenever a RIP router receives a data packet, it looks up its destination IP address in its routing table to determine where to forward that packet.

## The Metric Field

The metric contained in the routing table represents the total cost of moving a datagram from its point of origin to its specified destination. The Metric field in the routing table contains the sum total of the costs associated with the network links that comprise the end-to-end network path between the router and the specified destination.

## The Next Hop IP Address Field

The Next Hop IP Address field contains the IP address of the next router interface in the network path to the destination IP address. This field is only populated in a router's table if the destination IP address is on a network that is not directly connected to that router.

## The Route Change Flag Field

The Route Change Flag field is used to indicate whether the route to the destination IP address has changed recently. This field was deemed important because RIP records only one route per destination IP address.

## Route Timers

The two timers that are associated with each route are the route time-out and the route-flush timers. These timers work together to maintain the validity of each route stored in the routing table. The routing table maintenance process is described in more detail in the section "Updating the Routing Table," later in this chapter.

# Operational Mechanics

As explained in Chapter 13, "Routing in IP Networks," routers that use a distance-vector routing protocol must periodically pass copies of their routing tables to their immediate network neighbors. A router's routing table contains information about the distance between itself and known destinations. These destinations can be individual host computers, printers, or other networks.

Each recipient adds a distance vector; that is, its own distance "value," to the table and forwards the modified table to its immediate neighbors. This process occurs omnidirectionally between immediately neighboring routers. Figure 15.3 uses a simple RIP internetwork to illustrate the concept of immediate neighbors.

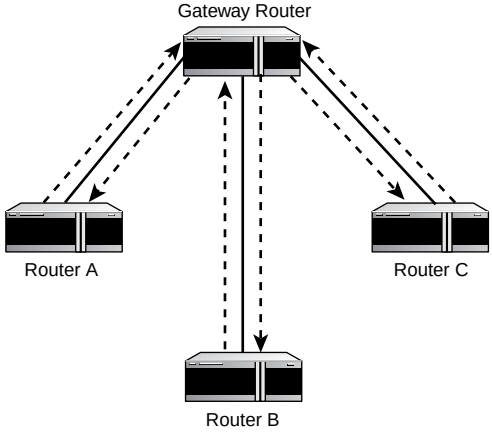



**FIGURE 15.3**
*Each RIP node advertises the contents of its routing table to its immediate neighbors.*

Figure 15.3 shows four routers. The gateway router is interconnected with each of the other three. It must exchange its routing information with these routers. Routers A, B, and C have only one connection each: to the gateway. Consequently, they can only exchange their information with the gateway directly. They can learn about each other's hosts through the information shared with the gateway. Table 15.1 shows the abbreviated contents of each of the three routers' routing tables. This information is shared with the gateway router.

**TABLE 15.1** Routing Table Contents

| *Router Name* | *Hostname* | *Next Hop* |
|---|---|---|
| A | 192.168.130.10 | Local |
| | 192.168.130.15 | Local |
| B | 192.168.125.2 | Local |
| | 192.168.125.9 | Local |
| C | 192.68.254.5 | Local |
| | 192.68.254.20 | Local |

The gateway router uses this information to build its own routing table. The abbreviated contents of this table are presented in Table 15.2.

**TABLE 15.2** Gateway Router Routing Table Contents

| *Hostname* | *Next Hop* | *Number of Hops* |
|---|---|---|
| 192.168.130.10 | A | 1 |
| 192.168.130.15 | A | 1 |
| 192.168.125.2 | B | 1 |
| 192.168.125.9 | B | 1 |
| 192.68.254.5 | C | 1 |
| 192.68.254.20 | C | 1 |

The routing information in Table 15.2 is then shared via routing information update packets with each of the other routers in the network. These routers use this information to round out their own routing tables. Table 15.3 shows the abbreviated contents of Router A's routing table after it has shared routing information with the gateway router.

**TABLE 15.3** Router A's Routing Table Contents

| *Hostname* | *Next Hop* | *Number of Hops* |
|---|---|---|
| 192.168.130.10 | Local | 0 |
| 192.168.130.15 | Local | 0 |
| 192.168.125.2 | Gateway | 2 |
| 192.168.125.9 | Gateway | 2 |
| 192.68.254.5 | Gateway | 2 |
| 192.68.254.20 | Gateway | 2 |

Router A knows that the gateway router is one hop away. Thus, seeing that the 192.168.125.x and 192.68.254.x hosts are also one hop away from the gateway, it adds the two numbers together, for a total of two hops to each machine.

This highly simplified step-by-step process results in each router learning about other routers and developing a cumulative perspective of the network as well as the distances between source and destination devices.

## Calculating Distance Vectors

A distance-vector routing protocol uses metrics to keep track of the distance separating it from all known destinations. This distance information enables the router to identify the most efficient next hop to a destination that resides within the network.

In RFC 1058 RIP, there is a single distance-vector metric: hop count. The default hop metric in RIP is set to 1. Thus, for each router that receives and forwards a packet, the hop count in the RIP packet Metric field is incremented by one. These distance metrics are used to construct a routing table. The routing table identifies the next hop for a packet to take to get to its destination at a minimal cost.

The earlier, proprietary RIP-like routing protocols typically used 1 as the only supported cost per hop. RFC 1058 RIP preserved this convention as a default hop-count value, but provisions were made for the router's administrator to select higher cost values. Such values would be beneficial in discriminating between links of differing performance capabilities. These capabilities could be the bandwidths available on different network links (that is, 56Kbps versus T1 private lines) or even the performance difference between a new router versus an older model.

Typically, a cost of 1 is assigned to each of a router's ports that connect to other networks. This is, apparently, an artifact from RIP's pre-RFC 1058 days, when the cost per hop

defaulted to 1 and was not modifiable. In a relatively small network that consisted of homogeneous transmission technologies, setting all ports to a cost of 1 would be reasonable. This is illustrated in Figure 15.4.

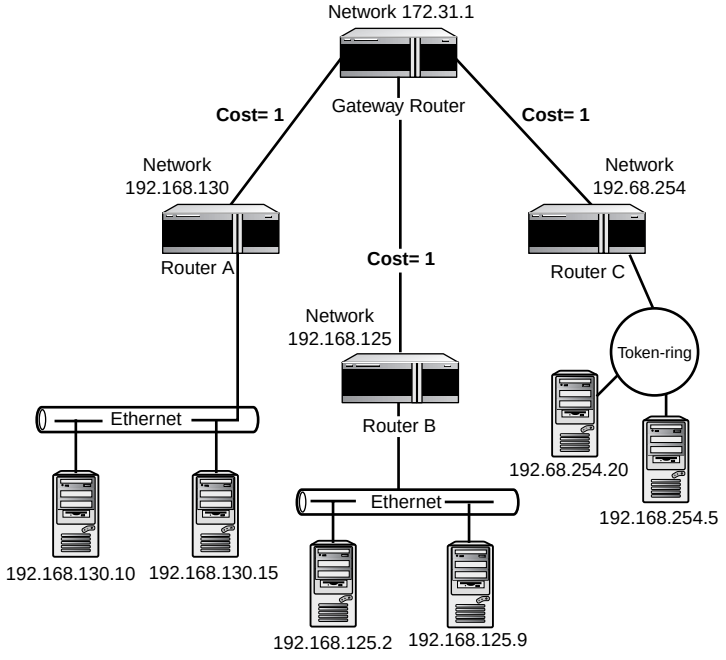



**FIGURE 15.4**
*A homogeneous network with equivalent costs.*

A router's administrator can change the default metric. For example, an administrator may increase the metric for slower-speed links to other routers. Although this might more accurately represent the costs or distances to a given destination, this practice is not recommended. Setting the metric to a value greater than 1 makes it correspondingly easier to reach the packet's maximum hop count of 16! Figure 15.5 demonstrates how quickly routes cane become invalid if route metrics are increased.

Figure 15.5 presents a slightly modified version of the WAN depicted in Figure 15.4. This illustration adds low-bandwidth redundant links to the topology depicted in Figure 15.4. The network administrator, to ensure that the alternate routes remained alternate routes, set the metric value of these alternate routes to 10. These higher costs preserve the bias toward the higher-bandwidth T1 transmission facilities. In the event of a failure of one of those T1 lines, the internetwork can continue to function normally, although some degraded performance levels might occur due to the lower available bandwidth on the 56 Kbps back-up facility. Figure 15.6 illustrates how the internetwork will react to a failure of a T1 line between the gateway and Router A.

**FIGURE 15.5**
*Hop counts are modified to differentiate between primary and alternate routes.*

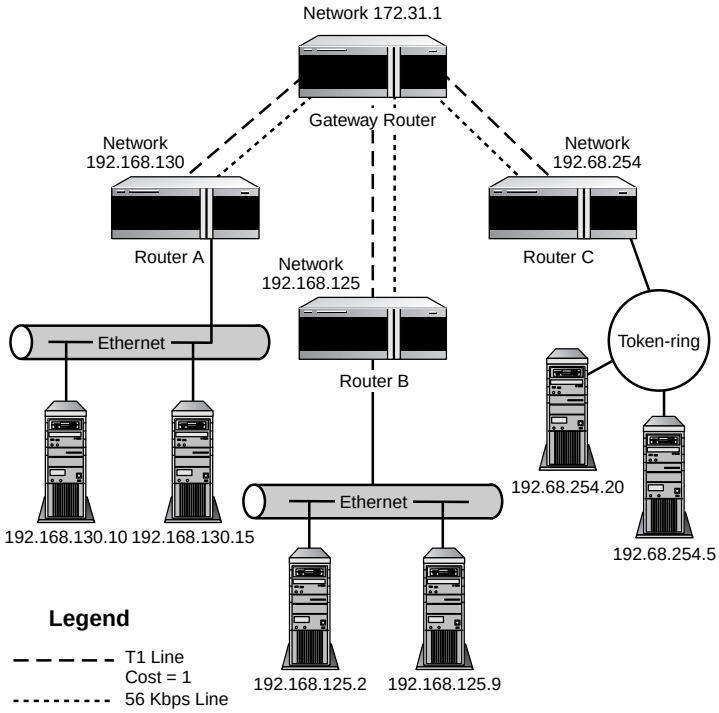
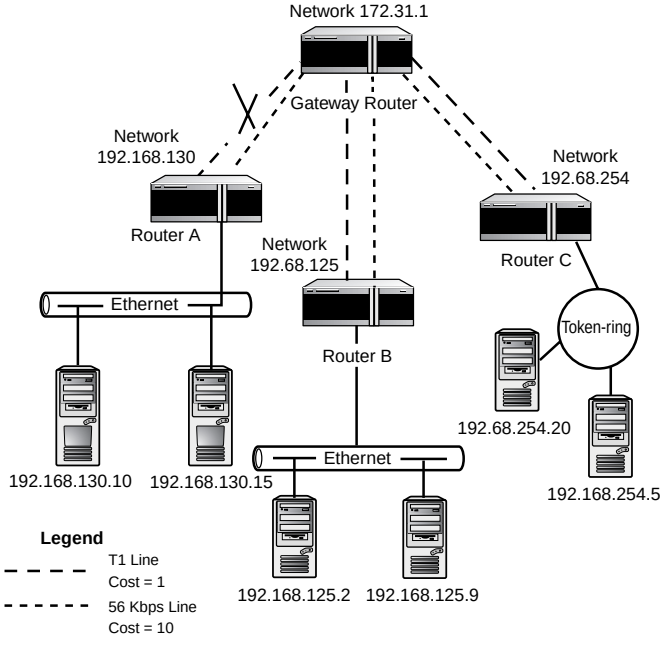



**FIGURE 15.6**
*Hop counts add up quickly, but the network remains functional.*



The alternate 56 Kbps transmission facility becomes the only way for Router A and the rest of the network to communicate. Router A's routing table, after the network converges upon a common understanding of this new topology, is summarized in Table 15.4.

**TABLE 15.4** Router A's Routing Table Contents with a Link Failure

| *Hostname* | *Next Hop* | *Number of Hops* |
|---|---|---|
| 192.168.130.10 | Local | 0 |
| 192.168.130.15 | Local | 0 |
| 192.168.125.2 | Gateway | 11 |
| 192.168.125.9 | Gateway | 11 |
| 192.68.254.5 | Gateway | 11 |
| 192.68.254.20 | Gateway | 11 |

Although a higher route cost is a more accurate reflection of the lower bandwidths offered by these alternate routes, it can introduce unwanted routing problems. In Figure 15.7, two of the T1 lines have failed, thus causing two of the alternate routes to become active simultaneously.

