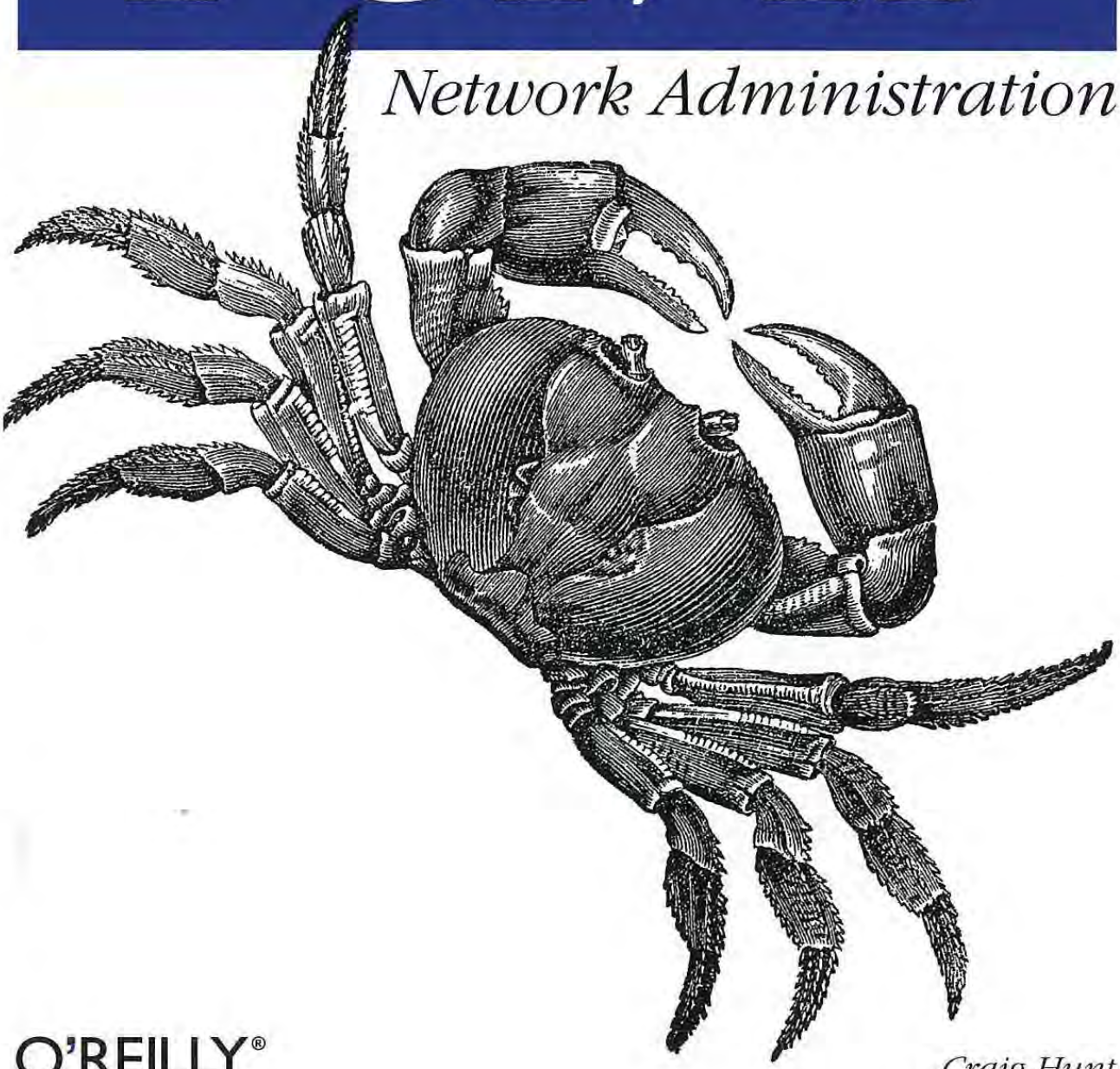
Help for UNIX System Administrators

2nd Edition

TCP/IP

Network Administration



O'REILLY®

Craig Hunt

TCP/IP Network Administration

TCP/IP Network Administration

Second Edition

Craig Hunt

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

TCP/IP Network Administration, Second Edition
by Craig Hunt

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editor: Mike Loukides

Update Editor: Gigi Estabrook

Production Editor: Nicole Gipson Arigo

Printing History:

August 1992:	First Edition.
March 1993:	Minor corrections.
September 1993:	Minor corrections.
January 1994:	Minor corrections.
May 1994:	Minor corrections.
January 1998:	Second Edition.

ISBN: 1-56592-322-7 [6/01]
[M]

TCP/IP Network Administration, Second Edition
by Craig Hunt

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editor: Mike Loukides

Update Editor: Gigi Estabrook

Production Editor: Nicole Gipson Arigo

Printing History:

August 1992:	First Edition.
March 1993:	Minor corrections.
September 1993:	Minor corrections.
January 1994:	Minor corrections.
May 1994:	Minor corrections.
January 1998:	Second Edition.

ISBN: 1-56592-322-7 [6/01]
[M]

In this chapter:

- *TCP/IP and the Internet*
- *A Data Communications Model*
- *TCP/IP Protocol Architecture*
- *Network Access Layer*
- *Internet Layer*
- *Transport Layer*
- *Application Layer*
- *Summary*

1

Overview of TCP/IP

All of us who use a UNIX desktop system—engineers, educators, scientists, and business people—have second careers as UNIX system administrators. Networking these computers gives us new tasks as network administrators.

Network administration and system administration are two different jobs. System administration tasks such as adding users and doing backups are isolated to one independent computer system. Not so with network administration. Once you place your computer on a network, it interacts with many other systems. The way you do network administration tasks has effects, good and bad, not only on your system but on other systems on the network. A sound understanding of basic network administration benefits everyone.

Networking computers dramatically enhances their ability to communicate—and most computers are used more for communication than computation. Many mainframes and supercomputers are busy crunching the numbers for business and science, but the number of such systems pales in comparison to the millions of systems busy moving mail to a remote colleague or retrieving information from a remote repository. Further, when you think of the hundreds of millions of desktop systems that are used primarily for preparing documents to communicate ideas from one person to another, it is easy to see why most computers can be viewed as communications devices.

The positive impact of computer communications increases with the number and type of computers that participate in the network. One of the great benefits of TCP/IP is that it provides interoperable communications between all types of hardware and all kinds of operating systems.

This book is a practical, step-by-step guide to configuring and managing TCP/IP networking software on UNIX computer systems. TCP/IP is the software package that dominates UNIX data communications. It is the leading communications software for UNIX local area networks and enterprise intranets, and for the foundation of the worldwide Internet.

The name "TCP/IP" refers to an entire suite of data communications protocols. The suite gets its name from two of the protocols that belong to it: the Transmission Control Protocol and the Internet Protocol. Although there are many other protocols in the suite, TCP and IP are certainly two of the most important.

The first part of this book discusses the basics of TCP/IP and how it moves data across a network. The second part explains how to configure and run TCP/IP on a UNIX system. Let's start with a little history.

TCP/IP and the Internet

In 1969 the Advanced Research Projects Agency (ARPA) funded a research and development project to create an experimental packet-switching network. This network, called the *ARPANET*, was built to study techniques for providing robust, reliable, vendor-independent data communications. Many techniques of modern data communications were developed in the ARPANET.

The experimental ARPANET was so successful that many of the organizations attached to it began to use it for daily data communications. In 1975 the ARPANET was converted from an experimental network to an operational network, and the responsibility for administering the network was given to the Defense Communications Agency (DCA).* However, development of the ARPANET did not stop just because it was being used as an operational network; the basic TCP/IP protocols were developed after the ARPANET was operational.

The TCP/IP protocols were adopted as Military Standards (MIL STD) in 1983, and all hosts connected to the network were required to convert to the new protocols. To ease this conversion, DARPA† funded Bolt, Beranek, and Newman (BBN) to implement TCP/IP in Berkeley (BSD) UNIX. Thus began the marriage of UNIX and TCP/IP.

About the time that TCP/IP was adopted as a standard, the term *Internet* came into common usage. In 1983, the old ARPANET was divided into MILNET, the

* DCA has since changed its name to Defense Information Systems Agency (DISA).

† During the 1980s and early 1990s, ARPA, which is part of the U.S. Department of Defense, was named Defense Advanced Research Projects Agency (DARPA). Currently known as ARPA, the agency is again preparing to change its name to DARPA. Whether it is known as ARPA or DARPA, the agency and its mission of funding advanced research has remained the same.

unclassified part of the Defense Data Network (DDN), and a new, smaller ARPANET. "Internet" was used to refer to the entire network: MILNET plus ARPANET.

In 1985 the National Science Foundation (NSF) created NSFNet and connected it to the then-existing Internet. The original NSFNet linked together the five NSF supercomputer centers. It was smaller than the ARPANET and no faster—56Kbps. Nonetheless, the creation of the NSFNet was a significant event in the history of the Internet because NSF brought with it a new vision of the use of the Internet. NSF wanted to extend the network to every scientist and engineer in the United States. To accomplish this, in 1987 NSF created a new, faster backbone and a three-tiered network topology that included the backbone, regional networks, and local networks.

In 1990, the ARPANET formally passed out of existence, and the NSFNet ceased its role as a primary Internet backbone network in 1995. Still, today the Internet is larger than ever and encompasses more than 95,000 networks worldwide. This network of networks is linked together in the United States at several major interconnection points:

- The three Network Access Points (NAPs) created by the NSF to ensure continued broad-based access to the Internet.
- The Federal Information Exchanges (FIXs) interconnect U.S. government networks.
- The Commercial Information Exchange (CIX) was the first interconnect specifically for commercial Internet Service Providers (ISPs).
- The Metropolitan Area Exchanges (MAEs) were also created to interconnect commercial ISPs.

The Internet has grown far beyond its original scope. The original networks and agencies that built the Internet no longer play an essential role for the current network. The Internet has evolved from a simple backbone network, through a three-tiered hierarchical structure, to a huge network of interconnected, distributed network hubs. It has grown exponentially since 1983—doubling in size every year. Through all of this incredible change one thing has remained constant: the Internet is built on the TCP/IP protocol suite.

A sign of the network's success is the confusion that surrounds the term *internet*. Originally it was used only as the name of the network built upon the Internet Protocol. Now *internet* is a generic term used to refer to an entire class of networks. An internet (lowercase "i") is any collection of separate physical networks, interconnected by a common protocol, to form a single logical network. The Internet (uppercase "I") is the worldwide collection of interconnected networks, which grew out of the original ARPANET, that uses *Internet Protocol* (IP) to link the

various physical networks into a single logical network. In this book, both "internet" and "Internet" refer to networks that are interconnected by TCP/IP.

Because TCP/IP is required for Internet connection, the growth of the Internet has spurred interest in TCP/IP. As more organizations become familiar with TCP/IP, they see that its power can be applied in other network applications. The Internet protocols are often used for local area networking, even when the local network is not connected to the Internet. TCP/IP is also widely used to build enterprise networks. TCP/IP-based enterprise networks that use Internet techniques and World Wide Web tools to disseminate internal corporate information are called *intranets*. TCP/IP is the foundation of all of these varied networks.

TCP/IP Features

The popularity of the TCP/IP protocols did not grow rapidly just because the protocols were there, or because connecting to the Internet mandated their use. They met an important need (worldwide data communication) at the right time, and they had several important features that allowed them to meet this need. These features are:

- Open protocol standards, freely available and developed independently from any specific computer hardware or operating system. Because it is so widely supported, TCP/IP is ideal for uniting different hardware and software, even if you don't communicate over the Internet.
- Independence from specific physical network hardware. This allows TCP/IP to integrate many different kinds of networks. TCP/IP can be run over an Ethernet, a token ring, a dial-up line, an FDDI net, and virtually any other kind of physical transmission medium.
- A common addressing scheme that allows any TCP/IP device to uniquely address any other device in the entire network, even if the network is as large as the worldwide Internet.
- Standardized high-level protocols for consistent, widely available user services.

Protocol Standards

Protocols are formal rules of behavior. In international relations, protocols minimize the problems caused by cultural differences when various nations work together. By agreeing to a common set of rules that are widely known and independent of any nation's customs, diplomatic protocols minimize misunderstandings; everyone knows how to act and how to interpret the actions of others. Similarly, when computers communicate, it is necessary to define a set of rules to govern their communications.

In data communications these sets of rules are also called *protocols*. In homogeneous networks, a single computer vendor specifies a set of communications rules designed to use the strengths of the vendor's operating system and hardware architecture. But homogeneous networks are like the culture of a single country—only the natives are truly at home in it. TCP/IP attempts to create a heterogeneous network with open protocols that are independent of operating system and architectural differences. TCP/IP protocols are available to everyone, and are developed and changed by consensus—not by the fiat of one manufacturer. Everyone is free to develop products to meet these open protocol specifications.

The open nature of TCP/IP protocols requires publicly available standards documents. All protocols in the TCP/IP protocol suite are defined in one of three Internet standards publications. A number of the protocols have been adopted as *Military Standards* (MIL STD). Others were published as *Internet Engineering Notes* (IEN)—though the IEN form of publication has now been abandoned. But most information about TCP/IP protocols is published as *Requests for Comments* (RFCs). RFCs contain the latest versions of the specifications of all standard TCP/IP protocols.* As the title "Request for Comments" implies, the style and content of these documents is much less rigid than most standards documents. RFCs contain a wide range of interesting and useful information, and are not limited to the formal specification of data communications protocols.

As a network system administrator, you will no doubt read many of the RFCs yourself. Some contain practical advice and guidance that is simple to understand. Other RFCs contain protocol implementation specifications defined in terminology that is unique to data communications.

A Data Communications Model

To discuss computer networking, it is necessary to use terms that have special meaning. Even other computer professionals may not be familiar with all the terms in the networking alphabet soup. As is always the case, English and computer-speak are not equivalent (or even necessarily compatible) languages. Although descriptions and examples should make the meaning of the networking jargon more apparent, sometimes terms are ambiguous. A common frame of reference is necessary for understanding data communications terminology.

An architectural model developed by the International Standards Organization (ISO) is frequently used to describe the structure and function of data communications protocols. This architectural model, which is called the *Open Systems Interconnect Reference Model* (OSI), provides a common reference for discussing

* Interested in finding out how Internet standards are created? Read *The Internet Standards Process*, RFC 1310.

communications. The terms defined by this model are well understood and widely used in the data communications community—so widely used, in fact, that it is difficult to discuss data communications without using OSI's terminology.

The OSI Reference Model contains seven *layers* that define the functions of data communications protocols. Each layer of the OSI model represents a function performed when data is transferred between cooperating applications across an intervening network. Figure 1-1 identifies each layer by name and provides a short functional description for it. Looking at this figure, the protocols are like a pile of building blocks stacked one upon another. Because of this appearance, the structure is often called a *stack* or *protocol stack*.

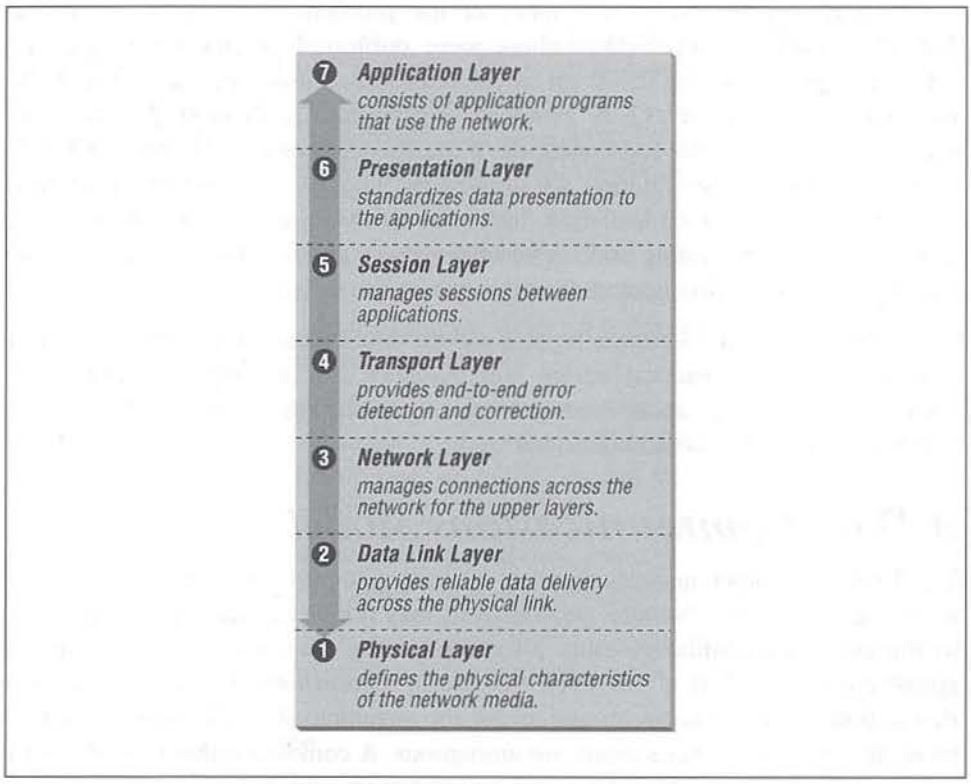


Figure 1-1: The OSI Reference Model

A layer does not define a single protocol—it defines a data communications function that may be performed by any number of protocols. Therefore, each layer may contain multiple protocols, each providing a service suitable to the function of that layer. For example, a file transfer protocol and an electronic mail protocol both provide user services, and both are part of the Application Layer.

Every protocol communicates with its peer. A *peer* is an implementation of the same protocol in the equivalent layer on a remote system; i.e., the local file transfer protocol is the peer of a remote file transfer protocol. Peer-level communications must be standardized for successful communications to take place. In the abstract, each protocol is concerned only with communicating to its peer; it does not care about the layer above or below it.

However, there must also be agreement on how to pass data between the layers on a single computer, because every layer is involved in sending data from a local application to an equivalent remote application. The upper layers rely on the lower layers to transfer the data over the underlying network. Data is passed down the stack from one layer to the next, until it is transmitted over the network by the Physical Layer protocols. At the remote end, the data is passed up the stack to the receiving application. The individual layers do not need to know how the layers above and below them function; they only need to know how to pass data to them. Isolating network communications functions in different layers minimizes the impact of technological change on the entire protocol suite. New applications can be added without changing the physical network, and new network hardware can be installed without rewriting the application software.

Although the OSI model is useful, the TCP/IP protocols don't match its structure exactly. Therefore, in our discussions of TCP/IP, we use the layers of the OSI model in the following way:

Application Layer

The Application Layer is the level of the protocol hierarchy where user-accessed network processes reside. In this text, a TCP/IP application is any network process that occurs above the Transport Layer. This includes all of the processes that users directly interact with, as well as other processes at this level that users are not necessarily aware of.

Presentation Layer

For cooperating applications to exchange data, they must agree about how data is represented. In OSI, this layer provides standard data presentation routines. This function is frequently handled within the applications in TCP/IP, though increasingly TCP/IP protocols such as XDR and MIME perform this function.

Session Layer

As with the Presentation Layer, the Session Layer is not identifiable as a separate layer in the TCP/IP protocol hierarchy. The OSI Session Layer manages the sessions (connection) between cooperating applications. In TCP/IP, this function largely occurs in the Transport Layer, and the term "session" is not used. For TCP/IP, the terms "socket" and "port" are used to describe the path over which cooperating applications communicate.

Transport Layer

Much of our discussion of TCP/IP is directed to the protocols that occur in the Transport Layer. The Transport Layer in the OSI reference model guarantees that the receiver gets the data exactly as it was sent. In TCP/IP this function is performed by the *Transmission Control Protocol* (TCP). However, TCP/IP offers a second Transport Layer service, *User Datagram Protocol* (UDP), that does not perform the end-to-end reliability checks.

Network Layer

The Network Layer manages connections across the network and isolates the upper layer protocols from the details of the underlying network. The Internet Protocol (IP), which isolates the upper layers from the underlying network and handles the addressing and delivery of data, is usually described as TCP/IP's Network Layer.

Data Link Layer

The reliable delivery of data across the underlying physical network is handled by the Data Link Layer. TCP/IP rarely creates protocols in the Data Link Layer. Most RFCs that relate to the Data Link Layer discuss how IP can make use of existing data link protocols.

Physical Layer

The Physical Layer defines the characteristics of the hardware needed to carry the data transmission signal. Features such as voltage levels, and the number and location of interface pins, are defined in this layer. Examples of standards at the Physical Layer are interface connectors such as RS232C and V.35, and standards for local area network wiring such as IEEE 802.3. TCP/IP does not define physical standards—it makes use of existing standards.

The terminology of the OSI reference model helps us describe TCP/IP, but to fully understand it, we must use an architectural model that more closely matches the structure of TCP/IP. The next section introduces the protocol model we'll use to describe TCP/IP.

TCP/IP Protocol Architecture

While there is no universal agreement about how to describe TCP/IP with a layered model, it is generally viewed as being composed of fewer layers than the seven used in the OSI model. Most descriptions of TCP/IP define three to five functional levels in the protocol architecture. The four-level model illustrated in Figure 1-2 is based on the three layers (Application, Host-to-Host, and Network Access) shown in the DOD Protocol Model in the *DDN Protocol Handbook—Volume 1*, with the addition of a separate Internet layer. This model provides a reasonable pictorial representation of the layers in the TCP/IP protocol hierarchy.

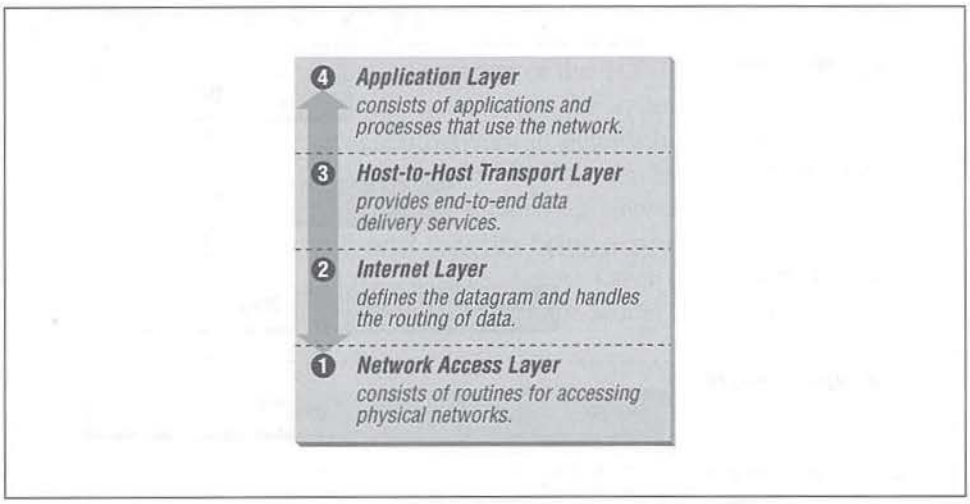


Figure 1-2: Layers in the TCP/IP protocol architecture

As in the OSI model, data is passed down the stack when it is being sent to the network, and up the stack when it is being received from the network. The four-layered structure of TCP/IP is seen in the way data is handled as it passes down the protocol stack from the Application Layer to the underlying physical network. Each layer in the stack adds control information to ensure proper delivery. This control information is called a *header* because it is placed in front of the data to be transmitted. Each layer treats all of the information it receives from the layer above as data and places its own header in front of that information. The addition of delivery information at every layer is called *encapsulation*. (See Figure 1-3 for an illustration of this.) When data is received, the opposite happens. Each layer strips off its header before passing the data on to the layer above. As information flows back up the stack, information received from a lower layer is interpreted as both a header and data.

Each layer has its own independent data structures. Conceptually, a layer is unaware of the data structures used by the layers above and below it. In reality, the data structures of a layer are designed to be compatible with the structures used by the surrounding layers for the sake of more efficient data transmission. Still, each layer has its own data structure and its own terminology to describe that structure.

