NEAR EAST UNIVERSITY

Faculty of Engineering

Department Of Computer Engineering

Transmission Control Protocol and Internet Protocol

Graduation Project
COM – 400

Student : **Mu'ayyad Dmour**

Supervisor: **Assoc.prof DOGAN IBRAHIM**

Nicosia - 2004

ACKNOWLEDGEMENTS

"First I would like to thank my supervisor Assoc. Prof. Dr. Dogan Ibrahim for his great advice and recommendations to finish this work properly.

Although I faced many problem collections data but has guiding me the appropriate references.(DR Dogan) thanks a lot for your invaluable and continual support.

Second, I would like to thank my family for their constant encouragement and support during the preparation of this work specially my brothers (Eyad and muhammad) .

Third, I thank all the staff of the faculty of engineering for giving me the facilities to practice and solving any problem I was facing during working in this project.

Forth I do not want to forget my best friends (Tarq mikki),(Waleed),(Adeeb) and all friends for helping me to finish this work in short time by their invaluable encouragement.

Finally thanks for all of my friends for their advices and support.

ABSTRACT

Transmission Control Protocol/Internet Protocol (TCP/IP) is an industry-standard suit of protocols designed for Wide Area Networks (WANs). The roots of the TCP/IP can be traced back to the packet switching network experiments conducted by the US Department of Defense Advanced Research Projects Agency (DARPA). IP is a connectionless protocol primarily responsible for addressing and routing packets between hosts, that is, a session is not established before exchanging data. IP is unreliable in that delivery is not guaranteed. An acknowledgement is not required when data is received. where as Transmission Control Protocol (TCP) is responsible for controlling the transmission of data from one host to another host. The TCP/IP utilities include File Transfer Protocol (FTP), Trivial File Transfer Protocol (TFTP), Remote Copy Protocol (RCP), Telnet, Remote Shell (RSH), Remote Execution (REXEC), Line Printer Remote (LPR), Line Printer Queue (LPQ), and Line Printer Daemon (LPD).

TABLE 0F CONTENTS

INTRODUCTION

In first chapter, we will see the relationship of OSI and TCP/IP layered architectures, a history of TCP/IP and the Internet, the structure of the Internet, Internet and IP addresses, and the Address Resolution Protocol. Using these concepts, we will then move on to look at the TCP/IP family of protocols in more detail.

The next chapter begins with the Internet Protocol (IP), showing how it is used and the format of its header information. The rest of the chapter covers gateway information necessary to piece together the rest of the protocols. We will start an in-depth look at the TCP/IP protocol family with the Internet Protocol. We will cover what IP is and how it does its task of passing datagrams between machines. The construction of the IP datagram and the format of the IP header will be shown in detail. The construction of the IP header is important to many TCP/IP family protocol members. We will also look at the Internet Control Message Protocol (ICMP), an important aspect of the TCP/IP system.

The third chapter moves to the next-higher layer in the TCP/IP layered architecture and looks at the Transmission Control Protocol (TCP). We will also look at the related User Datagram Protocol (UDP). TCP and UDP form the basis for all TCP/IP protocols. Here we will look at TCP in reasonable detail. Combined with the information in the last two chapters, we will now have the theory and background necessary to better understand TCP/IP utilities, such as Telnet and FTP, as well as other protocols that use or closely resemble TCP/IP, such as SMTP and TFTP.

In fourth chapter, we will see the basic functions performed by the socket API during establishment of a TCP or UDP call. We will also see the functions that are available to application programmers. Although the treatment has been at a high level, we should be able to see that working with sockets is not a complex, confusing task. Indeed, socket programming is surprisingly easy once we have tried it. Not everyone wants to write TCP or UDP applications, of course. However, understanding the basics of the socket API helps in understanding the protocol and troubleshooting.

CHAPTER 1

INTRODUCTION TO TCP/IP AND THE INTERNET

Just what is TCP/IP? It is a software-based communications protocol used in networking. Although the name TCP/IP implies that the entire scope of the product is a combination of two protocols—Transmission Control Protocol and Internet Protocol—the term TCP/IP refers not to a single entity combining two protocols, but a larger set of software programs that provides network services such as remote logins, remote file transfers, and electronic mail. TCP/IP provides a method for transferring information from one machine to another. A communications protocol should handle errors in transmission, manage the routing and delivery of data, and control the actual transmission by the use of predetermined status signals. TCP/IP accomplishes all of this.

OSI Reference Model is composed of seven layers. TCP/IP was designed with layers as well, although they do not correspond one-to-one with the OSI-RM layers. You can overlay the TCP/IP programs on this model to give you a rough idea of where all the TCP/IP layers reside. Figure 2.1 shows the basic elements of the TCP/IP family of protocols. We can see that TCP/IP is not involved in the bottom two layers of the OSI model (data link and physical) but begins in the network layer, where the Internet Protocol (IP) resides. In the transport layer, the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are involved. Above this, the utilities and protocols that make up the rest of the TCP/IP suite are built using the TCP or UDP and IP layers for their communications system.

Figure 2.1 shows that some of the upper-layer protocols depend on TCP (such as Telnet and FTP), whereas some depend on UDP (such as TFTP and RPC). Most upper-layer TCP/IP protocols use only one of the two transport protocols (TCP or UDP), although a few, including DNS (Domain Name System) can use both. A note of caution about TCP/IP: Despite the fact that TCP/IP is an open protocol, many companies have modified it for their own networking system. There can be incompatibilities because of these modifications, which, even though they might adhere to the official standards, might have other aspects that cause problems. Luckily, these types of changes are not

rampant, but you should be careful when choosing a TCP/IP product to ensure its compatibility with existing software and hardware.

Telnet - RemoteLogin
FTP - File Transfer Protocol
SMTP - Simple Mail Transfer Protocol
X - X Windows System
Kerberos - Security
DNS - Domain Name System
ASN - Abstract Syntax Notation
SNMP - Simple Network Management Protocol

NFS - Network File Server
RPC - Remote Procedure Calls
TFTP - Trivial File Transfer Protocol
TCP - Transmission Control Protocol
User Datagram Protocol
IP - Internet Protocol
ICMP - Internet Control Message Protocol

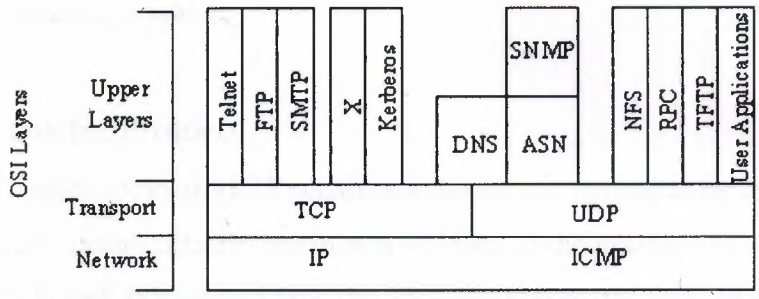

Figure 2.1. TCP/IP suite and OSI layers.

TCP/IP is dependent on the concept of clients and servers. This has nothing to do with a file server being accessed by a diskless workstation or PC. The term *client/server* has a simple meaning in TCP/IP: any device that initiates communications is the client, and the device that answers is the server. The server is responding to (serving) the client's requests.

1.1 An Overview of TCP/IP Components

To understand the roles of the many components of the TCP/IP protocol family, it is useful to know what you can do over a TCP/IP network. Then, once the applications are understood, the protocols that make it possible are a little easier to comprehend. The following list is not exhaustive but mentions the primary user applications that TCP/IP provides.

1.1.1 Telnet

The Telnet program provides a remote login capability. This lets a user on one machine log onto another machine and act as though he or she were directly in front of the second machine. The connection can be anywhere on the local network or on another network anywhere in the world, as long as the user has permission to log onto the remote system.

We can use Telnet when we need to perform actions on a machine across the country. This isn't often done except in a LAN or WAN context, but a few systems accessible through the Internet allow Telnet sessions while users play around with a new application or operating system

1.1.2 File Transfer Protocol

File Transfer Protocol (FTP) enables a file on one system to be copied to another system. The user doesn't actually log in as a full user to the machine he or she wants to access, as with Telnet, but instead uses the FTP program to enable access. Again, the correct permissions are necessary to provide access to the files.

Once the connection to a remote machine has been established, FTP enables us to copy one or more files to your machine. (The term *transfer* implies that the file is moved from one system to another but the original is not affected. Files are copied.) FTP is a widely used service on the Internet, as well as on many large LANs and WANs.

1.1.3 Simple Mail Transfer Protocol

Simple Mail Transfer Protocol (SMTP) is used for transferring electronic mail. SMTP is completely transparent to the user. Behind the scenes, SMTP connects to remote machines and transfers mail messages much like FTP transfers files. Users are almost never aware of SMTP working, and few system administrators have to bother with it. SMTP is a mostly trouble-free protocol and is in very wide use.

1.1.4 Kerberos

Kerberos is a widely supported security protocol. Kerberos uses a special application called an *authentication server* to validate passwords and encryption schemes. Kerberos is one of the more secure encryption systems used in

communications and is quite common in UNIX.

1.1.5 Domain Name System

Domain Name System (DNS) enables a computer with a common name to be converted to a special network address. For example, a PC called Darkstar cannot be accessed by another machine on the same network (or any other connected network) unless some method of checking the local machine name and replacing the name with the machine's hardware address is available. DNS provides a conversion from the common local name to the unique physical address of the device's network connection.

1.1.6 Simple Network Management Protocol

Simple Network Management Protocol (SNMP) provides status messages and problem reports across a network to an administrator. SNMP uses User Datagram Protocol (UDP) as a transport mechanism. SNMP employs slightly different terms from TCP/IP, working with managers and agents instead of clients and servers (although they mean essentially the same thing). An agent provides information about a device, whereas a manager communicates across a network with agents.

1.1.7 Network File System

Network File System (NFS) is a set of protocols developed by Sun Microsystems to enable multiple machines to access each other's directories transparently. They accomplish this by using a distributed file system scheme. NFS systems are common in large corporate environments, especially those that use UNIX workstations.

1.1.8 Remote Procedure Call

The Remote Procedure Call (RPC) protocol is a set of functions that enable an application to communicate with another machine (the server). It provides for programming functions, return codes, and predefined variables to support distributed computing.

1.1.9 Trivial File Transfer Protocol

Trivial File Transfer Protocol (TFTP) is a very simple, unsophisticated file transfer protocol that lacks security. It uses UDP as a transport. TFTP performs the same task as FTP, but uses a different transport protocol.

1.1.10 Transmission Control Protocol

Transmission Control Protocol (the TCP part of TCP/IP) is a communications protocol that provides reliable transfer of data. It is responsible for assembling data passed from higher-layer applications into standard packets and ensuring that the data is transferred correctly.

1.1.11 User Datagram Protocol

User Datagram Protocol (UDP) is a connectionless-oriented protocol, meaning that it does not provide for the retransmission of datagrams (unlike TCP, which is connection-oriented). UDP is not very reliable, but it does have specialized purposes. If the applications that use UDP have reliability checking built into them, the shortcomings of UDP are overcome.