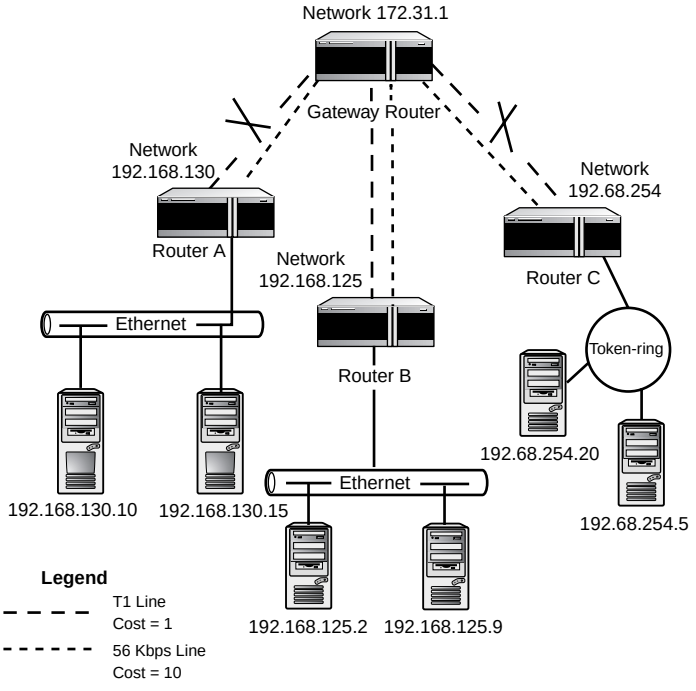



**FIGURE 15.7**
*Hop counts can add up to 16 too fast.*

Because both alternate links had a cost metric of 10, their simultaneous activation results in a route cost of greater than 16. The valid range for RIP's hop counter is from 0 through 16, with 16 representing an unreachable route. Thus, if the metrics (or cost) of a route exceed 16, the route is declared invalid, and a notification (a triggered update) is sent to all immediately neighboring routers.

Obviously, this problem can be avoided by leaving the default cost equal to 1. If it is absolutely necessary to increment the cost metric of a given hop, the new cost value should be selected with great care. The sum total of the route between any given pair of source and destination addresses in a network should never exceed 15. Table 15.5 demonstrates the impacts of a second link failure on Router A's routing table.

**TABLE 15.5** Router A's Routing Table Contents with Two Link Failures

| *Hostname* | *Next Hop* | *Number of Hops* |
|---|---|---|
| 192.168.130.10 | Local | 0 |
| 192.168.130.15 | Local | 0 |
| 192.168.125.2 | Gateway | 11 |
| 192.168.125.9 | Gateway | 11 |
| 192.68.254.5 | Gateway | 16 |
| 192.68.254.20 | Gateway | 16 |

As is evident in Table 15.5, the cost of the route between A and C exceeds 16, and all entries are declared invalid. Router A is still able to communicate with B because the total cost of that route is only 11.

## Updating the Routing Table

The fact that RIP records only one route per destination requires RIP to aggressively maintain the integrity of its routing table. It does so by requiring all active RIP routers to broadcast their routing table contents to neighboring RIP routers at a fixed interval. All updates received automatically supercede previous route information that was stored in the routing table.

RIP relies on three timers to maintain the routing table:

- The update timer
- The route time-out timer
- The route-flush timer

The update timer is used to initiate routing table updates at the node level. Each RIP node only uses one update timer. Conversely, the route time-out timer and the route-flush timer are kept for each route.

As such, separate time-out and route-flush timers are integrated in each routing table entry. Together, these timers enable RIP nodes to maintain the integrity of their routes as well as to proactively recover from failures in the network by initiating activity based on the passing of time. The following sections describe the processes used to maintain the routing tables.

## Initiating Table Updates

A table update is initiated every 30 seconds. The update timer is used to track this amount of time. Upon the expiration of this time, a RIP node launches a series of packets that contains its entire routing table.

These packets are broadcast to each neighboring node. Therefore, each RIP router should receive an update from each of its neighboring RIP nodes approximately every 30 seconds.

**Note**

In larger RIP-based autonomous systems, these periodic updates can create unacceptable levels of traffic. Thus, staggering the updates, from node to node is desirable. This is done automatically by RIP; each time the update timer is reset, a small, random amount of time is added to the clock.

If such an update fails to occur as expected, it indicates a failure or error somewhere in the internetwork. The failure may be something as simple as a dropped packet that contained the update. The failure could also be something as serious as a failed router, or virtually anything in-between these two extremes. Obviously, the appropriate course of action differs greatly along this spectrum of failures. It would be unwise to invalidate a series of routes just because the update packet was lost (remember, RIP update packets use an unreliable transport protocol to minimize overheads). Thus, it is reasonable to not take corrective action based upon a single missed update. To help discriminate between magnitudes of failures and errors, RIP uses timers to identify invalid routes.

## Identifying Invalid Routes

Routes can become invalid in one of two ways:

- A route can expire.
- A router can learn of a route's unavailability from another router.

In either event, the RIP router needs to modify its routing table to reflect the unavailability of a given route.

A route can expire if a router doesn't receive an update for it within a specified amount of time. For example, the route time-out timer is usually set to 180 seconds. This clock is initialized when the route becomes active or is updated.

180 seconds is approximately enough time for a router to receive six routing table updates from its neighbors (assuming that they initiate table updates every 30 seconds). If 180 seconds elapse and the RIP router hasn't received an update on that route, the RIP router assumes that the destination IP address is no longer reachable. Consequently, the router marks that routing entry in its table invalid. This is done by setting its routing metric to 16, and by setting the *route change flag*. This information is then communicated to the router's neighbors via the periodic routing table updates.

**Note**

To a RIP node, 16 equals infinity. Thus, simply setting the cost metric to 16 in its routing table entry can invalidate a route.

Neighboring nodes that receive notification of the route's new invalid status use that information to update their own routing tables. This is the second of the two ways that routes can become invalid in a routing table.

An invalid entry remains in the routing table for a very brief amount of time, as the router determines whether it should be purged. Even though the entry remains in the table, datagrams cannot be sent to that entry's destination address: RIP cannot forward datagrams to invalid destinations.

## Purging Invalid Routes

When a router recognizes a route as invalid, it initializes a second timer: the route-flush timer. Thus, 180 seconds after the last time the time-out timer was initialized, the route-flush timer is initialized. This timer is usually set for 90 seconds.

If the route update is still not received after 270 seconds (180 second time-out timer plus the 90-second route-flush timer), the route is removed (that is, flushed) from the routing table. The timer responsible for counting down the time to route flush is known as the route-flush timer. These timers are absolutely essential to RIP's ability to recover from network failures.

**Active Versus Passive Nodes**

It is important to note that for a RIP internetwork to function properly, every gateway within the network must participate. Participation can be active or passive, but all gateways must participate. Active nodes are those that actively engage in the sharing of routing information. They receive updates from their neighbors, and they forward copies of their routing table entries to those neighboring nodes.

Passive nodes receive updates from their neighbors, and use those updates to maintain their routing table. Passive nodes, however, do not actively distribute copies of their own routing table entries.

The ability to passively maintain a routing table was a particularly useful feature in the days before hardware routers, when `routed` was a daemon that ran on Unix processors. This kept routing overheads on the Unix host to a minimum.

# Addressing Considerations

The IETF ensured that RIP was fully backward compatible with all known RIP and `routed` variants. Given that these were highly proprietary, it was necessary that the open standard RIP not dictate an address type. Thus, the field labeled Address in a RIP packet may contain

- The host address
- The subnet number
- The network number
- A 0, which indicates a default route

Implicit in this flexibility is the fact that RIP permits calculating routes to either individual hosts, or to networks that contain numerous hosts. To accommodate this address flexibility in operation, RIP nodes use the most specific information available when they forward datagrams. For example, when a RIP router receives an IP packet, it must examine the destination address. It attempts to match this address with a destination IP address in its routing table. If it cannot find an entry for that destination host address, it then checks whether that destination address matches a known subnet or network number. If it cannot make a match at this level, the RIP router uses its default route to forward the datagram.

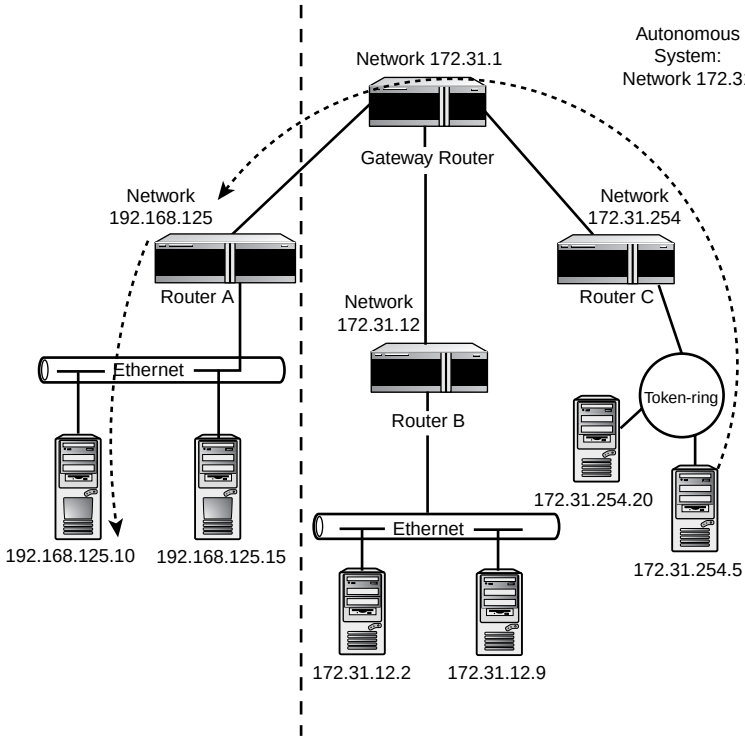
## Routing to a Gateway

Up to this point in the chapter, entries in the RIP routing table have been assumed to be routes to individual hosts. This was a simplifying assumption that better describes the

way that routing originally worked. Today, networks may be too large and well populated with hosts for host routing to be practical. Host-based routing unnecessarily inflates the size of routing tables, and slows down routing across the internetwork.

In real-world networks, routes are calculated to network addresses rather than host addresses. For example, if each host on any given network (or subnet) is accessible through the same gateway(s), the routing table can simply define that gateway as a destination IP address. All datagrams addressed to hosts within that network or subnetwork will be forwarded to that gateway. That gateway, then, will assume responsibility for forwarding them on to their ultimate destinations. Figure 15.8 illustrates this point; it preserves the topology of the previous few illustrations, but uses more conventional IP addresses.

**FIGURE 15.8**
*RIP nodes can deliver datagrams to gateways.*




In Figure 15.8, host 172.31.254.5 needs to transmit an IP packet to host number 192.168.125.10. This address is unknown to Router C. The router checks to see the subnet mask, which is set to 255.255.255.0. From this, it is easy to deduce that 192.168.125 is a subnet number. More importantly, Router C knows a route to that subnet. Router B

assumes that the gateway router at that subnet knows how to reach that host. Consequently, Router C forwards the packet to that gateway. This approach requires hosts to be known only to the router that is closest, and not known throughout a network. The finely dotted lines in Figure 15.8 illustrate the two parts of the IP datagram's journey: its trip from Router B to the Router A, and from A to host 192.168.125.10.

**Note**

RIPv1 does not support variable length subnet masks (VLSM). Consequently, there can be only one subnet mask for each network. It is quite likely that a network may contain multiple subnets, each with its own subnet address that uses the same mask length. RIP is also known as a "classful" routing protocol because it supports only the class-based IPv4 addresses. RIPv2 can support VLSM as well as CIDR (Classless Internet Domain Routing). CIDR was discussed in earlier chapters and is a mechanism for aggregating route table entries to a block of addresses by a single routing entry.

## Routing Between Gateways

In the case presented in the previous section, a potential routing problem exists. If Router C did not know the subnet mask of the destination IP address, and the host part of the address was not zero, it wouldn't be able to determine whether the address was a subnet number or a host address. Thus, the packet would be discarded as undeliverable.

To help avoid the ambiguity, routes to a subnet are not advertised outside the network that contains the subnet. The router at the border of this subnet functions as a gateway; it treats each subnet as an individual network. RIP updates are performed between immediate neighbors within each subnet, but the network gateway advertises a single network number only to its neighboring gateways in other networks.

The practical implication of this is that a border gateway will send different information to its neighbors. Immediate neighbors within the subnetted network will receive updates containing lists of all subnets directly connected to that gateway. The routing entries will list the number of each subnet.

Immediate neighbors outside the network will receive a single routing entry update that encompasses all the hosts on all the subnets contained within that network. The metric passed would be that associated with reaching the network, rather than including the costs of hops within the network. In this fashion, distant RIP routers assume that datagrams addressed to any host number within that subnet are known to, and reachable through, the network's border gateway router.

## Default Routes

The IP address 0.0.0.0 is used to describe a default route. Much like the way subnets can be summarized by routing to a network gateway, a default route can be used to route to multiple networks without explicitly defining and describing them. The only requirement is that there be at least one gateway between these networks that is prepared to handle the traffic generated.

To create a default route, a RIP entry needs to be created for the address 0.0.0.0. This special address is treated just like any other destination IP address. The next hop should be the destination IP address of the neighboring gateway router. This routing entry is used just like every other entry, with one important exception: The default route is used to route any datagram whose destination address doesn't match any other entries in the routing table.