Figure 1-4 shows the terms used by different layers of TCP/IP to refer to the data being transmitted. Applications using TCP refer to data as a *stream*, while applications using the User Datagram Protocol (UDP) refer to data as a *message*. TCP calls data a *segment*, and UDP calls its data structure a *packet*. The Internet layer views all data as blocks called *datagrams*. TCP/IP uses many different types of

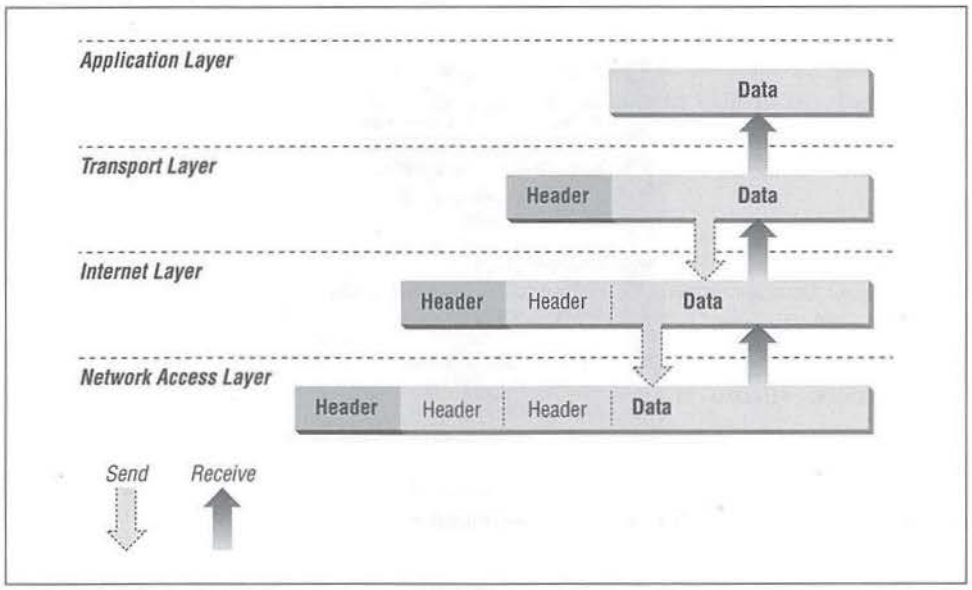


Figure 1-3: Data encapsulation

underlying networks, each of which may have a different terminology for the data it transmits. Most networks refer to transmitted data as *packets* or *frames*. In Figure 1-4 we show a network that transmits pieces of data it calls *frames*.

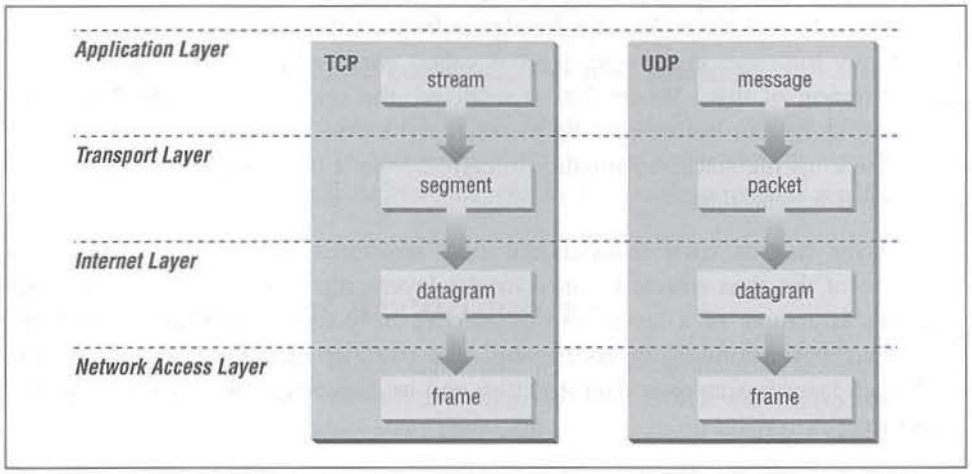


Figure 1-4: Data structures

Let's look more closely at the function of each layer, working our way up from the Network Access Layer to the Application Layer.

Network Access Layer

The *Network Access Layer* is the lowest layer of the TCP/IP protocol hierarchy. The protocols in this layer provide the means for the system to deliver data to the other devices on a directly attached network. It defines how to use the network to transmit an IP datagram. Unlike higher-level protocols, Network Access Layer protocols must know the details of the underlying network (its packet structure, addressing, etc.) to correctly format the data being transmitted to comply with the network constraints. The TCP/IP Network Access Layer can encompass the functions of all three lower layers of the OSI reference Model (Network, Data Link, and Physical).

The Network Access Layer is often ignored by users. The design of TCP/IP hides the function of the lower layers, and the better known protocols (IP, TCP, UDP, etc.) are all higher-level protocols. As new hardware technologies appear, new Network Access protocols must be developed so that TCP/IP networks can use the new hardware. Consequently, there are many access protocols—one for each physical network standard.

Functions performed at this level include encapsulation of IP datagrams into the frames transmitted by the network, and mapping of IP addresses to the physical addresses used by the network. One of TCP/IP's strengths is its universal addressing scheme. The IP address must be converted into an address that is appropriate for the physical network over which the datagram is transmitted.

Two examples of RFCs that define network access layer protocols are:

- RFC 826, *Address Resolution Protocol (ARP)*, which maps IP addresses to Ethernet addresses
- RFC 894, *A Standard for the Transmission of IP Datagrams over Ethernet Networks*, which specifies how IP datagrams are encapsulated for transmission over Ethernet networks

As implemented in UNIX, protocols in this layer often appear as a combination of device drivers and related programs. The modules that are identified with network device names usually encapsulate and deliver the data to the network, while separate programs perform related functions such as address mapping.

Internet Layer

The layer above the Network Access Layer in the protocol hierarchy is the *Internet Layer*. The Internet Protocol, RFC 791, is the heart of TCP/IP and the most important protocol in the Internet Layer. IP provides the basic packet delivery service on which TCP/IP networks are built. All protocols, in the layers above and below IP,

use the Internet Protocol to deliver data. All TCP/IP data flows through IP, incoming and outgoing, regardless of its final destination.

Internet Protocol

The Internet Protocol is the building block of the Internet. Its functions include:

- Defining the datagram, which is the basic unit of transmission in the Internet
- Defining the Internet addressing scheme
- Moving data between the Network Access Layer and the Host-to-Host Transport Layer
- Routing datagrams to remote hosts
- Performing fragmentation and re-assembly of datagrams

Before describing these functions in more detail, let's look at some of IP's characteristics. First, IP is a *connectionless protocol.* This means that IP does not exchange control information (called a "handshake") to establish an end-to-end connection before transmitting data. In contrast, a *connection-oriented protocol* exchanges control information with the remote system to verify that it is ready to receive data before any data is sent. When the handshaking is successful, the systems are said to have established a *connection.* Internet Protocol relies on protocols in other layers to establish the connection if they require connection-oriented service.

IP also relies on protocols in the other layers to provide error detection and error recovery. The Internet Protocol is sometimes called an *unreliable protocol* because it contains no error detection and recovery code. This is not to say that the protocol cannot be relied on—quite the contrary. IP can be relied upon to accurately deliver your data to the connected network, but it doesn't check whether that data was correctly received. Protocols in other layers of the TCP/IP architecture provide this checking when it is required.

The datagram

The TCP/IP protocols were built to transmit data over the ARPANET, which was a *packet switching network.* A *packet* is a block of data that carries with it the information necessary to deliver it—in a manner similar to a postal letter, which has an address written on its envelope. A packet switching network uses the addressing information in the packets to switch packets from one physical network to another, moving them toward their final destination. Each packet travels the network independently of any other packet.

The *datagram* is the packet format defined by Internet Protocol. Figure 1-5 is a pictorial representation of an IP datagram. The first five or six 32-bit words of the

datagram are control information called the *header*. By default, the header is five words long; the sixth word is optional. Because the header's length is variable, it includes a field called *Internet Header Length (IHL)* that indicates the header's length in words. The header contains all the information necessary to deliver the packet.

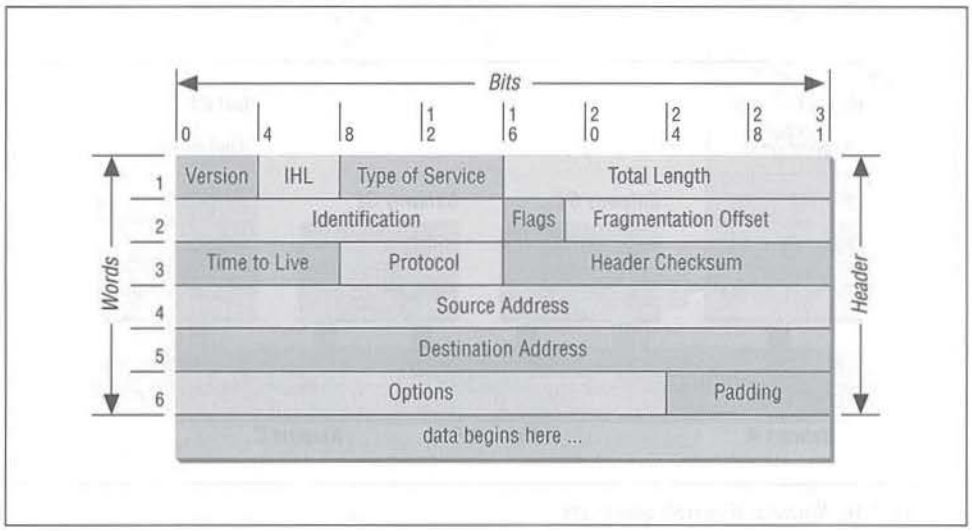


Figure 1-5: IP datagram format

The Internet Protocol delivers the datagram by checking the *Destination Address* in word 5 of the header. The Destination Address is a standard 32-bit IP address that identifies the destination network and the specific host on that network. (The format of IP addresses is explained in Chapter 2, *Delivering the Data.*) If the Destination Address is the address of a host on the local network, the packet is delivered directly to the destination. If the Destination Address is not on the local network, the packet is passed to a gateway for delivery. *Gateways* are devices that switch packets between the different physical networks. Deciding which gateway to use is called *routing*. IP makes the routing decision for each individual packet.

Routing datagrams

Internet gateways are commonly (and perhaps more accurately) referred to as *IP routers* because they use Internet Protocol to route packets between networks. In traditional TCP/IP jargon, there are only two types of network devices—*gateways* and *hosts*. Gateways forward packets between networks, and hosts don't. However, if a host is connected to more than one network (called a *multi-homed host*), it can forward packets between the networks. When a multi-homed host forwards packets, it acts just like any other gateway and is considered to be a gateway.

Current data communications terminology makes a distinction between gateways and routers,* but we'll use the terms *gateway* and *IP router* interchangeably.

Figure 1-6 shows the use of gateways to forward packets. The hosts (or *end systems*) process packets through all four protocol layers, while the gateways (or *intermediate systems*) process the packets only up to the Internet Layer where the routing decisions are made.

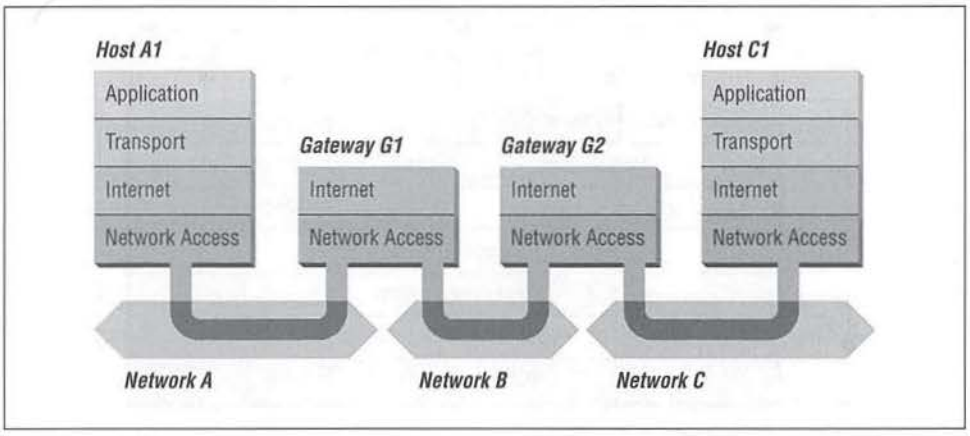


Figure 1-6: Routing through gateways

Systems can only deliver packets to other devices attached to the same physical network. Packets from *A1* destined for host *C1* are forwarded through gateways *G1* and *G2*. Host *A1* first delivers the packet to gateway *G1*, with which it shares network *A*. Gateway *G1* delivers the packet to *G2* over network *B*. Gateway *G2* then delivers the packet directly to host *C1*, because they are both attached to network *C*. Host *A1* has no knowledge of any gateways beyond gateway *G1*. It sends packets destined for both networks *C* and *B* to that local gateway, and then relies on that gateway to properly forward the packets along the path to their destinations. Likewise, host *C1* would send its packets to *G2*, in order to reach a host on network *A*, as well as any host on network *B*.

Figure 1-7 shows another view of routing. This figure emphasizes that the underlying physical networks that a datagram travels through may be different and even incompatible. Host *A1* on the token ring network routes the datagram through gateway *G1*, to reach host *C1* on the Ethernet. Gateway *G1* forwards the data through the X.25 network to gateway *G2*, for delivery to *C1*. The datagram traverses three physically different networks, but eventually arrives intact at *C1*.

* In current terminology, a gateway moves data between different protocols and a router moves data between different networks. So a system that moves mail between TCP/IP and OSI is a gateway, but a traditional IP gateway is a router.

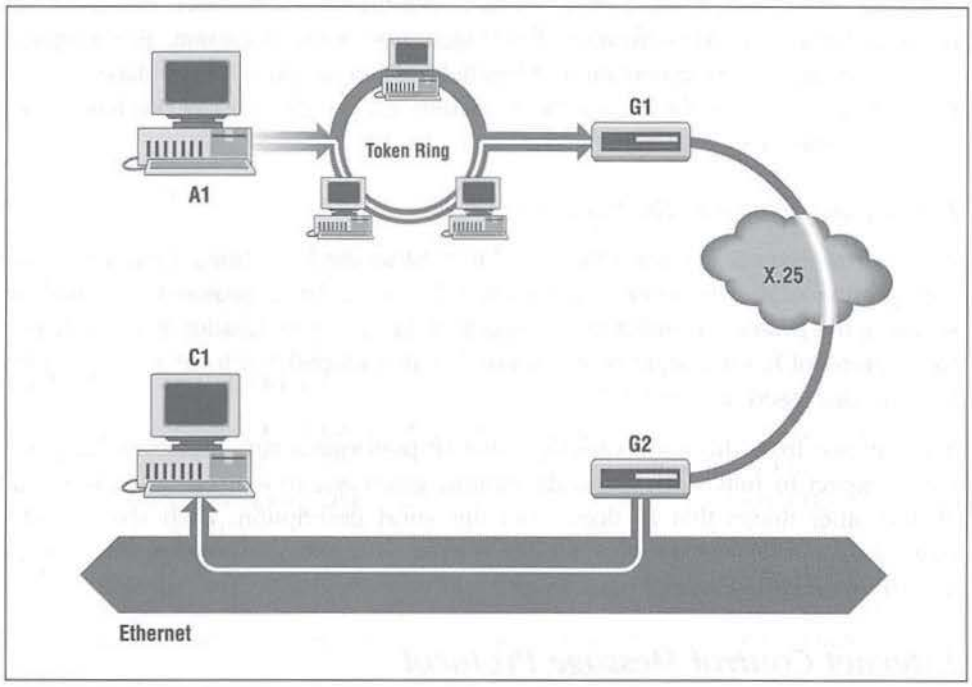


Figure 1-7: Networks, gateways, and hosts

Fragmenting datagrams

As a datagram is routed through different networks, it may be necessary for the IP module in a gateway to divide the datagram into smaller pieces. A datagram received from one network may be too large to be transmitted in a single packet on a different network. This condition occurs only when a gateway interconnects dissimilar physical networks.

Each type of network has a *maximum transmission unit* (MTU), which is the largest packet that it can transfer. If the datagram received from one network is longer than the other network's MTU, it is necessary to divide the datagram into smaller *fragments* for transmission. This process is called *fragmentation*. Think of a train delivering a load of steel. Each railway car can carry more steel than the trucks that will take it along the highway; so each railway car is unloaded onto many different trucks. In the same way that a railroad is physically different from a highway, an Ethernet is physically different from an X.25 network; IP must break an Ethernet's relatively large packets into smaller packets before it can transmit them over an X.25 network.

The format of each fragment is the same as the format of any normal datagram. Header word 2 contains information that identifies each datagram fragment and

provides information about how to re-assemble the fragments back into the original datagram. The Identification field identifies what datagram the fragment belongs to, and the Fragmentation Offset field tells what piece of the datagram this fragment is. The Flags field has a "More Fragments" bit that tells IP if it has assembled all of the datagram fragments.

Passing datagrams to the transport layer

When IP receives a datagram that is addressed to the local host, it must pass the data portion of the datagram to the correct Transport Layer protocol. This is done by using the *protocol number* from word 3 of the datagram header. Each Transport Layer protocol has a unique protocol number that identifies it to IP. Protocol numbers are discussed in Chapter 2.

You can see from this short overview that IP performs many important functions. Don't expect to fully understand datagrams, gateways, routing, IP addresses, and all the other things that IP does from this short description. Each chapter adds more details about these topics. So let's continue on with the other protocol in the TCP/IP Internet Layer.

Internet Control Message Protocol

An integral part of IP is the *Internet Control Message Protocol* (ICMP) defined in RFC 792. This protocol is part of the Internet Layer and uses the IP datagram delivery facility to send its messages. ICMP sends messages that perform the following control, error reporting, and informational functions for TCP/IP:

Flow control

When datagrams arrive too fast for processing, the destination host or an intermediate gateway sends an ICMP Source Quench Message back to the sender. This tells the source to stop sending datagrams temporarily.

Detecting unreachable destinations

When a destination is unreachable, the system detecting the problem sends a Destination Unreachable Message to the datagram's source. If the unreachable destination is a network or host, the message is sent by an intermediate gateway. But if the destination is an unreachable port, the destination host sends the message. (We discuss ports in Chapter 2.)

Redirecting routes

A gateway sends the ICMP Redirect Message to tell a host to use another gateway, presumably because the other gateway is a better choice. This message can be used only when the source host is on the same network as both gateways. To better understand this, refer to Figure 1-7. If a host on the X.25 network sent a datagram to *G1*, it would be possible for *G1* to redirect that host

to *G2* because the host, *G1*, and *G2* are all attached to the same network. On the other hand, if a host on the token ring network sent a datagram to *G1*, the host could not be redirected to use *G2*. This is because *G2* is not attached to the token ring.

Checking remote hosts

A host can send the ICMP Echo Message to see if a remote system's Internet Protocol is up and operational. When a system receives an echo message, it replies and sends the data from the packet back to the source host. The **ping** command uses this message.

Transport Layer

The protocol layer just above the Internet Layer is the *Host-to-Host Transport Layer*. This name is usually shortened to *Transport Layer*. The two most important protocols in the Transport Layer are *Transmission Control Protocol* (TCP) and *User Datagram Protocol* (UDP). TCP provides reliable data delivery service with end-to-end error detection and correction. UDP provides low-overhead, connectionless datagram delivery service. Both protocols deliver data between the Application Layer and the Internet Layer. Applications programmers can choose whichever service is more appropriate for their specific applications.

User Datagram Protocol

The User Datagram Protocol gives application programs direct access to a datagram delivery service, like the delivery service that IP provides. This allows applications to exchange messages over the network with a minimum of protocol overhead.

UDP is an unreliable, connectionless datagram protocol. As noted previously, "unreliable" merely means that there are no techniques in the protocol for verifying that the data reached the other end of the network correctly. Within your computer, UDP will deliver data correctly. UDP uses 16-bit *Source Port* and *Destination Port* numbers in word 1 of the message header, to deliver data to the correct applications process. Figure 1-8 shows the UDP message format.

Why do applications programmers choose UDP as a data transport service? There are a number of good reasons. If the amount of data being transmitted is small, the overhead of creating connections and ensuring reliable delivery may be greater than the work of re-transmitting the entire data set. In this case, UDP is the most efficient choice for a Transport Layer protocol. Applications that fit a *query-response* model are also excellent candidates for using UDP. The response can be used as a positive acknowledgment to the query. If a response isn't received within a certain time period, the application just sends another query. Still other

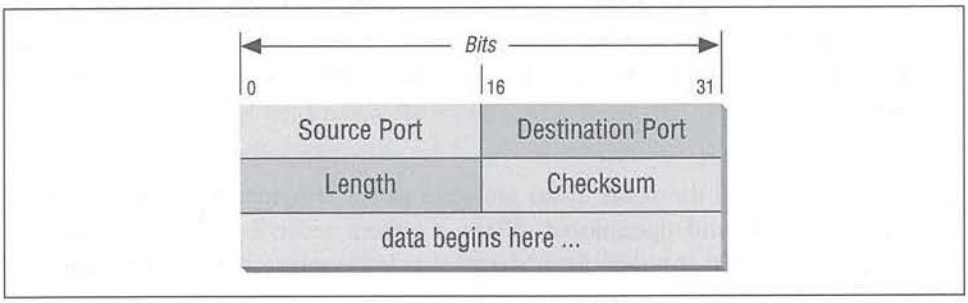


Figure 1-8: UDP message format

applications provide their own techniques for reliable data delivery, and don't require that service from the transport layer protocol. Imposing another layer of acknowledgment on any of these types of applications is inefficient.

Transmission Control Protocol

Applications that require the transport protocol to provide reliable data delivery use TCP because it verifies that data is delivered across the network accurately and in the proper sequence. TCP is a *reliable, connection-oriented, byte-stream* protocol. Let's look at each of the terms—reliable, connection-oriented, and byte-stream—in more detail.

TCP provides reliability with a mechanism called *Positive Acknowledgment with Re-transmission* (PAR). Simply stated, a system using PAR sends the data again, unless it hears from the remote system that the data arrived okay. The unit of data exchanged between cooperating TCP modules is called a *segment* (see Figure 1-9). Each segment contains a checksum that the recipient uses to verify that the data is undamaged. If the data segment is received undamaged, the receiver sends a *positive acknowledgment* back to the sender. If the data segment is damaged, the receiver discards it. After an appropriate time-out period, the sending TCP module re-transmits any segment for which no positive acknowledgment has been received.

TCP is connection-oriented. It establishes a logical end-to-end connection between the two communicating hosts. Control information, called a *handshake*, is exchanged between the two endpoints to establish a dialogue before data is transmitted. TCP indicates the control function of a segment by setting the appropriate bit in the Flags field in word 4 of the *segment header.*

The type of handshake used by TCP is called a *three-way handshake* because three segments are exchanged. Figure 1-10 shows the simplest form of the three-way handshake. Host *A* begins the connection by sending host *B* a segment with the "Synchronize sequence numbers" (SYN) bit set. This segment tells host *B* that

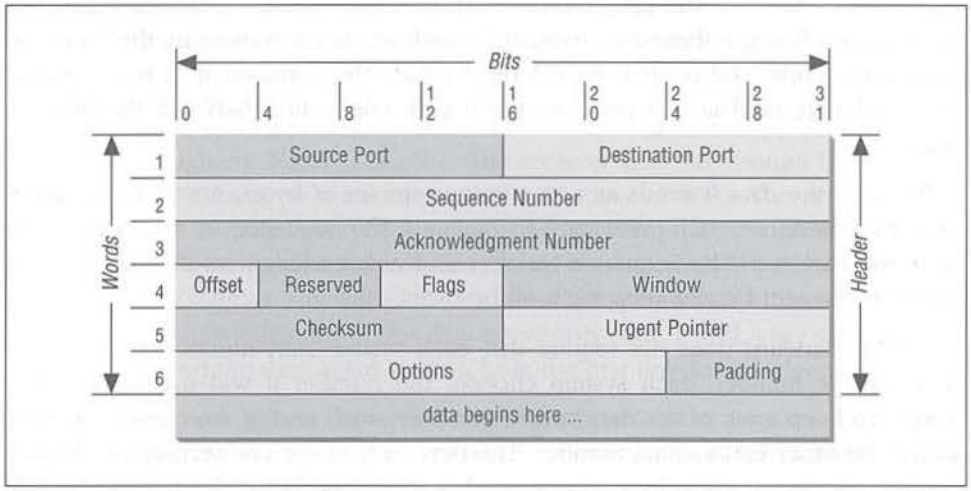


Figure 1-9: TCP segment format

A wishes to set up a connection, and it tells *B* what sequence number host *A* will use as a starting number for its segments. (Sequence numbers are used to keep data in the proper order.) Host *B* responds to *A* with a segment that has the "Acknowledgment" (ACK) and SYN bits set. *B*'s segment acknowledges the receipt of *A*'s segment, and informs *A* which Sequence Number host *B* will start with. Finally, host *A* sends a segment that acknowledges receipt of *B*'s segment, and transfers the first actual data.