1.1.12 Internet Protocol

Internet Protocol (IP) is responsible for moving the packets of data assembled by either TCP or UDP across networks. It uses a set of unique addresses for every device on the network to determine routing and destinations.

1.1.13 Internet Control Message Protocol

Internet Control Message Protocol (ICMP) is responsible for checking and generating messages on the status of devices on a network. It can be used to inform other devices of a failure in one particular machine. ICMP and IP usually work together.

1.2 TCP/IP History

The architecture of TCP/IP is often called the Internet architecture because TCP/IP and the Internet as so closely interwoven. We have seen how the Internet standards were developed by the Defense Advanced Research Projects Agency (DARPA) and eventually passed on to the Internet Society.

The Internet was originally proposed by the precursor of DARPA, called the Advanced Research Projects Agency (ARPA), as a method of testing the viability of packet-switching networks. (When ARPA's focus became military in nature, the name was changed.) During its tenure with the project, ARPA foresaw a network of leased lines connected by switching nodes. The network was called ARPANET, and the switching nodes were called Internet Message Processors, or IMPs. The ARPANET was initially to be comprised of four IMPs located at the University of California at Los Angeles, the University of California at Santa Barbara, the Stanford Research Institute, and the University of Utah. The original IMPs were to be Honeywell 316 minicomputers.

The contract for the installation of the network was won by Bolt, Beranek, and Newman (BBN), a company that had a strong influence on the development of the network in the following years. The contract was awarded in late 1968, followed by testing and refinement over the next five years. In 1971, ARPANET entered into regular service. Machines used the ARPANET by connecting to an IMP using the "1822" protocol—so called because that was the number of the technical paper describing the system. During the early years, the purpose and utility of the network was widely (and sometimes heatedly) discussed, leading to refinements and modifications as users requested more functionality from the system.

A commonly recognized need was the capability to transfer files from one machine to another, as well as the capability to support remote logins. Remote logins would enable a user in Santa Barbara to connect to a machine in Los Angeles over the network and function as though he or she were in front of the UCLA machine. The protocol then in use on the network wasn't capable of handling these new functionality requests, so new protocols were continually developed, refined, and tested.

Remote login and remote file transfer were finally implemented in a protocol called the Network Control Program (NCP). Later, electronic mail was added through File Transfer Protocol (FTP). Together with NCP's remote logins and file transfer, this formed the basic services for ARPANET. By 1973, it was clear that NCP was unable to handle the volume of traffic and proposed new functionality. A project was begun to develop a new protocol. The TCP/IP and gateway architectures were first proposed in 1974. The published article by Cerf and Kahn described a system that provided a standardized application protocol that also used end-to-end acknowledgments.

Neither of these concepts were really novel at the time, but more importantly (and with considerable vision), Cerf and Kahn suggested that the new protocol be independent of the underlying network and computer hardware. Also, they proposed universal connectivity throughout the network. These two ideas were radical in a world of proprietary hardware and software, because they would enable any kind of platform to participate in the network. The protocol was developed and became known as TCP/IP.

A series of RFCs (Requests for Comment, part of the process for adopting new Internet Standards) was issued in 1981, standardizing TCP/IP version 4 for the ARPANET. In 1982, TCP/IP supplanted NCP as the dominant protocol of the growing network, which was now connecting machines across the continent. It is estimated that a new computer was connected to ARPANET every 20 days during its first decade. (That might not seem like much compared to the current estimate of the Internet's size doubling every year, but in the early 1980s it was a phenomenal growth rate.)

During the development of ARPANET, it became obvious that nonmilitary researchers could use the network to their advantage, enabling faster communication of ideas as well as faster physical data transfer. A proposal to the National Science Foundation lead to funding for the Computer Science Network in 1981, joining the military with educational and research institutes to refine the network. This led to the splitting of the network into two different networks in 1984. MILNET was dedicated to unclassified military traffic, whereas ARPANET was left for research and other nonmilitary purposes. ARPANET's growth and subsequent demise came with the approval for the Office of Advanced Scientific Computing to develop wide access to supercomputers. They created NSFNET to connect six supercomputers spread across the country through T-1 lines (which operated at 1.544 Mbps). The Department of Defense finally declared ARPANET obsolete in 1990, when it was officially dismantled.

1.3 Berkeley UNIX Implementations and TCP/IP

TCP/IP became important when the Department of Defense started including the protocols as military standards, which were required for many contracts. TCP/IP became popular primarily because of the work done at UCB (Berkeley). UCB had been a center of UNIX development for years, but in 1983 they released a new version that

incorporated TCP/IP as an integral element. That version—4.2BSD (Berkeley System Distribution)—was made available to the world as public domain software.

The popularity of 4.2BSD spurred the popularity of TCP/IP, especially as more sites connected to the growing ARPANET. Berkeley released an enhanced version (which included the so-called Berkeley Utilities) in 1986 as 4.3BSD. An optimized TCP implementation followed in 1988 (4.3BSD/Tahoe). Practically every version of TCP/IP available today has its roots (and much of its code) in the Berkeley versions.

1.4 OSI and TCP/IP

The adoption of TCP/IP didn't conflict with the OSI standards because the two developed concurrently. In some ways, TCP/IP contributed to OSI, and vice-versa. Several important differences do exist, though, which arise from the basic requirements of TCP/IP which are:

- A common set of applications
- Dynamic routing
- Connectionless protocols at the networking level
- Universal connectivity
- Packet-switching

The differences between the OSI architecture and that of TCP/IP relate to the layers above the transport level and those at the network level. OSI has both the session layer and the presentation layer, whereas TCP/IP combines both into an application layer. The requirement for a connectionless protocol also required TCP/IP to combine OSI's physical layer and data link layer into a network level. TCP/IP also includes the session and presentation layers of the OSI model into TCP/IP's application layer. A schematic view of TCP/IP's layered structure compared with OSI's seven-layer model is shown in Figure 2.2. TCP/IP calls the different network level elements *subnetworks.*

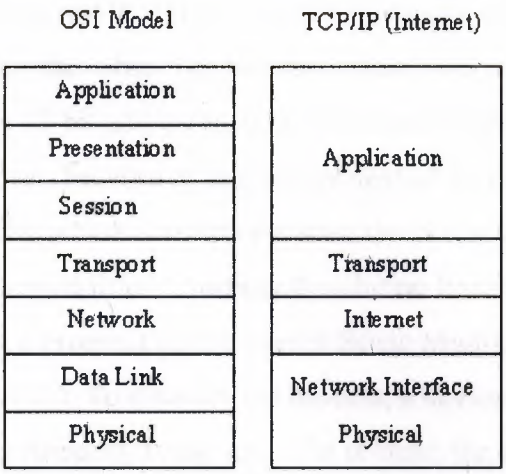

Figure 2.2. The OSI and TCP/IP layered structures.

Some fuss was made about the network level combination, although it soon became obvious that the argument was academic, as most implementations of the OSI model combined the physical and link levels on an intelligent controller (such as a network card). The combination of the two layers into a single layer had one major benefit: it enabled a subnetwork to be designed that was independent of any network protocols, because TCP/IP was oblivious to the details. This enabled proprietary, self-contained networks to implement the TCP/IP protocols for connectivity outside their closed systems.

The layered approach gave rise to the name TCP/IP. The transport layer uses the Transmission Control Protocol (TCP) or one of several variants, such as the User Datagram Protocol (UDP). (There are other protocols in use, but TCP and UDP are the most common.) There is, however, only one protocol for the network level—the Internet Protocol (IP). This is what assures the system of universal connectivity, one of the primary design goals.

There is a considerable amount of pressure from the user community to abandon the OSI model (and any future communications protocols developed that conform to it) in favor of TCP/IP. The argument hinges on some obvious reasons:

- TCP/IP is up and running and has a proven record.
- TCP/IP has an established, functioning management body.

Ethernet and TCP/IP work well together, with Ethernet providing the physical cabling (layers one and two) and TCP/IP the communications protocol (layers three and four) that is broadcast over the cable. The two have their own processes for packaging information: TCP/IP uses 32-bit addresses, whereas Ethernet uses a 48-bit scheme. The two work together, however, because of one component of TCP/IP called the Address Resolution Protocol (ARP), which converts between the two schemes. (I discuss ARP in more detail later, in the section titled "Address Resolution Protocol.")

Ethernet relies on a protocol called Carrier Sense Multiple Access with Collision Detect (CSMA/CD). To simplify the process, a device checks the network cable to see if anything is currently being sent. If it is clear, the device sends its data. If the cable is busy (carrier detect), the device waits for it to clear. If two devices transmit at the same time (a collision), the devices know because of their constant comparison of the cable traffic to the data in the sending buffer. If a collision occurs, the devices wait a random amount of time before trying again.

1.6 The Internet

As ARPANET grew out of a military-only network to add subnetworks in universities, corporations, and user communities, it became known as the Internet. There is no single network called the Internet, however. The term refers to the collective network of subnetworks. The one thing they all have in common is TCP/IP as a communications protocol.

As described in the first chapter, the organization of the Internet and adoption of new standards is controlled by the Internet Advisory Board (IAB). Among other things, the IAB coordinates several task forces, including the Internet Engineering Task Force (IETF) and Internet Research Task Force (IRTF). In a nutshell, the IRTF is concerned with ongoing research, whereas the IETF handles the implementation and engineering aspects associated with the Internet.

A body that has some bearing on the IAB is the Federal Networking Council (FNC), which serves as an intermediary between the IAB and the government. The FNC has an advisory capacity to the IAB and its task forces, as well as the responsibility for managing the government's use of the Internet and other networks. Because the government was responsible for funding the development of the Internet, it retains a considerable amount of control, as well as sponsoring some research and expansion of the Internet.

- Thousands of applications currently use TCP/IP and its well-documented application programming interfaces.
- TCP/IP is the basis for most UNIX systems, which are gaining the largest share of the operating system market (other than desktop single-user machines such as the PC and Macintosh).
- TCP/IP is vendor-independent.

Arguing rather strenuously against TCP/IP, surprisingly enough, is the US government—the very body that sponsored it in the first place. Their primary argument is that TCP/IP is not an internationally adopted standard, whereas OSI has that recognition. The Department of Defense has even begun to move its systems away from the TCP/IP protocol set. A compromise will probably result, with some aspects of OSI adopted into the still-evolving TCP/IP protocol suite.

1.5 TCP/IP and Ethernet

For many people the terms TCP/IP and Ethernet go together almost automatically, primarily for historical reasons, as well as the simple fact that there are more Ethernet-based TCP/IP networks than any other type. Ethernet was originally developed at Xerox's Palo Alto Research Center as a step toward an electronic office communications system, and it has since grown in capability and popularity.

Ethernet is a hardware system providing for the data link and physical layers of the OSI model. As part of the Ethernet standards, issues such as cable type and broadcast speeds are established. There are several different versions of Ethernet, each with a different data transfer rate. The most common is Ethernet version 2, also called 10Base5, Thick Ethernet, and IEEE 802.3 (after the number of the standard that defines the system adopted by the Institute of Electrical and Electronic Engineers). This system has a 10 Mbps rate.