Table 15.6 demonstrates the abbreviated contents of Router A's routing table with a default route. In this table, the two local hosts are the only ones explicitly identified. Any other locally generated transmission requests are automatically forwarded to the gateway router.

**TABLE 15.6** Router A's Routing Table Contents with a Default Route

| *Hostname* | *Next Hop* |
|---|---|
| 192.168.125.10 | Local |
| 192.168.125.15 | Local |
| 0.0.0.0 | Gateway |

# Topology Changes

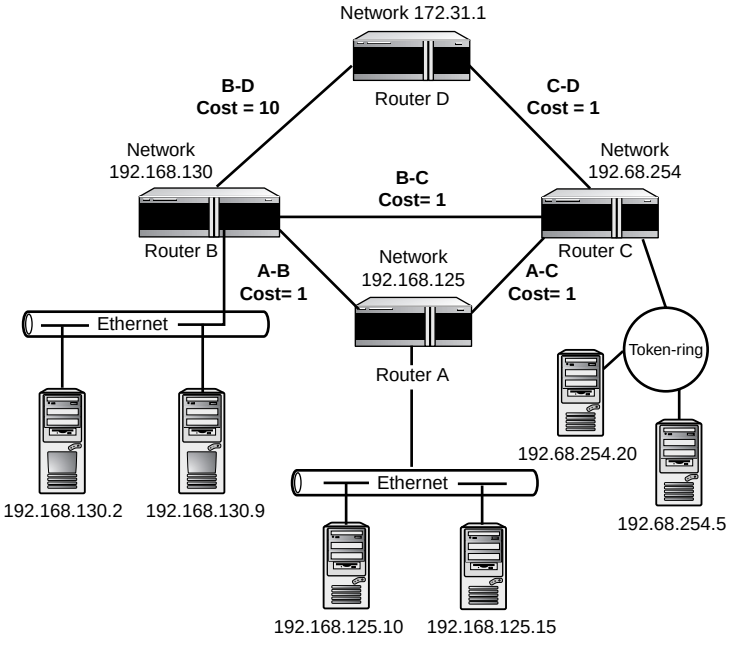
Up to this point, RIP's fundamental mechanisms and specifications have been examined in a rather static fashion. A deeper appreciation for RIP's mechanics, however, can be gained by looking at how these mechanisms interact to accommodate changes in network topology.

## Convergence

The most significant implication of a topology change in a RIP internetwork is that it changes the solution set of neighboring nodes. This change may also result in different results the next time the distance vectors are calculated. Thus, the new sets of neighboring nodes must then converge, from different starting points, on a consensus of what the new topology looks like. This process of developing a consensual perspective of the topology is known as *convergence*. In simple terms, the routers develop an agreement of what the network looks like *separately,* together.

Figure 15.9 illustrates convergence; it demonstrates two possible routes to Router D from Router A and Network 192.168.125. Router D is a gateway router. The primary route to Router D's network is via Router C. If this route were to fail, it would take some time for all the routers to converge upon a new topology that didn't include the link between Routers C and D.

**FIGURE 15.9**
*Two possible paths to Router D from Router A.*




As soon as the C–D link fails, it is no longer usable, but it may take quite a while for this fact to become known throughout the network. The first step in convergence is for D to realize that the link to C has failed. This assumes that Router D's update timer elapses before Router C's timer. Because this link was the one that should have carried updates from Router D to Router C, no updates can be received. Consequently, C (as well as A and B) is still unaware that the C–D link has failed. All routers in the internetwork will continue to forward datagrams addressed to Router D's network number through that link. This first stage in convergence is illustrated in Figure 15.10.

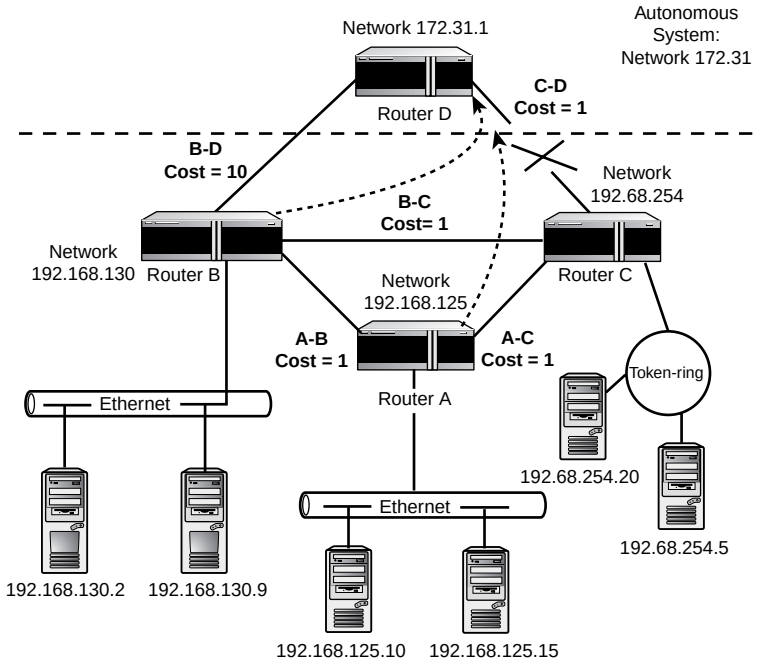



**FIGURE 15.10**
*Only Router D is aware of the link failure.*

Upon expiration of its update timer, Router D will attempt to notify its neighbors of its perception of the change in the network's topology. The only immediate neighbor that it will be able to contact is B. Upon receiving this update, B will update its routing table to set the route from B to D (via C) to infinity. This will allow it to resume communications with D, albeit via the B–D link. After B has updated its table, it can advertise its new-found perception of the topology to its other neighbors—A and C.

**Note**

Remember, a RIP node invalidates a route by setting its metric to 16—the RIP equivalent of infinity.

As soon as A and C have received updates, and have recalculated network costs, they can replace their obsolete entries that used the C–D link with the B–D link. The B–D route was previously rejected by all nodes, including B, as being more expensive than the C–D link. Its cost metric of 10 compared unfavorably with the C–D cost of 1 for each node. Now, with the failure of the C–D link, the B–D link features the lowest cost. Thus, this new route replaces the timed-out route in the neighbors' routing tables.

When all routers agree that the most efficient route to D is via B, they have converged. This is illustrated in Figure 15.11.

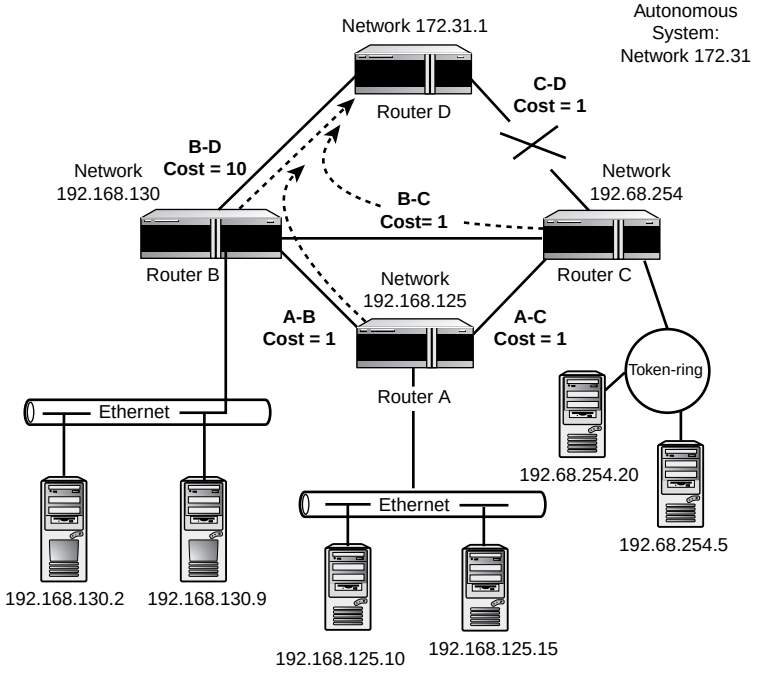



**FIGURE 15.11**
*The routers converge on B–D as the new route.*

The amount of time that will elapse before convergence completes is not easy to determine. It will vary greatly from network to network, based upon a wide variety of factors that include the robustness of the routers and transmission facilities, amount of traffic, and so on.

# The "Count-to-Infinity" Problem

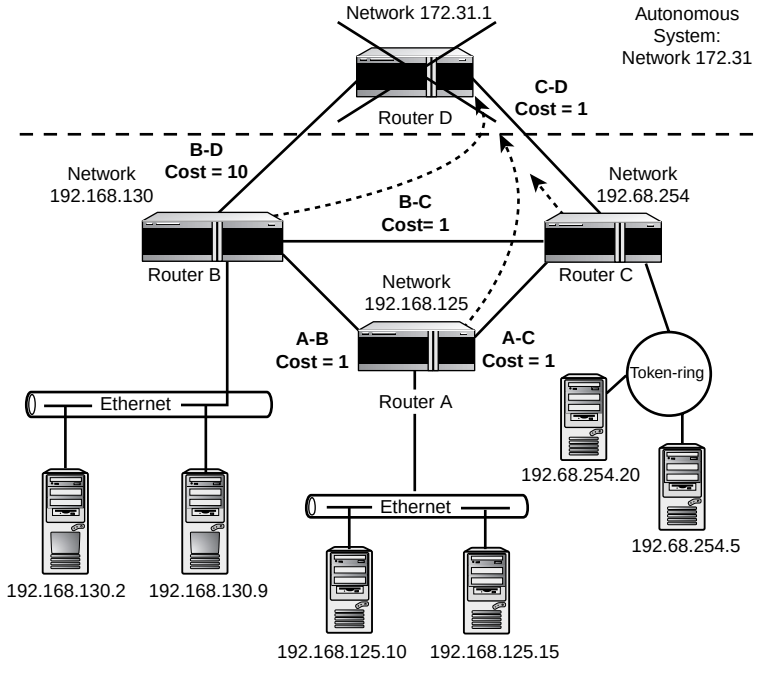
In the example presented in the preceding section, the only failure was transmission facility connecting C and D. The routers were able to converge on a new topology that restored access to gateway Router D's network via an alternate path. A much more disastrous failure would have been if D itself had failed. The convergence process in the previous example started when D was able to notify B of the link failure. If D, rather than its link to C, had failed, neither B nor C would receive an update informing them of the change in topology.

Converging on a new topology given this type of failure can result in a phenomenon known as *counting to infinity* or the *count to infinity problem.* When a network becomes

completely inaccessible, updates between remaining routers can steadily increment routing metrics to the inaccessible destination based on the mistaken belief that another router can access the lost destination. Left unchecked, the routers in that scenario will literally count to RIP's interpretation of infinity.

To illustrate the dangers from a routing perspective, inherent in this type of catastrophic failure, reconsider the topology presented in the convergence illustrations. In Figure 15.12, Router D has failed.

**FIGURE 15.12**
*Router D has failed.*




With the failure of Router D, all the hosts within its network are no longer accessible from the outside. Router C, after missing six consecutive updates from D, will invalidate its C–D route and advertise its unavailability. This is illustrated in Figure 15.13. Routers A and B will remain ignorant of the route's failure until notified by C.

At this point, both A and C believe that they can get to D via B. They recalculate their routes to include the higher costs of this detour. This is illustrated in Figure 15.14.