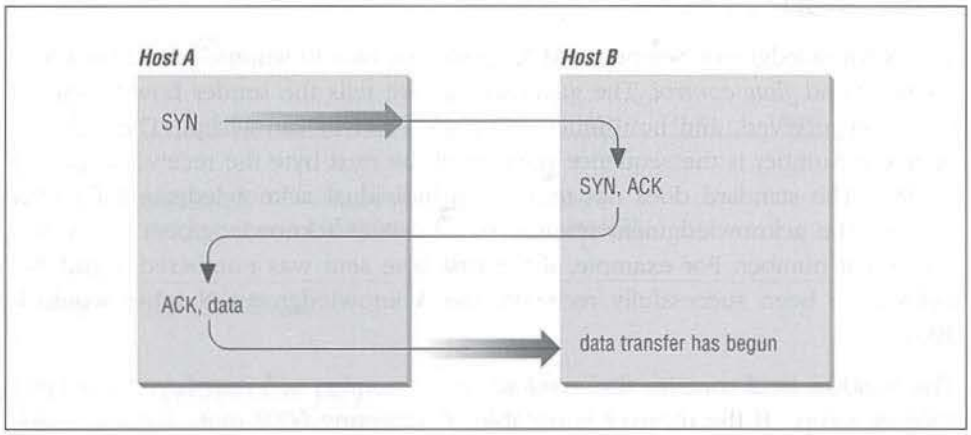


Figure 1-10: Three-way handshake

After this exchange, host *A*'s TCP has positive evidence that the remote TCP is alive and ready to receive data. As soon as the connection is established, data can

be transferred. When the cooperating modules have concluded the data transfers, they will exchange a three-way handshake with segments containing the "No more data from sender" bit (called the *FIN* bit) to close the connection. It is the end-to-end exchange of data that provides the logical connection between the two systems.

TCP views the data it sends as a continuous stream of bytes, not as independent packets. Therefore, TCP takes care to maintain the sequence in which bytes are sent and received. The Sequence Number and Acknowledgment Number fields in the TCP segment header keep track of the bytes.

The TCP standard does not require that each system start numbering bytes with any specific number; each system chooses the number it will use as a starting point. To keep track of the data stream correctly, each end of the connection must know the other end's initial number. The two ends of the connection synchronize byte-numbering systems by exchanging SYN segments during the handshake. The Sequence Number field in the SYN segment contains the *Initial Sequence Number* (ISN), which is the starting point for the byte-numbering system. For security reasons the ISN should be a random number, though it is often 0.

Each byte of data is numbered sequentially from the ISN, so the first real byte of data sent has a sequence number of ISN+1. The Sequence Number in the header of a data segment identifies the sequential position in the data stream of the first data byte in the segment. For example, if the first byte in the data stream was sequence number 1 (ISN=0) and 4000 bytes of data have already been transferred, then the first byte of data in the current segment is byte 4001, and the Sequence Number would be 4001.

The Acknowledgment Segment (ACK) performs two functions: *positive acknowledgment* and *flow control*. The acknowledgment tells the sender how much data has been received, and how much more the receiver can accept. The Acknowledgment Number is the sequence number of the next byte the receiver expects to receive. The standard does not require an individual acknowledgment for every packet. The acknowledgment number is a positive acknowledgment of all bytes up to that number. For example, if the first byte sent was numbered 1 and 2000 bytes have been successfully received, the Acknowledgment Number would be 2001.

The Window field contains the *window*, or the number of bytes the remote end is able to accept. If the receiver is capable of accepting 6000 more bytes, the window would be 6000. The window indicates to the sender that it can continue sending segments as long as the total number of bytes that it sends is smaller than the window of bytes that the receiver can accept. The receiver controls the flow of

bytes from the sender by changing the size of the window. A zero window tells the sender to cease transmission until it receives a non-zero window value.

Figure 1-11 shows a TCP data stream that starts with an Initial Sequence Number of 0. The receiving system has received and acknowledged 2000 bytes, so the current Acknowledgment Number is 2001. The receiver also has enough buffer space for another 6000 bytes, so it has advertised a window of 6000. The sender is currently sending a segment of 1000 bytes starting with Sequence Number 4001. The sender has received no acknowledgment for the bytes from 2001 on, but continues sending data as long as it is within the window. If the sender fills the window and receives no acknowledgment of the data previously sent, it will, after an appropriate time-out, send the data again starting from the first unacknowledged byte.

In Figure 1-11, re-transmission would start from byte 2001 if no further acknowledgments are received. This procedure ensures that data is reliably received at the far end of the network.

TCP is also responsible for delivering data received from IP to the correct application. The application that the data is bound for is identified by a 16-bit number called the *port number*. The *Source Port* and *Destination Port* are contained in the first word of the segment header. Correctly passing data to and from the Application Layer is an important part of what the Transport Layer services do.

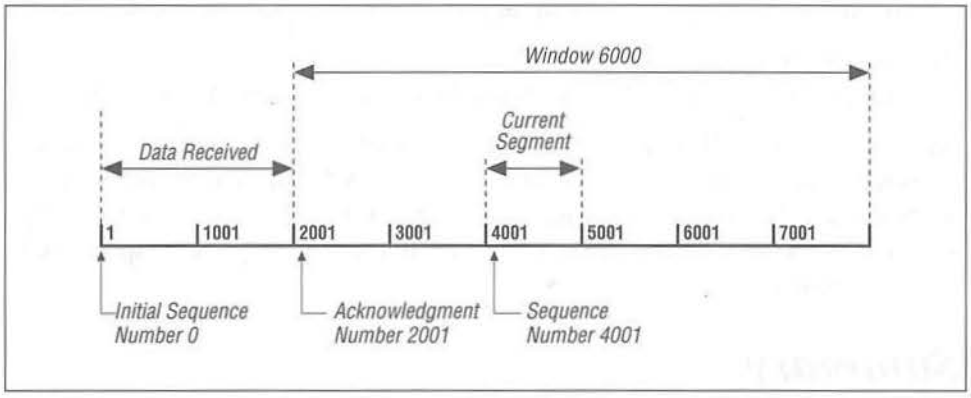


Figure 1-11: TCP data stream

Application Layer

At the top of the TCP/IP protocol architecture is the *Application Layer*. This layer includes all processes that use the Transport Layer protocols to deliver data. There are many applications protocols. Most provide user services, and new services are always being added to this layer.

The most widely known and implemented applications protocols are:

telnet
: The Network Terminal Protocol, which provides remote login over the network.

FTP
: The File Transfer Protocol, which is used for interactive file transfer.

SMTP
: The Simple Mail Transfer Protocol, which delivers electronic mail.

HTTP
: The Hypertext Transfer Protocol, which delivers Web pages over the network.

While HTTP, FTP, SMTP, and telnet are the most widely implemented TCP/IP applications, you will work with many others as both a user and a system administrator. Some other commonly used TCP/IP applications are:

Domain Name Service (DNS)
: Also called *name service*, this application maps IP addresses to the names assigned to network devices. DNS is discussed in detail in this book.

Open Shortest Path First (OSPF)
: Routing is central to the way TCP/IP works. OSPF is used by network devices to exchange routing information. Routing is also a major topic of this book.

Network Filesystem (NFS)
: This protocol allows files to be shared by various hosts on the network.

Some protocols, such as telnet and FTP, can only be used if the user has some knowledge of the network. Other protocols, like OSPF, run without the user even knowing that they exist. As system administrator, you are aware of all these applications and all the protocols in the other TCP/IP layers. And you're responsible for configuring them!

Summary

In this chapter we discussed the structure of TCP/IP, the protocol suite upon which the Internet is built. We have seen that TCP/IP is a hierarchy of four layers: Applications, Host-to-Host Transport, Internet, and Network Access. We have examined the function of each of these layers. In the next chapter we look at how the IP packet, the datagram, moves through a network when data is delivered between hosts.

In this chapter:

- *Addressing, Routing, and Multiplexing*
- *The IP Address*
- *Subnets*
- *Internet Routing Architecture*
- *The Routing Table*
- *Address Resolution*
- *Protocols, Ports, and Sockets*
- *Summary*

2

Delivering the Data

In Chapter 1, *Overview of TCP/IP*, we touched on the basic architecture and design of the TCP/IP protocols. From that discussion, we know that TCP/IP is a hierarchy of four layers. In this chapter, we explore in finer detail how data moves between the protocol layers and the systems on the network. We examine the structure of Internet addresses, including how addresses route data to its final destination, and how addressing rules are locally redefined to create subnets. We also look at the protocol and port numbers used to deliver data to the correct applications. These additional details move us from an overview of TCP/IP to the specific implementation details that affect your system's configuration.

Addressing, Routing, and Multiplexing

To deliver data between two Internet hosts, it is necessary to move the data across the network to the correct host, and within that host to the correct user or process. TCP/IP uses three schemes to accomplish these tasks:

Addressing
: IP addresses, which uniquely identify every host on the network, deliver data to the correct host.

Routing
: Gateways deliver data to the correct network.

Multiplexing
: Protocol and port numbers deliver data to the correct software module within the host.

Each of these functions—addressing between hosts, routing between networks, and multiplexing between layers—is necessary to send data between two

cooperating applications across the Internet. Let's examine each of these functions in detail.

To illustrate these concepts and provide consistent examples, we use an imaginary corporate network. Our imaginary company sells packaged nuts to the Army. Our company network is made up of several networks at our packing plant and sales office, as well as a connection to the Internet. We are responsible for managing the Ethernet in the computing center. This network's structure, or *topology*, is shown in Figure 2-1.

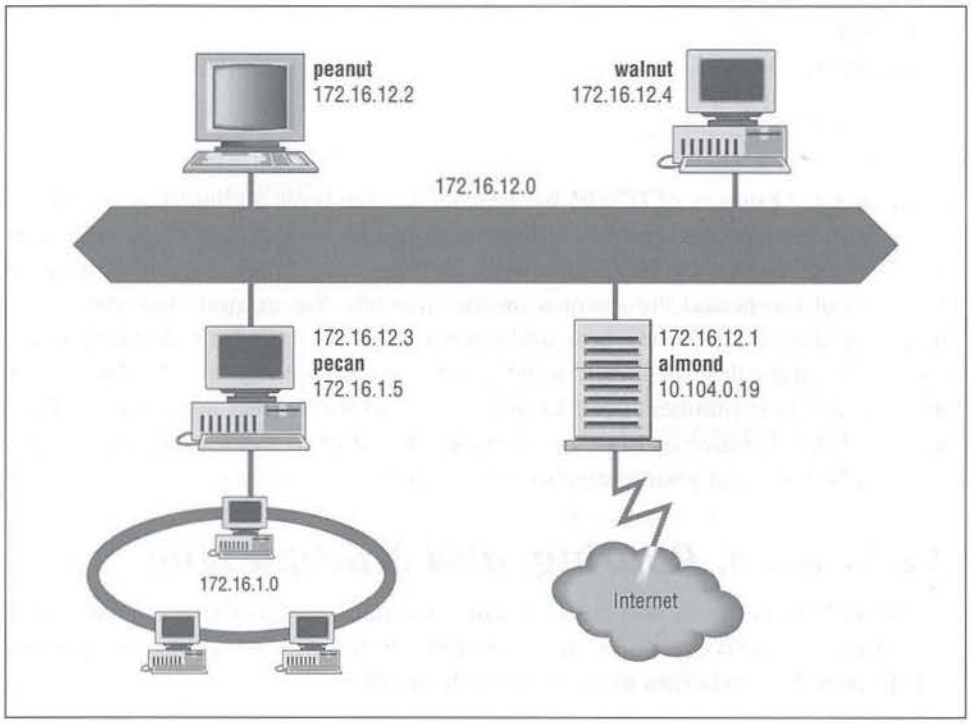


Figure 2-1: Sample network

The icons in the figure represent computer systems. There are, of course, several other imaginary systems on our imaginary network. You'll just have to use your imagination! But we'll use the hosts *peanut* (a workstation) and *almond* (a system that serves as a gateway) for most of our examples. The thick line is our computer center Ethernet and the circle is the local network that connects our various corporate networks. The cloud is the Internet. What the numbers are, how they're used, and how datagrams are delivered are the topics of this chapter.

The IP Address

The Internet Protocol moves data between hosts in the form of datagrams. Each datagram is delivered to the address contained in the Destination Address (word 5) of the datagram's header. The Destination Address is a standard 32-bit IP address that contains sufficient information to uniquely identify a network and a specific host on that network.

An IP address contains a *network part* and a *host part*, but the format of these parts is not the same in every IP address. The number of address bits used to identify the network, and the number used to identify the host, vary according to the prefix length of the address. There are two ways the prefix length is determined: by address class or by a CIDR address mask. We begin with a discussion of traditional IP address classes.

Address Classes

Originally, the IP address space was divided into a few fixed-length structures called *address classes*. The three main address classes are *class A*, *class B*, and *class C*. By examining the first few bits of an address, IP software can quickly determine the class, and therefore the structure, of an address. IP follows these rules to determine the address class:

- If the first bit of an IP address is 0, it is the address of a *class A network*. The first bit of a class A address identifies the address class. The next 7 bits identify the network, and the last 24 bits identify the host. There are fewer than 128 class A network numbers, but each class A network can be composed of millions of hosts.
- If the first 2 bits of the address are 1 0, it is a *class B network* address. The first 2 bits identify class; the next 14 bits identify the network, and the last 16 bits identify the host. There are thousands of class B network numbers and each class B network can contain thousands of hosts.
- If the first 3 bits of the address are 1 1 0, it is a *class C network* address. In a class C address, the first 3 bits are class identifiers; the next 21 bits are the network address, and the last 8 bits identify the host. There are millions of class C network numbers, but each class C network is composed of fewer than 254 hosts.
- If the first 4 bits of the address are 1 1 1 0, it is a multicast address. These addresses are sometimes called *class D* addresses, but they don't really refer to specific networks. Multicast addresses are used to address groups of computers all at one time. Multicast addresses identify a group of computers that

share a common application, such as a video conference, as opposed to a group of computers that share a common network.

- If the first four bits of the address are 1 1 1 1, it is a special reserved address. These addresses are sometimes called *class E* addresses, but they don't really refer to specific networks. No numbers are currently assigned in this range.

Luckily, this is not as complicated as it sounds. IP addresses are usually written as four decimal numbers separated by dots (periods).* Each of the four numbers is in the range 0–255 (the decimal values possible for a single byte). Because the bits that identify class are contiguous with the network bits of the address, we can lump them together and look at the address as composed of full bytes of network address and full bytes of host address. If the value of the first byte is:

- Less than 128, the address is class A; the first byte is the network number, and the next three bytes are the host address.
- From 128 to 191, the address is class B; the first two bytes identify the network, and the last two bytes identify the host.
- From 192 to 223, the address is class C; the first three bytes are the network address, and the last byte is the host number.
- From 224 to 239, the address is multicast. There is no network part. The entire address identifies a specific multicast group.
- Greater than 239, the address is reserved. We can ignore reserved addresses.

Figure 2-2 illustrates how the address structure varies with address class. The class A address is 10.104.0.19. The first bit of this address is 0, so the address is interpreted as host 104.0.19 on network 10. One byte specifies the network and three bytes specify the host. In the address 172.16.12.1, the two high-order bits are 1 0 so the address refers to host 12.1 on network 172.16. Two bytes identify the network and two identify the host. Finally, in the class C example, 192.168.16.1, the three high-order bits are 1 1 0, so this is the address of host 1 on network 192.168.16—three network bytes and one host byte.

The IP address, which provides universal addressing across all of the networks of the Internet, is one of the great strengths of the TCP/IP protocol suite. However, the original class structure of the IP address has weaknesses. The TCP/IP designers did not envision the enormous scale of today's network. When TCP/IP was being designed, networking was limited to large organizations that could afford substantial computer systems. The idea of a powerful UNIX system on every desktop did not exist. At that time, a 32-bit address seemed so large that it was divided into

* Addresses are occasionally written in other formats, e.g., as hexadecimal numbers. However, the "dot" notation form is the most widely used. Whatever the notation, the structure of the address is the same.

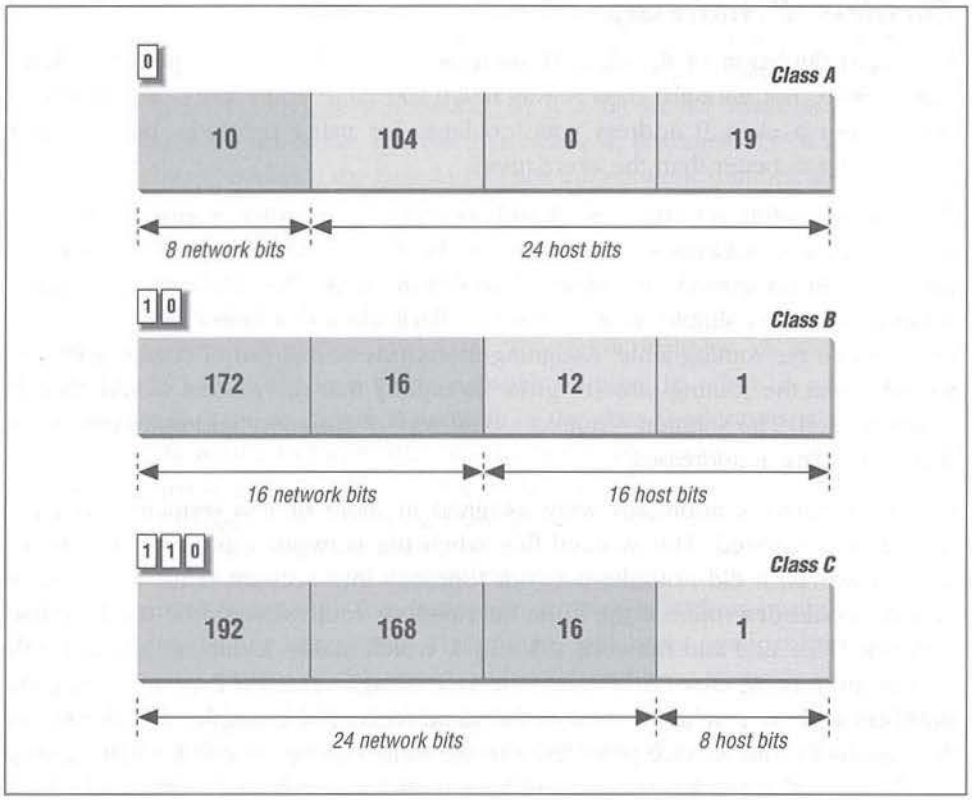


Figure 2-2: IP address structure

classes to reduce the processing load on routers, even though dividing the address into classes sharply reduced the number of host addresses actually available for use. For example, assigning a large network a single class B address, instead of six class C addresses, reduced the load on the router because the router needed to keep only one route for that entire organization. However, an organization that was given the class B address probably did not have 64,000 computers, so most of the host addresses available to the organization were never assigned.

The class-structured address design was critically strained by the rapid growth of the Internet. At one point it appeared that all class B addresses might be rapidly exhausted.* To prevent this, a new way of looking at IP addresses without a class structure was developed.

* The source for this prediction is the draft of *Supernetting: an Address Assignment and Aggregation Strategy*, by V. Fuller, T. Li, J. Yu, and K. Varadhan, March 1992.

Classless IP Addresses

The rapid depletion of the class B addresses showed that three primary address classes were not enough: class A was much too large and class C was much too small. Even a class B address was too large for many networks but was used because it was better than the alternatives.

The obvious solution to the class B address crisis was to force organizations to use multiple class C addresses. There were millions of these addresses available and they were in no immediate danger of depletion. As is often the case, the obvious solution is not as simple as it may seem. Each class C address requires its own entry within the routing table. Assigning thousands or millions of class C addresses would cause the routing table to grow so rapidly that the routers would soon be overwhelmed. The solution required a new way of assigning addresses and a new way of looking at addresses.

Originally network addresses were assigned in more or less sequential order as they were requested. This worked fine when the network was small and centralized. However, it did not take network topology into account. Thus only random chance would determine if the same intermediate routers would be used to reach network 195.4.12.0 and network 195.4.13.0, which makes it difficult to reduce the size of the routing table. Addresses can only be aggregated if they are contiguous numbers and are reachable through the same route. For example, if addresses are contiguous for one service provider, a single route can be created for that aggregation because that service provide will have a limited number of routes to the Internet. But if one network address is in France and the next contiguous address is in Australia, creating a consolidated route for these addresses does not work.

Today, large, contiguous blocks of addresses are assigned to large network service providers in a manner that better reflects the topology of the network. The service providers then allocate chunks of these address blocks to the organizations to which they provide network services. This alleviates the short-term shortage of class B addresses and, because the assignment of addressees reflects the topology of the network, it permits route aggregation. Under this new scheme, we know that network 195.4.12.0 and network 195.4.13.0 are reachable through the same intermediate routers. In fact, both of these addresses are in the range of the addresses assigned to Europe, 194.0.0.0 to 195.255.255.255. Assigning addresses that reflect the topology of the network enables route aggregation, but does not implement it. As long as network 195.4.12.0 and network 195.4.13.0 are interpreted as separate class C addresses, they will require separate entries in the routing table. A new, flexible way of defining addresses is needed.

Evaluating addresses according to the class rules discussed above limits the length of network numbers to 8, 16, or 24 bits—1, 2, or 3 bytes. The IP address,

however, is not really byte-oriented. It is 32 contiguous bits. A more flexible way to interpret the network and host portions of an address is with a *bit mask*. An address bit mask works in this way: if a bit is on in the mask, that equivalent bit in the address is interpreted as a network bit; if a bit in the mask is off, the bit belongs to the host part of the address. For example, if address 195.4.12.0 is interpreted as a class C address, the first 24 bits are the network number and the last 8 bits are the host address. The network mask that represents this is 255.255.255.0, 24 bits on and 8 bits off. The bit mask that is derived from the traditional class structure is called the *default mask* or the *natural mask*. However, with bit masks we are no longer limited by the address class structure. A mask of 255.255.0.0 can be applied to network address 195.4.0.0. This mask includes all addresses from 195.4.0.0 to 195.4.255.255 in a single network number. In effect, it creates a network number as large as a class B network in the class C address space. Using bit masks to create networks larger than the natural mask is called *supernetting*, and the use of a mask instead of the address class to determine the destination network is called *Classless Inter-Domain Routing* (CIDR).*

CIDR requires modifications to the routers and routing protocols. The protocols need to distribute, along with the destination addresses, address masks that define how the addresses are interpreted. The routers and hosts need to know how to interpret these addresses as "classless" addresses and how to apply the bit mask that accompanies the address. Older routing protocols, such as *Routing Information Protocol* (RIP), and older operating systems do not support CIDR address masks. As the incorporation of the mask information in the routing table shows, new operating systems like Linux 2.0.0 do support CIDR.

```
# route
Kernel routing table
Destination  Gateway      Genmask          Flags MSS    Window Use Iface
172.16.26.32 *            255.255.255.224 U     1500   0        2 eth0
195.4.0.0    129.6.26.62  255.255.0.0     UG    1500   0        0 eth0
loopback     *            255.0.0.0       U     3584   0        1 lo
default      129.6.26.62  *               UG    1500   0        3 eth0
```

Specifying both the address and the mask is cumbersome when writing out addresses. A shorthand notation has been developed for writing CIDR addresses. Instead of writing network 172.16.26.32 with a mask of 255.255.255.224, we can write 172.16.26.32/27. The format of this notation is *address/prefix-length*, where *prefix-length* is the number of bits in the network portion of the address. Without this notation, the address 172.16.26.32 could easily be interpreted as a host address. RFC 1878 list all 32 possible prefix values. But little documentation is needed because the CIDR prefix is much easier to understand and remember than are address classes. I know that 10.104.0.19 is a class A address, but writing it as

* CIDR is pronounced "cider."

10.104.0.19/8 shows me that this address has 8 bits for the network number and therefore 24 bits for the host number. I don't have to remember anything about the class A address structure.

CIDR is an interim solution, though it is capable of providing address and routing relief for many more years. The long-term solution is to replace the current addressing scheme with a new one. In the TCP/IP protocol suite addressing is defined by the IP protocol. Therefore, to define a new address structure, the Internet Engineering Task Force (IETF) created a new version of IP called IPv6.* IPv6 has a very large 128-bit address, so address depletion is not an issue. The large address also makes it possible to use a hierarchical address structure to reduce the burden on routers while still maintaining more than enough addresses for future network growth. Other benefits of IPv6 are:

- Improved security built into the protocol
- Simplified, fixed-length, word-aligned headers to speed header processing and reduce overhead
- Improved techniques for handling header options

IPv6 has several good features, but it is still a few years from widespread availability. In the meantime, the current generation of TCP/IP should be more than adequate for your network needs. On your network you will use IP and standard IP addressing.

Final notes on IP addresses

Not all network addresses or host addresses are available for use. We have already said that the addresses with a first byte greater than 223 cannot be used as host addresses. There are also two large pieces of the address space, 0.0.0.0/8 and 127.0.0.0/8, that are reserved for special uses. Network 0 designates the *default route* and network 127 is the *loopback address*. The default route is used to simplify the routing information that IP must handle. The loopback address simplifies network applications by allowing the local host to be addressed in the same manner as a remote host. We use these special network addresses when configuring a host.