There are several commonly used variants of Ethernet, such as Thin Ethernet (called 10Base2), which can operate over thinner cable (such as the coaxial cable used in cable television systems), and Twisted-Pair Ethernet (10BaseT), which uses simple twisted-pair wires similar to telephone cable. The latter variant is popular for small companies because it is inexpensive, easy to wire, and has no strict requirements for distance between machines.

1.6.1 The Structure of the Internet

As mentioned earlier, the Internet is not a single network but a collection of networks that communicate with each other through gateways. For the purposes of this chapter, a *gateway* (sometimes called a *router*) is defined as a system that performs relay functions between networks, as shown in Figure 2.3. The different networks connected to each other through gateways are often called subnetworks, because they are a smaller part of the larger overall network. This does not imply that a subnetwork is small or dependent on the larger network. Subnetworks are complete networks, but they are connected through a gateway as a part of a larger internetwork, or in this case the Internet.

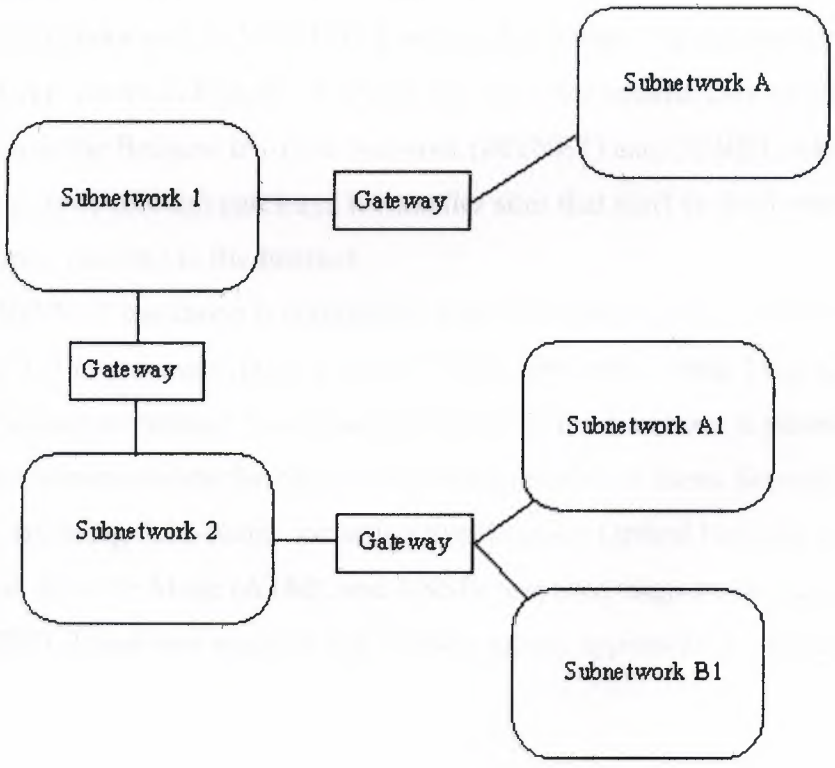

Figure 2.3. Gateways act as relays between subnetworks.

With TCP/IP, all interconnections between physical networks are through gateways. An important point to remember for use later is that gateways route information packets based on their destination network name, not the destination

machine. Gateways are supposed to be completely transparent to the user, which alleviates the gateway from handling user applications (unless the machine that is acting as a gateway is also someone's work machine or a local network server, as is often the case with small networks). Put simply, the gateway's sole task is to receive a Protocol Data Unit (PDU) from either the internetwork or the local network and either route it on to the next gateway or pass it into the local network for routing to the proper user.

Gateways work with any kind of hardware and operating system, as long as they are designed to communicate with the other gateways they are attached to (which in this case means that it uses TCP/IP). Whether the gateway is leading to a Macintosh network, a set of IBM PCs, or mainframes from a dozen different companies doesn't matter to the gateway or the PDUs it handles.

In the United States, the Internet has the NFSNET as its backbone, as shown in Figure 2.4. Among the primary networks connected to the NFSNET are NASA's Space Physics Analysis Network (SPAN), the Computer Science Network (CSNET), and several other networks such as WESTNET and the San Diego Supercomputer Network *(SDSCNET), not shown in Figure 2.4. There are also other smaller user-oriented* networks such as the Because It's Time Network (BITNET) and UUNET, which provide connectivity through gateways for smaller sites that can't or don't want to establish a direct gateway to the Internet.

The NFSNET backbone is comprised of approximately 3,000 research sites, connected by T-3 leased lines running at 44.736 Megabits per second. Tests are currently underway to increase the operational speed of the backbone to enable more throughput and accommodate the rapidly increasing number of users. Several technologies are being field-tested, including Synchronous Optical Network (SONET), Asynchronous Transfer Mode (ATM), and ANSI's proposed High-Performance Parallel Interface (HPPI). These new systems can produce speeds approaching 1 Gigabit per second.

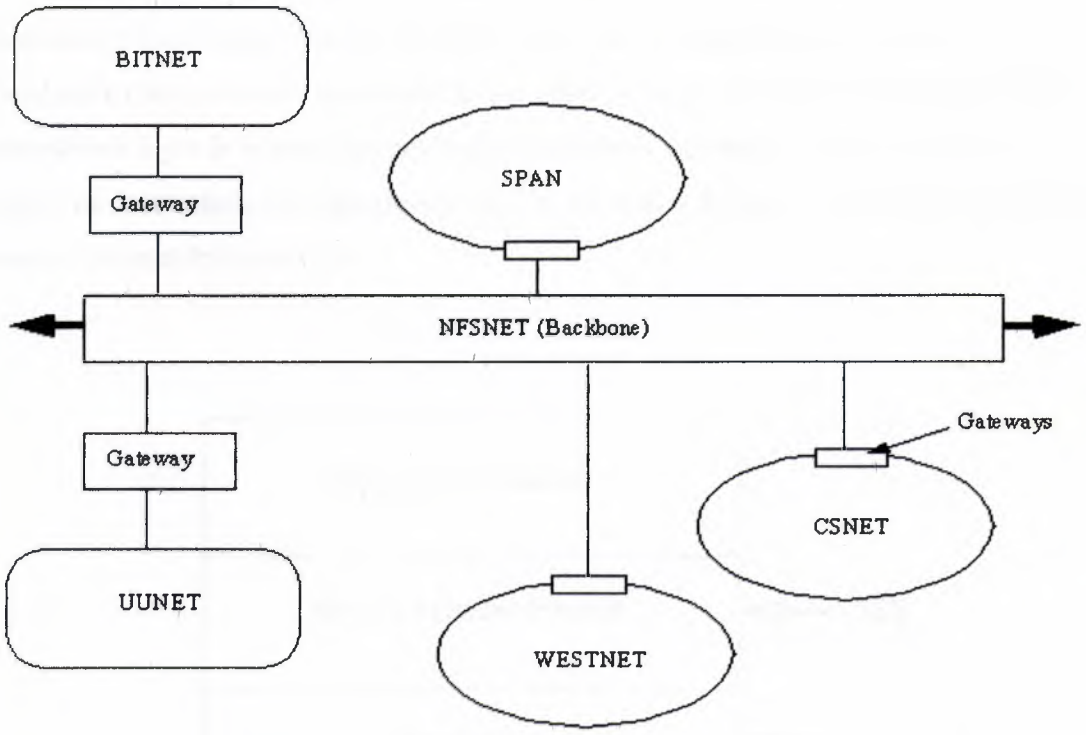

Figure 2.4. The US Internet network.

1.6.2 The Internet Layers

Most internetworks, including the Internet, can be thought of as a layered architecture (yes, even more layers!) to simplify understanding. The layer concept helps in the task of developing applications for internetworks. The layering also shows how the different parts of TCP/IP work together. The more logical structure brought about by using a layering process has already been seen in the first chapter for the OSI model, so applying it to the Internet makes sense. Be careful to think of these layers as conceptual only; they are not really physical or software layers as such (unlike the OSI or TCP/IP layers).

It is convenient to think of the Internet as having four layers. This layered Internet architecture is shown in Figure 2.5. These layers should not be confused with the architecture of each machine, as described in the OSI seven-layer model. Instead, they are a method of seeing how the internetwork, network, TCP/IP, and the individual machines work together. Independent machines reside in the subnetwork layer at the bottom of the architecture, connected together in a local area network (LAN) and referred to as the subnetwork, a term you saw in the last section.

On top of the subnetwork layer is the internetwork layer, which provides the functionality for communications between networks through gateways. Each subnetwork uses gateways to connect to the other subnetworks in the internetwork. The internetwork layer is where data gets transferred from gateway to gateway until it reaches its destination and then passes into the subnetwork layer. The internetwork layer runs the Internet Protocol (IP).

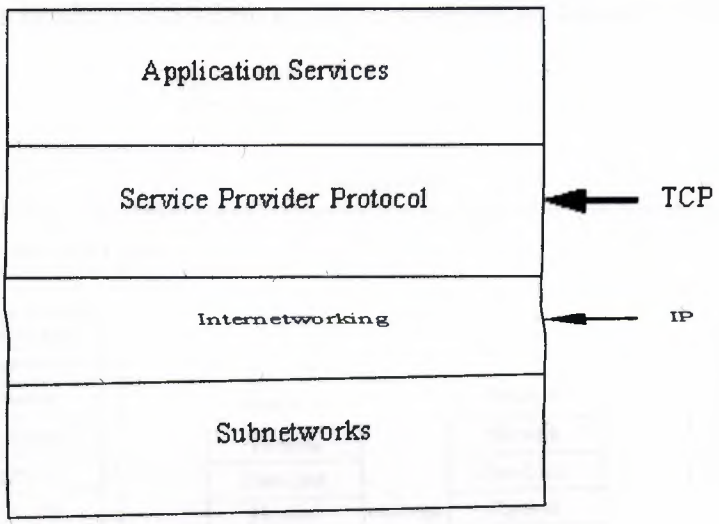

Figure 2.5. The Internet architecture.

The service provider protocol layer is responsible for the overall end-to-end communications of the network. This is the layer that runs the Transmission Control Protocol (TCP) and other protocols. It handles the data traffic flow itself and ensures reliability for the message transfer.

The top layer is the application services layer, which supports the interfaces to the user applications. This layer interfaces to electronic mail, remote file transfers, and remote access. Several protocols are used in this layer, many of which you will read about later. To see how the Internet architecture model works, a simple example is useful. Assume that an application on one machine wants to transfer a datagram to an application on another machine in a different subnetwork. Without all the signals between layers, and simplifying the architecture a little, the process is shown in Figure

2.6. The layers in the sending and receiving machines are the OSI layers, with the equivalent Internet architecture layers indicated.

The data is sent down the layers of the sending machine, assembling the datagram with the Protocol Control Information (PCI) as it goes. From the physical layer, the datagram (which is sometimes called a *frame* after the data link layer has added its header and trailing information) is sent out to the local area network. The LAN routes the information to the gateway out to the internetwork. During this process, the LAN has no concern about the message contained in the datagram. Some networks, however, alter the header information to show, among other things, the machines it has passed through.