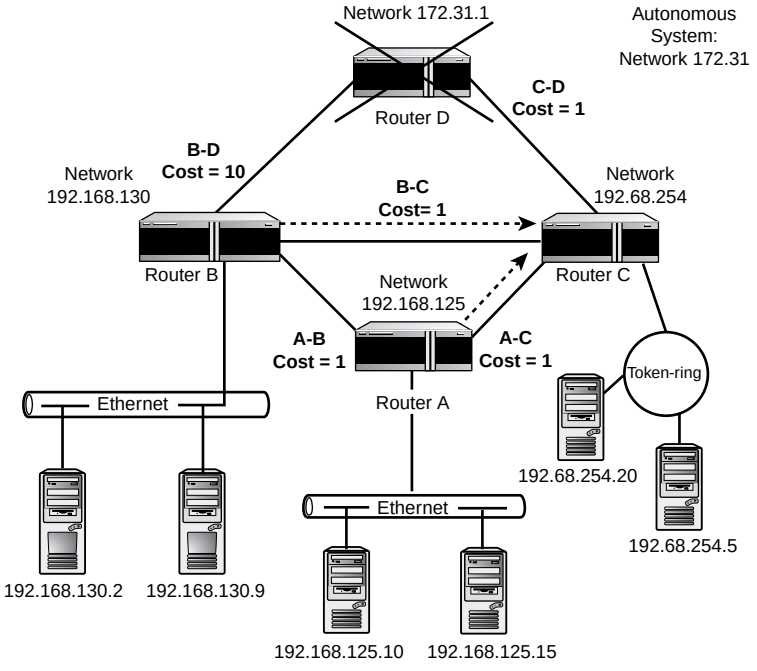



**FIGURE 15.13**
*Router C invalidates its C–D route.*

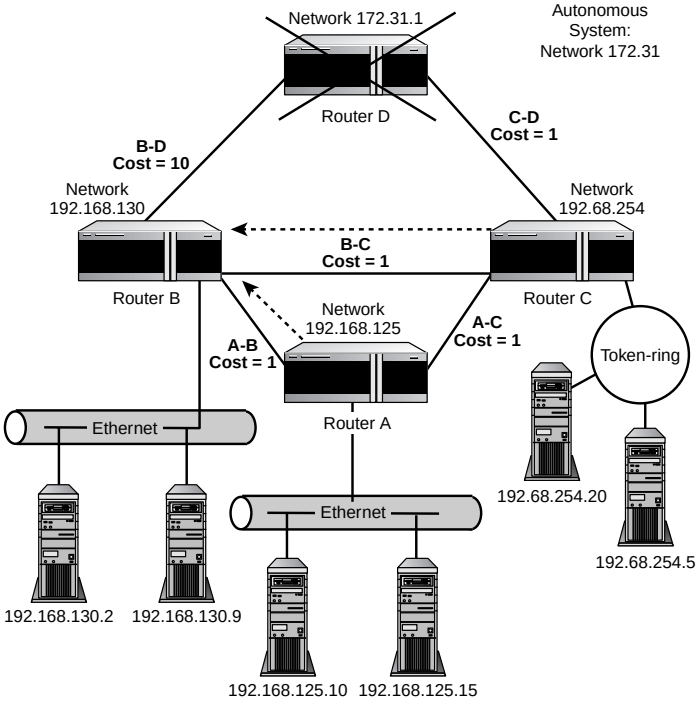



**FIGURE 15.14**
*A and C believe that they can access D through B.*

These routers send their next updates to B, an immediate neighbor of both routers. Router B, having timed-out its own route to D, believes it can still access D through either A or C. Obviously, it cannot because those routers are relying on the link that B just invalidated. In essence, a loop is formed between A, B, and C that is fed by the mistaken belief that both A and C can still reach the unreachable Router D through each other. This is because both have a connection to B, which has the connection to D.

With each iteration of updates, the cost metrics are incremented to account for the next extra hop that is added to the loop already calculated. This form of looping is induced by the time delay that characterizes independent convergence through neighbor-transmitted updates.

In theory, the nodes will eventually realize that D is unreachable. However, it is virtually impossible to tell how much time would be required to achieve this convergence. This example illustrates precisely why RIP's interpretation of infinity is set so low. Whenever a network becomes inaccessible, the incrementing of metrics through routine updates must be halted as soon as practical. Unfortunately, this means placing an upper limit on how high the nodes will count before declaring a destination unreachable. Any upper limit directly translates into a limitation on the maximum size of the routed network's diameter. In the case of RIP, its original designers felt that 15 hops were more than adequate for an autonomous system. Systems larger than this could utilize a more sophisticated routing protocol.

RIP supports three means of avoiding the count-to-infinity loop problem:

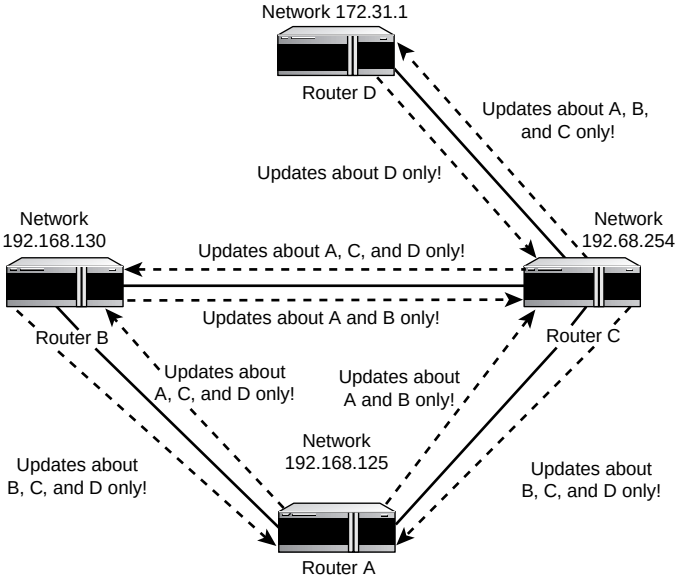
- Split horizon
- Split horizon with poisoned reverse
- Triggered updates

## Split Horizon

It should be fairly obvious that the looping problem described in the preceding section could be prevented with the application of logic. The term used to describe this logic is *split horizon*. Although RIP does not support pure split horizon, understanding it will facilitate understanding its somewhat more complicated variant, *split horizon with poisoned reverse*.

The essence of split horizon is the assumption that a RIP node won't advertise an update to a particular route to a particular neighbor, if that route was originally learned from that neighbor. Figure 15.15 illustrates this point.

**FIGURE 15.15**
*A split horizon.*



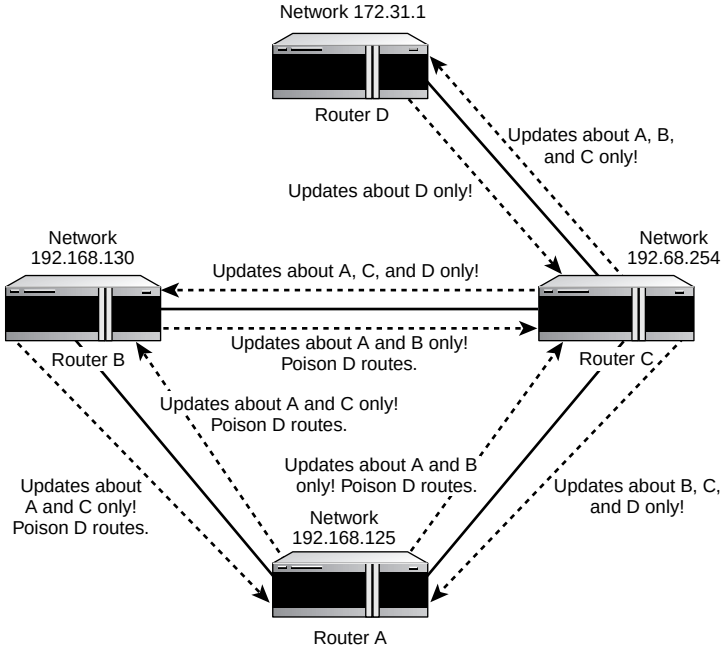

In Figure 15.15, the routers support the split horizon logic. Thus, Router C (which supports the only path to Router D) cannot receive updates from Router A about Network D. This is because A relies on C (and even B) for this route information. Router A must omit from its routing table information about routes learned from C. This simple approach to splitting loops can be relatively effective, but it does have a serious functional limitation: By omitting reverse routes from advertising, each node must wait for the route to the unreachable destination to time-out.

In RIP, a time-out occurs only after six update messages fail to update a route. Thus, five opportunities exist for a misinformed node to misinform other nodes about an unreachable destination. It is this time delay that creates the opportunity for invalid routing information to start the loop. Due to this limitation, RIP supports a slightly modified version known as split horizon with poisoned reverse.

## Split Horizon with Poisoned Reverse

The simple split horizon scheme attempts to control loops by ceasing to propagate information back to its originator. Although this can be effective, there are more effective ways to stop a loop. Split horizon with poisoned reverse takes a much more proactive approach to stopping loops: This technique actually poisons the looped route by setting its metric to infinity. This is illustrated in Figure 15.16.

**FIGURE 15.16**
*A split horizon with poisoned reverse.*




As illustrated in Figure 15.16, Router A can provide information to Router B about how to reach Router D, but this route carries a metric of 16. Thus, Router B cannot update its routing table with information about a better way to reach the destination. In fact, A is advertising that it cannot reach D, which is a true statement. This form of route advertising effectively breaks loops immediately.

Generally speaking, split horizon with poisoned reverse is much safer in distance-vector networks than just split horizon. However, neither is perfect. Split horizon with poisoned reverse will effectively prevent routing loops in topologies with just two gateways. In larger internetworks, however, RIP is still subject to the counting-to-infinity problem. To ensure that such infinite loops are caught as early as possible, RIP supports a triggered update.

## Triggered Updates

Networks that feature three gateways to a common network are still susceptible to loops caused by mutual deception of the gateways. Figure 15.17 illustrates this point. This diagram features three gateways to Router D: A, B, and C.

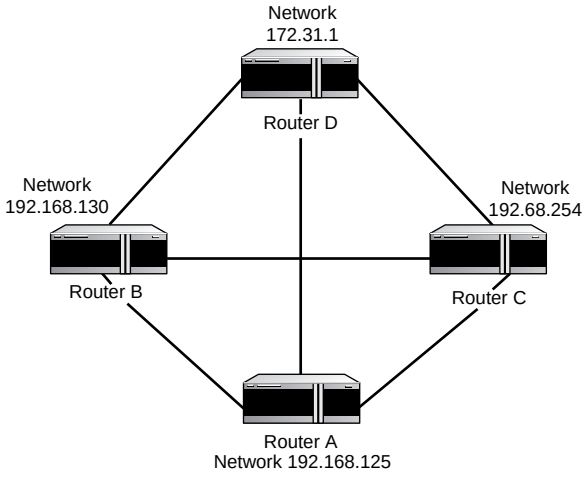



**FIGURE 15.17**
*Three gateways to D.*

In the event that Router D fails, Router A may believe that B can still access D. Router B may believe that C can still access D, and C may believe that A can still access D. The net effect is a continuous loop to infinity. This is illustrated in Figure 15.18.

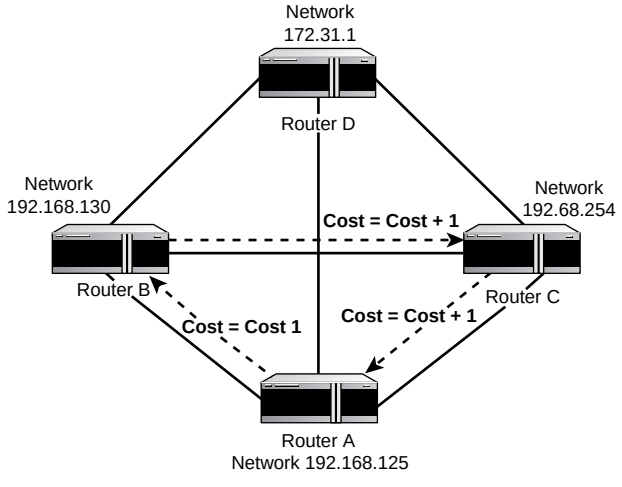



**FIGURE 15.18**
*Counting to infinity with three gateways.*

Split horizon logic would be ineffective in this scenario due to the time delay before routes can be invalidated. RIP uses a different technique, known as a *triggered update*, to accelerate convergence. A triggered update is a rule in the protocol that requires gateways to immediately broadcast an update message whenever it changes a route metric, regardless of how much time remains in the 30-second update timer.

The previous sections demonstrate how time is the Achilles' heel of split horizons, with or without reverse poisoning. Triggered updates are designed to overcome this vulnerability by reducing time delay to an absolute minimum.

## Hold-Down Timers

Triggered updates are not a panacea! Updates are not propagated instantaneously throughout a network. Thus, it is possible (however unlikely) that another gateway could have just transmitted a periodic update before receiving a triggered update from another gateway. In this scenario, vestiges of an invalid route could re-propagate throughout the network. Although the likelihood of this occurring is extremely low, it is still possible for counting-to-infinity loops to occur within a RIP network despite the use of triggered updates.

The solution to this potential problem is the use of a hold-down timer. A hold-down timer works in conjunction with the triggered update logic. In essence, as soon as a triggered update has been made, a clock starts counting down to zero. Until it decrements to zero, the router will not accept any updates from any neighbors for that route or destination.

This prevents a RIP router from accepting updates for a route that has been invalidated for a configurable amount of time. This prevents a router from being misled into believing that another router may have a viable route to an otherwise invalid destination.

# Limitations of RIP

Despite its lengthy heritage, RIP is not without its limitations. It was marvelously suited to calculating routes during the early days of internetworking; however, technological advance has radically changed the way that internetworks are built and used. Consequently, RIP is rapidly approaching obsolescence in today's internetwork.

Some of RIP's greatest limitations include the following:

- Inability to support paths longer than 15 hops
- Reliance upon fixed metrics to calculate routes
- Network intensity of table updates
- Relatively slow convergence
- Lack of support for dynamic load balancing

## Hop Count Limit

RIP was designed for use in relatively small autonomous systems. As such, it enforces a strict hop count limit of 15 hops. As packets are forwarded by a routing device, their hop counters are incremented by the cost of the link that it is being transmitted over. If the

hop counter hits 15, and the packet isn't at its addressed destination, that destination is considered unreachable, and the packet is discarded.