There are also some host addresses reserved for special uses. In all network classes, host numbers 0 and 255 are reserved. An IP address with all host bits set to 0 identifies the network itself. For example, 10.0.0.0 refers to network 10, and 172.16.0.0 refers to network 172.16. Addresses in this form are used in routing table listings to refer to entire networks. An IP address with all host bits set to 1 is

* The current release of IP is IP version 4 (IPv4). IP version 5 is an experimental Stream Transport (ST) protocol used for real-time data delivery.

a *broadcast address.** A broadcast address is used to simultaneously address every host on a network. The broadcast address for network 172.16 is 172.16.255.255. A datagram sent to this address is delivered to every individual host on network 172.16.

IP addresses are often called host addresses. While this is common usage, it is slightly misleading. IP addresses are assigned to network interfaces, not to computer systems. A gateway, such as *almond* (see Figure 2-1), has a different address for each network to which it is connected. The gateway is known to other devices by the address associated with the network that it shares with those devices. For example, *peanut* addresses *almond* as 172.16.12.1, while external hosts address it as 10.104.0.19.

Systems can be addressed in three different ways. Individual systems are directly addressed by a host address, which is called a *unicast address.* A unicast packet is addressed to one individual host. Groups of systems can be addressed using a *multicast address,* e.g., 224.0.0.9. Routers along the path from the source to destination recognize the special address and route copies of the packet to each member of the multicast group.† All systems on a network are addressed using the broadcast address, e.g., 172.16.255.255. The broadcast address depends on the broadcast capabilities of the underlying physical network.

IP uses the network portion of the address to route the datagram between networks. The full address, including the host information, is used to make final delivery when the datagram reaches the destination network.

Subnets

The structure of an IP address can be locally modified by using host address bits as additional network address bits. Essentially, the "dividing line" between network address bits and host address bits is moved, creating additional networks, but reducing the maximum number of hosts that can belong to each network. These newly designated network bits define a network within the larger network, called a *subnet.*

Organizations usually decide to subnet in order to overcome topological or organizational problems. Subnetting allows decentralized management of host addressing. With the standard addressing scheme, a central administrator is responsible for managing host addresses for the entire network. By subnetting, the administrator can delegate address assignment to smaller organizations within the overall

* Unfortunately, there are implementation-specific variations in broadcast addresses. Chapter 5, *Basic Configuration,* discusses these variations.

† This is only partially true. Multicasting is not supported by every router. Sometimes it is necessary to tunnel through routers and networks by encapsulating the multicast packet inside of a unicast packet.

organization—which may be a political expedient, if not a technical requirement. If you don't want to deal with the data processing department, assign them their own subnet and let them manage it themselves.

Subnetting can also be used to overcome hardware differences and distance limitations. IP routers can link dissimilar physical networks together, but only if each physical network has its own unique network address. Subnetting divides a single network address into many unique subnet addresses, so that each physical network can have its own unique address.

A subnet is defined by changing the bit mask of the IP address. A *subnet mask* functions in the same way as a normal address mask: an "on" bit is interpreted as a network bit; an "off" bit belongs to the host part of the address. The difference is that a subnet mask is only used locally. In the outside world the address is still interpreted as a standard IP address.

Assume we have been assigned network address 172.16.0.0/16. The subnet mask associated with that address is 255.255.0.0. The most commonly used subnet mask, and the one we use in most of our examples, extends the network portion of the address by an additional byte, e.g., 172.16.0.0/24. The subnet mask that does this is 255.255.255.0; all bits on in the first three bytes, and all bits off in the last byte. The first two bytes define the original network; the third byte defines the the subnet address; the fourth byte defines the host on that subnet.

Many network administrators prefer byte-oriented masks because they are easy to read and understand when addresses are written in dotted decimal notation. However, limiting subnet masks to byte boundaries does not take advantage of their true power. The subnet mask is bit-oriented. We could subdivide 172.16.0.0/16 into 16 subnets with the mask 255.255.240.0, i.e. 172.16.0.0/20. Applying this mask defines the four high-order bits of the third byte as the subnet part of the address, and the remaining 12 bits—four bits of the third byte and all of the fourth byte—as the host portion of the address. This creates 16 subnets that each contain more than four thousand host addresses, which may well be better suited to our network and organization. For example, we may have a small number of large subdivisions. Table 2-1 shows the subnets and host addresses produced by applying this subnet masks to network address 172.16.0.0/16.

Table 2-1: Effect of a Subnet Mask

Network Number	First Address	Last Address
172.16.0.0	172.16.0.1	172.16.15.254
172.16.16.0	172.16.16.1	172.16.31.254
172.16.32.0	172.16.32.1	172.16.47.254
172.16.48.0	172.16.48.1	172.16.63.254

Table 2-1: Effect of a Subnet Mask (continued)

Network Number	First Address	Last Address
172.16.64.0	172.16.64.1	172.16.79.254
172.16.80.0	172.16.80.1	172.16.95.254
172.16.96.0	172.16.96.1	172.16.111.254
172.16.112.0	172.16.112.1	172.16.127.254
172.16.128.0	172.16.128.1	172.16.143.254
172.16.144.0	172.16.144.1	172.16.159.254
172.16.160.0	172.16.160.1	172.16.175.254
172.16.176.0	172.16.176.1	172.16.191.254
172.16.192.0	172.16.192.1	172.16.207.254
172.16.208.0	172.16.208.1	172.16.223.254
172.16.224.0	172.16.224.1	172.16.239.254
172.16.240.0	172.16.240.1	172.16.254.254

You don't have to manually calculate a table like Table 2-1 to know what subnets and host addresses are produced by a subnet mask. The calculations have already been done for you. RFC 1878 lists all possible subnet masks and the valid addresses they produce.

Organizations have been discouraged from subnetting class C addresses because of the fear that subnetting reduces the number of host addresses to increase the number of network addresses. A class C network is limited to fewer than 255 host addresses. Further limiting the number of hosts would reduce the utility of a class C address. The mask 255.255.255.192 divides a class C address into four subnets of 64 host addresses. The fear is that the subnet address of all 0s and the subnet address of all 1s will not be usable. This leaves only two subnets; and because host addresses of all 1s and all 0s are also unusable, the remaining two subnets can only address 62 hosts. Therefore the address space of this class C network number is reduced from 254 hosts to 124 hosts. The fear of subnetting class C addresses is no longer justified.

Originally, the RFCs implied that you should not use subnet numbers of all 0s or all 1s. However, RFC 1812, *Requirements for IP Version 4 Routers*, makes it clear that subnets of all 0s and all 1s are legal and should be supported by all routers. Some older routers do not allow the use of these addresses despite the newer RFCs. Updating router software or hardware should make it possible for you to reliably subnet class C addresses.

Class C subnets are used when very small networks are needed for specialized network equipment, such as terminal servers, cluster controllers or routers. In some configurations an entire subnet may be consumed for the link between two routers. In this case only two host addresses are needed, one for the router at each end of the link. A subnet mask of 255.255.255.252 applied to a class C address

creates 64 subnets each containing four host addresses. In a special case this might be just what is needed.

Internet Routing Architecture

Chapter 1 described the evolution of the Internet architecture over the years. Along with these architectural changes have come changes in the way that routing information is disseminated within the network.

In the original Internet structure, there was a hierarchy of gateways. This hierarchy reflected the fact that the Internet was built upon the existing ARPANET. When the Internet was created, the ARPANET was the backbone of the network: a central delivery medium to carry long-distance traffic. This central system was called the *core*, and the centrally managed gateways that interconnected it were called the *core gateways*.

In that hierarchical structure, routing information about all of the networks in the Internet was passed into the core gateways. The core gateways processed the information, and then exchanged it among themselves using the *Gateway to Gateway Protocol* (GGP). The processed routing information was then passed back out to the external gateways. The core gateways maintained accurate routing information for the entire Internet.

Using the hierarchical core router model to distribute routing information has a major weakness: every route must be processed by the core. This places a tremendous processing burden on the core, and as the Internet grew larger the burden increased. In network-speak, we say that this routing model does not "scale well." For this reason, a new model emerged.

Even in the days of a single Internet, core groups of independent networks called *autonomous systems* (AS) existed outside of the core. The term "autonomous system" has a formal meaning in TCP/IP routing. An autonomous system is not merely an independent network. It is a collection of networks and gateways with its own internal mechanism for collecting routing information and passing it to other independent network systems. The routing information passed to the other network systems is called *reachability information*. Reachability information simply says which networks can be reached through that autonomous system. The *Exterior Gateway Protocol* (EGP) was the protocol used to pass reachability information between autonomous systems and into the core (see Figure 2-3).

The new routing model is based on co-equal collections of autonomous systems, called *routing domains*. Routing domains exchange routing information with other domains using *Border Gateway Protocol* (BGP). Each routing domain processes the information it receives from other domains. Unlike the hierarchical model, this model does not depend on a single core system to choose the "best" routes. Each

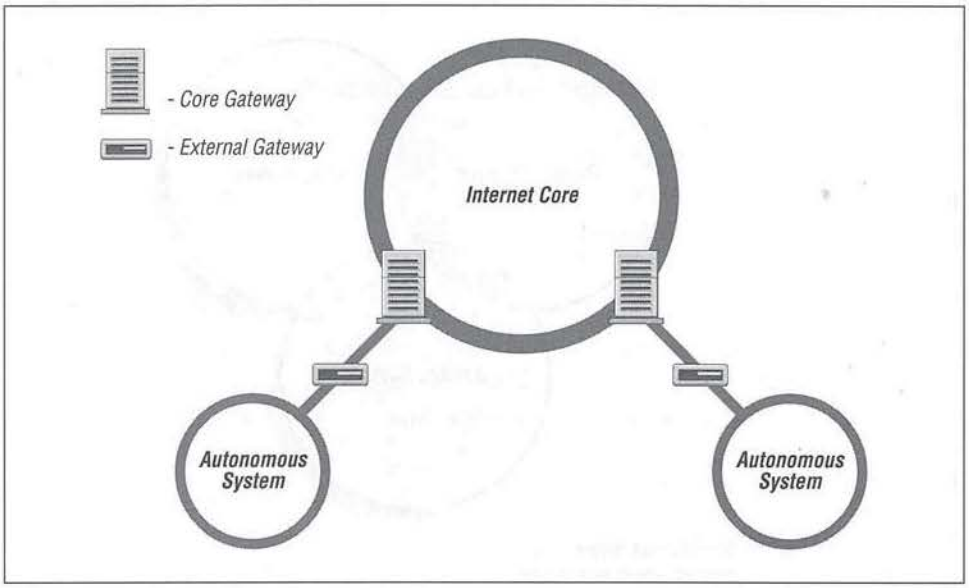


Figure 2-3: Gateway hierarchy

routing domain does this processing for itself; therefore, this model is more expandable. Figure 2-4 represents this model with three intersecting circles. Each circle is a routing domain. The overlapping areas are border areas, where routing information is shared. The domains share information, but do not rely on any one system to provide all routing information.

The problem with this model is: how are "best" routes determined in a global network if there is no central routing authority, like the core, that is trusted to determine the "best" routes? In the days of the NSFNET, the *policy routing database* (PRDB) was used to determine whether the reachability information advertised by an autonomous system was valid. But now, even the NSFNET does not play a central role.

To fill this void, NSF created the *Routing Arbiter* (RA) servers when it created the *Network Access Points* (NAPs) that replaced the role of the NSFNET. A route arbiter is located at each NAP. The server provides access to the *Routing Arbiter Database* (RADB), which replaced the PRDB. Internet Service Providers can query servers to validate the reachability information advertised by an autonomous system.

Many ISPs do not use the route servers. Instead they depend on formal and informal bilateral agreements. In essence, two ISPs get together and decide what reachability information each will accept from the other. They create, in effect, local routing policies. This is a slow manual process that probably will not be flexible enough for a rapidly growing Internet.

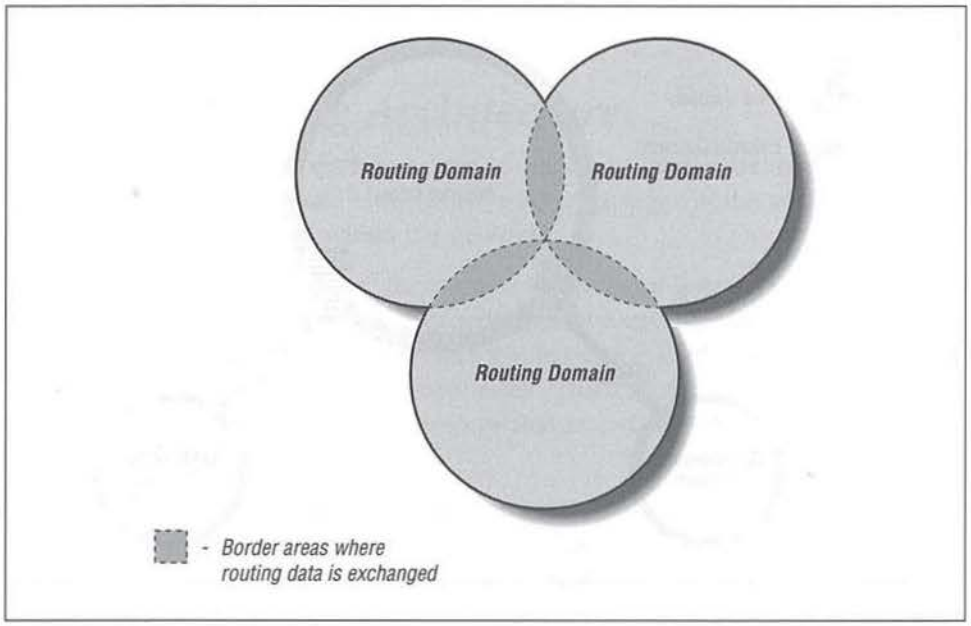


Figure 2-4: Routing domains

The RADB is only part of the *Internet Routing Registry* (IRR). As befits a distributed routing architecture, there are multiple organizations that validate and register routing information. Europeans were the pioneers in this. The Reseaux IP Europeens (RIPE) Network Control Center (NCC) provides the routing registry for European IP networks. Big network carriers, like MCI and ANS, provide registries for their customers. All of the registries share a common format based on the RIPE-181 standard.

Creating an effective routing architecture continues to be a major challenge for the Internet that will certainly evolve over time. No matter how it is derived, eventually the routing information winds up in your local gateway, where it is used by IP to make routing decisions.

The Routing Table

Gateways route data between networks; but all network devices, hosts as well as gateways, must make routing decisions. For most hosts, the routing decisions are simple:

- If the destination host is on the local network, the data is delivered to the destination host.

- If the destination host is on a remote network, the data is forwarded to a local gateway.

Because routing is network-oriented, IP makes routing decisions based on the network portion of the address. The IP module determines the network part of the destination's IP address by applying the network mask to the address. If the destination network is the local network, the mask that is applied may be the local subnet mask. If no mask is provided with the address, the address class determines the network portion of the address.

After determining the destination network, the IP module looks up the network in the local *routing table.** Packets are routed toward their destination as directed by the routing table. The routing table may be built by the system administrator or by routing protocols, but the end result is the same; IP routing decisions are simple table look-ups.

You can display the routing table's contents with the **netstat –nr** command. The **–r** option tells **netstat** to display the routing table, and the **–n** option tells **netstat** to display the table in numeric form. It's useful to display the routing table in numeric form because the destination of most routes is a network, and networks are usually referred to by network numbers.

On a Solaris system, the **netstat** command displays the routing table with the following fields:

Destination
The destination network (or host).

Gateway
The gateway to use to reach the specified destination.

Flags
The flags describe certain characteristics of this route. The possible flag values are:

U Indicates that the route is up and operational.

H Indicates this is a route to a specific host (most routes are to networks).

G Means the route uses a gateway. The system's network interfaces provide routes to directly connected networks. All other routes use remote gateways. Directly connected networks do not have the G flag set; all other routes do.

D Means that this route was added because of an ICMP Redirect Message. When a system learns of a route via an ICMP Redirect, it adds the route to

* This table is also called the *forwarding table.*

its routing table, so that additional packets bound for that destination will not need to be redirected. The system uses the D flag to mark these routes.

Ref
The number of times the route has been referenced to establish a connection.

Use
The number of packets transmitted via this route.

Interface
The name of the network interface* used by this route.

The only two fields important for our current discussion are the destination and gateway fields. The following is a sample routing table:

```
% netstat -nr
Routing Table:
Destination  Gateway        Flags  Ref    Use   Interface
-----------  -----------    -----  ----  -----  ---------
127.0.0.1    127.0.0.1      UH       1    298       lo0
default      172.16.12.1    UG       2  50360
172.16.12.0  172.16.12.2    U       40 111379       le0
172.16.2.0   172.16.12.3    UG       4   1179
172.16.1.0   172.16.12.3    UG      10   1113
172.16.3.0   172.16.12.3    UG       2   1379
172.16.4.0   172.16.12.3    UG       4   1119
```

The first table entry is the *loopback route* for the local host. This is the loopback address mentioned earlier as a reserved network number. Because every system uses the loopback route to send datagrams to itself, this entry is in every host's routing table. The H flag is set because it is a route to a specific host (127.0.0.1), not a route to an entire network (127.0.0.0). We'll see the loopback facility again when we discuss kernel configuration and the **ifconfig** command. For now, however, our real interest is in external routes.

Another unique entry in the routing table is the entry with the word "default" in the destination field. This entry is for the *default route*, and the gateway specified in this entry is the *default gateway*. The default route is the other reserved network number mentioned earlier: 0.0.0.0. The default gateway is used whenever there is no specific route in the table for a destination network address. For example, this routing table has no entry for network 192.168.16.0. If IP receives any datagrams addressed to this network, it will send the datagram via the default gateway 172.16.12.1.

* The network interface is the network access hardware and software that IP uses to communicate with the physical network. See Chapter 6, *Configuring the Interface*, for details.

You can tell from the sample routing table display that this host (*peanut*) is directly connected to network 172.16.12.0. The routing table entry for that network does not specify an external gateway; i.e., the routing table entry for 172.16.12.0 does not have the G flag set. Therefore, *peanut* must be directly connected to that network.

All of the gateways that appear in a routing table are on networks directly connected to the local system. In the sample shown above this means that, regardless of the destination address, the gateway addresses all begin with 172.16.12. This is the only network to which *peanut* is directly attached, and therefore it is the only network to which *peanut* can directly deliver data. The gateways that *peanut* uses to reach the rest of the Internet must be on *peanut*'s subnet.

In Figure 2-5 the IP layer of each host and gateway on our imaginary network is replaced by a small piece of a routing table, showing destination networks and the gateways used to reach those destinations. When the source host (172.16.12.2) sends data to the destination host (172.16.1.2), it first determines that 172.16.1.2 is the local network's official address and applies the subnet mask. (Network 172.16.0.0 is subnetted using the mask 255.255.255.0.) After applying the subnet mask, IP knows that the destination's network address is 172.16.1.0. The routing table in the source host shows that data bound for 172.16.1.0 should be sent to gateway 172.16.12.3. Gateway 172.16.12.3 makes direct delivery through its 172.16.1.5 interface. Examining the routing tables shows that all systems list only gateways on networks they are directly connected to. Note that 172.16.12.1 is the default gateway for both 172.16.12.2 and 172.16.12.3. But because 172.16.1.2 cannot reach network 172.16.12.0 directly, it has a different default route.

A routing table does not contain end-to-end routes. A route points only to the next gateway, called the *next hop*, along the path to the destination network.* The host relies on the local gateway to deliver the data, and the gateway relies on other gateways. As a datagram moves from one gateway to another, it should eventually reach one that is directly connected to its destination network. It is this last gateway that finally delivers the data to the destination host.

Address Resolution

The IP address and the routing table direct a datagram to a specific physical network, but when data travels across a network, it must obey the physical layer protocols used by that network. The physical networks that underlay the TCP/IP network do not understand IP addressing. Physical networks have their own addressing schemes, and there are as many different addressing schemes as there

* As we'll see in Chapter 7, *Configuring Routing*, some routing protocols, such as OSPF and BGP, obtain end-to-end routing information. Nevertheless, the packet is still passed to the next-hop router.

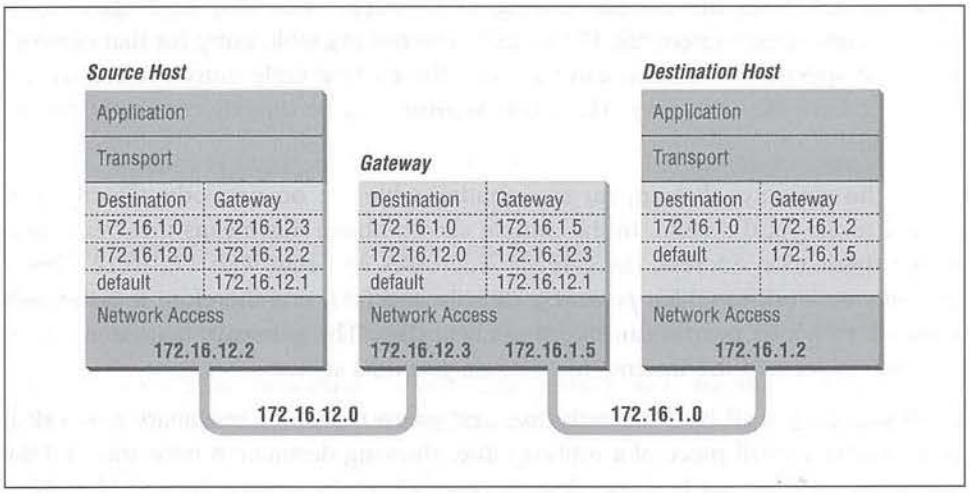


Figure 2-5: Table-based routing

are different types of physical networks. One task of the network access protocols is to map IP addresses to physical network addresses.

The most common example of this network access layer function is the translation of IP addresses to Ethernet addresses. The protocol that performs this function is *Address Resolution Protocol* (ARP), which is defined in RFC 826.

The ARP software maintains a table of translations between IP addresses and Ethernet addresses. This table is built dynamically. When ARP receives a request to translate an IP address, it checks for the address in its table. If the address is found, it returns the Ethernet address to the requesting software. If the address is not found in the table, ARP broadcasts a packet to every host on the Ethernet. The packet contains the IP address for which an Ethernet address is sought. If a receiving host identifies the IP address as its own, it responds by sending its Ethernet address back to the requesting host. The response is then cached in the ARP table.

The **arp** command displays the contents of the ARP table. To display the entire ARP table, use the **arp –a** command. Individual entries can be displayed by specifying a hostname on the **arp** command line. For example, to check the entry for *peanut* in the ARP table on *almond*, enter:

```
% arp peanut
peanut (172.16.12.2) at 8:0:20:0:e:c8
```

Checking all entries in the table with the **–a** option produces the following output:

```
% arp -a
Net to Media Table
```

```
Device   IP Address               Mask      Flags   Phys Addr
------ ----------------- --------------- ----- ---------------
le0    peanut.nuts.com   255.255.255.255       08:00:20:00:0e:c8
le0    acorn.nuts.com    255.255.255.255       08:00:02:05:21:33
le0    almond.nuts.com   255.255.255.255 SP    08:00:20:22:fd:51
le0    pecan.nuts.com    255.255.255.255       00:20:af:1e:7e:5f
le0    BASE-ADDRESS.MCAST.NET 240.0.0.0       SM    01:00:5e:00:00:00
```

This table tells you that when *almond* forwards datagrams addressed to *peanut*, it puts those datagrams into Ethernet frames and sends them to Ethernet address 08:00:20:00:0e:c8.

Three of the entries in the sample table (*peanut*, *acorn*, and *pecan*) were added dynamically as a result of queries by *almond*. Two of the entries (*almond* and *BASE-ADDRESS.MCAST.NET*) are static entries added as a result of the configuration of *almond*. We know this because both of these entries have an S, for "static," in the Flags field. The special *BASE-ADDRESS.MCAST.NET* entry is for all multicast addresses. The M flag means "mapping" and is only used for the multicast entry. On a broadcast medium like Ethernet, the Ethernet broadcast address is used to make final delivery to a multicast group.

The P flag on the *almond* entry means that this entry will be "published." The "publish" flag indicates that when an ARP query is received for the IP address of *almond*, this system answers it with the Ethernet address 08:00:20:22:fd:51. This is logical because this is the ARP table on *almond*. However, it is also possible to publish Ethernet addresses for other hosts, not just for the local host. Answering ARP queries for other computers is called *proxy ARP*.