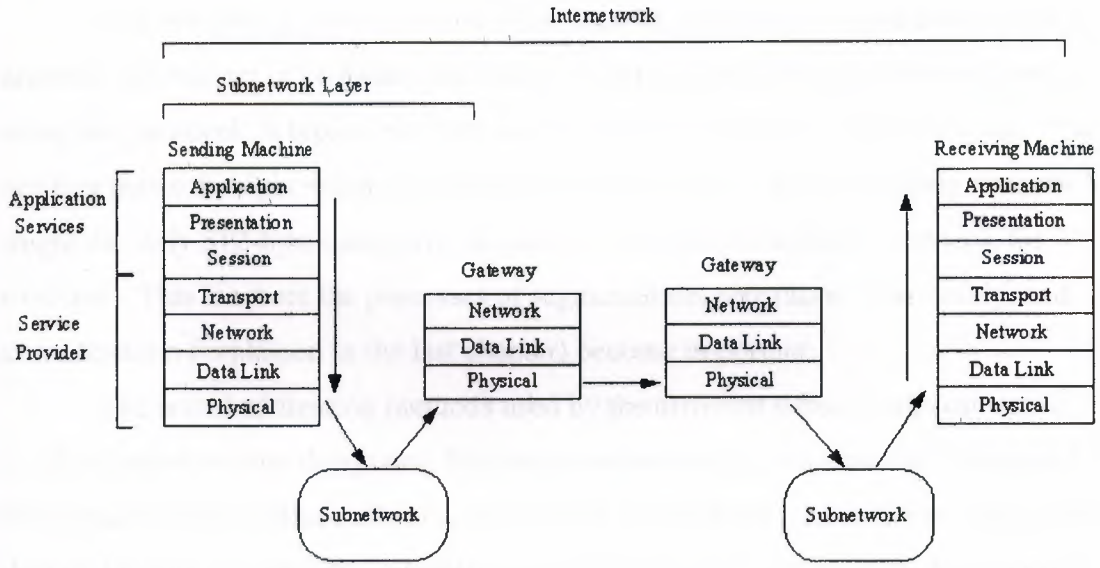

Figure 2.6. Transfer of a datagram over an internetwork.

From the gateway, the frame passes from gateway to gateway along the internetwork until it arrives at the destination subnetwork. At each step, the gateway analyzes the datagram's header to determine if it is for the subnetwork the gateway leads to. If not, it routes the datagram back out over the internetwork. This analysis is performed in the physical layer, eliminating the need to pass the frame up and down through different layers on each gateway. The header can be altered at each gateway to reflect its routing path.

When the datagram is finally received at the destination subnetwork's gateway, the gateway recognizes that the datagram is at its correct subnetwork and routes it into

the LAN and eventually to the target machine. The routing is accomplished by reading the header information. When the datagram reaches the destination machine, it passes up through the layers, with each layer stripping off its PCI header and then passing the result on up. At long last, the application layer on the destination machine processes the final header and passes the message to the correct application.

If the datagram was not data to be processed but a request for a service, such as a remote file transfer, the correct layer on the destination machine would decode the request and route the file back over the internetwork to the original machine. Quite a process!

1.6.3 Internetwork Problems

Not everything goes smoothly when transferring data from one subnetwork to another. All manner of problems can occur, despite the fact that the entire network is using one protocol. A typical problem is a limitation on the size of the datagram. The sending network might support datagrams of 1,024 bytes, but the receiving network might use only 512-byte datagrams (because of a different hardware protocol, for example). This is where the processes of segmentation, separation, reassembly, and concatenation (explained in the last chapter) become important.

The actual addressing methods used by the different subnetworks can cause conflicts when routing datagrams. Because communicating subnetworks might not have the same network control software, the network-based header information might differ, despite the fact that the communications methods are based on TCP/IP. An associated problem occurs when dealing with the differences between physical and logical machine names. In the same manner, a network that requires encryption instead of clear-text datagrams can affect the decoding of header information. Therefore, differences in the security implemented on the subnetworks can affect datagram traffic. These differences can all be resolved with software, but the problems associated with addressing methods can become considerable.

Another common problem is the different networks' tolerance for timing problems. Time-out and retry values might differ, so when two subnetworks are trying to establish communication, one might have given up and moved on to another task while the second is still waiting patiently for an acknowledgment signal. Also, if two subnetworks are communicating properly and one gets busy and has to pause the communications process for a short while, the amount of time before the other network

assumes a disconnection and gives up might be important. Coordinating the timing over the internetwork can become very complicated.

Routing methods and the speed of the machines on the network can also affect the internetwork's performance. If a gateway is managed by a particularly slow machine, the traffic coming through the gateway can back up, causing delays and incomplete transmissions for the entire internetwork. Developing an internetwork system that can dynamically adapt to loads and reroute datagrams when a bottleneck occurs is very important.

There are other factors to consider, such as network management and troubleshooting information, but you should begin to see that simply connecting networks together without due thought does not work. The many different network operating systems and hardware platforms require a logical, well-developed approach to the internetwork. This is outside the scope of TCP/IP, which is simply concerned with the transmission of the datagrams. The TCP/IP implementations on each platform, however, must be able to handle the problems mentioned.

1.7 Internet Addresses

Network addresses are analogous to mailing addresses in that they tell a system where to deliver a datagram. Three terms commonly used in the Internet relate to addressing: name, address, and route.

A *name* is a specific identification of a machine, a user, or an application. It is usually unique and provides an absolute target for the datagram. An *address* typically identifies where the target is located, usually its physical or logical location in a network. A *route* tells the system how to get a datagram to the address.

You use the recipient's name often, either specifying a user name or a machine name, and an application does the same thing transparently to you. From the name, a network software package called the *name server* tries to resolve the address and the route, making that aspect unimportant to you. When you send electronic mail, you simply indicate the recipient's name, relying on the name server to figure out how to get the mail message to them.

Using a name server has one other primary advantage besides making the addressing and routing unimportant to the end user: It gives the system or network administrator a lot of freedom to change the network as required, without having to tell

each user's machine about any changes. As long as an application can access the name server, any routing changes can be ignored by the application and users.

Naming conventions differ depending on the platform, the network, and the software release, but following is a typical Ethernet-based Internet subnetwork as an example. There are several types of addressing you need to look at, including the LAN system, as well as the wider internetwork addressing conventions.

1.7.1 Subnetwork Addressing

On a single network, several pieces of information are necessary to ensure the correct delivery of data. The primary components are the physical address and the data link address.

1.7.1.1 The Physical Address

Each device on a network that communicates with others has a unique *physical address,* sometimes called the *hardware address.* On any given network, there is only one occurrence of each address; otherwise, the name server has no way of identifying the target device unambiguously. For hardware, the addresses are usually encoded into a network interface card, set either by switches or by software. With respect to the OSI model, the address is located in the physical layer. In the physical layer, the analysis of each incoming datagram (or protocol data unit) is performed. If the recipient's address matches the physical address of the device, the datagram can be passed up the layers. If the addresses don't match, the datagram is ignored. Keeping this analysis in the bottom layer of the OSI model prevents unnecessary delays, because otherwise the datagram would have to be passed up to other layers for analysis.

The length of the physical address varies depending on the networking system, but Ethernet and several others use 48 bits in each address. For communication to occur, two addresses are required: one each for the sending and receiving devices. The IEEE is now handling the task of assigning universal physical addresses for subnetworks (a task previously performed by Xerox, as they developed Ethernet). For each subnetwork, the IEEE assigns an organization unique identifier (OUI) that is 24 bits long, enabling the organization to assign the other 24 bits however it wants. (Actually, two of the 24 bits assigned as an OUI are control bits, so only 22 bits identify the subnetwork. Because

this provides 2^{22} combinations, it is possible to run out of OUIs in the future if the current rate of growth is sustained.)

The format of the OUI is shown in Figure 2.7. The least significant bit of the address (the lowest bit number) is the individual or group address bit. If the bit is set to 0, the address refers to an individual address; a setting of 1 means that the rest of the address field identifies a group address that needs further resolution. If the entire OUI is set to 1s, the address has a special meaning which is that all stations on the network are assumed to be the destination.

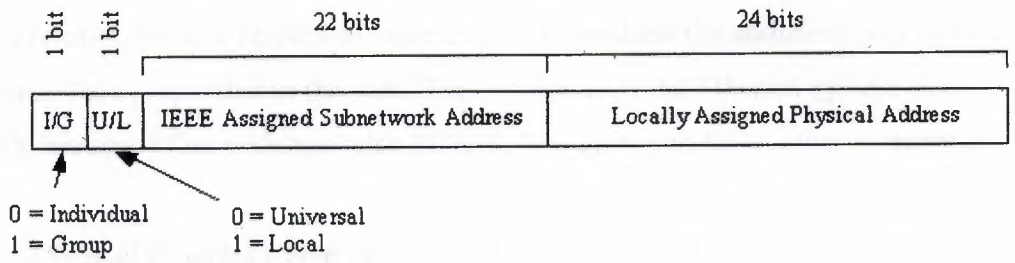

Figure 2.7. Layout of the organization unique identifier.

The second bit is the *local* or *universal* bit. If set to zero, it has been set by the universal administration body. This is the setting for IEEE-assigned OUIs. If it has a value of 1, the OUI has been locally assigned and would cause addressing problems if decoded as an IEEE-assigned address. The remaining 22 bits make up the physical address of the subnetwork, as assigned by the IEEE. The second set of 24 bits identifies local network addresses and is administered locally. If an organization runs out of physical addresses (there are about 16 million addresses possible from 24 bits), the IEEE has the capacity to assign a second subnetwork address.

The combination of 24 bits from the OUI and 24 locally assigned bits is called a media access control (MAC) address. When a packet of data is assembled for transfer across an internetwork, there are two sets of MACs: one from the sending machine and one for the receiving machine.

1.7.1.2 The Data Link Address

The IEEE Ethernet standards (and several other allied standards) use another address called the link layer address (abbreviated as LSAP for link service access point). The LSAP identifies the type of link protocol used in the data link layer. As with the physical address, a datagram carries both sending and receiving LSAPs. The IEEE also enables a code that identifies the EtherType assignment, which identifies the upper layer protocol (ULP) running on the network (almost always a LAN).

1.7.1.3 Ethernet Frames

The layout of information in each transmitted packet of data differs depending on the protocol, but it is helpful to examine one to see how the addresses and related information are prepended to the data. This section uses the Ethernet system as an example because of its wide use with TCP/IP. It is quite similar to other systems as well.

A typical Ethernet frame (remember that a frame is the term for a network-ready datagram) is shown in Figure 2.8. The preamble is a set of bits that are used primarily to synchronize the communication process and account for any random noise in the first few bits that are sent. At the end of the preamble is a sequence of bits that are the start frame delimiter (SFD), which indicates that the frame follows immediately.

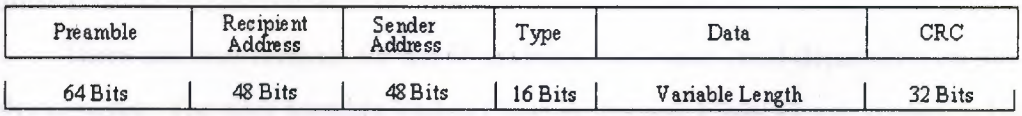

Figure 2.8. The Ethernet frame.