This effectively fixes maximum network diameter at 15 hops. This is sufficiently high to build a fairly large network, depending on how cleverly it was designed, but is still severely limited in comparison to the scalability of other, more modern, routing protocols. Thus, if you have anything but a very small network, RIP probably isn't the right routing protocol to use.

## Fixed Metrics

The discussion about hop counts nicely sets the stage for an examination of RIP's next fundamental limitation: its fixed-cost metrics. Although cost metrics can be configured by the administrator, they are static in nature. RIP cannot update them in real-time to accommodate changes it encounters in the network. The cost metrics defined by the administrator remain fixed, until updated manually.

This means that RIP is particularly unsuited for highly dynamic networks, where route calculations must be made in real-time in response to changes in the network's condition. For example, if a network supports time-sensitive applications, it is reasonable to use a routing protocol that can calculate routes based on the measured delay of its transmission facilities or even the existing load on a given facility. RIP uses fixed metrics. Thus, it is not capable of supporting real-time route calculation.

## Network Intensity of Table Updates

A RIP node broadcasts its routing tables omnidirectionally every 30 seconds. In large networks with many nodes, this can consume a fair amount of bandwidth.

## Slow Convergence

In human terms, waiting 30 seconds for an update is hardly inconvenient. Routers and computers, however, operate at much higher speeds than humans do. Thus, having to wait 30 seconds for an update can have demonstrably adverse effects. This point is demonstrated in the section "Topology Changes" earlier in this chapter.

Much more damaging than merely waiting 30 seconds for an update, however, is having to wait up to 180 seconds to invalidate a route. And this is just the amount of time needed for just one router to begin convergence. Depending on how many routers are internetworked, and their topology, it may take repeated updates to completely converge on a new topology. The slowness with which RIP routers converge creates a wealth of opportunities for vestiges of invalid routes to be falsely advertised as still available. Obviously, this compromises the performance of the network, both in the aggregate and in the perception of individual users.

This chapter should have amply demonstrated the dangers inherent in RIP's slow convergence.

## Lack of Load Balancing

Another of RIP's significant limitations is its inability to dynamically load balance. Figure 15.19 illustrates a router with two serial connections to another router in its internetwork. Ideally, the router in this illustration would split the traffic as evenly as possible between the two serial connections. This would keep congestion to a minimum on both links and optimize performance.

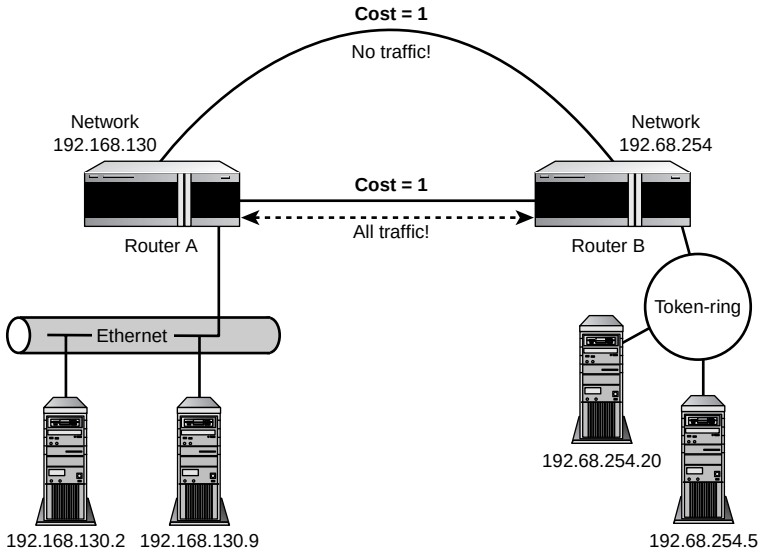



**FIGURE 15.19**
*A router with redundant serial connections.*

Unfortunately, RIP is unable to perform such dynamic load balancing. It would use whichever of the two physical connections that it knew about first. RIP would forward all of its traffic over that connection even though the second connection was available for use. The only way that this scenario would change would be if the router in Figure 15.19 received a routing update informing it of a change in the metrics to any given destination. If this update meant that the second link was the lowest-cost path to a destination, it would begin using that link and cease using the first link.

RIP's inherent lack of load-balancing capability reinforces its intended use in simple networks. Simple networks, by their very nature, tend to have few (if any) redundant routes. Consequently, load balancing was not perceived as a design requirement, and support for it was not developed.

# Summary

RIP's ease of configuration, flexibility, and ease-of-use have made it a highly successful routing protocol. Since its development, there have been tremendous advances in computing, networking, and internetworking technologies. The cumulative effects of these advances have taken their toll on RIP's popularity. In fact, many other routing protocols in use today are technically superior to RIP. Despite the success of these protocols, RIP remains a highly useful routing protocol, provided you understand the practical implications of its limitations, and use it accordingly. The improvements in the newer version of RIP, called RIPv2, was also discussed.

CHAPTER 16

# Open Shortest Path First (OSPF)

*by Mark A. Sportack*

## IN THIS CHAPTER

As the 1980s drew to a close, the fundamental limitations of distance-vector routing were becoming increasingly apparent. One attempt to improve the scalability of networks was to base routing decisions on link states, rather than hop count or other distance vectors. A *link* is the connection between two routers in a network. The status of that link can include such attributes as its transmission speed and delay levels.

This chapter provides an in-depth look at the Internet Engineering Task Force's (IETF's) version of a link-state, interior gateway routing protocol: Open Shortest Path First (OSPF). OSPF was first specified in RFC 1131. This short-lived specification was quickly made obsolete by RFC 1247. The differences between these two OSPFs were substantial enough that the RFC 1247 OSPF was called OSPF Version 2. OSPF Version 2 continued to mature and evolve. Subsequent modifications were outlined in RFCs 1583, 2178, and 2328 (which is the current version). The current version of OSPF is version 2. Because the Internet and IP are both highly dynamic, it is likely that OSPF will continue to evolve over time to keep pace.

# The Origins of OSPF

The IETF, in response to the increased need for building larger and larger IP-based networks, formed a working group specifically to develop an open, link-state routing protocol for use in large, heterogeneous IP networks. This new routing protocol was based on the moderately successful series of proprietary, vendor-specific, Shortest Path First (SPF) routing protocols that had proliferated in the market. All SPF routing protocols, including the IETF's OSPF, were directly based on a mathematical algorithm known as the *Dijkstra Algorithm*. This algorithm enables the selection of routes based upon link states, as opposed to just distance vectors.

The OSPF routing protocol was developed by the IETF during the late 1980s. OSPF was, quite literally, an open version of the SPF class of routing protocols. The original OSPF was specified in RFC 1131. This first version (OSPF Version 1) was quickly superseded by a greatly improved version that was documented in RFC 1247. The RFC 1247 OSPF was dubbed OSPF Version 2 to explicitly denote its substantial improvements in stability and functionality. Numerous updates have been made to this version of OSPF. Each has been crafted as an open standard using the IETF as a forum. Subsequent specifications were published in RFCs 1583, 2178, and 2328.

Rather than examine the iterative development of the current, open-standard OSPF, this chapter will focus on the capabilities, features, and uses of the latest OSPF version.

# Understanding OSPF

OSPF was designed specifically as an IP routing protocol for use within autonomous systems. As such, it is incapable of transporting datagrams of other routable network protocols, such as IPX or AppleTalk. If your network must accommodate multiple routable protocols, you might want to consider a different routing protocol than OSPF.

OSPF calculates routes based on the destination IP address found in IP datagram headers, and no provisions are made for calculating routes to non-IP destinations. Additionally, the various OSPF messages are encapsulated directly in IP: No other protocols (TCP, UDP, and so on) are needed for delivery.

OSPF was also designed to quickly detect topological changes in the autonomous system, and to converge on a new topology after detecting a change. Routing decisions are based on the state of the links interconnecting the routers in the autonomous system. Each of these routers maintains an identical database that tracks link states in the network. Included in this database is the state of the router. This includes its usable interfaces, known-reachable neighbors, and link-state information.

Routing table updates, known as *link-state advertisements* (LSAs) are transmitted directly to all other neighbors within a router's area. The technical term for this update process is *flooding*, a rather unflattering term with a negative connotation that belies the actual performance characteristics of OSPF.

In practice, OSPF networks converge very quickly. All routers within the network run the same routing algorithm and transmit routing table updates directly to each other. This information is used to construct an image of the network and its links. Each router's image of the network uses a *tree* structure, with itself as the *root*. This tree, known as the *shortest-path tree*, tracks the shortest path to each destination within the autonomous system. Destinations outside the autonomous system may be acquired via border gateways to those external networks, and appear as *leaves* on the shortest-path tree structure. Link-state data cannot be maintained on such destinations and/or networks simply because they are outside the OSPF network. Thus, they cannot appear as branches in the shortest-path tree.

## OSPF Areas

One of the key reasons for the rapidity of OSPF's convergence is its use of areas. Remember, the two main goals that the IETF sought to achieve with OSPF were

- Improved network scalability
- Rapid convergence times

The key to both goals lies in compartmentalizing a network into smaller regions. These regions are known as *areas*. An area is a collection of networked end systems, routers, and transmission facilities. Each area is defined with a unique area number that is configured into each router. Router interfaces that are defined with the same area number become part of the same area. Ideally, these areas are not arbitrarily defined. Instead, the boundaries of an area should be selected to minimize the amount of traffic between different areas. Each area should reflect actual traffic patterns rather than geographic or political boundaries. Of course, this ideal is theoretical and might prove impractical in your particular environment.

The number of areas an OSPF network can support is limited by the size of its Area ID field. This field is a 32-bit binary number. Thus, the theoretical maximum number of networks is a 32-bit binary number with all of its bits equal to 1. The decimal equivalent of this number is 4,294,967,295. Obviously, the practical maximum number of areas you can support is much less than this theoretical maximum. In practice, how well the network is designed will determine the practical maximum number of areas you can support within it. Figure 16.1 illustrates a fairly simple OSPF network with just three areas, numbered 0, 1, and 2.

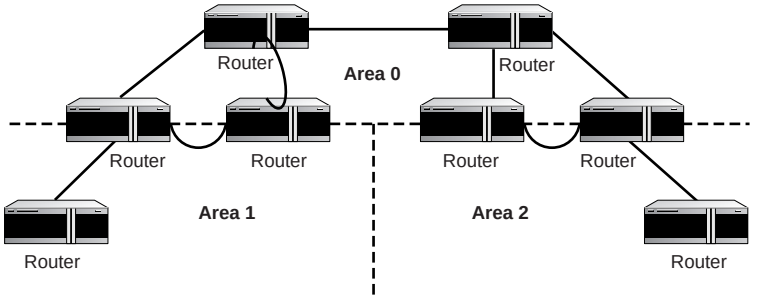



**FIGURE 16.1**
*A small OSPF network with three areas.*

## Router Types

It is important to remember that OSPF is a link-state routing protocol. Therefore, the links and the router interfaces that they attach to are defined as members of an area. Based upon area membership, three different types of routers can exist within an OSPF network:

- Internal routers
- Area border routers
- Backbone routers

Figure 16.2 uses the network depicted in Figure 16.1 to identify the three different types of routers in the network.

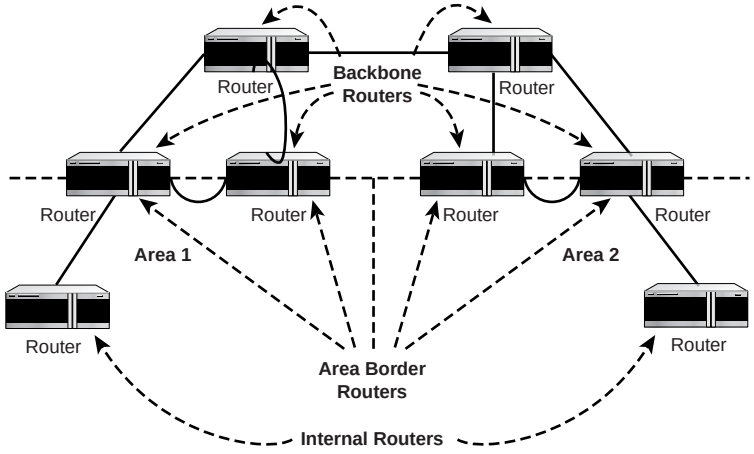



**FIGURE 16.2**
*Area border routers, internal routers, and backbone routers in an OSPF network.*

As shown in Figure 16.2, a router with multiple interfaces may belong to two or more areas. Such routers become *area border routers*. A *backbone* router is one that has at least one interface defined as belonging to Area 0. It is possible for an area border router to also be a backbone router. Any area border router that interconnects a numbered area with Area 0 is both an area border router and a backbone router.

An *internal router* features interfaces that are all defined as the same area, but not Area 0. Using these three basic types of routers, constructing highly efficient and scalable OSPF networks is possible.