For example: assume that *acorn* is the server for a remote system named *hazel* connected via a dial-up telephone line. Instead of setting up routing to the remote system, the administrator of *acorn* could place a static, published entry in the ARP table with the IP address of *hazel* and the Ethernet address of *acorn*. Now when *acorn* hears an ARP query for the IP address of *hazel*, it answers with its own Ethernet address. The other systems on the network therefore send packets destined for *hazel* to *acorn*. *acorn* then forwards the packets on to *hazel* over the telephone line. Proxy ARP is used to answer queries for systems that can't answer for themselves.

ARP tables normally don't require any attention because they are built automatically by the ARP protocol, which is very stable. However, if things go wrong, the ARP table can be manually adjusted. See Chapter 11, *Troubleshooting TCP/IP*, the section called "Troubleshooting with the arp Command."

Protocols, Ports, and Sockets

Once data is routed through the network and delivered to a specific host, it must be delivered to the correct user or process. As the data moves up or down the TCP/IP layers, a mechanism is needed to deliver it to the correct protocols in each layer. The system must be able to combine data from many applications into a few transport protocols, and from the transport protocols into the Internet Protocol. Combining many sources of data into a single data stream is called *multiplexing*.

Data arriving from the network must be *demultiplexed*: divided for delivery to multiple processes. To accomplish this task, IP uses *protocol numbers* to identify transport protocols, and the transport protocols use *port numbers* to identify applications.

Some protocol and port numbers are reserved to identify *well-known services*. Well-known services are standard network protocols, such as FTP and telnet, that are commonly used throughout the network. The protocol numbers and port numbers allocated to well-known services are documented in the *Assigned Numbers* RFC. UNIX systems define protocol and port numbers in two simple text files.

Protocol Numbers

The protocol number is a single byte in the third word of the datagram header. The value identifies the protocol in the layer above IP to which the data should be passed.

On a UNIX system, the protocol numbers are defined in */etc/protocols*. This file is a simple table containing the protocol name and the protocol number associated with that name. The format of the table is a single entry per line, consisting of the official protocol name, separated by whitespace from the protocol number. The protocol number is separated by whitespace from the "alias" for the protocol name. Comments in the table begin with #. An */etc/protocols* file is shown below:

```
% cat /etc/protocols
#ident  "@(#)protocols  1.2     90/02/03 SMI"   /* SVr4.0 1.1   */

#
# Internet (IP) protocols
#
ip      0       IP      # internet protocol, pseudo protocol number
icmp    1       ICMP    # internet control message protocol
ggp     3       GGP     # gateway-gateway protocol
tcp     6       TCP     # transmission control protocol
egp     8       EGP     # exterior gateway protocol
pup     12      PUP     # PARC universal packet protocol
udp     17      UDP     # user datagram protocol
hmp     20      HMP     # host monitoring protocol
```

```
xns-idp 22      XNS-IDP # Xerox NS IDP
rdp     27      RDP     # "reliable datagram" protocol
```

The listing shown above is the contents of the */etc/protocols* file from a Solaris 2.5.1 workstation. This list of numbers is by no means complete. If you refer to the Protocol Numbers section of the *Assigned Numbers* RFC, you'll see many more protocol numbers. However, a system needs to include only the numbers of the protocols that it actually uses. Even the list shown above is more than this specific workstation needed, but the additional entries do no harm.

What exactly does this table mean? When a datagram arrives and its destination address matches the local IP address, the IP layer knows that the datagram has to be delivered to one of the transport protocols above it. To decide which protocol should receive the datagram, IP looks at the datagram's protocol number. Using this table you can see that, if the datagram's protocol number is 6, IP delivers the datagram to TCP. If the protocol number is 17, IP delivers the datagram to UDP. TCP and UDP are the two transport layer services we are concerned with, but all of the protocols listed in the table use IP datagram delivery service directly. Some, such as ICMP, EGP, and GGP, have already been mentioned. You don't need to be concerned with the minor protocols.

Port Numbers

After IP passes incoming data to the transport protocol, the transport protocol passes the data to the correct application process. Application processes (also called *network services*) are identified by port numbers, which are 16-bit values. The source port number, which identifies the process that sent the data, and the destination port number, which identifies the process that is to receive the data, are contained in the first header word of each TCP segment and UDP packet.

On UNIX systems, port numbers are defined in the */etc/services* file. There are many more network applications than there are transport layer protocols, as the size of the table shows. Port numbers below 256 are reserved for well-known services (like FTP and telnet) and are defined in the *Assigned Numbers* RFC. Ports numbered from 256 to 1024 are used for UNIX-specific services, services like **rlogin** that were originally developed for UNIX systems. However, most of them are no longer UNIX-specific.

Port numbers are not unique between transport layer protocols; the numbers are only unique within a specific transport protocol. In other words, TCP and UDP can, and do, both assign the same port numbers. It is the combination of protocol and port numbers that uniquely identifies the specific process to which the data should be delivered.

A partial */etc/services* file from a Solaris 2.5.1 workstation is shown below. The format of this file is very similar to the */etc/protocols* file. Each single-line entry starts with the official name of the service, separated by whitespace from the port number/protocol pairing associated with that service. The port numbers are paired with transport protocol names, because different transport protocols may use the same port number. An optional list of aliases for the official service name may be provided after the port number/protocol pair.

```
peanut% head -20 /etc/services
#ident  "@(#)services   1.13    95/07/28 SMI"   /* SVr4.0 1.8   */

#
# Network services, Internet style
#
tcpmux          1/tcp
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
```

This table, combined with the */etc/protocols* table, provides all of the information necessary to deliver data to the correct application. A datagram arrives at its destination based on the destination address in the fifth word of the datagram header. Using the protocol number in the third word of the datagram header, IP delivers the data from the datagram to the proper transport layer protocol. The first word of the data delivered to the transport protocol contains the destination port number that tells the transport protocol to pass the data up to a specific application. Figure 2-6 shows this delivery process.

Despite its size, the */etc/services* file does not contain the port number of every well-known application. You won't find the port number of every *Remote Procedure Call* (RPC) service in the *services* file. Sun developed a different technique for reserving ports for RPC services that doesn't involve registering well-known port numbers. When an RPC service starts, it picks any unused port number and registers that number with the **portmapper**. The **portmapper** is a program that keeps track of the port numbers being used by RPC services. When a client wants to use an RPC service, it queries the **portmapper** running on the server to discover the port assigned to the service. The client can find **portmapper** because it is assigned

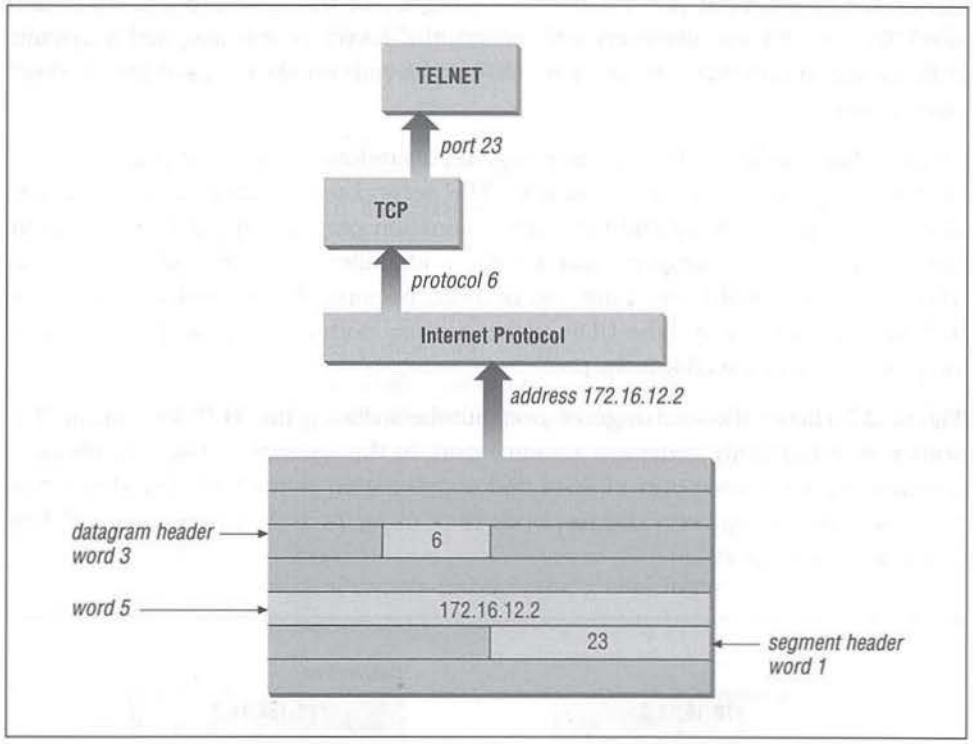


Figure 2-6: Protocol and port numbers

well-known port 111. **portmapper** makes it possible to install well-known services without formally obtaining a well-known port.

Sockets

Well-known ports are standardized port numbers that enable remote computers to know which port to connect to for a particular network service. This simplifies the connection process because both the sender and receiver know in advance that data bound for a specific process will use a specific port. For example, all systems that offer telnet do so on port 23.

There is a second type of port number called a *dynamically allocated port*. As the name implies, dynamically allocated ports are not pre-assigned. They are assigned to processes when needed. The system ensures that it does not assign the same port number to two processes, and that the numbers assigned are above the range of standard port numbers.

Dynamically allocated ports provide the flexibility needed to support multiple users. If a telnet user is assigned port number 23 for both the source and

destination ports, what port numbers are assigned to the second concurrent telnet user? To uniquely identify every connection, the source port is assigned a dynamically allocated port number, and the well-known port number is used for the destination port.

In the telnet example, the first user is given a random source port number and a destination port number of 23 (telnet). The second user is given a different random source port number and the same destination port. It is the pair of port numbers, source and destination, that uniquely identifies each network connection. The destination host knows the source port, because it is provided in both the TCP segment header and the UDP packet header. Both hosts know the destination port because it is a well-known port.

Figure 2-7 shows the exchange of port numbers during the TCP handshake. The source host randomly generates a source port, in this example 3044. It sends out a segment with a source port of 3044 and a destination port of 23. The destination host receives the segment, and responds back using 23 as its source port and 3044 as its destination port.

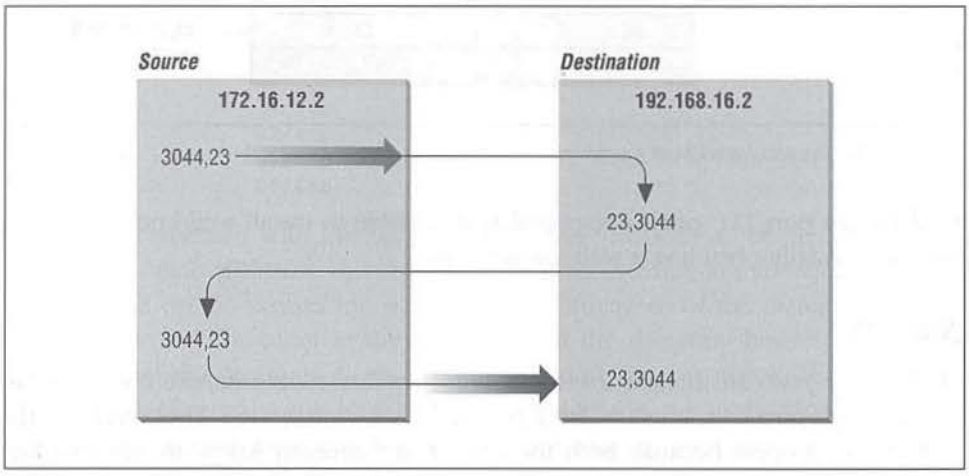


Figure 2-7: Passing port numbers

The combination of an IP address and a port number is called a *socket*. A socket uniquely identifies a single network process within the entire Internet. Sometimes the terms "socket" and "port number" are used interchangeably. In fact, well-known services are frequently referred to as "well-known sockets." In the context of this discussion, a "socket" is the combination of an IP address and a port number. A pair of sockets, one socket for the receiving host and one for the sending host, define the connection for connection-oriented protocols such as TCP.

Let's build on the example of dynamically assigned ports and well-known ports. Assume a user on host 172.16.12.2 uses telnet to connect to host 192.168.16.2. Host 172.16.12.2 is the source host. The user is dynamically assigned a unique port number—3382. The connection is made to the telnet service on the remote host which is, according to the standard, assigned well-known port 23. The socket for the source side of the connection is 172.16.12.2.3382 (IP address 172.16.12.2 plus port number 3382). For the destination side of the connection, the socket is 192.168.16.2.23 (address 192.168.16.2 plus port 23). The port of the destination socket is known by both systems because it is a well-known port. The port of the source socket is known, because the source host informed the destination host of the source socket when the connection request was made. The socket pair is therefore known by both the source and destination computers. The combination of the two sockets uniquely identifies this connection; no other connection in the Internet has this socket pair.

Summary

This chapter shows how data moves through the global Internet from one specific process on the source computer to a single cooperating process on the other side of the world. TCP/IP uses globally unique addresses to identify any computer in the world. It uses protocol numbers and port numbers to uniquely identify a single process running on that computer.

Routing directs the datagrams destined for a remote process through the maze of the global network. Routing uses part of the IP address to identify the destination network. Every system maintains a routing table that describes how to reach remote networks. The routing table usually contains a default route that is used if the table does not contain a specific route to the remote network. A route only identifies the next computer along the path to the destination. TCP/IP uses hop-by-hop routing to move datagrams one step closer to the destination until the datagram finally reaches the destination network.

At the destination network, final delivery is made by using the full IP address (including the host part) and converting that address to a physical layer address. An example of the type of protocol used to convert IP addresses to physical layer addresses is *Address Resolution Protocol* (ARP). It converts IP addresses to Ethernet addresses for final delivery.

The first two chapters described the structure of the TCP/IP protocol stack and the way in which it moves data across a network. In the next chapter we move up the protocol stack to look at the type of services the network provides to simplify configuration and use.

In this chapter:
- Approaching a Problem
- Diagnostic Tools
- Testing Basic Connectivity
- Troubleshooting Network Access
- Checking Routing
- Checking Name Service
- Analyzing Protocol Problems
- Protocol Case Study
- Simple Network Management Protocol
- Summary

11

Troubleshooting TCP/IP

Network administration tasks fall into two very different categories: configuration and troubleshooting. Configuration tasks prepare for the expected; they require detailed knowledge of command syntax, but are usually simple and predictable. Once a system is properly configured, there is rarely any reason to change it. The configuration process is repeated each time a new operating system release is installed, but with very few changes.

In contrast, network troubleshooting deals with the unexpected. Troubleshooting frequently requires knowledge that is conceptual rather than detailed. Network problems are usually unique and sometimes difficult to resolve. Troubleshooting is an important part of maintaining a stable, reliable network service.

In this chapter, we discuss the tools you will use to ensure that the network is in good running condition. However, good tools are not enough. No troubleshooting tool is effective if applied haphazardly. Effective troubleshooting requires a methodical approach to the problem, and a basic understanding of how the network works. We'll start our discussion by looking at ways to approach a network problem.

Approaching a Problem

To approach a problem properly, you need a basic understanding of TCP/IP. The first few chapters of this book discuss the basics of TCP/IP and provide enough background information to troubleshoot most network problems. Knowledge of how TCP/IP routes data through the network, between individual hosts, and between the layers in the protocol stack, is important for understanding a network problem. But detailed knowledge of each protocol usually isn't necessary. When you need these details, look them up in a definitive reference—don't try to recall them from memory.

Not all TCP/IP problems are alike, and not all problems can be approached in the same manner. But the key to solving any problem is understanding what the problem is. This is not as easy as it may seem. The "surface" problem is sometimes misleading, and the "real" problem is frequently obscured by many layers of software. Once you understand the true nature of the problem, the solution to the problem is often obvious.

First, gather detailed information about exactly what's happening. When a user reports a problem, talk to her. Find out which application failed. What is the remote host's name and IP address? What is the user's hostname and address? What error message was displayed? If possible, verify the problem by having the user run the application while you talk her through it. If possible, duplicate the problem on your own system.

Testing from the user's system, and other systems, find out:

- Does the problem occur in other applications on the user's host, or is only one application having trouble? If only one application is involved, the application may be misconfigured or disabled on the remote host. Because of security concerns, many systems disable some services.
- Does the problem occur with only one remote host, all remote hosts, or only certain "groups" of remote hosts? If only one remote host is involved, the problem could easily be with that host. If all remote hosts are involved, the problem is probably with the user's system (particularly if no other hosts on your local network are experiencing the same problem). If only hosts on certain subnets or external networks are involved, the problem may be related to routing.
- Does the problem occur on other local systems? Make sure you check other systems on the same subnet. If the problem only occurs on the user's host, concentrate testing on that system. If the problem affects every system on a subnet, concentrate on the router for that subnet.

Once you know the symptoms of the problem, visualize each protocol and device that handles the data. Visualizing the problem will help you avoid oversimplification, and keep you from assuming that you know the cause even before you start testing. Using your TCP/IP knowledge, narrow your attack to the most likely causes of the problem, but keep an open mind.

Troubleshooting Hints

Below we offer several useful troubleshooting hints. They are not part of a troubleshooting methodology—just good ideas to keep in mind.

- Approach problems methodically. Allow the information gathered from each test to guide your testing. Don't jump on a hunch into another test scenario without ensuring that you can pick up your original scenario where you left off.
- Work carefully through the problem, dividing it into manageable pieces. Test each piece before moving on to the next. For example, when testing a network connection, test each part of the network until you find the problem.
- Keep good records of the tests you have completed and their results. Keep a historical record of the problem in case it reappears.
- Keep an open mind. Don't assume too much about the cause of the problem. Some people believe their network is always at fault, while others assume the remote end is always the problem. Some are so sure they know the cause of a problem that they ignore the evidence of the tests. Don't fall into these traps. Test each possibility and base your actions on the evidence of the tests.
- Be aware of security barriers. Security firewalls sometimes block **ping**, **traceroute**, and even ICMP error messages. If problems seem to cluster around a specific remote site, find out if they have a firewall.
- Pay attention to error messages. Error messages are often vague, but they frequently contain important hints for solving the problem.
- Duplicate the reported problem yourself. Don't rely too heavily on the user's problem report. The user has probably only seen this problem from the application level. If necessary, obtain the user's data files to duplicate the problem. Even if you cannot duplicate the problem, log the details of the reported problem for your records.
- Most problems are caused by human error. You can prevent some of these errors by providing information and training on network configuration and usage.
- Keep your users informed. This reduces the number of duplicated trouble reports, and the duplication of effort when several system administrators work

on the same problem without knowing others are already working on it. If you're lucky, someone may have seen the problem before and have a helpful suggestion about how to resolve it.

- Don't speculate about the cause of the problem while talking to the user. Save your speculations for discussions with your networking colleagues. Your speculations may be accepted by the user as gospel, and become rumors. These rumors can cause users to avoid using legitimate network services and may undermine confidence in your network. Users want solutions to their problems; they're not interested in speculative techno-babble.
- Stick to a few simple troubleshooting tools. For most TCP/IP software problems, the tools discussed in this chapter are sufficient. Just learning how to use a new tool is often more time-consuming than solving the problem with an old familiar tool.
- Thoroughly test the problem at your end of the network before locating the owner of the remote system to coordinate testing with him. The greatest difficulty of network troubleshooting is that you do not always control the systems at both ends of the network. In many cases, you may not even know who does control the remote system.* The more information you have about your end, the simpler the job will be when you have to contact the remote administrator.
- Don't neglect the obvious. A loose or damaged cable is always a possible problem. Check plugs, connectors, cables, and switches. Small things can cause big problems.

Diagnostic Tools

Because most problems have a simple cause, developing a clear idea of the problem often provides the solution. Unfortunately, this is not always true, so in this section we begin to discuss the tools that can help you attack the most intractable problems. Many diagnostic tools are available, ranging from commercial systems with specialized hardware and software that may cost thousands of dollars, to free software that is available from the Internet. Many software tools are provided with your UNIX system. You should also keep some hardware tools handy.

To maintain the network's equipment and wiring you need some simple hand tools. A pair of needle-nose pliers and a few screwdrivers may be sufficient, but you may also need specialized tools. For example, attaching RJ45 connectors to Unshielded Twisted Pair (UTP) cable requires special crimping tools. It is usually easiest to buy a ready-made network maintenance toolkit from your cable vendor.

* Chapter 13 explains how to find out who is responsible for a remote network

A full-featured cable tester is also useful. Modern cable testers are small hand-held units with a keypad and LCD display that test both thinnet or UTP cable. Tests are selected from the keyboard and results are displayed on the LCD screen. It is not necessary to interpret the results because the unit does that for you and displays the error condition in a simple text message. For example, a cable test might produce the message "Short at 74 feet." This tells you that the cable is shorted 74 feet away from the tester. What could be simpler? The proper test tools make it easier to locate, and therefore fix, cable problems.

A laptop computer can be a most useful piece of test equipment when properly configured. Install TCP/IP software on the laptop. Take it to the location where the user reports a network problem. Disconnect the Ethernet cable from the back of the user's system and attach it to the laptop. Configure the laptop with an appropriate address for the user's subnet and reboot it. Then **ping** various systems on the network and attach to one of the user's servers. If everything works, the fault is probably in the user's computer. The user trusts this test because it demonstrates something she does every day. She will have more confidence in the laptop than an unidentifiable piece of test equipment displaying the message "No faults found." If the test fails, the fault is probably in the network equipment or wiring. That's the time to bring out the cable tester.

Another advantage of using a laptop as a piece of test equipment is its inherent versatility. It runs a wide variety of test, diagnostic, and management software. Install UNIX on the laptop and run the software discussed in the rest of this chapter from your desktop or your laptop.

This book emphasizes free or "built-in" software diagnostic tools that run on UNIX systems. The software tools used in this chapter, and many more, are described in RFC 1470, *FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices.* A catchy title, and a very useful RFC! The tools listed in that catalog and discussed in this book are:

ifconfig

Provides information about the basic configuration of the interface. It is useful for detecting bad IP addresses, incorrect subnet masks, and improper broadcast addresses. Chapter 6, *Configuring the Interface*, covers **ifconfig** in detail. This tool is provided with the UNIX operating system.

arp

Provides information about Ethernet/IP address translation. It can be used to detect systems on the local network that are configured with the wrong IP address. **arp** is covered in this chapter, and is used in an example in Chapter 2, *Delivering the Data*. **arp** is delivered as part of UNIX.

netstat
: Provides a variety of information. It is commonly used to display detailed statistics about each network interface, network sockets, and the network routing table. **netstat** is used repeatedly in this book, most extensively in Chapters 2, 6, and 7. **netstat** is delivered as part of UNIX.

ping
: Indicates whether a remote host can be reached. **ping** also displays statistics about packet loss and delivery time. **ping** is discussed in Chapter 1 and used in Chapter 7. **ping** also comes as part of UNIX.

nslookup
: Provides information about the DNS name service. **nslookup** is covered in detail in Chapter 8, *Configuring DNS Name Service*. It comes as part of the BIND software package.

dig
: Also provides information about name service, and is similar to **nslookup**.

ripquery
: Provides information about the contents of the RIP update packets being sent or received by your system. It is provided as part of the **gated** software package, but it does not require that you run **gated**. It will work with any system running RIP.

traceroute
: Prints information about each routing hop that packets take going from your system to a remote system.

snoop
: Analyzes the individual packets exchanged between hosts on a network. **snoop** is a TCP/IP protocol analyzer that examines the contents of packets, including their headers. It is most useful for analyzing protocol problems. **tcpdump** is a tool similar to **snoop** that is available via anonymous FTP from the Internet.

This chapter discusses each of these tools, even those covered earlier in the text. We start with **ping**, which is used in more troubleshooting situations than any other diagnostic tool.

Testing Basic Connectivity

The **ping** command tests whether a remote host can be reached from your computer. This simple function is extremely useful for testing the network connection, independent of the application in which the original problem was detected. **ping** allows you to determine whether further testing should be directed toward the

network connection (the lower layers) or the application (the upper layers). If **ping** shows that packets can travel to the remote system and back, the user's problem is probably in the upper layers. If packets can't make the round trip, lower protocol layers are probably at fault.

Frequently a user reports a network problem by stating that he can't **telnet** (or **ftp**, or send email, or whatever) to some remote host. He then immediately qualifies this statement with the announcement that it worked before. In cases like this, where the ability to connect to the remote host is in question, **ping** is a very useful tool.

Using the hostname provided by the user, **ping** the remote host. If your **ping** is successful, have the user **ping** the host. If the user's **ping** is also successful, concentrate your further analysis on the specific application that the user is having trouble with. Perhaps the user is attempting to **telnet** to a host that only provides anonymous **ftp**. Perhaps the host was down when the user tried his application. Have the user try it again, while you watch or listen to every detail of what he is doing. If he is doing everything right and the application still fails, detailed analysis of the application with **snoop** and coordination with the remote system administrator may be needed.