The recipient and sender addresses follow in IEEE 48-bit format, followed by a 16-bit type indicator that is used to identify the protocol. The data follows the type indicator. The Data field is between 46 and 1,500 bytes in length. If the data is less than 46 bytes, it is padded with 0s until it is 46 bytes long. Any padding is not counted in the calculations of the data field's total length, which is used in one part of the IP header. The next chapter covers IP headers.

At the end of the frame is the cyclic redundancy check (CRC) count, which is used to ensure that the frame's contents have not been modified during the transmission process. Each gateway along the transmission route calculates a CRC value for the frame and compares it to the value at the end of the frame. If the two match, the frame can be sent farther along the network or into the subnetwork. If they differ, a modification to the frame must have occurred, and the frame is discarded (to be later retransmitted by the sending machine when a timer expires). In some protocols, such as the IEEE 802.3, the overall layout of the frame is the same, with slight variations in the contents. With 802.3, the 16 bits used by Ethernet to identify the protocol type are replaced with a 16-bit value for the length of the data block. Also, the data area itself is prepended by a new field.

1.8 IP Addresses

TCP/IP uses a 32-bit address to identify a machine on a network and the network to which it is attached. IP addresses identify a machine's connection to the network, not the machine itself—an important distinction. Whenever a machine's location on the network changes, the IP address must be changed, too. The IP address is the set of numbers many people see on their workstations or terminals, such as 127.40.8.72, which uniquely identifies the device. IP (or Internet) addresses are assigned only by the Network Information Center (NIC), although if a network is not connected to the Internet, that network can determine its own numbering. For all Internet accesses, the IP address must be registered with the NIC.

There are four formats for the IP address, with each used depending on the size of the network. The four formats, called Class A through Class D, are shown in Figure 2.9. The class is identified by the first few bit sequences, shown in the figure as one bit for Class A and up to four bits for Class D. The class can be determined from the first three (high-order) bits. In fact, in most cases, the first two bits are enough, because there are few Class D networks.

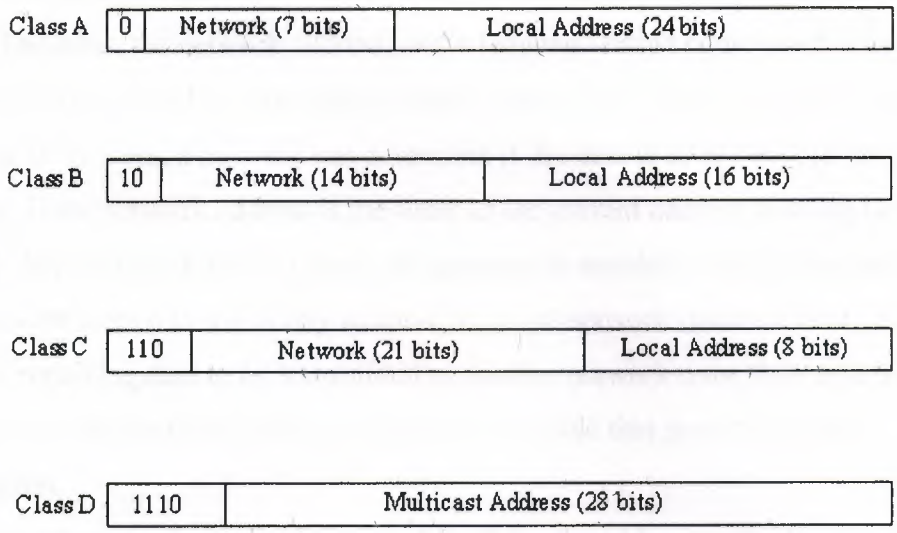

Figure 2.9. The four IP address class structures.

Class A addresses are for large networks that have many machines. The 24 bits for the local address (also frequently called the host address) are needed in these cases. The network address is kept to 7 bits, which limits the number of networks that can be identified. Class B addresses are for intermediate networks, with 16-bit local or host addresses and 14-bit network addresses. Class C networks have only 8 bits for the local or host address, limiting the number of devices to 256. There are 21 bits for the network address. Finally, Class D networks are used for multicasting purposes, when a general broadcast to more than one device is required. The lengths of each section of the IP address have been carefully chosen to provide maximum flexibility in assigning both network and local addresses.

IP addresses are four sets of 8 bits, for a total 32 bits. You often represent these bits as separated by a period for convenience, so the IP address format can be thought of as network.local.local.local for Class A or network.network.network.local for Class C. The IP addresses are usually written out in their decimal equivalents, instead of the long binary strings. This is the familiar host address number that network users are used to seeing, such as 147.10.13.28, which would indicate that the network address is 147.10 and the local or host address is 13.28. Of course, the actual address is a set of 1s and 0s. The decimal notation used for IP addresses is properly called *dotted quad notation*—a bit of trivia for your next dinner party.

The IP addresses can be translated to common names and letters. This can pose a problem, though, because there must be some method of unambiguously relating the physical address, the network address, and a language-based name (such a tpci_ws_4 or bobs_machine). "The Domain Name System" looks at this aspect of address naming. From the IP address, a network can determine if the data is to be sent out through a gateway. If the network address is the same as the current address (routing to a local network device, called a *direct host*), the gateway is avoided, but all other network addresses are routed to a gateway to leave the local network (*indirect host*). The gateway receiving data to be transmitted to another network must then determine the routing from the data's IP address and an internal table that provides routing information.

As mentioned, if an address is set to all 1s, the address applies to all addresses on the network. (See the previous section titled "Physical Addresses.") The same rule applies to IP addresses, so that an IP address of 32 1s is considered a broadcast message to all networks and all devices. It is possible to broadcast to all machines in a network by altering the local or host address to all 1s, so that the address 147.10.255.255 for a Class B network (identified as network 147.10) would be received by all devices on that network (255.255 being the local addresses composed of all 1s), but the data would not leave the network. There are two contradictory ways to indicate broadcasts. The later versions of TCP/IP use 1s, but earlier BSD systems use 0s. This causes a lot of confusion. All the devices on a network must know which broadcast convention is used; otherwise, datagrams can be stuck on the network forever!

A slight twist is coding the network address as all 0s, which means the originating network or the local address being set to 0s, which refers to the originating device only (usually used only when a device is trying to determine its IP address). The all-zero network address format is used when the network IP address is not known but other devices on the network can still interpret the local address. If this were transmitted to another network, it could cause confusion! By convention, no local device is given a physical address of 0.

It is possible for a device to have more than one IP address if it is connected to more than one network, as is the case with gateways. These devices are called *multihomed,* because they have a unique address for each network they are connected to. In practice, it is best to have a dedicate machine for a multihomed gateway; otherwise, the applications on that machine can get confused as to which address they should use

when building datagrams. Two networks can have the same network address if they are connected by a gateway. This can cause problems for addressing, because the gateway must be able to differentiate which network the physical address is on. This problem is looked at again in the next section, showing how it can be solved.

1.9 Address Resolution Protocol

Determining addresses can be difficult because every machine on the network might not have a list of all the addresses of the other machines or devices. Sending data from one machine to another if the recipient machine's physical address is not known can cause a problem if there is no resolution system for determining the addresses. Having to constantly update a table of addresses on each machine would be a network administration nightmare. The problem is not restricted to machine addresses within a small network, because if the remote destination network addresses are unknown, routing and delivery problems will also occur.

The Address Resolution Protocol (ARP) helps solve these problems. ARP's job is to convert IP addresses to physical addresses (network and local) and in doing so, eliminate the need for applications to know about the physical addresses. Essentially, ARP is a table with a list of the IP addresses and their corresponding physical addresses. The table is called an *ARP cache.* The layout of an ARP cache is shown in Figure 2.10. Each row corresponds to one device, with four pieces of information for each device:

	IF INDEX	PHYSICAL ADDRESS	IP ADDRESS	TYPE
Entry 1				
Entry 2				
Entry 3				
Entry n				

Figure 2.10. The ARP cache address translation table layout.

- IF Index: The physical port (interface)
- Physical Address: The physical address of the device
- IP Address: The IP address corresponding to the physical address
- Type: The type of entry in the ARP cache

1.9.1 Mapping Types

The mapping type is one of four possible values indicating the status of the entry in the ARP cache. A value of 2 means the entry is invalid; a value of 3 means the mapping is dynamic (the entry can change); a value of 4 means static (the entry doesn't change); and a value of 1 means none of the above. When the ARP receives a recipient device's IP address, it searches the ARP cache for a match. If it finds one, it returns the physical address. If the ARP cache doesn't find a match for an IP address, it sends a message out on the network. The message, called an *ARP request,* is a broadcast that is received by all devices on the local network. (You might remember that a broadcast has all 1s in the address.) The ARP request contains the IP address of the intended recipient device. If a device recognizes the IP address as belonging to it, the device sends a reply message containing its physical address back to the machine that generated the ARP request, which places the information into its ARP cache for future use. In this manner, the ARP cache can determine the physical address for any machine based on its IP address.

Whenever an ARP request is received by an ARP cache, it uses the information in the request to update its own table. Thus, the system can accommodate changing physical addresses and new additions to the network dynamically without having to generate an ARP request of its own. Without the use of an ARP cache, all the ARP requests and replies would generate a lot of network traffic, which can have a serious impact on network performance. Some simpler network schemes abandon the cache and simply use broadcast messages each time. This is feasible only when the number of devices is low enough to avoid network traffic problems.

The layout of the ARP request is shown in Figure 2.11. When an ARP request is sent, all fields in the layout are used except the Recipient Hardware Address (which the request is trying to identify). In an ARP reply, all the fields are used.

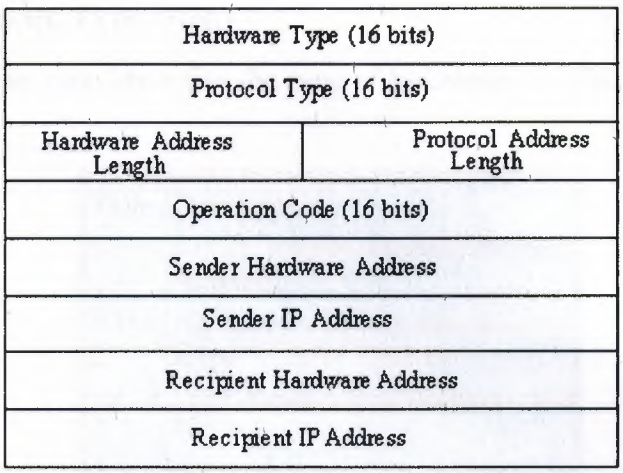

Figure 2.11. The ARP request and ARP reply layout.

This layout, which is combined with the network system's protocols into a protocol data unit (PDU), has several fields. The fields and their purposes are as follows:

- Hardware Type: The type of hardware interface
- Protocol Type: The type of protocol the sending device is using
- Hardware Address Length: The length of each hardware address in the datagram, given in bytes
- Protocol Address Length: The length of the protocol address in the datagram, given in bytes
- Operation Code (Opcode): The Opcode indicates whether the datagram is an ARP request or an ARP reply. If the datagram is a request, the value is set to 1. If it is a reply, the value is set to 2.
- Sender Hardware Address: The hardware address of the sending device
- Sender IP Address: The IP address of the sending device
- Recipient IP Address: The IP Address of the recipient
- Recipient Hardware Address: The hardware address of the recipient device

Some of these fields need a little more explanation to show their legal values and field usage. The following sections describe these fields in more detail.