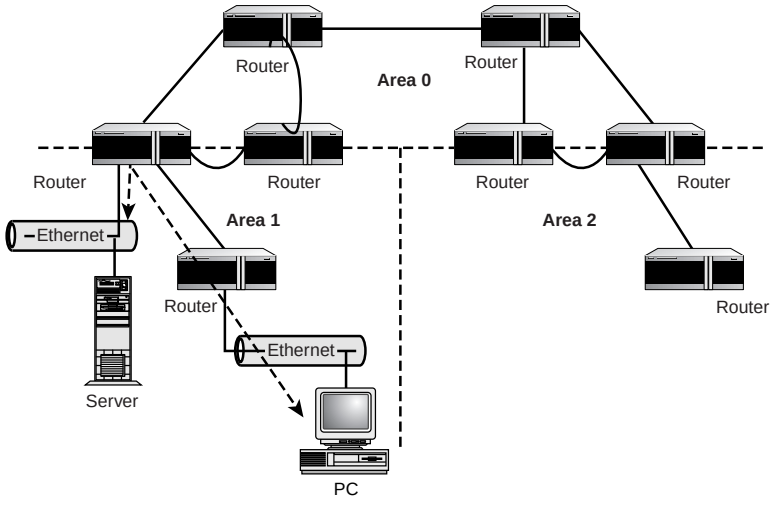
## Routing Types

Given the three different types of OSPF routers illustrated in Figure 16.2, it is important to note that OSPF supports two different types of routing:

- Intra-area routing
- Inter-area routing

These names are fairly self-evident. *Intra-area routing* is self-contained, and limited to just the routers internal to a single area. Using the sample network first illustrated in Figure 16.1, Figure 16.3 demonstrates intra-area communications in an OSPF network.

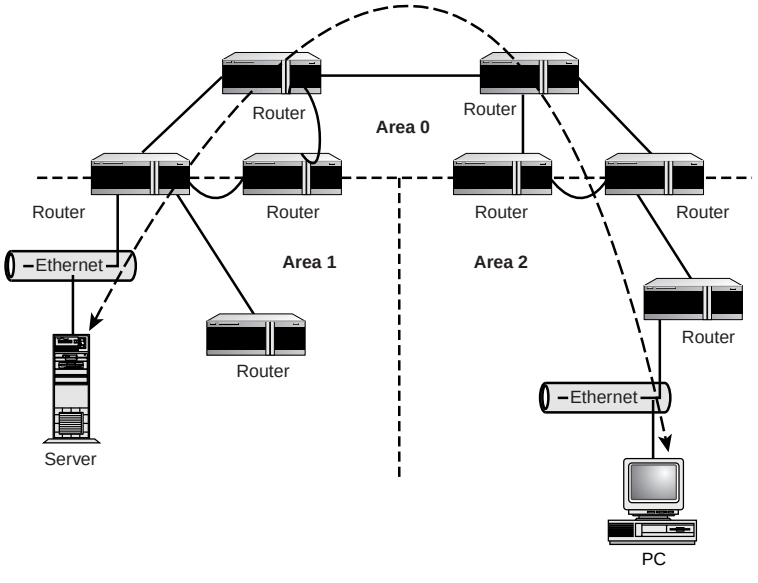
**FIGURE 16.3**
*Intra-area communications in an OSPF network.*




*Inter-area routing* requires the exchange of data between different areas. All inter-area routing must be conducted through Area 0. Nonzero area numbers are not permitted to directly communicate with each other. This hierarchical restriction ensures that OSPF networks scale gracefully, without becoming confusing morasses of links and routers. This type of hierarchical architecture is sometimes called *hierarchical routing*.

Figure 16.4 demonstrates the proper use of Area 0 to facilitate inter-area communications in an OSPF network.

**FIGURE 16.4**
*Inter-area communications in an OSPF network.*



The preceding examples demonstrate, at a high level, how communications work within an OSPF network. However, OSPF can also be used to communicate routing information between OSPF networks, rather than just areas within a single network. This use of OSPF is examined in the following section.

## Routing Between Networks

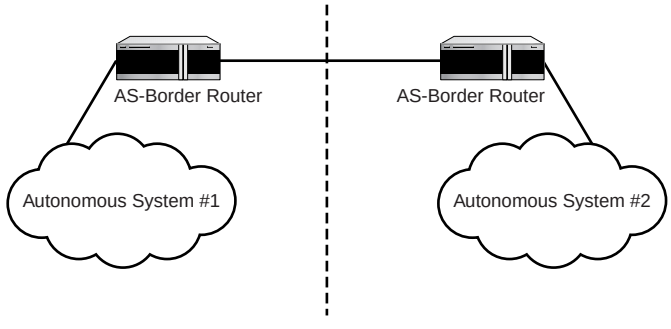
OSPF can be used to internetwork separate networks. Such networks could be another, complete OSPF network, or utilize a completely different routing protocol. Internetworking an OSPF network with a different routing protocol is a complicated task, and uses a technique known as *route redistribution*. This term describes the summarizing and redistributing of routing information from one network into another network. Routing information from the non-OSPF network is summarized and redistributed into the OSPF network.

The OSPF network tags all routes learned in this manner as *external*. Internetworking two different OSPF networks is easier, because there is no need to convert one routing protocol's route cost information into a format that the other protocol can understand. Additionally, OSPF enables the creation of autonomous systems. An *autonomous system* (AS) is a self-contained network. Ostensibly, an AS would feature a single network administrator or group of administrators, and use a single routing protocol.

The actual definition of an AS is somewhat fluid. In truth, it almost doesn't matter. What does matter is that OSPF permits the assignment of an AS number to a network. One very large OSPF network could be segmented into two or more autonomous systems. These systems would be interconnected via a fourth type of OSPF router, *autonomous system border router* (ASBR). The ASBR summarizes all the routing information for its AS, and forwards that summarization to its counterpart ASBR in the neighboring AS. In this regard, the ASBR functions much like an area border router. The difference, obviously, is that they comprise the border between separate autonomous systems instead of areas within a single autonomous system or network.

Figure 16.5 demonstrates internetworking autonomous systems using ASBRs.

**FIGURE 16.5**
*Internetworked OSPF autonomous systems.*



# Routing Updates

One of the reasons OSPF is so scalable is its routing update mechanism. OSPF uses an LSA to share routing information among OSPF nodes. These advertisements are propagated completely throughout an area, but not beyond an area.

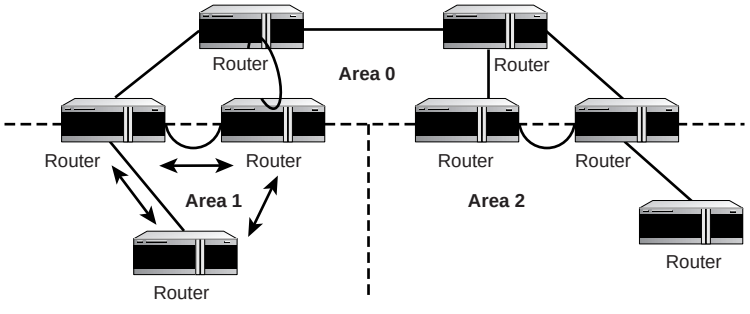
Thus, each router within a given area knows the topology of its area. However, the topology of any given area is not known outside of that area. Given that there are actually four different types of OSPF routers—internal area, area border, autonomous system border, and backbone—it is clear that each router type has a different set of peers that LSAs must be exchanged with.

## Internal Area Routers

Internal area routers must exchange LSAs directly with each other router in its area. This includes every internal area router as well as any area border routers that may also be a member in its area. Figure 16.6 demonstrates the forwarding, or *flooding*, of LSAs throughout Area 1 of the sample OSPF network presented in this chapter's previous illustrations. It is important to note that same-area OSPF routers needn't be directly connected to each other to share LSA information. An OSPF router directly addresses LSA packets to every known router in its area, and forwards those packets using any available links.

The actual mechanism used to flood LSAs is multicasting, which is more efficient than broadcasting. Only OSPF routing processes that join the multicast group 224.0.0.5 receive LSAs. Also OSPF multicasts are efficient because they are carried directly inside an IP datagram using a protocol type 89. OSPF does not use any transport protocol such as TCP or UDP. Contrast this with RIP that uses UDP port 520.

**FIGURE 16.6**
*LSA flooding within Area 1.*



A subtle implication of Figure 16.6 is that convergence can occur quite rapidly. There are two reasons for this. The first is that an OSPF router directly addresses and transmits LSAs to all routers in its area simultaneously (known as flooding). This is in stark contrast to the neighbor-by-neighbor approach used by RIP to drive convergence. The result is an almost instantaneous convergence on a new topology within that area.

Convergence is also expedited through the definition and use of areas. Topological data are not propagated beyond an area's borders. Thus, convergence needn't occur among all routers in the autonomous system, just among the routers in the impacted area. This feature both expedites convergence and enhances the stability of the network because only a subset of the routers in the autonomous system experiences the instability that is innate in convergence.

## Area Border Routers

Area border routers are responsible for maintaining topology information in their databases for each of the areas that they contain interfaces to. Thus, if an area border router interconnects two different areas, it must exchange LSAs with peers in both networks. As with internal area routers, these LSAs are addressed and transmitted directly to their peers in each area. This is illustrated in Figure 16.7.

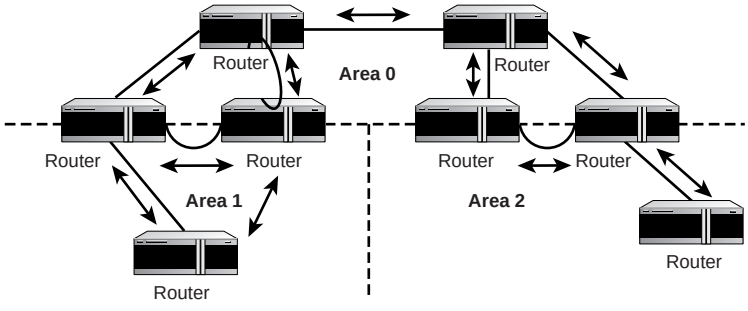



**FIGURE 16.7**
*Intra-area LSA flooding in an OSPF network by area border routers.*

Another of the performance-enhancing features of OSPF is *route summarization*. Topological information about an area is not shared with other routers that are outside that area. Instead, the area border router summarizes all the addresses contained in all the areas it is connected to. This summarized routing data is then shared, via a link-state advertisement (LSA) packet with peer routers in each of the areas it interconnects. OSPF uses several different types of LSAs; each one has a different function. The LSA used to share summarized routing data is known as a *Type 3 LSA*. All the OSPF LSA types are described throughout the remainder of this chapter.

In Figure 16.7, the area border router advertises this summarized data directly to all routers in Area 0. OSPF prevents areas numbered `greater than or equal to` 1 from directly connecting to each other. All such interconnections must occur via Area 0. Thus, it is implied that area border routers interconnect Area 0 with at least one, nonzero numbered area.

## Backbone Routers

Backbone routers are responsible for maintaining topology information for the backbone, as well as propagating summarized topology information for each of the other areas within the autonomous system.

Figure 16.8 illustrates the exchange of LSAs by backbone routers.

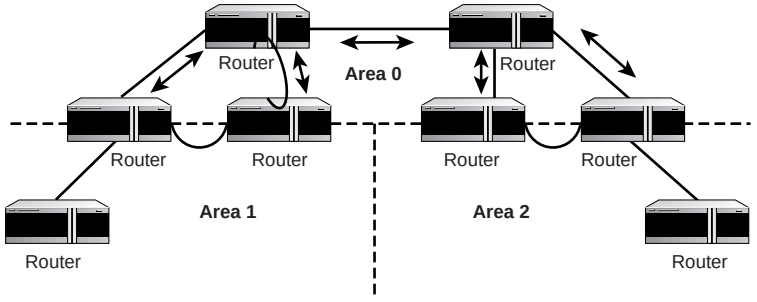



**FIGURE 16.8**
*Intra-area LSA flooding in an OSPF network by backbone routers.*

Although the distinctions between backbone, area border, and internal area routers might seem clear and distinct, there is room for confusion because of the capability of the router to support multiple I/O port connections to other routers. Each port, in theory, could be connected to a different area. Consequently, the router forms a border between the various areas that its interface ports connect to.

# Exploring OSPF Data Structures

OSPF is a fairly complex routing protocol, with many performance- and stability-enhancing features. Thus, it shouldn't be a surprise to find that it uses an extensive array of data structures. Each structure, or message type, is intended to perform a specific task. All of them share a common header, known as the *OSPF header*. The OSPF header is 24 octets long and has the following fields:

- *Version Number*—The first octet of an OSPF header is allocated to the identification of the version number.