If your **ping** is successful and the user's **ping** fails, concentrate testing on the user's system configuration, and on those things that are different about the user's path to the remote host, when compared to your path to the remote host.

If your **ping** fails, or the user's **ping** fails, pay close attention to any error messages. The error messages displayed by **ping** are helpful guides for planning further testing. The details of the messages may vary from implementation to implementation, but there are only a few basic types of errors:

Unknown host

The remote host's name cannot be resolved by name service into an IP address. The name servers could be at fault (either your local server or the remote system's server), the name could be incorrect, or something could be wrong with the network between your system and the remote server. If you know the remote host's IP address, try to **ping** that. If you can reach the host using its IP address, the problem is with name service. Use **nslookup** or **dig** to test the local and remote servers, and to check the accuracy of the host name the user gave you.

Network unreachable

The local system does not have a route to the remote system. If the numeric IP address was used on the **ping** command line, re-enter the **ping** command using the hostname. This eliminates the possibility that the IP address was entered incorrectly, or that you were given the wrong address. If a routing

protocol is being used, make sure it is running and check the routing table with **netstat**. If RIP is being used, **ripquery** will check the contents of the RIP updates being received. If a static default route is being used, re-install it. If everything seems fine on the host, check its default gateway for routing problems.

No answer

The remote system did not respond. Most network utilities have some version of this message. Some **ping** implementations print the message "100% packet loss." **telnet** prints the message "Connection timed out" and **sendmail** returns the error "cannot connect." All of these errors mean the same thing. The local system has a route to the remote system, but it receives no response from the remote system to any of the packets it sends.

There are many possible causes of this problem. The remote host may be down. Either the local or the remote host may be configured incorrectly. A gateway or circuit between the local host and the remote host may be down. The remote host may have routing problems. Only additional testing can isolate the cause of the problem. Carefully check the local configuration using **netstat** and **ifconfig**. Check the route to the remote system with **traceroute**. Contact the administrator of the remote system and report the problem.

All of the tools mentioned here will be discussed later in this chapter. However, before leaving **ping**, let's look more closely at the command and the statistics it displays.

The ping Command

The basic format of the **ping** command on a Solaris system is:*

ping *host* [*packetsize*] [*count*]

host

The hostname or IP address of the remote host being tested. Use the hostname or address provided by the user in the trouble report.

packetsize

Defines the size in bytes of the test packets. This field is required only if the *count* field is going to be used. Use the default *packetsize* of 56 bytes.

count

The number of packets to be sent in the test. Use the *count* field, and set the value low. Otherwise, the **ping** command may continue to send test packets

* Check your system's documentation. **ping** varies slightly from system to system. On Linux, the format shown above would be: **ping** [**-c** *count*] [**-s** *packetsize*] *host*

until you interrupt it, usually by pressing CTRL-C (^C). Sending excessive numbers of test packets is not a good use of network bandwidth and system resources. Usually five packets are sufficient for a test.

To check that *ns.uu.net* can be reached from *almond*, we send five 56-byte packets with the following command:

```
% ping -s ns.uu.net 56 5
PING ns.uu.net: 56 data bytes
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=0. time=32.8 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=1. time=15.3 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=2. time=13.1 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=3. time=32.4 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=4. time=28.1 ms

----ns.uu.net PING Statistics----
5 packets transmitted, 5 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 13.1/24.3/32.8
```

The –s option is included because *almond* is a Solaris workstation, and we want packet-by-packet statistics. Without the –s option, Sun's **ping** command only prints a summary line saying "ns.uu.net is alive." Other **ping** implementations do not require the –s option; they display the statistics by default.

This test shows an extremely good wide area network link to *ns.uu.net* with no packet loss and a fast response. The round-trip between *peanut* and *ns.uu.net* took an average of only 24.3 milliseconds. A small packet loss, and a round-trip time an order of magnitude higher, would not be abnormal for a connection made across a wide area network. The statistics displayed by the **ping** command can indicate low-level network problems. The key statistics are:

- The sequence in which the packets are arriving, as shown by the ICMP sequence number (`icmp_seq`) displayed for each packet.
- How long it takes a packet to make the round trip, displayed in milliseconds after the string `time=`.
- The percentage of packets lost, displayed in a summary line at the end of the **ping** output.

If the packet loss is high, the response time is very slow, or packets are arriving out of order, there could be a network hardware problem. If you see these conditions when communicating over great distances on a wide area network, there is nothing to worry about. TCP/IP was designed to deal with unreliable networks, and some wide area networks suffer a lot of packet loss. But if these problems are seen on a local area network, they indicate trouble.

On a local network cable segment, the round-trip time should be near 0, there should be little or no packet loss, and the packets should arrive in order. If these

things are not true, there is a problem with the network hardware. On an Ethernet the problem could be improper cable termination, a bad cable segment, or a bad piece of "active" hardware, such as a hub, switch, or transceiver. Check the cable with a cable tester as described earlier. Good hubs and switches often have built-in diagnostic software that can be checked. Cheap hubs and transceivers may require the "brute force" method of disconnecting individual pieces of hardware until the problem goes away.

The results of a simple ping test, even if the ping is successful, can help you direct further testing toward the most likely causes of the problem. But other diagnostic tools are needed to examine the problem more closely and find the underlying cause.

Troubleshooting Network Access

The "no answer" and "cannot connect" errors indicate a problem in the lower layers of the network protocols. If the preliminary tests point to this type of problem, concentrate your testing on routing and on the network interface. Use the ifconfig, netstat, and arp commands to test the Network Access Layer.

Troubleshooting with the ifconfig Command

ifconfig checks the network interface configuration. Use this command to verify the user's configuration if the user's system has been recently configured, or if the user's system cannot reach the remote host while other systems on the same network can.

When ifconfig is entered with an interface name and no other arguments, it displays the current values assigned to that interface. For example, checking interface le0 on a Solaris system gives this report:

```
% ifconfig le0
le0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 172.16.55.105 netmask ffffff00 broadcast 172.16.55.255
```

The ifconfig command displays two lines of output. The first line of the display shows the interface's name and its characteristics. Check for these characteristics:

UP The interface is enabled for use. If the interface is "down," have the system's superuser bring the interface "up" with the ifconfig command (e.g., ifconfig le0 up). If the interface won't come up, replace the interface cable and try again. If it still fails, have the interface hardware checked.

RUNNING
This interface is operational. If the interface is not "running," the driver for this interface may not be properly installed. The system administrator should

review all of the steps necessary to install this interface, looking for errors or missed steps.

The second line of **ifconfig** output shows the IP address, the subnet mask (written in hexadecimal), and the broadcast address. Check these three fields to make sure the network interface is properly configured.

Two common interface configuration problems are misconfigured subnet masks and incorrect IP addresses. A bad subnet mask is indicated when the host can reach other hosts on its local subnet and remote hosts on distant networks, but it cannot reach hosts on other local subnets. **ifconfig** quickly reveals if a bad subnet mask is set.

An incorrectly set IP address can be a subtle problem. If the network part of the address is incorrect, every **ping** will fail with the "no answer" error. In this case, using **ifconfig** will reveal the incorrect address. However, if the host part of the address is wrong, the problem can be more difficult to detect. A small system, such as a PC that only connects out to other systems and never accepts incoming connections, can run for a long time with the wrong address without its user noticing the problem. Additionally, the system that suffers the ill effects may not be the one that is misconfigured. It is possible for someone to accidentally use your IP address on his system, and for his mistake to cause your system intermittent communications problems. An example of this problem is discussed later. This type of configuration error cannot be discovered by **ifconfig**, because the error is on a remote host. The **arp** command is used for this type of problem.

Troubleshooting with the arp Command

The **arp** command is used to analyze problems with IP to Ethernet address translation. The **arp** command has three useful options for troubleshooting:

-a Display all ARP entries in the table.

-d *hostname*
Delete an entry from the ARP table.

-s *hostname ether-address*
Add a new entry to the table.

With these three options you can view the contents of the ARP table, delete a problem entry, and install a corrected entry. The ability to install a corrected entry is useful in "buying time" while you look for the permanent fix.

Use **arp** if you suspect that incorrect entries are getting into the address resolution table. One clear indication of problems with the ARP table is a report that the "wrong" host responded to some command, like **ftp** or **telnet**. Intermittent problems that affect only certain hosts can also indicate that the ARP table has been

corrupted. ARP table problems are usually caused by two systems using the same IP address. The problems appear intermittent, because the entry that appears in the table is the address of the host that responded quickest to the last ARP request. Sometimes the "correct" host responds first, and sometimes the "wrong" host responds first.

If you suspect that two systems are using the same IP address, display the address resolution table with the **arp** –a command. Here's an example from a Solaris system:*

```
% arp -a
Net to Media Table
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
le0    peanut.nuts.com      255.255.255.255       08:00:20:05:21:33
le0    pecan.nuts.com       255.255.255.255       00:00:0c:e0:80:b1
le0    almond.nuts.com      255.255.255.255  SP   08:00:20:22:fd:51
le0    BASE-ADDRESS.MCAST.NET 240.0.0.0      SM   01:00:5e:00:00:00
```

It is easiest to verify that the IP and Ethernet address pairs are correct if you have a record of each host's correct Ethernet address. For this reason you should record each host's Ethernet and IP address when it is added to your network. If you have such a record, you'll quickly see if anything is wrong with the table.

If you don't have this type of record, the first three bytes of the Ethernet address can help you to detect a problem. The first three bytes of the address identify the equipment manufacturer. A list of these identifying prefixes is found in the *Assigned Numbers* RFC, in the section entitled "Ethernet Vendor Address Components." This information is also available at *ftp://ftp.isi.edu/in-notes/iana/assignments/ethernet-numbers.*

From the vendor prefixes we see that two of the ARP entries displayed in our example are Sun systems (8:0:20). If *pecan* is also supposed to be a Sun, the 0:0:0c Cisco prefix indicates that a Cisco router has been mistakenly configured with *pecan*'s IP address.

If neither checking a record of correct assignments nor checking the manufacturer prefix helps you identify the source of the errant ARP, try using **telnet** to connect to the IP address shown in the ARP entry. If the device supports **telnet**, the login banner might help you identify the incorrectly configured host.

* The format in which the ARP table is displayed may vary slightly between systems.

ARP problem case study

A user called in asking if the server was down, and reported the following problem. The user's workstation, called *cashew*, appeared to "lock up" for minutes at a time when certain commands were used, while other commands worked with no problems. The network commands that involved the NIS name server all caused the lock-up problem, but some unrelated commands also caused the problem. The user reported seeing the error message:

```
NFS getattr failed for server almond: RPC: Timed out
```

The server *almond* was providing *cashew* with NIS and NFS services. The commands that failed on *cashew* were commands that required NIS service, or that were stored in the centrally maintained */usr/local* directory exported from *almond*. The commands that ran correctly were installed locally on the user's workstation. No one else reported a problem with the server, and we were able to **ping** *cashew* from *almond* and get good responses.

We had the user check the */usr/adm/messages* file for recent error messages, and she discovered this:

```
Mar  6 13:38:23 cashew vmunix: duplicate IP address!!
        sent from ethernet address: 0:0:c0:4:38:1a
```

This message indicates that the workstation detected another host on the Ethernet responding to its IP address. The "imposter" used the Ethernet address 0:0:c0:4:38:1a in its ARP response. The correct Ethernet address for *cashew* is 8:0:20:e:12:37.

We checked *almond*'s ARP table and found that it had the incorrect ARP entry for *cashew*. We deleted the bad *cashew* entry with the **arp -d** command, and installed the correct entry with the **-s** option, as shown below:

```
# arp -d cashew
cashew (172.16.180.130) deleted
# arp -s cashew 8:0:20:e:12:37
```

ARP entries received via the ARP protocol are temporary. The values are held in the table for a finite lifetime and are deleted when that lifetime expires. New values are then obtained via the ARP protocol. Therefore, if some remote interfaces change, the local table adjusts and communications continue. Usually this is a good idea, but if someone is using the wrong IP address, that bad address can keep reappearing in the ARP table even if it is deleted. However, manually entered values are permanent; they stay in the table and can only be deleted manually. This allowed us to install a correct entry in the table, without worrying about it being overwritten by a bad address.

This quick fix resolved *cashew's* immediate problem, but we still needed to find the culprit. We checked the */etc/ethers* file to see if we had an entry for Ethernet address 0:0:c0:4:38:1a, but we didn't. From the first three bytes of this address, 0:0:c0, we knew that the device was a Western Digital card. Since our network has only UNIX workstations and PCs, we assumed the Western Digital card was installed in a PC. We also guessed that the problem address was recently installed because the user had never had the problem before. We sent out an urgent announcement to all users asking if anyone had recently installed a new PC, reconfigured a PC, or installed TCP/IP software on a PC. We got one response. When we checked his system, we found out that he had entered the address 172.16.180.130 when he should have entered 172.16.180.138. The address was corrected and the problem did not recur.

Nothing fancy was needed to solve this problem. Once we checked the error messages, we knew what the problem was and how to solve it. Involving the entire network user community allowed us to quickly locate the problem system and to avoid a room-to-room search for the PC. Reluctance to involve users and make them part of the solution is one of the costliest, and most common, mistakes made by network administrators.

Checking the Interface with netstat

If the preliminary tests lead you to suspect that the connection to the local area network is unreliable, the **netstat –i** command can provide useful information. The example below shows the output from the **netstat –i** command:

```
% netstat -i
Name Mtu  Net/Dest Address   Ipkts Ierrs Opkts Oerrs Collis Queue
le0  1500 nuts.com almond    442697  2     633424  2     50679  0
lo0  1536 loopback localhost  53040  0     53040   0     0      0
```

The line for the loopback interface, lo0, can be ignored. Only the line for the real network interface is significant, and only the last five fields on that line provide significant troubleshooting information.

Let's look at the last field first. There should be no packets queued (Queue) that cannot be transmitted. If the interface is up and running, and the system cannot deliver packets to the network, suspect a bad drop cable or a bad interface. Replace the cable and see if the problem goes away. If it doesn't, call the vendor for interface hardware repairs.

The input errors (Ierrs) and the output errors (Oerrs) should be close to 0. Regardless of how much traffic has passed through this interface, 100 errors in either of these fields is high. High output errors could indicate a saturated local network or a bad physical connection between the host and the network. High input errors could indicate that the network is saturated, the local host is overloaded, or there

is a physical network problem. Tools, such as ping statistics or a cable tester, can help you determine if it is a physical network problem. Evaluating the collision rate can help you determine if the local Ethernet is saturated.

A high value in the collision field (Collis) is normal, but if the percentage of output packets that result in a collision is too high, it indicates that the network is saturated. Collision rates greater than 5% bear watching. If high collision rates are seen consistently, and are seen among a broad sampling of systems on the network, you may need to subdivide the network to reduce traffic load.

Collision rates are a percentage of output packets. Don't use the total number of packets sent and received; use the values in the Opkts and Collis fields when determining the collision rate. For example, the output in the netstat sample above shows 50679 collisions out of 633424 outgoing packets. That's a collision rate of 8%. This sample network could be overworked; check the statistics on other hosts on this network. If the other systems also show a high collision rate, consider subdividing this network.

Subdividing an Ethernet

To reduce the collision rate, you must reduce the amount of traffic on the network segment. A simple way to do this is to create multiple segments out of the single segment. Each new segment will have fewer hosts and, therefore, less traffic. We'll see, however, that it's not quite this simple.

The most effective way to subdivide an Ethernet is to install an Ethernet switch. Each port on the switch is essentially a separate Ethernet. So a 16-port switch gives you 16 Ethernets to work with when balancing the load. On most switches the ports can be used in a variety of ways (see Figure 11-1). Lightly used systems can be attached to a hub that is then attached to one of the switch ports to allow the systems to share a single segment. Servers and demanding systems can be given dedicated ports so that they don't need to share a segment with anyone. Additionally, some switches provide a few Fast Ethernet 100 Mbps ports. These are called asymmetric switches because different ports operate at different speeds. Use the Fast Ethernet ports to connect heavily used servers. If you're buying a new switch, buy a 10/100 switch with auto-sensing ports. This allows every port to be used at either 100 Mbps or at 10 Mbps, which give you the maximum configuration flexibility.

Figure 11-1 shows an 8-port 10/100 Ethernet switch. Ports 1 and 2 are wired to Ethernet hubs. A few systems are connected to each hub. When new systems are added they are distributed evenly among the hubs to prevent any one segment from becoming overloaded. Additional hubs can be added to the available switch ports for future expansion. Port 4 attaches a demanding system with its own

private segment. Port 6 operates at 100 Mbps and attaches a heavily used server. A port can be reserved for a future 100 Mbps connection to a second 10/100 Ethernet switch for even more expansion.

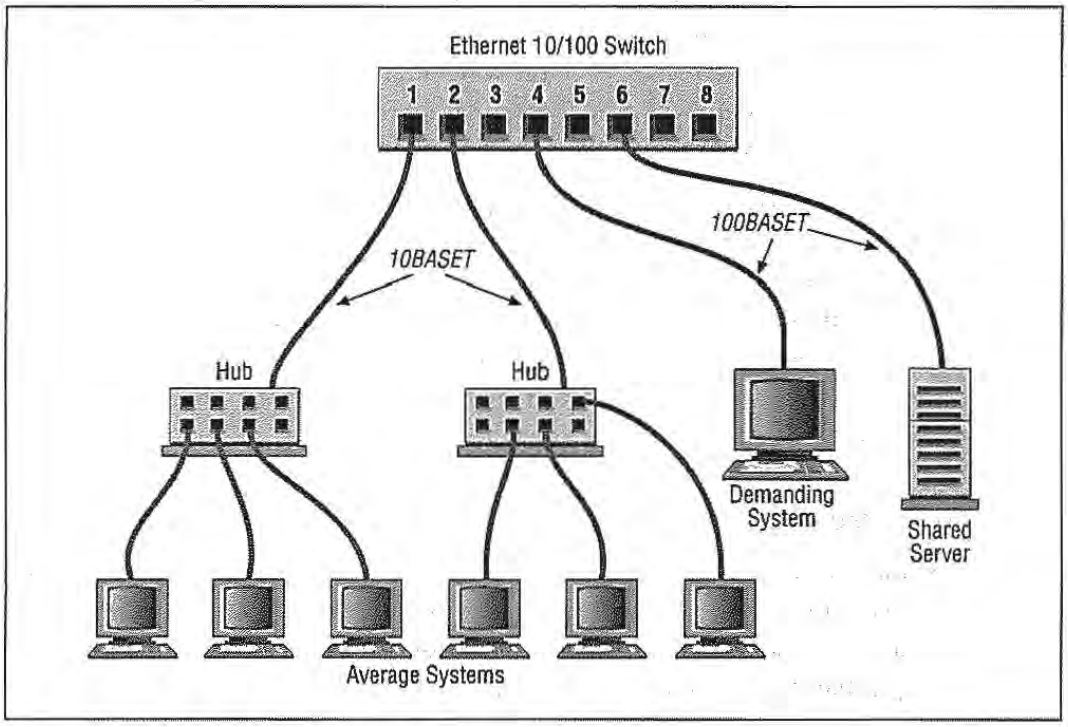


Figure 11-1: Subdividing an Ethernet with switches

Before allocating the ports on your switch, evaluate what services are in demand, and who talks to whom. Then develop a plan that reduces the amount of traffic flowing over any segment. For example, if the demanding system on Port 4 uses lots of bandwidth because it is constantly talking to one of the systems on Port 1, all of the systems on Port 1 will suffer because of this traffic. The computer that the demanding system communicates with should be moved to one of the vacant ports or to the same port (4) as the demanding system. Use your switch to the greatest advantage by balancing the load.

Should you segment an old coaxial cable Ethernet by cutting the cable and joining it back together through a router or a bridge? No. If you have an old network that is finally reaching saturation, it is time to install a new network built on a more robust technology. A *shared media* network, a network where everyone is on the same cable (in this example, a coaxial cable Ethernet) is an accident waiting to happen. Design a network that a user cannot bring down by merely disconnecting his system, or even by accidentally cutting a wire in his office. Use *Unshielded Twisted Pair* (UTP) cable, ideally Category 5 cable, to create a 10BaseT Ethernet or

100BaseT Fast Ethernet that wires equipment located in the user's office to a hub securely stored in a wire closet. The network components in the user's office should be sufficiently isolated from the network so that damage to those components does not damage the entire network. The new network will solve your collision problem and reduce the amount of hardware troubleshooting you are called upon to do.

Network hardware problems

Some of the tests discussed in this section can show a network hardware problem. If a hardware problem is indicated, contact the people responsible for the hardware. If the problem appears to be in a leased telephone line, contact the telephone company. If the problem appears to be in a wide area network, contact the management of that network. Don't sit on a problem expecting it to go away. It could easily get worse.

If the problem is in your local area network, you will have to handle it yourself. Some tools, such as the cable tester described above, can help. But frequently the only way to approach a hardware problem is by brute force—disconnecting pieces of hardware until you find the one causing the problem. It is most convenient to do this at the switch or hub. If you identify a device causing the problem, repair or replace it. Remember that the problem can be the cable itself, rather than any particular device.

Checking Routing

The "network unreachable" error message clearly indicates a routing problem. If the problem is in the local host's routing table, it is easy to detect and resolve. First, use **netstat -nr** and **grep** to see whether or not a valid route to your destination is installed in the routing table. This example checks for a specific route to network 128.8.0.0:

```
% netstat -nr | grep '128\.8\.0'
128.8.0.0      26.20.0.16      UG      0     37     std0
```

This same test, run on a system that did not have this route in its routing table, would return no response at all. For example, a user reports that the "network is down" because he cannot **ftp** to *sunsite.unc.edu*, and a **ping** test returns the following results:

```
% ping -s sunsite.unc.edu 56 2
PING sunsite.unc.edu: 56 data bytes
sendto: Network is unreachable
ping: wrote sunsite.unc.edu 64 chars, ret=-1
sendto: Network is unreachable
ping: wrote sunsite.unc.edu 64 chars, ret=-1
```

```
----sunsite.unc.edu PING Statistics----
2 packets transmitted, 0 packets received, 100% packet loss
```

Based on the "network unreachable" error message, check the user's routing table. In our example, we're looking for a route to *sunsite.unc.edu*. The IP address* of sunsite.unc.edu is 152.2.254.81, which is a class B address. Remember that routes are network-oriented. So we check for a route to network 152.2.0.0:

```
% netstat -nr | grep '152\.2\.0\.0'
%
```

This test shows that there is no *specific* route to 152.2.0.0. If a route was found, grep would display it. Since there's no specific route to the destination, remember to look for a default route. This example shows a successful check for a default route:

```
% netstat -nr | grep def
default        172.16.12.1      UG    0   101277   le0
```

If netstat shows the correct specific route, or a valid default route, the problem is not in the routing table. In that case, use traceroute, as described later in this chapter, to trace the route all the way to its destination.

If netstat doesn't return the expected route, it's a local routing problem. There are two ways to approach local routing problems, depending on whether the system uses static or dynamic routing. If you're using static routing, install the missing route using the route add command. Remember, most systems that use static routing rely on a default route, so the missing route could be the default route. Make sure that the startup files add the needed route to the table whenever the system reboots. See Chapter 7, *Configuring Routing*, for details about the route add command.

If you're using dynamic routing, make sure that the routing program is running. For example, the command below makes sure that gated is running:

```
% ps `cat /etc/gated.pid`
  PID TT STAT  TIME COMMAND
27711 ?  S   304:59 gated -tep /etc/log/gated.log
```

If the correct routing daemon is not running, restart it and specify tracing. Tracing allows you to check for problems that might be causing the daemon to terminate abnormally.

* Use nslookup to find the IP address if you don't know it. nslookup is discussed later in this chapter.

Checking RIP Updates

If the routing daemon is running and the local system receives routing updates via Routing Information Protocol (RIP), use **ripquery** to check the updates received from your RIP suppliers. For example, to check the RIP updates being received from *almond* and *pecan*, the *peanut* administrator enters the following command:

```
% ripquery -1 -n -r almond pecan
44 bytes from almond.nuts.com(172.16.12.1):
        0.0.0.0, metric 3
        10.0.0.0, metric 0
264 bytes from pecan.nuts.com(172.16.12.3):
        172.16.5.0, metric 2
        172.16.3.0, metric 2
                .
                .
                .
        172.16.12.0, metric 2
        172.16.13.0, metric 2
```

After an initial line identifying the gateway, **ripquery** shows the contents of the incoming RIP packets, one line per route. The first line of the report above indicates that **ripquery** received a response from *almond*. That line is followed by two lines for the two routes advertised by *almond*. *almond* advertises the default route (destination 0.0.0.0) with a metric of 3, and its direct route to Milnet (destination 10.0.0.0) with a metric of 0. Next, **ripquery** shows the routes advertised by *pecan*. These are the routes to the other *nuts-net* subnets.