1.9.2 The Hardware Type Field

The hardware type identifies the type of hardware interface. Legal values are as follows:

Type	*Description*
1	Ethernet
2	Experimental Ethernet
3	X.25
4	Proteon ProNET (Token Ring)
5	Chaos
6	IEEE 802.X
7	ARCnet

1.9.3 The Protocol Type Field

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

Decimal	*Description*
512	XEROX PUP
513	PUP Address Translation
1536	XEROX NS IDP
2048	Internet Protocol (IP)
2049	X.75
2050	NBS
2051	ECMA

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

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

1.10 ARP and IP Addresses

Two (or more) networks connected by a gateway can have the same network address. The gateway has to determine which network the physical address or IP address corresponds with. The gateway can do this with a modified ARP, called the Proxy ARP (sometimes called Promiscuous ARP). A proxy ARP creates an ARP cache consisting of entries from both networks, with the gateway able to transfer datagrams

from one network to the other. The gateway has to manage the ARP requests and replies that cross the two networks.

An obvious flaw with the ARP system is that if a device doesn't know its own IP address, there is no way to generate requests and replies. This can happen when a new device (typically a diskless workstation) is added to the network. The only address the device is aware of is the physical address set either by switches on the network interface or by software. A simple solution is the Reverse Address Resolution Protocol (RARP), which works the reverse of ARP, sending out the physical address and expecting back an IP address. The reply containing the IP address is sent by an RARP server, a machine that can supply the information. Although the originating device sends the message as a broadcast, RARP rules stipulate that only the RARP server can generate a reply. (Many networks assign more than one RARP server, both to spread the processing load and to act as a backup in case of problems.)

1.11 The Domain Name System

Instead of using the full 32-bit IP address, many systems adopt more meaningful names for their devices and networks. Network names usually reflect the organization's name (such as tpci.com and bobs_cement). Individual device names within a network can range from descriptive names on small networks (such as tims_machine and laser_1) to more complex naming conventions on larger networks (such as hpws_23 and tpci704). Translating between these names and the IP addresses would be practically impossible on an Internet-wide scale. To solve the problem of network names, the Network Information Center (NIC) maintains a list of network names and the corresponding network gateway addresses. This system grew from a simple flat-file list (which was searched for matches) to a more complicated system called the Domain Name System (DNS) when the networks became too numerous for the flat-file system to function efficiently.

DNS uses a hierarchical architecture, much like the UNIX filesystem. The first level of naming divides networks into the category of subnetworks, such as com for commercial, mil for military, edu for education, and so on. Below each of these is another division that identifies the individual subnetwork, usually one for each organization. This is called the *domain name* and is unique. The organization's system manager can further divide the company's subnetworks as desired, with each network called a *subdomain.* For example, the system merlin.abc_corp.com has the domain name

abc_corp.com, whereas the network merlin.abc_corp is a subdomain of merlin.abc_corp.com. A network can be identified with an *absolute name* (such as merlin.abc_corp.com) or a *relative name* (such as merlin) that uses part of the complete domain name.

Seven first-level domain names have been established by the NIC so far. These are as follows:

.arpa	An ARPANET-Internet identification
.com	Commercial company
.edu	Educational institution
.gov	Any governmental body
.mil	Military
.net	Networks used by Internet Service Providers
.org	Anything that doesn't fall into one of the other categories

The NIC also allows for a country designator to be appended. There are designators for all countries in the world, such as .ca for Canada and .uk for the United Kingdom. DNS uses two systems to establish and track domain names. A *name resolver* on each network examines information in a domain name. If it can't find the full IP address, it queries a *name server,* which has the full NIC information available. The name resolver tries to complete the addressing information using its own database, which it updates in much the same manner as the ARP system (discussed earlier) when it must query a name server. If a queried name server cannot resolve the address, it can query another name server, and so on, across the entire internetwork.

There is a considerable amount of information stored in the name resolver and name server, as well as a whole set of protocols for querying between the two. The details, luckily, are not important to an understanding of TCP/IP, although the overall concept of the address resolution is important when understanding how the Internet translates between domain names and IP addresses.

CHAPTER 2

THE INTERNET PROTOCOL (IP)

A good understanding of IP is necessary to continue on to TCP and UDP, because the IP is the component that handles the movement of datagrams across a network. Knowing how a datagram must be assembled and how it is moved through the networks helps you understand how the higher-level layers work with IP. For almost all protocols in the TCP/IP family, IP is the essential element that packages data and ensures that it is sent to its destination. This chapter contains, unfortunately, even more detail on headers, protocols, and messaging. This level of information is necessary in order for us to deal with understanding the applications and their interaction with IP, as well as troubleshooting the system. There is enough here that we can refer back to this chapter whenever needed.

2.1 Internet Protocol

The Internet Protocol (IP) is a primary protocol of the OSI model, as well as an integral part of TCP/IP (as the name suggests). Although the word "Internet" appears in the protocol's name, it is not restricted to use with the Internet. It is true that all machines on the Internet can use or understand IP, but IP can also be used on dedicated networks that have no relation to the Internet at all. IP defines a protocol, not a connection. Indeed, IP is a very good choice for any network that needs an efficient protocol for machine-to-machine communications, although it faces some competition from protocols like Novell NetWare's IPX on small to medium local area networks that use NetWare as a C server operating system.

What does IP do? Its main tasks are addressing of datagrams of information between computers and managing the fragmentation process of these datagrams. The protocol has a formal definition of the layout of a datagram of information and the formation of a header composed of information about the datagram. IP is responsible for the routing of a datagram, determining where it will be sent, and devising alternate routes in case of problems. Another important aspect of IP's purpose has to do with unreliable delivery of a datagram. Unreliable in the IP sense means that the delivery of the datagram is not guaranteed, because it can get delayed, misrouted, or mangled in the

breakdown and reassembly of message fragments. IP has nothing to do with flow control or reliability: there is no inherent capability to verify that a sent message is correctly received. IP does not have a checksum for the data contents of a datagram, only for the header information. The verification and flow control tasks are left to other components in the layer model. (For that matter, IP doesn't even properly handle the forwarding of datagrams. IP can make a guess as to the best routing to move a datagram to the next node along a path, but it does not inherently verify that the chosen path is the fastest or most efficient route.) Part of the IP system defines how gateways manage datagrams, how and when they should produce error messages, and how to recover from problems that might arise.

We saw how data can be broken into smaller sections for transmission and then reassembled at another location, a process called fragmentation and reassembly. IP provides for a maximum packet size of 65,535 bytes, which is much larger than most networks can handle, hence the need for fragmentation. IP has the capability to automatically divide a datagram of information into smaller datagrams if necessary, using the principles. When the first datagram of a larger message that has been divided into fragments arrives at the destination, a *reassembly timer* is started by the receiving machine's IP layer. If all the pieces of the entire datagram are not received when the timer reaches a predetermined value, all the datagrams that have been received are discarded. The receiving machine knows the order in which the pieces are to be reassembled because of a field in the IP header. One consequence of this process is that a fragmented message has a lower chance of arrival than an unfragmented message, which is why most applications try to avoid fragmentation whenever possible.

IP is connectionless, meaning that it doesn't worry about which nodes a datagram passes through along the path, or even at which machines the datagram starts and ends. This information is in the header, but the process of analyzing and passing on a datagram has nothing to do with IP analyzing the sending and receiving IP addresses. IP handles the addressing of a datagram with the full 32-bit Internet address, even though the transport protocol addresses use 8 bits. A new version of IP, called version 6 or IPng (IP Next Generation) can handle much larger headers, as you will see toward the end of today's material in the section titled "IPng: IP Version 6."

2.1.1 The Internet Protocol Datagram Header

It is tempting to compare IP to a hardware network such as Ethernet because of the basic similarities in packaging information. Yesterday you saw how Ethernet assembles a frame by combining the application data with a header block containing address information. IP does the same, except the contents of the header are specific to IP. When Ethernet receives an IP-assembled datagram (which includes the IP header), it adds its header to the front to create a frame—a process called *encapsulation.* One of the primary differences between the IP and Ethernet headers is that Ethernet's header contains the physical address of the destination machine, whereas the IP header contains the IP address. You might recall from yesterday's discussion that the translation between the two addresses is performed by the Address Resolution Protocol.

The datagram is the transfer unit used by IP, sometimes more specifically called an Internet datagram, or IP datagram. The specifications that define IP (as well as most of the other protocols and services in the TCP/IP family of protocols) define headers and tails in terms of words, where a word is 32 bits. Some operating systems use a different word length, although 32 bits per word is the more-often encountered value (some minicomputers and larger systems use 64 bits per word, for example). There are eight bits to a byte, so a 32-bit word is the same as four bytes on most systems.

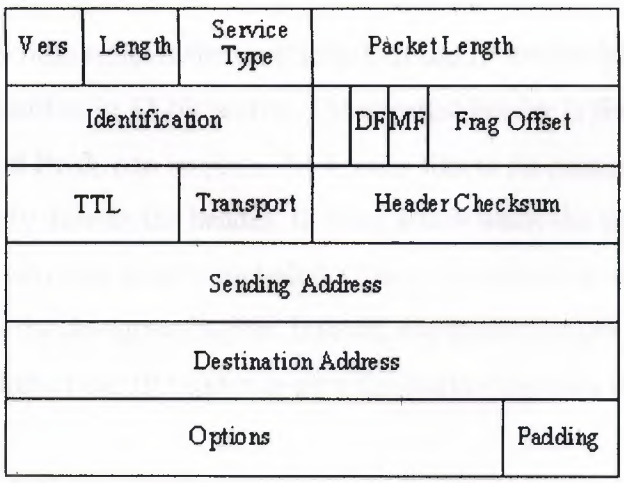

Figure 3.1. The IP header layout.

The IP header is six 32-bit words in length (24 bytes total) when all the optional fields are included in the header. The shortest header allowed by IP uses five words (20 bytes total). To understand all the fields in the header, it is useful to remember that IP has no hardware dependence but must account for all versions of IP software it can encounter (providing full backward-compatibility with previous versions of IP). The IP header layout is shown schematically in Figure 3.1. The different fields in the IP header are examined in more detail in the following subsections.

2.1.1.1 Version Number

This is a 4-bit field that contains the IP version number the protocol software is using. The version number is required so that receiving IP software knows how to decode the rest of the header, which changes with each new release of the IP standards. The most widely used version is 4, although several systems are now testing version 6 (called IPng). The Internet and most LANs do not support IP version 6 at present. Part of the protocol definition stipulates that the receiving software must first check the version number of incoming datagrams before proceeding to analyze the rest of the header and encapsulated data. If the software cannot handle the version used to build the datagram, the receiving machine's IP layer rejects the datagram and ignores the contents completely.