- *Type*—The second octet identifies which of the five OSPF packet types is appended to this header structure. The five types (HELLO, database description, link-state request, link-state update, and link-state acknowledgment) are identified numerically.
- *Packet Length*—The next two octets of the OSPF header are used to inform the node receiving the packet of its total length. The total length includes the packet's payload as well as its header.
- *Router ID*—Each router in an area is assigned a unique, four-octet identification number. An OSPF router populates this field with its ID number before transmitting any OSPF messages to other routers.
- *Area ID*—Four octets of the header are used to identify the area identification number.
- *Checksum*—Each OSPF header contains a two-octet checksum field that can be used to detect damage done to the message in transit. The originator runs a mathematical algorithm against each message and stores the results in this field. The recipient node runs an identical algorithm against the received message and compares its result with the result stored in the checksum field. If the message arrives undamaged, the two results will be identical. A mismatch indicates that the OSPF packet was damaged in transit. The recipient simply discards any damaged packets.
- *Authentication Type*—OSPF can guard against the types of attacks that can result in spurious routing information by authenticating the originator of each OSPF message. The Authentication Type field is a two-octet field that identifies which of the various forms of authentication is being used on this message.
- *Authentication*—The last nine octets of the header are used to carry any authentication data that may be needed by the recipient to authenticate the originator of the message. OSPF enables the network's administrator to specify various levels of authentication that range from `NONE`, to `SIMPLE`, to the strong `MD5` authentication mechanism.

This basic structure contains all the information an OSPF node needs to determine whether the packet should be accepted for further processing, or discarded. Packets that have been damaged in transit (as indicated by the Checksum field) will be discarded, as will packets that cannot be authenticated.

OSPF uses five different packet types. Each is designed to support a different, highly specific function within the network. These five are

- HELLO packets (Type 1)
- Database description packets (Type 2)
- Link-state request packets (Type 3)
- Link-state update packets (Type 4)
- Link-state acknowledgment packets (Type 5)

These five packet types are sometimes referred to by their numbers, rather than by name. Thus, an OSPF Type 5 packet is really a link-state acknowledgment packet. All these packet types use the OSPF header.

**Note**

Due to the sheer number of variations on the five basic OSPF data structures, an exhaustive review of their sizes and structures is beyond the scope of this chapter. Instead, this chapter's coverage is limited to a description of the purpose and usage of each data structure.

## The HELLO Packet

OSPF contains a protocol (the HELLO protocol) that is used to establish and maintain relationships between neighboring nodes. These relationships are called adjacencies. *Adjacencies* are the basis for the exchange of routing data in OSPF.

It is through the use of this protocol, and packet type, that an OSPF node discovers the other OSPF nodes in its area. Its name is intentionally significant; the HELLO protocol establishes communications between potential neighboring routers. The HELLO protocol uses a special sub-packet structure that is appended to the standard 24-octet OSPF header. Together, these structures form a *HELLO packet*.

All routers in an OSPF network must adhere to certain conventions that must be uniform throughout the network. These conventions include

- The network mask
- The interval at which HELLO packets will be broadcast (the *HELLO interval*)
- The amount of time that must elapse before a non-responding router will be declared dead (that is, the *router dead interval*) by the other routers in the network

All routers in an OSPF network must agree to use the same value for each of these parameters, or the network might not operate properly. These parameters are exchanged using HELLO packets. Together, they form the basis for neighborly communications. They ensure that neighbor relationships (known as *adjacencies*) are not formed between routers in different networks, and that all members of the network agree on how frequently to stay in contact with each other.

The HELLO packet also includes a listing of other routers (using their unique router IDs) that the source router has recently been in contact with. This field, the Neighbor field, facilitates the neighbor discovery process. The HELLO packet also contains several other fields, such as Designated Router, Backup Designated Router, and others.

These fields are useful in maintaining adjacencies, and support the operation of the OSPF network in both periods of stability and convergence. The specific roles of the Designated Router and Backup Designated Router are described in later sections of this chapter.

## The Database Description Packet

The database description (DD) packet is exchanged between two OSPF routers as they initialize an adjacency. This packet type is used to describe, but not actually convey, the contents of an OSPF router's link-state database. Because this database may be quite lengthy, multiple database description packets might be needed to describe the entire contents of a database. In fact, a field is reserved for identifying the sequence of database description packets. Resequencing ensures that the recipient is able to faithfully replicate the description of the transmitted database description.

The DD exchange process also follows a poll/response method, in which one of the routers is designated as the master. The other functions as the slave. The master router sends its routing table contents to the slave. The slave's responsibilities are just to acknowledge received DD packets. Obviously, the relationship between slave and master varies with each DD exchange. All routers within the network will function, at different times, as both master and slave during this process.

## The Link-State Request Packet

The third type of OSPF packet is the link-state request packet. This packet is used to request specific pieces of a neighboring router's link-state database. Ostensibly, after receiving a DD update, an OSPF router may discover that the neighbor's information is either more current, or more complete, than its own. If so, the router sends a link-state request packet, or packets, to its neighbor (the one with the more recent information) to request more specific link-state routing information.

The request for more information must be very specific. It must specify which data is being requested by using the following criteria:

- Link-state (LS) type number (1 through 5)
- LS ID
- Advertising router

Together, these criteria identify a specific subset of an OSPF database, but not its instance. An instance is the same subset of information, with a temporal boundary (that is, a time stamp). Remember, OSPF is a dynamic routing protocol: It can be expected to automatically update network perspectives in reaction to changes in the state of links in the network. Thus, the recipient of an LS request packet interprets it to be the most recent iteration of this particular piece of its routing database.

## The Link-State Update Packet

The link-state update packet is used to actually transport LSAs to neighboring nodes. These updates are generated in response to an LSA request. There are five different LSA packet types. These packet types are identified by their type number, which ranges from 1 through 5.

**Note**

Potential for confusion exists because OSPF regards link-state advertisements, generically, as LSAs. Yet, the actual mechanism that is used to update routing tables is the link-state update packet—LSU, if one were to use its acronym. If this isn't confusing enough, there is another packet structure, the link-state acknowledgment packet, whose acronym would be LSA. For unknown and unspecified reasons, this packet is known as *link-state acknowledgment*, whereas *LSA* refers generically to the family of update packets.

These packet types, and their respective LSA numbers, are described as follows:

- *Router LSA (Type 1)*—Router LSAs describe the states and costs of a router's links to the area. All such links must be described in a single LSA packet. Also, a router must originate a router LSA for each area it belongs to. Thus, an area border router would generate multiple router LSAs, whereas an interior area router need generate only one such update.
- *Network LSA (Type 2)*—A network LSA is similar to a router LSA in that it, too, describes link-state and cost information for all routers that are attached in the network. The difference between a router and network LSA is that the network LSA is an aggregation of all the link-state and cost information in the network. Only the network's *designated router* tracks this information, and can generate a network LSA.
- *Summary LSA–IP Network (Type 3)*—The Type 3 LSA is somewhat awkwardly referred to as the *summary LSA–IP*, which is probably why the architects of OSPF implemented a numbering scheme for LSAs! Only area border routers in an OSPF network can generate this LSA type. This LSA type is used to communicate summarized routing information about the area to neighboring areas in the OSPF network. It is usually preferable to summarize default routes rather than propagate summarized OSPF information into other networks.
- *Summary LSA–Autonomous System Boundary Router (Type 4*)—The Type 4 LSA is a close relative of the Type 3 LSA. The distinction between these two LSA types is that Type 3 describes inter-area routes, whereas Type 4 describes routes that are external to the OSPF network.

- *AS–External LSA (Type 5)*—The fifth type of LSA is the autonomous system–external LSA. As its name implies, these LSAs are used to describe destinations that are outside of the OSPF network. These destinations can be either specific hosts or external network addresses. An OSPF node that functions as the ASBR to the external autonomous system is responsible for propagating this external routing information throughout all the OSPF areas to which it belongs.

These LSA types are used to describe different aspects of the OSPF routing domain. They are directly addressed to each router in the OSPF area and transmitted simultaneously. This flooding ensures that all routers in an OSPF area have all the same information about the five different aspects (LSA types) of their network. A router's complete collection of LSA data is stored in a link-state database. The contents of this database, when subjected to the Dijkstra Algorithm, result in the creation of the OSPF routing table. The difference between the table and the database is that the database contains a complete collection of raw data, whereas the routing table contains a list of shortest paths to known destinations via specific router interface ports.

Rather than examine the structure of each LSA type, it should be sufficient to merely examine their headers.

## LSA Header

All the LSAs use a common header format. This header is 20 octets long and is appended to the standard 24-octet OSPF header. The LSA header is designed to uniquely identify each LSA. Thus, it contains information about the LSA type, the link-state ID, and the advertising router's ID. The following are the LSA header fields:

- *LS Age*—The first two octets of the LSA header contain the age of the LSA. This age is the number of seconds that have elapsed since the LSA was originated.
- *OSPF Options*—The next octet consists of a series of flags that identify the various optional services that an OSPF network can support.
- *LS Type*—The one-octet LS type identifies which of the five possible types the LSA contains. The format of each LSA type is different. Thus, it is imperative to identify which type of data is appended to this header.
- *Link-State ID*—The Link-State ID field is a four-octet field that identifies the specific portion of the network environment that the LSA describes. This field is closely related to the previous header field, LS type. In fact, the contents of this field directly depend on the LS type. For example, in a router LSA the link-state ID contains the OSPF router ID of the packet's originator—the *advertising router*.
- *Advertising Router*—The advertising router is the router that originated this LSA. Thus, the Advertising Router field contains the OSPF router ID of LSA's originator. Given that OSPF router IDs are four octets long, this field must be the same length.

- *LS Sequence Number*—OSPF routers increment the sequence number for each LSA generated. Thus, a router that receives two instances of the same LSA has two options for determining which of the two is the most recent. The LS Sequence Number field is four octets long, and can be checked to determine how long the LSA has been traversing the network. It is theoretically possible for a newer LSA to have a greater LSA age than an older LSA, particularly in large and complex OSPF networks. Thus, recipient routers compare the LS sequence number. The higher number was the most recently generated. This mechanism doesn't suffer from the vicissitudes of dynamic routing and should be considered a more reliable means of determining the currency of an LSA.
- *LS Checksum*—The three-octet LS checksum is used to detect damage to LSAs en route to their destination. *Checksums* are simple mathematical algorithms. Their output depends on their input, and they are highly consistent. Fed the same input, a checksum algorithm will always return the same output. The LS Checksum field uses part of the contents of the LSA packet (includes the header, except for the LS Age and Checksum fields) to derive a checksum value. The source node runs an algorithm known as the *Fletcher Algorithm* and stores the results in the LS Checksum field. The destination node performs the same mathematical exercise and compares its result to the result stored in the Checksum field. If the values are different, it is relatively safe to assume that damage has occurred in transit. Consequently, a retransmission request is generated.
- *LS Length*—Predictably, the LS Length field informs the recipient of the LSA's length, in octets. This field is one octet in length.

The remainder of an LSA packet's body contains a list of LSAs. Each LSA describes one of the five distinct aspects of an OSPF network, as identified by the LSA number. Thus, a router LSA packet would advertise information about routers known to exist within an area.

## Processing LSA Updates

OSPF differs substantially from other routing tables in that its updates are not directly usable by recipient nodes. Updates received from other routers contain information about the network *from that router's perspective*. Thus, the received LSA data must be subjected to a router's Dijkstra Algorithm to convert it to its own perspective before that data can be interpreted or used.

Ostensibly, LSAs are transmitted because a router detects a change in the state of a link or links. Thus, after receiving an LSA of any type, an OSPF router must check the contents of that LSA against the appropriate portion of its own routing database. This can't be done until after the router uses the new data to form a new perspective of the network, which

is done via the SPF algorithm. The result of this output is the router's new perspective of the network. These results are compared with the existing OSPF routing database to see whether any of its routes have been affected by the network's change in state.

If one or more existing routes must change as a result of the state change, the router builds a new routing database using the new information.

### Duplicate LSAs

Given that LSAs are flooded throughout an OSPF area, it is possible that multiple occurrences, known as *instances*, of the same LSA type will exist simultaneously. The stability of an OSPF network, therefore, requires a router to be able to identify the most current instance of the duplicated LSA. A router that has received two or more instances of the same LSA type examines the LS Age, LS Sequence Number, and the LS Checksum fields in the LSA headers. Only the information contained in the newest LSA is accepted, and subjected to the processes described in the preceding section.

## The Link-State Acknowledgment Packet

The fifth type of OSPF packet is the link-state acknowledgment packet. OSPF features a *reliable* distribution of LSA packets (remember that LSA stands for link-state advertisement, not link-state acknowledgement). Reliability means that receipt of the packet must be acknowledged. Otherwise, the source node would have no way of knowing whether the LSA actually reached its intended destination. Thus, some mechanism was needed to acknowledge receipt of LSAs. This mechanism is the link-state acknowledgment packet.

The link-state acknowledgment packet uniquely identifies the LSA packet that it is acknowledging receipt of. This identification is based on the header information contained in the LSA's header, including LS sequence number and advertising router. There needn't be a one-to-one correlation between LSAs and acknowledgement packets. Multiple LSAs can be acknowledged with a single acknowledgment packet.