The three **ripquery** options used in this example are:

–1 Sends the query as a RIP version 1 packet. By default, queries are sent as RIP version 2 packets. Older systems may only support RIP version 1.

–n Causes **ripquery** to display all output in numeric form. **ripquery** attempts to resolve all IP addresses to names if the **–n** option is not specified. It's a good idea to use the **–n** option; it produces a cleaner display, and you don't waste time resolving names.

–r Directs **ripquery** to use the RIP REQUEST command, instead of the RIP POLL command, to query the RIP supplier. RIP POLL is not universally supported. You are more likely to get a successful response if you specify **–r** on the **ripquery** command line.

The routes returned in these updates should be the routes you expect. If they are not, or if no routes are returned, check the configuration of the RIP suppliers. Routing configuration problems cause RIP suppliers to advertise routes that they shouldn't, or to fail to advertise the routes that they should. You can detect these problems only by applying your knowledge of your network configuration. You must know what is right to detect what is wrong. Don't expect to see error

messages or strange garbled routes. For example, assume that in the previous test *pecan* returned the following update:

```
264 bytes from pecan.nuts.com(172.16.12.3):
        0.0.0.0, metric 2
        172.16.3.0, metric 2
                .
                .
                .
        172.16.12.0, metric 2
        172.16.13.0, metric 2
```

This update shows that *pecan* is advertising itself as a default gateway with a lower cost (2 versus 3) than *almond*. This would cause every host on this subnet to use *pecan* as its default gateway. If this is not what you wanted, the routing configuration of *pecan* should be corrected.*

Tracing Routes

If the local routing table and RIP suppliers are correct, the problem may be occurring some distance away from the local host. Remote routing problems can cause the "no answer" error message, as well as the "network unreachable" error message. But the "network unreachable" message does not always signify a routing problem. It can mean that the remote network cannot be reached because something is down between the local host and the remote destination. **traceroute** is the program that can help you locate these problems.

traceroute traces the route of UDP packets from the local host to a remote host. It prints the name (if it can be determined) and IP address of each gateway along the route to the remote host.

traceroute uses two techniques, small *ttl* (time-to-live) values and an invalid port number, to trace packets to their destination. **traceroute** sends out UDP packets with small ttl values to detect the intermediate gateways. The ttl values start at 1 and increase in increments of 1 for each group of three UDP packets sent. When a gateway receives a packet, it decrements the ttl. If the ttl is then 0, the packet is not forwarded and an ICMP "Time Exceeded" message is returned to the source of the packet. **traceroute** displays one line of output for each gateway from which it receives a "Time Exceeded" message. Figure 11-2 shows a sample of the single line of output that is displayed for a gateway, and it shows the meaning of each field in the line. When the destination host receives a packet from **traceroute**, it returns an ICMP "Unreachable Port" message. This happens because **traceroute** intentionally uses an invalid port number (33434) to force this error. When **traceroute** receives the "Unreachable Port" message, it knows that it has reached

* Correct routing configuration is discussed in Chapter 7.

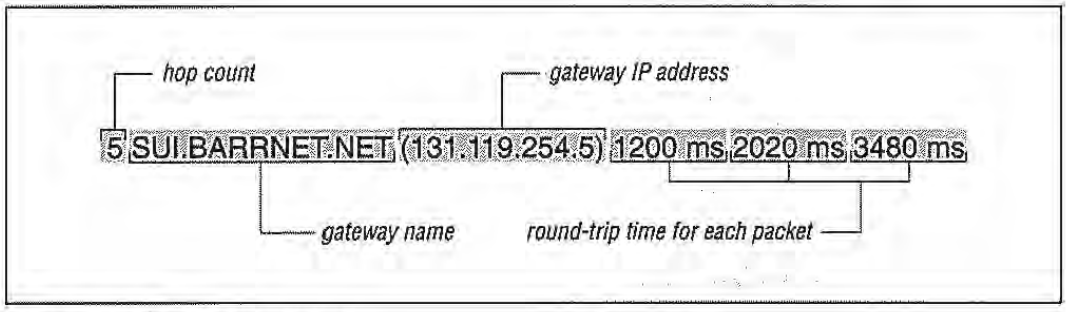


Figure 11-2: traceroute output

the destination host, and it terminates the trace. So, traceroute is able to develop a list of the gateways, starting at one hop away and increasing one hop at a time until the remote host is reached. Figure 11-3 illustrates the flow of packets tracing to a host three hops away. The following shows a traceroute to *ds.internic.net* from a Linux system hanging off BBN PlaNET. traceroute sends out three packets at each ttl value. If no response is received to a packet, traceroute prints an asterisk (*). If a response is received, traceroute displays the name and address of the gateway that responded, and the packet's round-trip time in milliseconds.

```
% traceroute ds.internic.net
traceroute to ds.internic.net (198.49.45.10), 30 hops max, 40 byte packets
 1  gw-55.nuts.com (172.16.55.200)  0.95 ms  0.91 ms  0.91 ms
 2  172.16.230.254 (172.16.230.254)  1.51 ms  1.33 ms  1.29 ms
 3  gw225.nuts.com (172.16.2.252)  4.13 ms  1.94 ms  2.20 ms
 4  192.221.253.2 (192.221.253.2)  52.90 ms  81.19 ms  58.09 ms
 5  washdc1-br2.bbnplanet.net (4.0.36.17)  6.5 ms  5.8 ms  5.88 ms
 6  nyc1-br1.bbnplanet.net (4.0.1.114)  13.24 ms  12.71 ms  12.96 ms
 7  nyc1-br2.bbnplanet.net (4.0.1.178)  14.64 ms  13.32 ms  12.21 ms
 8  cambridge1-br1.bbnplanet.net (4.0.2.86)  28.84 ms  27.78 ms  23.56 ms
 9  cambridge1-cr14.bbnplanet.net (199.94.205.14) 19.9 ms  24.7 ms 22.3 ms
10  attbcstol1.bbnplanet.net (206.34.99.38)  34.31 ms  36.63 ms  32.21 ms
11  ds0.internic.net (198.49.45.10)  33.19 ms  33.34 ms *
```

This trace shows that 10 intermediate gateways are involved, that packets are making the trip, and that round-trip travel time for packets from this host to *ds.internic.net* is about 33 ms.

Variations and bugs in the implementation of ICMP on different types of gateways, and the unpredictable nature of the path a datagram can take through a network, can cause some odd displays. For this reason, you shouldn't examine the output of traceroute too closely. The most important things in the traceroute output are:

- Did the packet get to its remote destination?
- If not, where did it stop?

In the code below we show another trace of the path to *ds.internic.net*. This time the trace does not go all the way through to the InterNIC.

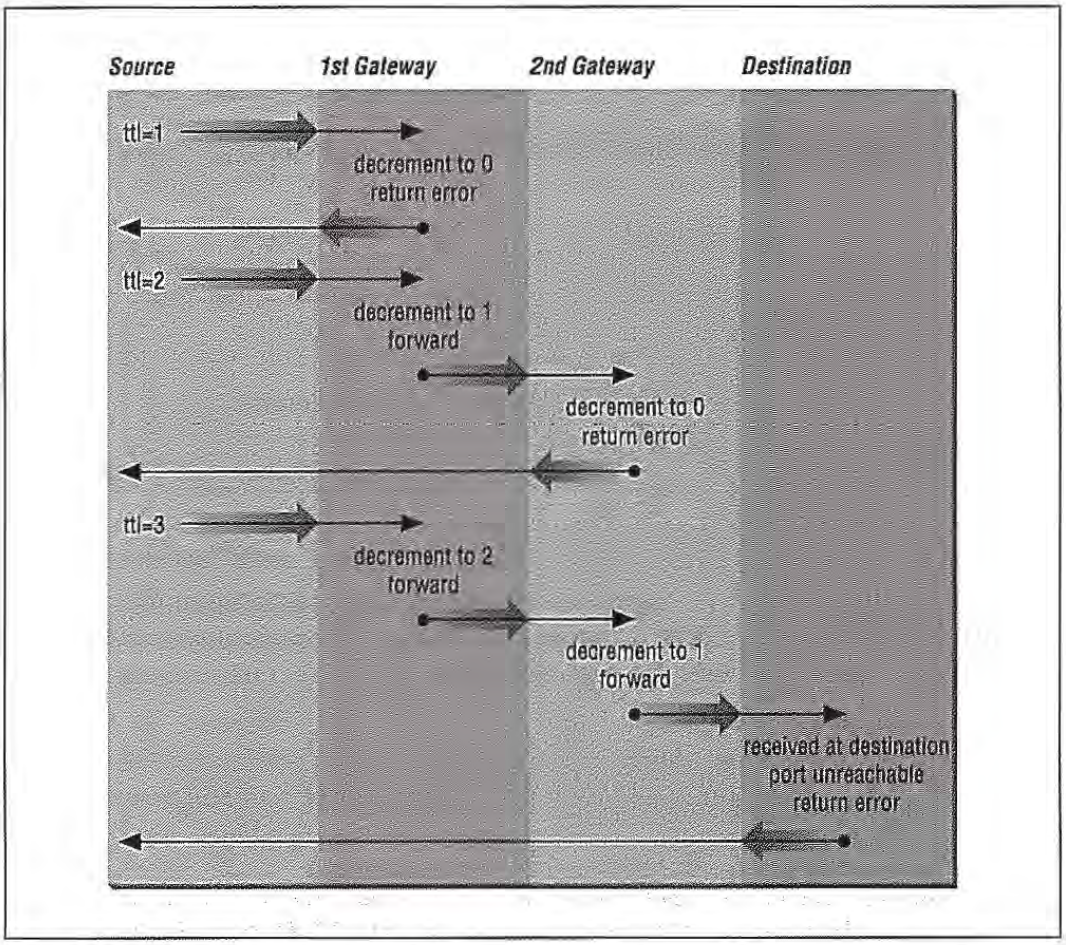


Figure 11-3: Flow of traceroute packets

```
% traceroute ds.internic.net
traceroute to ds.internic.net (198.49.45.10), 30 hops max,
     40 byte packets
 1  gw-55.nuts.com (172.16.55.200)  0.959 ms  0.917 ms  0.913 ms
 2  172.16.230.254 (172.16.230.254)  1.518 ms  1.337 ms  1.296 ms
 3  gw225.nuts.com (172.16.2.252)  4.137 ms  1.945 ms  2.209 ms
 4  192.221.253.2 (192.221.253.2)  52.903 ms  81.19 ms  58.097 ms
 5  washdc1-br2.bbnplanet.net (4.0.36.17)  6.5 ms  5.8 ms  5.888 ms
 6  nyc1-br1.bbnplanet.net (4.0.1.114)  13.244 ms  12.717 ms  12.968 ms
 7  nyc1-br2.bbnplanet.net (4.0.1.178)  14.649 ms  13.323 ms  12.212 ms
 8  cambridge1-br1.bbnplanet.net (4.0.2.86)  28.842 ms  27.784 ms
     23.561 ms
 9  * * *
10  * * *
       .
       .
       .
29  * * *
30  * * *
```

When **traceroute** fails to get packets through to the remote end system, the trace trails off, displaying a series of three asterisks at each hop count until the count reaches 30. If this happens, contact the administrator of the remote host you're trying to reach, and the administrator of the last gateway displayed in the trace. Describe the problem to them; they may be able to help.* In our example, the last gateway that responded to our packets was *cambridge1-br1.bbnplanet.net.* We would contact this system administrator, and the administrator of *ds.internic.net.*

Checking Name Service

Name server problems are indicated when the "unknown host" error message is returned by the user's application. Name server problems can usually be diagnosed with **nslookup** or **dig**. **nslookup** is discussed in detail in Chapter 8. **dig** is an alternative tool with similar functionality that is discussed in this chapter. Before looking at **dig**, let's take another look at **nslookup** and see how it is used to troubleshoot name service.

Three features of **nslookup** covered in Chapter 8 are particularly important for troubleshooting remote name server problems. These features are its ability to:

- Locate the authoritative servers for the remote domain using the NS query
- Obtain all records about the remote host using the ANY query
- Browse all entries in the remote zone using **nslookup**'s **ls** and **view** commands

When troubleshooting a remote server problem, directly query the authoritative servers returned by the NS query. Don't rely on information returned by non-authoritative servers. If the problems that have been reported are intermittent, query all of the authoritative servers in turn and compare their answers. Intermittent name server problems are sometimes caused by the remote servers returning different answers to the same query.

The ANY query returns all records about a host, thus giving the broadest range of troubleshooting information. Simply knowing what information is (and isn't) available can solve a lot of problems. For example, if the query returns an MX record but no A record, it is easy to understand why the user couldn't **telnet** to that host! Many hosts are accessible to mail that are not accessible by other network services. In this case, the user is confused and is trying to use the remote host in an inappropriate manner.

If you are unable to locate any information about the hostname that the user gave you, perhaps the hostname is incorrect. Given that the hostname you have is

* Chapter 13, *Internet Information Resources*, explains how to find out who is responsible for a specific computer.

wrong, looking for the correct name is like trying to find a needle in a haystack. However, **nslookup** can help. Use **nslookup**'s **ls** command to dump the remote zone file, and redirect the listing to a file. Then use **nslookup**'s **view** command to browse through the file, looking for names similar to the one the user supplied. Many problems are caused by a mistaken hostname.

All of the **nslookup** features and commands mentioned here are used in Chapter 8. However, some examples using these commands to solve real name server problems will be helpful. The three examples that follow are based on actual trouble reports.*

Some systems work, others don't

A user reported that she could resolve a certain hostname from her workstation, but could not resolve the same hostname from the central system. However, the central system could resolve other hostnames. We ran several tests and found that we could resolve the hostname on some systems and not on others. There seemed to be no predictable pattern to the failure. So we used **nslookup** to check the remote servers.

```
% nslookup
Default Server:  almond.nuts.com
Address:  172.16.12.1

> set type=NS
> foo.edu.
Server:  almond.nuts.com
Address:  172.16.12.1

foo.edu         nameserver = gerbil.foo.edu
foo.edu         nameserver = red.big.com
foo.edu         nameserver = shrew.foo.edu
gerbil.foo.edu    inet address = 198.97.99.2
red.big.com    inet address = 184.6.16.2
shrew.foo.edu     inet address = 198.97.99.1
> set type=ANY
> server gerbil.foo.edu
Default Server:  gerbil.foo.edu
Address:  198.97.99.2

> hamster.foo.edu
Server:  gerbil.foo.edu
Address:  198.97.99.2

hamster.foo.edu         inet address = 198.97.99.8
> server red.big.com
Default Server:  red.big.com
```

* The host and server names are fictitious, but the problems were real.

```
Address:  184.6.16.2
> hamster.foo.edu
Server:  red.big.com
Address:  184.6.16.2

*** red.big.com can't find hamster.foo.edu: Non-existent domain
```

This sample nslookup session contains several steps. The first step is to locate the authoritative servers for the host name in question (*hamster.foo.edu*). We set the query type to NS to get the name server records, and query for the domain (*foo.edu*) in which the hostname is found. This returns three names of authoritative servers: *gerbil.foo.edu*, *red.big.com*, and *shrew.foo.edu*.

Next, we set the query type to ANY to look for any records related to the hostname in question. Then we set the server to the first server in the list, *gerbil.foo.edu*, and query for *hamster.foo.edu*. This returns an address record. So server *gerbil.foo.edu* works fine. We repeat the test using *red.big.com* as the server, and it fails. No records are returned.

The next step is to get SOA records from each server and see if they are the same:

```
> set type=SOA
> foo.edu.
Server:  red.big.com
Address:  184.6.16.2

foo.edu         origin = gerbil.foo.edu
        mail addr = amanda.gerbil.foo.edu
        serial=10164, refresh=43200, retry=3600, expire=3600000,
        min=2592000
> server gerbil.foo.edu
Default Server:  gerbil.foo.edu
Address:  198.97.99.2

> foo.edu.
Server:  gerbil.foo.edu
Address:  198.97.99.2

foo.edu         origin = gerbil.foo.edu
        mail addr = amanda.gerbil.foo.edu
        serial=10164, refresh=43200, retry=3600, expire=3600000,
        min=2592000

> exit
```

If the SOA records have different serial numbers, perhaps the zone file, and therefore the hostname, has not yet been downloaded to the secondary server. If the serial numbers are the same and the data is different, as in this case, there is a definite problem. Contact the remote domain administrator and notify her of the problem. The administrator's mailing address is shown in the "mail addr" field of

the SOA record. In our example, we would send mail to *amanda@gerbil.foo.edu* reporting the problem.

The data is here and the server can't find it!

This problem was reported by the administrator of one of our secondary name servers. The administrator reported that his server could not resolve a certain hostname in a domain for which his server was a secondary server. The primary server was, however, able to resolve the name. The administrator dumped his cache (more on dumping the server cache in the next section), and he could see in the dump that his server had the correct entry for the host. But his server still would not resolve that hostname to an IP address!

The problem was replicated on several other secondary servers. The primary server would resolve the name; the secondary servers wouldn't. All servers had the same SOA serial number, and a dump of the cache on each server showed that they all had the correct address records for the hostname in question. So why wouldn't they resolve the hostname to an address?

Visualizing the difference between the way primary and secondary servers load their data made us suspicious of the zone file transfer. Primary servers load the data directly from local disk files. Secondary servers transfer the data from the primary server via a zone file transfer. Perhaps the zone files were getting corrupted. We displayed the zone file on one of the secondary servers, and it showed the following data:

```
% cat /usr/etc/sales.nuts.com.hosts
PCpma       IN      A       172.16.64.159
            IN      HINFO   "pc" "n3/800salesnutscom"
PCrkc       IN      A       172.16.64.155
            IN      HINFO   "pc" "n3/800salesnutscom"
PCafc       IN      A       172.16.64.189
            IN      HINFO   "pc" "n3/800salesnutscom"
accu        IN      A       172.16.65.27
cmgds1      IN      A       172.16.130.40
cmg         IN      A       172.16.130.30
PCgns       IN      A       172.16.64.167
            IN      HINFO   "pc" "(3/800salesnutscom"
gw          IN      A       172.16.65.254
zephyr      IN      A       172.16.64.188
            IN      HINFO   "Sun" "sparcstation"
ejw         IN      A       172.16.65.17
PCecp       IN      A       172.16.64.193
            IN      HINFO   "pc" "n^Lsparcstationstcom"
```

Notice the odd display in the last field of the HINFO statement for each PC.* This data might have been corrupted in the transfer or it might be bad on the primary server. We used **nslookup** to check that.

```
% nslookup
Default Server:  almond.nuts.com
Address:  172.16.12.1

> server acorn.sales.nuts.com
Default Server:  acorn.sales.nuts.com
Address:  172.16.6.1

> set query=HINFO
> PCwlg.sales.nuts.com
Server:  acorn.sales.nuts.com
Address:  172.16.6.1

PCwlg.sales.nuts.com     CPU=pc  OS=ov
packet size error (0xf7fff590 != 0xf7fff528)
> exit
```

In this **nslookup** example, we set the server to *acorn.sales.nuts.com*, which is the primary server for *sales.nuts.com*. Next we queried for the HINFO record for one of the hosts that appeared to have a corrupted record. The "packet size error" message clearly indicates that **nslookup** was even having trouble retrieving the HINFO record directly from the primary server. We contacted the administrator of the primary server and told him about the problem, pointing out the records that appeared to be in error. He discovered that he had forgotten to put an operating system entry on some of the HINFO records. He corrected this, and it fixed the problem.

Cache corruption

The problem described above was caused by having the name server cache corrupted by bad data. Cache corruption can occur even if your system is not a secondary server. Sometimes the root server entries in the cache become corrupted. Dumping the cache can help diagnose these types of problems.

For example, a user reported intermittent name server failures. She had no trouble with any hostnames within the local domain, or with some names outside the local domain, but names in several different remote domains would not resolve. **nslookup** tests produced no solid clues, so the name server cache was dumped and examined for problems. The root server entries were corrupted, so **named** was reloaded to clear the cache and reread the *named.ca* file. Here's how it was done.

---

* See Appendix D, *A dhcpd Reference*, for a detailed description of the HINFO statement.

The SIGINT signal causes named to dump the name server cache to the file */var/tmp/named_dump.db*. The following command passes named this signal:

```
# kill -INT `cat /etc/named.pid`
```

The process ID of named can be obtained from */etc/named.pid*, as in the example above, because named writes its process ID in that file during startup.*

Once SIGINT causes named to snapshot its cache to the file, we can then examine the first part of the file to see if the names and addresses of the root servers are correct. For example:

```
# head -10 /var/tmp/named_dump.db
; Dumped at Wed Sep 18 08:45:58 1991
; --- Cache & Data ---
$ORIGIN .
.       80805   IN      SOA     NS.NIC.DDN.MIL. HOSTMASTER.NIC.DDN.MIL.
            ( 910909 10800 900 604800 86400 )
        479912  IN      NS      NS.NIC.DDN.MIL.
        479912  IN      NS      AOS.BRL.MIL.
        479912  IN      NS      A.ISI.EDU.
        479912  IN      NS      C.NYSER.NET.
        479912  IN      NS      TERP.UMD.EDU.
```

The cache shown above is clean. If intermittent name server problems lead you to suspect a cache corruption problem, examine the cache and check the names and addresses of all the root servers. The following symptoms might indicate a problem with the root server cache:

- Incorrect root server names. The section on */etc/named.ca* in Chapter 8 explains how you can locate the correct root server names. The easiest way to do this is to get the file *domain/named.root* from the InterNIC.
- No address or an incorrect address for any of the servers. Again, the correct addresses are in *domain/named.root*.
- A name other than root (.) in the name field of the first root server NS record, or the wildcard character (*) occurring in the name field of a root or top-level name server. The structure of NS records is described in Appendix D.

A "bad cache" with multiple errors might look like this:

```
# head -10 /var/tmp/named_dump.db
; Dumped at Wed Sep 18 08:45:58 1991
; --- Cache & Data ---
$ORIGIN .
arpa    80805   IN      SOA     SRI-NIC.ARPA.  HOSTMASTER.SRI-NIC.ARPA.
            ( 910909 10800 900 604800 86400 )
        479912  IN      NS      NS.NIC.DDN.MIL.
```

* On our Linux system the process ID is written to */var/run/named.pid*.

```
          479912   IN      NS      AOS.BRL.MIL.
          479912   IN      NS      A.ISI.EDU.
          479912   IN      NS      C.NYSER.NET.
          479912   IN      NS      TERP.UMD.EDU.
*         479912   IN      NS      NS.FOO.MIL.
```

This contrived example has three glaring errors. The "arpa" entry in the first field of the SOA record is invalid, and is the most infamous form of cache corruption. The last NS record is also invalid. NS.FOO.MIL. is not a valid root server, and an asterisk (*) in the first field of a root server record is not normal.

If you see problems like these, force **named** to reload its cache with the SIGHUP signal as shown below:

```
# kill -HUP `cat /etc/named.pid`
```

This clears the cache and reloads the valid root server entries from your *named.ca* file.

If you know which system is corrupting your cache, instruct your system to ignore updates from the culprit by using the **bogusns** statement in the */etc/named.boot* file. The **bogusns** statement lists the IP addresses of name servers whose information cannot be trusted. For example, in the previous section we described a problem where *acorn.sales.nuts.com* (172.16.16.1) was causing cache corruption with improperly formatted HINFO records. The following entry in the *named.boot* file blocks queries to *acorn.sales.nuts.com* and thus blocks the cache corruption:

```
bogusns 172.16.16.1
```

The **bogusns** entry is only a temporary measure. It is designed to keep things running while the remote domain administrator has a chance to diagnose and repair the problem. Once the remote system is fixed, remove the **bogusns** entry from *named.boot*.

dig: An Alternative to nslookup

An alternative to **nslookup** for making name service queries is **dig**. **dig** queries are usually entered as single-line commands, while **nslookup** is usually run as an interactive session. But the **dig** command performs essentially the same function as **nslookup**. Which you use is mostly a matter of personal choice. They both work well.

As an example, we'll use **dig** to ask the root server *terp.umd.edu* for the NS records for the *mit.edu* domain. To do this, enter the following command:

```
% dig @terp.umd.edu mit.edu ns
```

In this example, *@terp.umd.edu* is the server that is being queried. The server can be identified by name or IP address. If you're troubleshooting a problem in a

remote domain, specify an authoritative server for that domain. In this example we're asking for the names of servers for a top-level domain (*mit.edu*), so we ask a root server.

If you don't specify a server explicitly, dig uses the local name server, or the name server defined in the */etc/resolv.conf* file. (Chapter 8 describes *resolv.conf*.) Optionally, you can set the environment variable LOCALRES to the name of an alternate *resolv.conf* file. This alternate file will then be used in place of */etc/resolv.conf* for dig queries. Setting the LOCALRES variable will only affect dig. Other programs that use name service will continue to use */etc/resolv.conf*.