2.1.1.2 Header Length

This 4-bit field reflects the total length of the IP header built by the sending machine; it is specified in 32-bit words. The shortest header is five words (20 bytes), but the use of optional fields can increase the header size to its maximum of six words (24 bytes). To properly decode the header, IP must know when the header ends and the data begins, which is why this field is included. (There is no start-of-data marker to show where the data in the datagram begins. Instead, the header length is used to compute an offset from the start of the IP header to give the start of the data block.)

2.1.1.3 Type of Service

The 8-bit (1 byte) Service Type field instructs IP how to process the datagram properly. The field's 8 bits are read and assigned as shown in Figure 3.2, which shows the layout of the Service Type field inside the larger IP header shown in Figure 3.1. The

first 3 bits indicate the datagram's precedence, with a value from 0 (normal) through 7 (network control). The higher the number, the more important the datagram and, in theory at least, the faster the datagram should be routed to its destination. In practice, though, most implementations of TCP/IP and practically all hardware that uses TCP/IP ignores this field, treating all datagrams with the same priority.

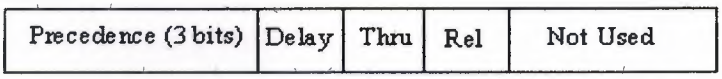

Figure 3.2. The 8-bit Service Type field layout.

The next three bits are 1-bit flags that control the delay, throughput, and reliability of the datagram. If the bit is set to 0, the setting is normal. A bit set to 1 implies low delay, high throughput, and high reliability for the respective flags. The last two bits of the field are not used. Most of these bits are ignored by current IP implementations, and all datagrams are treated with the same delay, throughput, and reliability settings.

For most purposes, the values of all the bits in the Service Type field are set to 0 because differences in precedence, delay, throughput, and reliability between machines are virtually nonexistent unless a special network has been established. Although these flags would be useful in establishing the best routing method for a datagram, no currently available UNIX-based IP system bothers to evaluate the bits in these fields. (Although it is conceivable that the code could be modified for high security or high reliability networks.)

2.1.1.4 Datagram Length (or Packet Length)

This field gives the total length of the datagram, including the header, in bytes. The length of the data area itself can be computed by subtracting the header length from this value. The size of the total datagram length field is 16 bits, hence the 65,535 bytes maximum length of a datagram (including the header). This field is used to determine the length value to be passed to the transport protocol to set the total frame length.

2.1.1.5 Identification

This field holds a number that is a unique identifier created by the sending node. This number is required when reassembling fragmented messages, ensuring that the fragments of one message are not intermixed with others. Each chunk of data received by the IP layer from a higher protocol layer is assigned one of these identification numbers when the data arrives. If a datagram is fragmented, each fragment has the same identification number.

2.1.1.6 Flags

The Flags field is a 3-bit field, the first bit of which is left unused (it is ignored by the protocol and usually has no value written to it). The remaining two bits are dedicated to flags called DF (Don't Fragment) and MF (More Fragments), which control the handling of the datagrams when fragmentation is desirable. If the DF flag is set to 1, the datagram cannot be fragmented under any circumstances. If the current IP layer software cannot send the datagram on to another machine without fragmenting it, and this bit is set to 1, the datagram is discarded and an error message is sent back to the sending device.

If the MF flag is set to 1, the current datagram is followed by more packets (sometimes called *subpackets*), which must be reassembled to re-create the full message. The last fragment that is sent as part of a larger message has its MF flag set to 0 (off) so that the receiving device knows when to stop waiting for datagrams. Because the order of the fragments' arrival might not correspond to the order in which they were sent, the MF flag is used in conjunction with the Fragment Offset field (the next field in the IP header) to indicate to the receiving machine the full extent of the message.

2.1.1.7 Fragment Offset

If the MF (More Fragments) flag bit is set to 1 (indicating fragmentation of a larger datagram), the fragment offset contains the position in the complete message of the submessage contained within the current datagram. This enables IP to reassemble fragmented packets in the proper order. Offsets are always given relative to the beginning of the message. This is a 13-bit field, so offsets are calculated in units of 8 bytes, corresponding to the maximum packet length of 65,535 bytes. Using the identification number to indicate which message a receiving datagram belongs to, the IP

layer on a receiving machine can then use the fragment offset to reassemble the entire message.

2.1.1.8 Time to Live (TTL)

This field gives the amount of time in seconds that a datagram can remain on the network before it is discarded. This is set by the sending node when the datagram is assembled. Usually the TTL field is set to 15 or 30 seconds. The TCP/IP standards stipulate that the TTL field must be decreased by at least one second for each node that processes the packet, even if the processing time is less than one second. Also, when a datagram is received by a gateway, the arrival time is tagged so that if the datagram must wait to be processed, that time counts against its TTL. Hence, if a gateway is particularly overloaded and can't get to the datagram in short order, the TTL timer can expire while awaiting processing, and the datagram is abandoned.

If the TTL field reaches 0, the datagram must be discarded by the current node, but a message is sent back to the sending machine when the packet is dropped. The sending machine can then resend the datagram. The rules governing the TTL field are designed to prevent IP packets from endlessly circulating through networks.

2.1.1.9 Transport Protocol

This field holds the identification number of the transport protocol to which the packet has been handed. The numbers are defined by the Network Information Center (NIC), which governs the Internet. There are currently about 50 protocols defined and assigned a transport protocol number. The two most important protocols are ICMP (detailed in the section titled "Internet Control Message Protocol (ICMP)" later today), which is number 1, and TCP, which is number 6. The full list of numbers is not necessary here because most of the protocols are never encountered by users. (If you really want this information, it's in several RFCs mentioned in the apendixes.)

2.1.1.10 Header Checksum

The number in this field of the IP header is a checksum for the protocol header field (but not the data fields) to enable faster processing. Because the Time to Live (TTL) field is decremented at each node, the checksum also changes with every

machine the datagram passes through. The checksum algorithm takes the ones-complement of the 16-bit sum of all 16-bit words.

This is a fast, efficient algorithm, but it misses some unusual corruption circumstances such as the loss of an entire 16-bit word that contains only 0s. However, because the data checksums used by both TCP and UDP cover the entire packet, these types of errors usually can be caught as the frame is assembled for the network transport.

2.1.1.11 Sending Address and Destination Address

These fields contain the 32-bit IP addresses of the sending and destination devices. These fields are established when the datagram is created and are not altered during the routing.

2.1.1.12 Options

The Options field is optional, composed of several codes of variable length. If more than one option is used in the datagram, the options appear consecutively in the IP header. All the options are controlled by a byte that is usually divided into three fields: a 1-bit copy flag, a 2-bit option class, and a 5-bit option number. The copy flag is used to stipulate how the option is handled when fragmentation is necessary in a gateway. When the bit is set to 0, the option should be copied to the first datagram but not subsequent ones. If the bit is set to 1, the option is copied to all the datagrams.

The option class and option number indicate the type of option and its particular value. At present, there are only two option classes set. (With only 2 bits to work with in the field, a maximum of four options could be set.) When the value is 0, the option applies to datagram or network control. A value of 2 means the option is for debugging or administration purposes. Values of 1 and 3 are unused. Currently supported values for the option class and number are given in Table 3.1.

Table 3.1. Valid option class and numbers for IP headers.

Option Class	*Option Number*	*Description*
0	0	Marks the end of the options list
0	1	No option (used for padding)
0	2	Security options (military purposes only)
0	3	Loose source routing
0	7	Activates routing record (adds fields)
0	9	Strict source routing
2	4	Timestamping active (adds fields)

Of most interest to us are options that enable the routing and timestamps to be recorded. These are used to provide a record of a datagram's passage across the internetwork, which can be useful for diagnostic purposes. Both these options add information to a list contained within the datagram. (The timestamp has an interesting format: it is expressed in milliseconds since midnight, Universal Time. Unfortunately, because most systems have widely differing time settings—even when corrected to Universal Time—the timestamps should be treated with more than a little suspicion.)

There are two kinds of routing indicated within the Options field: loose and strict. *Loose routing* provides a series of IP addresses that the machine must pass through, but it enables any route to be used to get to each of these addresses (usually gateways). *Strict routing* enables no deviations from the specified route. If the route can't be followed, the datagram is abandoned. Strict routing is frequently used for testing routes but rarely for transmission of user datagrams because of the higher chances of the datagram being lost or abandoned.

2.1.1.13 Padding

The content of the padding area depends on the options selected. The padding is usually used to ensure that the datagram header is a round number of bytes.

2.1.2 A Datagram's Life

To understand how IP and other TCP/IP layers work to package and send a datagram from one machine to another, I take a simplified look at a typical datagram's passage. When an application must send a datagram out on the network, it performs a few simple steps. First, it constructs the IP datagram within the legal lengths stipulated by the local IP implementation. The checksum is calculated for the data, and then the IP header is constructed. Next, the first hop (machine) of the route to the destination must be determined to route the datagram to the destination machine directly over the local network, or to a gateway if the internetwork is used. If routing is important, this information is added to the header using an option. Finally, the datagram is passed to the network for its manipulation of the datagram.

As a datagram passes along the internetwork, each gateway performs a series of tests. After the network layer has stripped off its own header, the gateway IP layer calculates the checksum and verifies the integrity of the datagram. If the checksums don't match, the datagram is discarded and an error message is returned to the sending device. Next, the TTL field is decremented and checked. If the datagram has expired, it is discarded and an error message is sent back to the sending machine. After determining the next hop of the route, either by analysis of the target address or from a specified routing instruction within the Options field of the IP header, the datagram is rebuilt with the new TTL value and new checksum.

If fragmentation is necessary because of an increase in the datagram's length or a limitation in the software, the datagram is divided, and new datagrams with the correct header information are assembled. If a routing or timestamp is required, it is added as well. Finally, the datagram is passed back to the network layer. When the datagram is finally received at the destination device, the system performs a checksum calculation and—assuming the two sums match—checks to see if there are other fragments. If more datagrams are required to reassemble the entire message, the system waits, meanwhile running a timer to ensure that the datagrams arrive within a reasonable time. If all the parts of the larger message have arrived but the device can't reassemble them before the timer reaches 0, the datagram is discarded and an error message is returned to the sender. Finally, the IP header is stripped off, the original message is reconstructed if it was fragmented, and the message is passed up the layers to the upper layer application. If a reply was required, it is then generated and sent back to the sending device.

When extra information is added to the datagram for routing or timestamp recording, the length of the datagram can increase. Handling all these conditions is part of IP's forte, for which practically every problem has a resolution system.

2.2 Internet Control Message Protocol (ICMP)

Many problems can occur in routing a message from sender to receiver. The TTL timer might expire; fragmented datagrams might not arrive with all segments intact; a gateway might misroute a datagram, and so on. Letting the sending device know of a problem with a datagram is important, as is correctly handling error conditions within the network routing itself. The Internet Control Message Protocol (ICMP) was developed for this task.

ICMP is an error-reporting system. It is an integral part of IP and must be included in every IP implementation. This provides for consistent, understandable error messages and signals across the different versions of IP and different operating systems. It is useful to think of ICMP as one IP package designed specifically to talk to another IP package across the network: in other words, ICMP is the IP layer's communications system. Messages generated by ICMP are treated by the rest of the network as any other datagram, but they are interpreted differently by the IP layer software. ICMP messages have a header built in the same manner as any IP datagram, and ICMP datagrams are not differentiated at any point from normal data-carrying datagrams until a receiving machine's IP layer processes the datagram properly.