# Calculating Routes

OSPF, despite its complexity, calculates the costs of a route in one of two remarkably simple ways:

- A non–bandwidth-sensitive default value can be used for each OSPF interface.
- OSPF can automatically calculate the cost of using individual router interfaces.

Regardless of which method is employed, the cost of any given route is calculated by calculating the costs of all interfaces encountered along that route. A record is kept of the calculated costs to known destinations in OSPF's shortest-path tree.

# Using Autocalculation

OSPF can automatically calculate the cost of an interface. This algorithm is based on the amount of bandwidth that each interface type supports. The sum of the calculated values of all interfaces in a given route forms the basis for OSPF routing decisions. These values enable OSPF to calculate routes based, at a minimum, on the bandwidth available per link in redundant routes. Figure 16.9 presents a sample network to demonstrate this point.

In Figure 16.9, the cost of the WAN route between a host in network 193.1.3.0 and an end system in network 193.1.4.0 is 138. This cost is the sum of the two T1 links between those networks, each with a cost of 64, plus the cost of the Ethernet interface to network 193.1.4.0. The cost of the Ethernet interfaces at the origination and destination points are not included in the OSPF cost calculation because the OSPF only calculates the costs of outbound router interfaces.

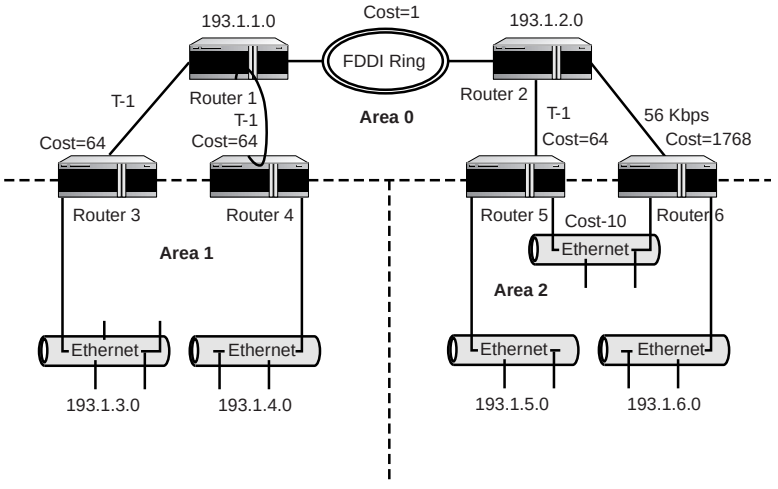



**FIGURE 16.9**
*Autocalculated costs of the links.*

Table 16.1 summarizes the automatically calculated costs for each of the interfaces used in Figure 16.9's network diagram.

**TABLE 16.1** Calculated Costs Per Interface Type

| *Interface Type* | *Calculated Cost* |
|---|---|
| 100Mbps FDDI | 1 |
| 10Mbps Ethernet | 10 |
| 1.544Mbps T1 serial link | 64 |
| 56Kbps serial link | 1,768 |

# Using Default Route Costs

It is usually in your best interest to have OSPF automatically calculate route costs, although this may not be possible. For example, older routers might not support the autocalculation feature. In such cases, all interfaces will have the same OSPF cost. Thus, a T3 will have exactly the same cost as a 56Kbps leased line. Clearly, these two facilities offer very different levels of performance. This disparity should form the basis of informed routing decisions.

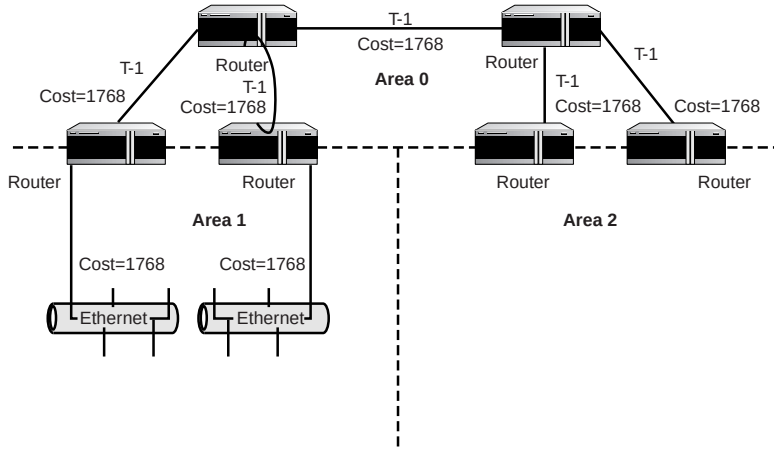
There are, however, circumstances that might make the use of default route costs acceptable. For example, if your network consists of relatively homogeneous transmission facilities, default values would be acceptable. Alternatively, you can manually change the cost metrics for specific interfaces. This would enable you to shape traffic patterns in your OSPF network as you see fit, while still using predominantly default routing costs.

## Homogeneous Networks

In a homogeneous network, all the transmission facilities are the same. For example, all the LAN interfaces would be 10Mbps Ethernet and all the serial WAN interfaces would be T1s. In such a scenario, using the default values would not likely cause routing problems. This would be particularly true if there were little, if any, route redundancy.

To illustrate this point, consider the network diagram in Figure 16.10.

**FIGURE 16.10**
*Acceptable use of OSPF's default interface values.*



In Figure 16.10, a default value of 1,768 was assigned to each of the interfaces. All the WAN links, however, are T1s. Given that they are all the same, it doesn't matter whether the value assigned them is 1, 128, 1,768, or 1,000,000! Routing decisions, in a homogeneous network, become a simple matter of counting and comparing hops (albeit in multiples of the interface costs). This would be true regardless of how much, or how little, route redundancy existed in the network.

Obviously, in a complex network with substantial route redundancy *and* a disparity in the actual transmission technologies used, the default value would not enable selection of optimal routes to any given destination.

### Manually Setting Values

In some networks, accepting OSPF's default costs, and then manually resetting those specific links that differ the most from the default might be desirable. For example, your network's default cost value might be 1,768—the calculated value for a 56Kbps serial link.

If all but one or two of the links in your network offered the same bandwidth, you could accept the default values and then reset the values for those particular links.

Whether you use automatically calculated routing costs, default costs, or manually configured costs is immaterial to OSPF nodes. They will accept all such cost values and develop a shortest-path tree perspective of the network.

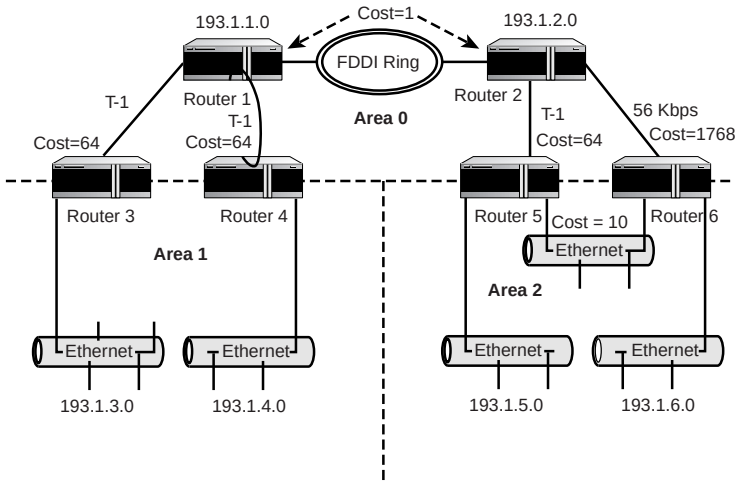
## The Shortest-Path Tree

The purpose of the various LSA mechanisms is to enable each router to develop a perspective of the network's topology. This topology is arranged in the shape of a tree. The OSPF router forms the tree's root. The tree gives the complete path to all known destination addresses, either network or host, even though only the next hop is actually used in forwarding datagrams. The reason for this is simple. Tracking complete paths to destinations makes comparing redundant paths, and selecting the best one to each known destination, possible. If there are multiple paths of equal cost, they are all discovered and used for load balancing by OSPF. Traffic is dynamically balanced equally across all such available links, thereby increasing the overall throughput for the route.

### Router 3's Perspective

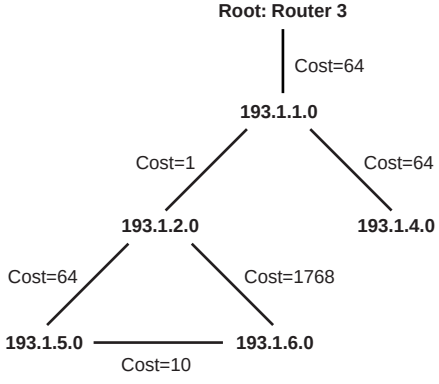
To better understand the concept of the shortest-path tree, consider the network diagram presented in Figure 16.11. The simple network depicted is a small OSPF network. The administrator has enabled autocalculation of routing costs. It is important to note that the Ethernet installed between routers 5 and 6 creates an alternate path for both networks 193.1.5.0 and 193.1.6.0. Thus, it has an OSPF autocalculated cost of 10, whereas similar costs are not assigned to the other Ethernet networks in this example.

**FIGURE 16.11**
*An OSPF network with routing costs.*




The shortest-path tree for this network (illustrated in Figure 16.10) would vary from router to router. Figure 16.12, for example, presents this tree from the perspective of Router 3.

**FIGURE 16.12**
*Router 3's shortest-path tree.*




As is evident in Figure 16.12, the tree structure greatly facilitates the calculation of routing costs to any given destination. The root router (Router 3—193.1.3.0, in this case) can quickly sum the costs associated with each interface encountered along a route to a given destination. From Router 3's perspective, routing costs to each of the networks are calculated for you in Table 16.2. For destinations that are more than one hop away, the interface costs are calculated in parentheses. This will enable you to trace the path through the network in Figure 16.12.

**TABLE 16.2** Costs from Router 3 to Known Destinations

| *Destination* | *Hops Away* | *Cumulative Cost* |
|---|---|---|
| 193.1.3.0 | — | 0 |
| 193.1.1.0 | 1 | 64 |
| 193.1.2.0 | 2 | 65 (64 + 1) |
| 193.1.4.0 | 2 | 128 (64 + 64) |
| 193.1.5.0 | 3 | 129 (64 + 1 + 64) |
| 193.1.6.0 | 3 | 1,833 (64 + 1 + 1768) |
| 193.1.6.0 | 4 | 139 (64 + 1 + 64 + 10) |

In this example, there are two possible routes to network 193.1.6.0. The one route contains fewer hops, but has a much higher cost due to the low-speed serial link between Routers 2 and 6. The alternate route has a higher hop count, but a much lower overall cost. In this case, OSPF would discard the higher-cost route and use the lower-cost route exclusively, even if it had more hops. If these two redundant routes had the same overall cost, OSPF would have maintained both routes as separate entries in its routing table and balanced the traffic as equally as possible between them.

## Router 2's Perspective

Each router's perspective of the network is different. Although it would be somewhat monotonous to examine each router's perspective, a second example may prove useful in demonstrating the impact that perspective has on the shortest-path tree. Figure 16.13 demonstrates the shortest-path tree for Router 2.

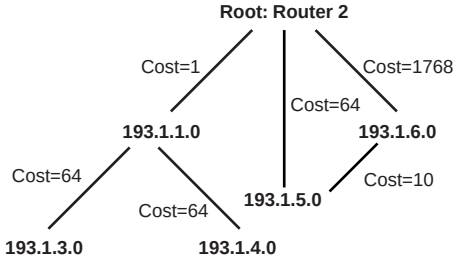



**FIGURE 16.13**
*Router 2's shortest-path tree.*

Table 16.3 provides the summarized routing costs to all known destinations from Router 2's perspective.

**TABLE 16.3** Costs from Router 2 to Known Destinations

| *Destination* | *Hops Away* | *Cumulative Cost* |
|---|---|---|
| 193.1.2.0 | — | 0 |
| 193.1.5.0 | 1 | 64 |
| 193.1.6.0 | 1 | 1,768 |
| 193.1.6.0 | 2 | 74 (64 + 10) |
| 193.1.1.0 | 1 | 1 |
| 193.1.3.0 | 2 | 65 (1 + 64) |
| 193.1.4.0 | 2 | 65 (1 + 64) |

Comparing Tables 16.2 and 16.3 demonstrates that the cumulative distances between a source and destination in a network can vary based on the starting point. Perspective, it seems, is everything. This is why OSPF routers use data obtained from other routers via LSA updates to develop their own perspective of the network, rather than directly update their routing tables with that information.

# Summary

OSPF is one of the most powerful and feature-rich open routing protocols available. Its complexity is also a source of weakness because designing, building, and operating an OSPF internetwork require more expertise and effort than a similar network using almost any other routing protocol. Accepting the default values for the routing costs will greatly simplify the design of an OSPF network. As your knowledge of both OSPF and your network's operational characteristics increase, you can slowly fine-tune its performance by manipulating the OSPF variables.

Extreme care must be used in designing the areas, and the network's topology. Done properly, your OSPF network will reward you with solid performance and quick convergence.