The last item on our sample command line is *ns*. This is the query type. A query type is a value that requests a specific type of DNS information. It is similar to the value used in nslookup's set type command. Table 11-1 shows the possible dig query types and their meanings.

Table 11-1: dig Query Types

Query Type	DNS Record Requested
a	Address records
any	Any type of record
mx	Mail Exchange records
ns	Name Server records
soa	Start of Authority records
hinfo	Host Info records
axfr	All records in the zone
txt	Text records

Notice that the function of nslookup's ls command is performed by the dig query type axfr.

dig also has an option that is useful for locating a hostname when you have only an IP address. If you only have the IP address of a host, you may want to find out the hostname because numeric addresses are more prone to typos. Having the hostname can reduce the user's problems. The *in-addr.arpa* domain converts addresses to hostnames, and dig provides a simple way to enter *in-addr.arpa* domain queries. Using the -x option, you can query for a number to name conversion without having to manually reverse the numbers and add "in-addr.arpa." For example, to query for the hostname of IP address 18.72.0.3, simply enter:

```
% dig -x 18.72.0.3

; <<>> DiG 2.1 <<>> -x
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 0, Addit: 0
```

```
;; QUESTIONS:
;;      3.0.72.18.in-addr.arpa, type = ANY, class = IN

;; ANSWERS:
3.0.72.18.in-addr.arpa. 21600   PTR     BITSY.MIT.EDU.

;; Total query time: 74 msec
;; FROM: peanut to SERVER: default -- 172.16.12.1
;; WHEN: Sat Jul 12 11:12:55 1997
;; MSG SIZE  sent: 40  rcvd: 67
```

The answer to our query is BITSY.MIT.EDU, but dig displays lots of other output. The first five lines and the last four lines provide information and statistics about the query. For our purposes, the only important information is the answer.*

Analyzing Protocol Problems

Problems caused by bad TCP/IP configurations are much more common than problems caused by bad TCP/IP protocol implementations. Most of the problems you encounter will succumb to analysis using the simple tools we have already discussed. But on occasion, you may need to analyze the protocol interaction between two systems. In the worst case, you may need to analyze the packets in the data stream bit by bit. Protocol analyzers help you do this.

snoop is the tool we'll use. It is provided with the Solaris operating system.† Although we use snoop in all of our examples, the concepts introduced in this section should be applicable to the analyzer that you use, because most protocol analyzers function in basically the same way. Protocol analyzers allow you to select, or filter, the packets you want to examine, and to examine those packets byte by byte. We'll discuss both of these functions.

Protocol analyzers watch all the packets on the network. Therefore, you only need *one* system that runs analyzer software on the affected part of the network. One Solaris system with snoop can monitor the network traffic and tell you what the other hosts are (or aren't) doing. This, of course, assumes a shared media network. If you use an Ethernet switch, only the traffic on an individual segment can be seen. Some switches provide a monitor port. For others you may need to take your monitor to the location of the problem.

---

* To see a single-line answer to this query, pipe dig's output to grep; e.g., dig –x 18.72.0.3 | grep PTR.

† If you don't use Solaris, try tcpdump. It is available via anonymous FTP on the Internet and is similar to snoop.

Packet Filters

snoop reads all the packets on an Ethernet. It does this by placing the Ethernet interface into *promiscuous mode*. Normally, an Ethernet interface only passes packets up to the higher layer protocols that are destined for the local host. In promiscuous mode, all packets are accepted and passed to the higher layer. This allows snoop to view all packets and to select packets for analysis, based on a filter you define. Filters can be defined to capture packets from, or to, specific hosts, protocols, and ports, or combinations of all these. As an example, let's look at a very simple snoop filter. The following snoop command displays all packets sent between the hosts *almond* and *peanut*:

```
# snoop host almond and host peanut
Using device /dev/le (promiscuous mode)
peanut.nuts.com -> almond.nuts.com ICMP Echo request
almond.nuts.com -> peanut.nuts.com ICMP Echo reply
peanut.nuts.com -> almond.nuts.com RLOGIN C port=1023
almond.nuts.com -> peanut.nuts.com RLOGIN R port=1023
^C
```

The filter "host almond and host peanut" selects only those packets that are from *peanut* to *almond*, or from *almond* to *peanut*. The filter is constructed from a set of primitives, and associated hostnames, protocol names, and port numbers. The primitives can be modified and combined with the operators **and**, **or**, and **not**. The filter may be omitted; this causes snoop to display all packets from the network.

Table 11-2 shows the primitives used to build snoop filters. There are a few additional primitives and some variations that perform the same functions, but these are the essential primitive. See the snoop manpage for additional details.

Table 11-2: Expression Primitives

Primitive	Matches Packets
dst host | net | port *destination*	To *destination* host, net, or port
src host | net | port *source*	From *source* host, net, or port
host *destination*	To or from *destination* host
net *destination*	To or from *destination* network
port *destination*	To or from *destination* port
ether *address*	To or from Ethernet *address*
protocol	Of *protocol* type (icmp, udp, or tcp)

Using these primitives with the operators **and** and **or**, complex filters can be constructed. However, filters are usually simple. Capturing the traffic between two hosts is probably the most common filter. You may further limit the data captured to a specific protocol, but often you're not sure which protocol will reveal the problem. Just because the user sees the problem in ftp or telnet does not mean that is where the problem actually occurs. Analysis must often start by capturing

all packets, and can only be narrowed after test evidence points to some specific problem.

Modifying analyzer output

The example in the previous section shows that snoop displays a single line of summary information for each packet received. All lines show the source and destination addresses, and the protocol being used (ICMP and RLOGIN in the example). The lines that summarize the ICMP packets identify the packet types (Echo request and Echo reply in the example). The lines that summarize the application protocol packets display the source port and the first 20 characters of the packet data.

This summary information is sufficient to gain insight into how packets flow between two hosts and into potential problems. However, troubleshooting protocol problems requires more detailed information about each packet. snoop has options that give you control over what information is displayed. To display the data contained in a packet, use the **-x** option. It causes the entire contents of the packet to be dumped in hex and ASCII. In most cases, you don't need to see the entire packet; usually, the headers are sufficient to troubleshoot a protocol problem. The **-v** option displays the headers in a well-formatted and very detailed manner. Because of the large number of lines displayed for each packet, only use **-v** when you need it.

The following example shows an ICMP Echo Request packet displayed with the **-v** option. The same type of packet was summarized in the first line of the previous example.

```
# snoop -v host almond and host macadamia
Using device /dev/le (promiscuous mode)
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 3 arrived at 16:56:57.90
ETHER:  Packet size = 98 bytes
ETHER:  Destination = 8:0:20:22:fd:51, Sun
ETHER:  Source      = 0:0:c0:9a:d0:db, Western Digital
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 84 bytes
IP:   Identification = 3049
```

```
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 64 seconds/hops
IP:   Protocol = 1 (ICMP)
IP:   Header checksum = fde0
IP:   Source address = 172.16.55.106, macadamia.nuts.com
IP:   Destination address = 172.16.12.1, almond.nuts.com
IP:   No options
IP:
ICMP:  ----- ICMP Header -----
ICMP:
ICMP:  Type = 8 (Echo request)
ICMP:  Code = 0
ICMP:  Checksum = ac54
ICMP:
```

The detailed formatting done by **snoop** maps the bytes received from the network to the header structure. Look at the description of the various header fields in Chapter 1, *Overview of TCP/IP*, and Appendix F, *Selected TCP/IP Headers*, for more information.

Protocol Case Study

This example is an actual case that was solved by protocol analysis. The problem was reported as an occasional **ftp** failure with the error message:

```
netout: Option not supported by protocol
421 Service not available, remote server has closed connection
```

Only one user reported the problem, and it occurred only when transferring large files from a workstation to the central computer, via our FDDI backbone network.

We obtained the user's data file and were able to duplicate the problem from other workstations, but only when we transferred the file to the same central system via the backbone network. Figure 11-4 graphically summarizes the tests we ran to duplicate the problem.

We notified all users of the problem. In response, we received reports that others had also experienced it, but again only when transferring to the central system, and only when transferring via the backbone. They had not reported it, because they rarely saw it. But the additional reports gave us some evidence that the problem did not relate to any recent network changes.

Because the problem had been duplicated on other systems, it probably was not a configuration problem on the user's system. The **ftp** failure could also be avoided if the backbone routers and the central system did not interact. So we concentrated our attention on those systems. We checked the routing tables and ARP

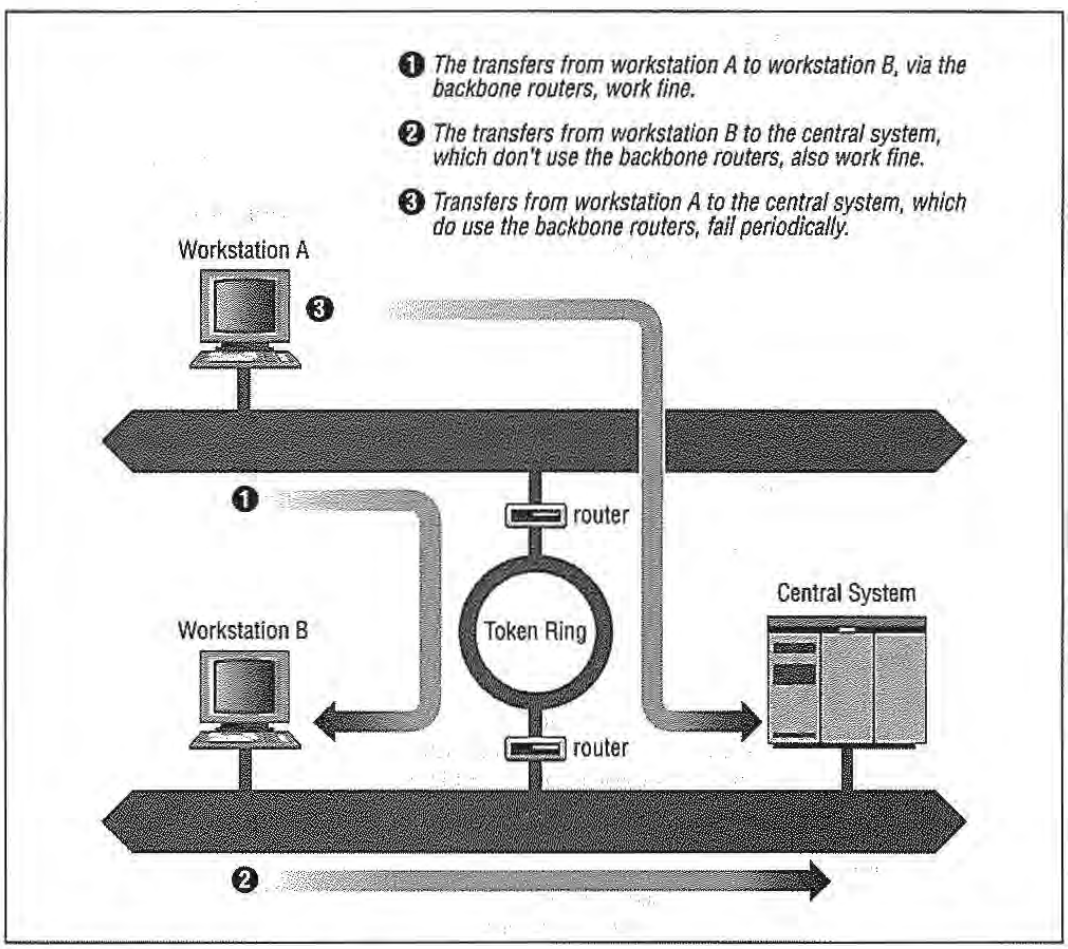


Figure 11-4: FTP test summary

tables, and ran **ping** tests on the central system and the routers. No problems were observed.

Based on this preliminary analysis, the **ftp** failure appeared to be a possible protocol interaction problem between a certain brand of routers and a central computer. We made that assessment because the transfer routinely failed when these two brands of systems were involved, but never failed in any other circumstance. If the router or the central system were misconfigured, they should fail when transferring data to other hosts. If the problem was an intermittent physical problem, it should occur randomly regardless of the hosts involved. Instead, this problem occurred predictably, and only between two specific brands of computers. Perhaps there was something incompatible in the way these two systems implemented TCP/IP.

Therefore, we used **snoop** to capture the TCP/IP headers during several **ftp** test runs. Reviewing the dumps showed that all transfers that failed with the "netout" error message had an ICMP Parameter Error packet near the end of the session,

usually about 50 packets before the final close. No successful transfer had this ICMP packet. Note that the error did *not* occur in the last packet in the data stream, as you might expect. It is common for an error to be detected, and for the data stream to continue for some time before the connection is actually shut down. Don't assume that an error will always be at the end of a data stream.

Here are the headers from the key packets. First, the IP header of the packet from the backbone router that caused the central system to send the error:

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 1 arrived at 16:56:36.39
ETHER:  Packet size = 60 bytes
ETHER:  Destination = 8:0:25:30:6:51, CDC
ETHER:  Source      = 0:0:93:e0:a0:bf, Proteon
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 552 bytes
IP:   Identification = 8a22
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 57 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = ffff
IP:   Source address = 172.16.55.106, fs.nuts.com
IP:   Destination address = 172.16.51.252, bnos.nuts.com
IP:   No options
IP:
```

And this is the ICMP Parameter Error packet sent from the central system in response to that packet:

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 3 arrived at 16:56:57.90
ETHER:  Packet size = 98 bytes
ETHER:  Destination = 0:0:93:e0:a0:bf, Proteon
ETHER:  Source      = 8:0:25:30:6:51, CDC
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
```

```
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 56 bytes
IP:   Identification = 000c
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 59 seconds/hops
IP:   Protocol = 1 (ICMP)
IP:   Header checksum = 8a0b
IP:   Source address = 172.16.51.252, bnos.nuts.com
IP:   Destination address = 172.16.55.106, fs.nuts.com
IP:   No options
IP:
ICMP:  ----- ICMP Header -----
ICMP:
ICMP:  Type = 12 (Parameter problem)
ICMP:  Code = 0
ICMP:  Checksum = 0d9f
ICMP:  Pointer = 10
```

Each packet header is broken out bit-by-bit and mapped to the appropriate TCP/IP header fields. From this detailed analysis of each packet, we see that the router issued an IP Header Checksum of 0xffff, and that the central system objected to this checksum. We know that the central system objected to the checksum because it returned an ICMP Parameter Error with a Pointer of 10. The Parameter Error indicates that there is something wrong with the data the system has just received, and the Pointer identifies the specific data that the system thinks is in error. The tenth byte of the router's IP header is the IP Header Checksum. The data field of the ICMP error message returns the header that it believes is in error. When we displayed that data we noticed that when the central system returned the header, the checksum field was "corrected" to 0000. Clearly the central system disagreed with the router's checksum calculation.

Occasional checksum errors will occur. They can be caused by transmission problems, and are intended to detect these types of problems. Every protocol suite has a mechanism for recovering from checksum errors. So how should they be handled in TCP/IP?

To determine the correct protocol action in this situation, we turned to the authoritative sources—the RFCs. RFC 791, *Internet Protocol*, provided information about the checksum calculation, but the best source for this particular problem was RFC 1122, *Requirements for Internet Hosts—Communication Layers*, by R. Braden. This

RFC provided two specific references that define the action to be taken. These excerpts are from page 29 of RFC 1122:

> In the following, the action specified in certain cases is to "silently discard" a received datagram. This means that the datagram will be discarded without further processing and that the host will not send any ICMP error message (see Section 3.2.2) as a result....
>
> ...
>
> A host MUST verify the IP header checksum on every received datagram and silently discard every datagram that has a bad checksum.

Therefore, when a system receives a packet with a bad checksum, it is not supposed to do anything with it. The packet should be discarded, and the system should wait for the next packet to arrive. The system should not respond with an error message. A system cannot respond to a bad IP header checksum, because it cannot really know where the packet came from. If the header checksum is in doubt, how do you know if the addresses in the header are correct? And if you don't know for sure where the packet came from, how can you respond to it?

IP relies on the upper-layer protocols to recover from these problems. If TCP is used (as it was in this case), the sending TCP eventually notices that the recipient has never acknowledged the segment, and it sends the segment again. If UDP is used, the sending application is responsible for recovering from the error. In neither case does recovery rely on an error message returned from the recipient.

Therefore, for an incorrect checksum, the central system should have simply discarded the bad packet. The vendor was informed of this problem and, much to their credit, they sent us a fix for the software within two weeks. Not only that, the fix worked perfectly!

Not all problems are resolved so cleanly. But the technique of analysis is the same no matter what the problem.

Simple Network Management Protocol

Troubleshooting is necessary to recover from problems, but the ultimate goal of the network administrator is to avoid problems. That is also the goal of network management software. The network management software used on TCP/IP networks is based on the *Simple Network Management Protocol* (SNMP).

SNMP is a client/server protocol. In SNMP terminology, it is described as a *manager/agent protocol.* The *agent* (the server) runs on the device being managed, which is called the *Managed Network Entity.* The agent monitors the status of the device and reports that status to the manager.

The *manager* (the client) runs on the *Network Management Station* (NMS). The NMS collects information from all of the different devices that are being managed, consolidates it, and presents it to the network administrator. This design places all of the data manipulation tools and most of the human interaction on the NMS. Concentrating the bulk of the work on the manager means that the agent software is small and easy to implement. Correspondingly, most TCP/IP network equipment comes with an SNMP management agent.

SNMP is a request/response protocol. UDP port 161 is its well-known port. SNMP uses UDP as its transport protocol because it has no need for the overhead of TCP. "Reliability" is not required because each request generates a response. If the SNMP application does not receive a response, it simply re-issues the request. "Sequencing" is not needed because each request and each response travels as a single datagram.

The request and response messages that SNMP sends in the datagrams are called *Protocol Data Units* (PDU). The five PDUs used by SNMP are listed in Table 11-3. These message types allow the manager to request management information, and when appropriate, to modify that information. The messages also allow the agent to respond to manager requests and to notify the manager of unusual situations.

Table 11-3: SNMP Protocol Data Units

PDU	Use
GetRequest	Manager requests an update.
GetNextRequest	Manager requests the next entry in a table.
GetResponse	Agent answers a manager request.
SetRequest	Manager modifies data on the managed device.
Trap	Agent alerts manager of an unusual event.

The NMS periodically requests the status of each managed device (GetRequest) and each agent responds with the status of its device (GetResponse). Making periodic requests is called *polling*. Polling reduces the burden on the agent because the NMS decides when polls are needed, and the agent simply responds. Polling also reduces the burden on the network because the polls originate from a single system at a predictable rate. The shortcoming of polling is that it does not allow for real-time updates. If a problem occurs on a managed device, the manager does not find out until the agent is polled. To handle this, SNMP uses a modified polling system called *trap-directed polling*.

A *trap* is an interrupt signaled by a predefined event. When a trap event occurs, the SNMP agent does not wait for the manager to poll; instead it immediately sends information to the manager. Traps allow the agent to inform the manager of unusual events while allowing the manager to maintain control of polling. SNMP

traps are sent on UDP port 162. The manager sends polls on port 161 and listens for traps on port 162. Table 11-4 lists the trap events defined in the RFCs.

Table 11-4: Generic Traps Defined in the RFCs

Trap	Meaning
coldStart	Agent restarted; possible configuration changes
warmStart	Agent reinitialized without configuration changes
enterpriseSpecific	An event significant to this hardware or software
authenticationFailure	Agent received an unauthenticated message
linkDown	Agent detected a network link failure
linkUp	Agent detected a network link coming up
egpNeighborLoss	The device's EGP neighbor is down

The last three entries in this table show the roots of SNMP in *Simple Gateway Management Protocol* (SGMP), which was a tool for tracking the status of network routers. Routers are generally the only devices that have multiple network links to keep track of and are the only devices that run *Exterior Gateway Protocol* (EGP).* These traps are not significant for most systems.

The most important trap may be the **enterpriseSpecific** trap. The events that signal this trap are defined differently by every vendor's SNMP agent software. Therefore it is possible for the trap to be tuned to events that are significant for that system. SNMP uses the term "enterprise" to refer to something that is privately defined by a vendor or organization as opposed to something that is globally defined by an RFC.

SNMP has twice as much jargon as the rest of networking—and that's saying something! Managed Network Entity, NMS, PDU, trap, polling, enterprise—that's just the beginning! We also need to mention (below) what SMI is, what a MIB is, and what ANS.1 is used for. Why this bewildering array of acronyms and buzzwords? I think there are two main reasons:

- Network management covers a wide range of different devices, from repeaters to mainframe computers. A "vendor-neutral" language is needed to define terms for the manufacturers of all of this different equipment.
- SNMP is based on the *Common Management Information Protocol* (CMIP) that was created by the *International Standards Organization* (ISO). Formal international standards always spend a lot of time defining terms because it is important to make terms clear when they are used by people from many different cultures who speak many different languages.

* EGP is covered in Chapter 7.

Now that you know why you have to suffer through all of this jargon, let's define a few more important terms.

The *Structure of Management Information* (SMI) defines how data should be presented in an SNMP environment. The SMI is documented in RFC 1155 and RFC 1065, *Structure and Identification of Management Information for TCP/IP-based Internets.* The SMI defines how managed objects are named, the syntax in which they are defined, and how they are encoded for transmission over the network. The SMI is based on previous ISO work.

Each managed object is given a globally unique name called an *object identifier.* The object identifier is part of a hierarchical name space that is managed by the ISO. The hierarchical name structure is used, just like it is in DNS, to guarantee that each name is globally unique. In an object identifier, each level of the hierarchy is identified by a number.

Objects are defined just as formally as they are named. The syntax used to define managed objects is *Abstract Syntax Notation One (ASN.1).* ASN.1 is ISO Standard 8824, *Specification of Abstract Syntax Notation One (ASN.1).* It is a very formal set of language rules for defining data. It makes the data definition independent of incompatibilities between systems and character sets. ASN.1 also includes a set of rules for encoding data for transfer over a network. These rules are defined in ISO Standard 8825, *Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).* The *Basic Encoding Rules* (BER) define that bit 8 of an octet is sent first, that 2's complement is used for signed integers, and other nitty-gritty details of data transmission.

Every object managed by SNMP has a unique object identifier defined by the ASN.1 syntax and encoding defined by BER. When all of these unique objects are grouped together, they are called the *Management Information Base* (MIB). The MIB refers to all information that is managed by SNMP. However, we usually refer to "a MIB" or "the MIBs" (plural), meaning the individual databases of management information formally defined by an RFC or privately defined by a vendor.

MIBI and MIBII are standards defined by RFCs. MIBII is a superset of MIBI, and is the standard MIB for monitoring TCP/IP. It provides such information as the number of packets transmitted into and out of an interface, and the number of errors that occurred sending and receiving those packets—useful information for spotting usage trends and potential trouble spots. Every agent supports MIBI or MIBII.

Some systems also provide a private MIB in addition to the standard MIBII. Private MIBs add to the monitoring capability by providing system-specific information. Most UNIX systems do not provide private MIBs. Private MIBs are most common on network hardware like routers, hubs, and switches.

No matter what MIBs are provided by the agents, it is the monitoring software that displays the information for the system administrator. A private MIB won't do you any good unless your network monitoring software also supports that MIB. For this reason, most administrators prefer to purchase a monitor from the vendor that supplies the bulk of their network equipment. Another possibility is to select a monitor that includes a *MIB compiler*, which gives you the most flexibility. A MIB compiler reads in the ASN.1 description of a MIB and adds the MIB to the monitor. A MIB compiler makes the monitor *extensible* because if you can get the ASN.1 source from the network equipment vendor, you can add the vendor's private MIB to your monitor.

MIB compilers are only part of the advanced features offered by some monitors. Some of the features offered are:

Network maps
: Some monitors automatically draw a map of the network. Colors are used to indicate the state (up, down, etc.) of the devices on the network. At a glance, the network manager sees the overall state of the network.

Tabular data displays
: Data displayed in tables or rendered into charts is used to make comparisons between different devices. Some monitors output data that can then be read into a standard spreadsheet or graphing program.

Filters
: Filters sift the data coming in from the agents in order to detect certain conditions.

Alarms
: Alarms indicate when "thresholds" are exceeded or special events occur. For example, you may want an alarm to trigger when your server exceeds some specified number of transmit errors.

Don't be put off by the jargon. All of this detail is necessary to formally define a network management scheme that is independent of the managed systems, but you don't need to memorize it. You need to know that a MIB is a collection of management information, that an NMS is the network management station, and that an agent runs in each managed device in order to make intelligent decisions when selecting an SNMP monitor. This information provides that necessary background. The features available in network monitors vary widely; so does the price. Select an SNMP monitor that is suitable for the complexity of your network and the size of your budget.