In almost all cases, error messages sent by ICMP are routed back to the original datagram's sending machine. This is because only the sender's and destination device's IP addresses are included in the header. Because the error doesn't mean anything to the destination device, the sender is the logical recipient of the error message. The sender can then determine from the ICMP message the type of error that occurred and establish how to best resend the failed datagram.

ICMP messages go through two encapsulations, as do all IP messages: incorporation into a regular IP datagram and then into the network frame. This is shown in Figure 3.3. ICMP headers have a different format than IP headers, though, and the format differs slightly depending on the type of message. However, all ICMP headers start with the same three fields: a message type, a code field, and a checksum for the ICMP message. Figure 3.4 shows the layout of the ICMP message.

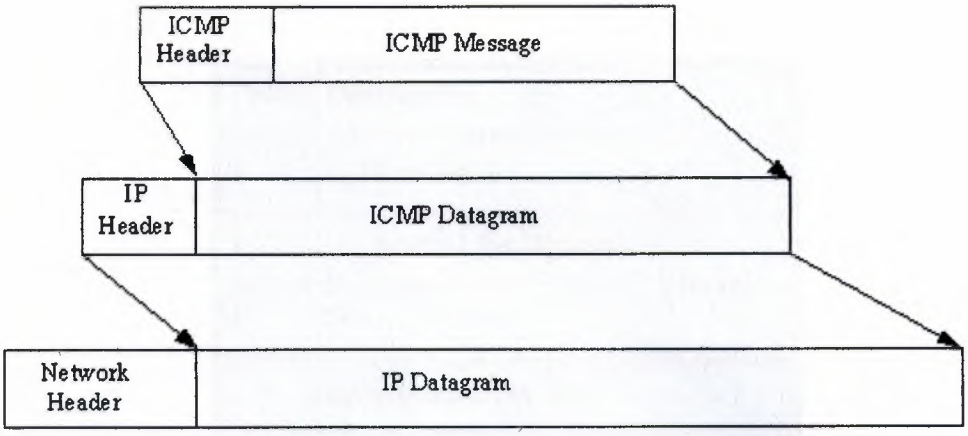

Figure 3.3. Two-step encapsulation of an ICMP message.

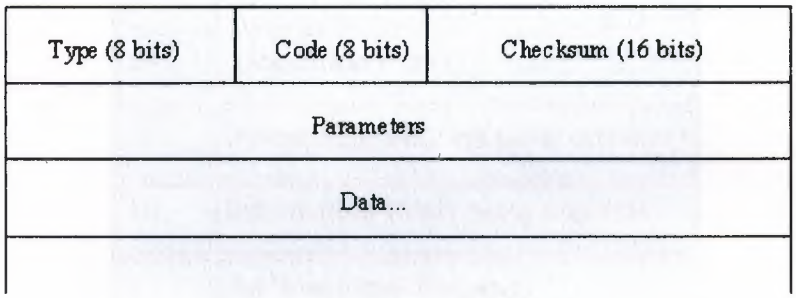

Figure 3.4. The layout of an ICMP message.

Usually, any ICMP message that is reporting a problem with delivery also includes the header and first 64 bits of the data field from the datagram for which the problem occurred. Including the 64 bits of the original datagram accomplishes two things. First, it enables the sending device to match the datagram fragment to the original datagram by comparison. Also, because most of the protocols involved are defined at the start of the datagram, the inclusion of the original datagram fragment allows for some diagnostics to be performed by the machine receiving the ICMP message.

The 8-bit Message Type field in the ICMP header (shown in Figure 3.4) can have one of the values shown in Table 3.2.

Table 3.2. Valid values for the ICMP Message Type field.

Value	*Description*
0	Echo Reply
3	Destination Not Reachable
4	Source Quench
5	Redirection Required
8	Echo Request
11	Time to Live Exceeded
12	Parameter Problem
13	Timestamp Request
14	Timestamp Reply
15	Information Request (now obsolete)
16	Information Reply (now obsolete)
17	Address Mask Request
18	Address Mask Reply

The Code field expands on the message type, providing a little more information for the receiving machine. The checksum in the ICMP header is calculated in the same manner as the normal IP header checksum. The layout of the ICMP message is slightly different for each type of message. Figure 3.5 shows the layouts of each type of ICMP message header. The Destination Unreachable and Time Exceeded messages are self-explanatory, although they are used in other circumstances, too, such as when a datagram must be fragmented but the Don't Fragment flag is set. This results in a Destination Unreachable message being returned to the sending machine.

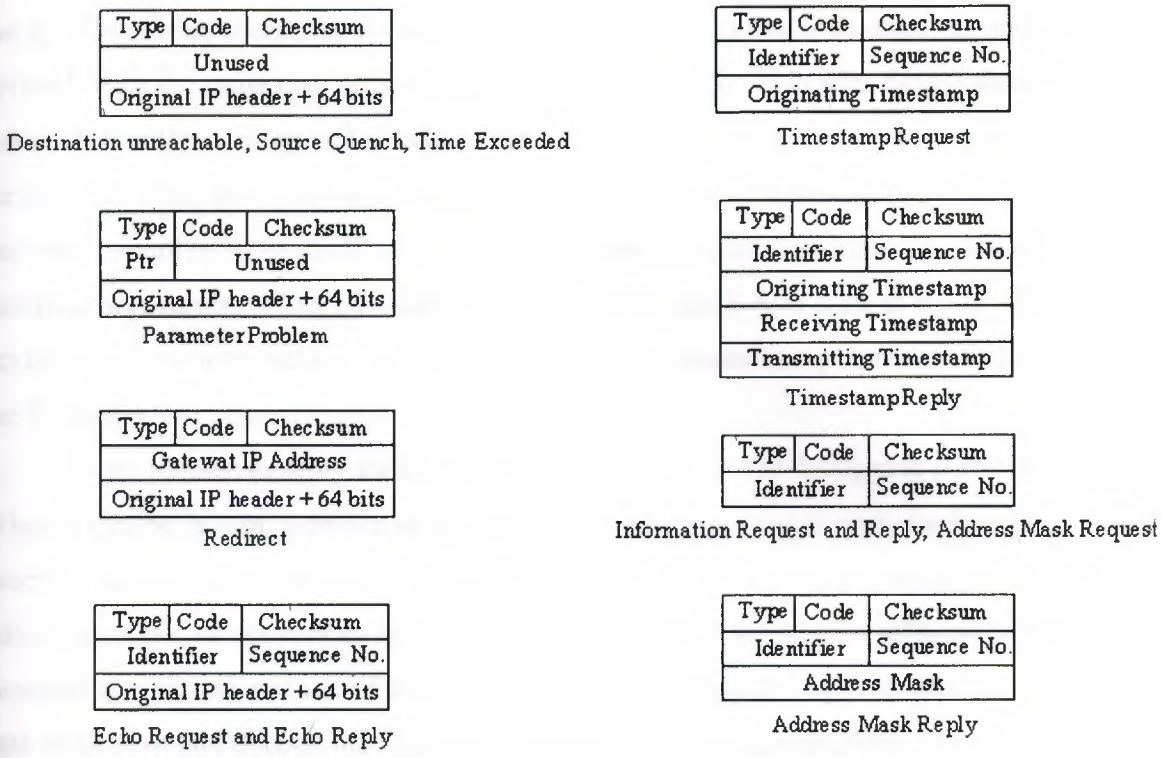

Figure 3.5. ICMP message header layouts.

The Source Quench ICMP message is used to control the rate at which datagrams are transmitted, although this is a very rudimentary form of flow control. When a device receives a Source Quench message, it should reduce the transmittal rate over the network until the Source Quench messages cease. The messages are typically generated by a gateway or host that either has a full receiving buffer or has slowed processing of incoming datagrams because of other factors. If the buffer is full, the device is supposed to issue a Source Quench message for each datagram that is discarded. Some implementations issue Source Quench messages when the buffer exceeds a certain percentage to slow down reception of new datagrams and enable the device to clear the buffer.

Redirection messages are sent to a gateway in the path when a better route is available. For example, if a gateway has just received a datagram from another gateway but on checking its datafiles finds a better route, it sends the Redirection message back to that gateway with the IP address of the better route. When a Redirection message is sent, an integer is placed in the code field of the header to indicate the conditions for which the rerouting applies. A value of 0 means that datagrams for any device on the

destination network should be redirected. A value of 1 indicates that only datagrams for the specific device should be rerouted. A value of 2 implies that only datagrams for the network with the same type of service (read from one of the IP header fields) should be rerouted. Finally, a value of 3 reroutes only for the same host with the same type of service. The Parameter Problem message is used whenever a semantic or syntactic error has been encountered in the IP header. This can happen when options are used with incorrect arguments. When a Parameter Problem message is sent back to the sending device, the Parameter field in the ICMP error message contains a pointer to the byte in the IP header that caused the problem. (See Figure 3.5.)

Echo request or reply messages are commonly used for debugging purposes. When a request is sent, a device or gateway down the path sends a reply back to the specified device. These request/reply pairs are useful for identifying routing problems, failed gateways, or network cabling problems. The simple act of processing an ICMP message also acts as a check of the network, because each gateway or device along the path must correctly decode the headers and then pass the datagram along. Any failure along the way could be with the implementation of the IP software. A commonly used request/reply system is the ping command. The ping command sends a series of requests and waits for replies. Timestamp requests and replies enable the timing of message passing along the network to be monitored. When combined with strict routing, this can be useful in identifying bottlenecks. Address mask requests and replies are used for testing within a specific network or subnetwork.

2.3 IPng: IP Version 6

When IP version 4 (the current release) was developed, the use of a 32-bit IP address seemed more than enough to handle the projected use of the Internet. With the incredible growth rate of the Internet over the last few years, however, the 32-bit IP address might become a problem. To counter this limit, IP Next Generation, usually called IP version 6 (IPv6), is under development. Several proposals for IPng implementation are currently being studied, the most popular of which are TUBA (TCP and UDP with Bigger Addresses), CATNIP (Common Architecture for the Internet), and SIPP (Simple Internet Protocol Plus). None of the three meet all the proposed changes for version 6, but a compromise or modification based on one of these proposals is likely. What does IPng have to offer? The list of changes tells you the main features of IPng in a nutshell:

- 128-bit network address instead of 32-bit
- More efficient IP header with extensions for applications and options
- No header checksum
- A flow label for quality-of-service requirements
- Prevention of intermediate fragmentation of datagrams
- Built-in security for authentication and encryption

Next we look at IPng in a little more detail to show the changes that affect most users, as well as network programmers and network administrators. I start with a look at the IPng header. Remember that at present IPng is still under development and is not widely deployed except on test networks.

2.3.1 IPng Datagram

As mentioned earlier, the header for IPng datagrams has been modified over the earlier version 4 header. The changes are mostly to provide support for the new, longer 128-bit IP addresses and to remove obsolete and unneeded fields. The basic layout of the IPng header is shown in Figure 3.6. As you can see, there are quite a few changes from the IP header used in IP version 4 (see Figure 3.1).

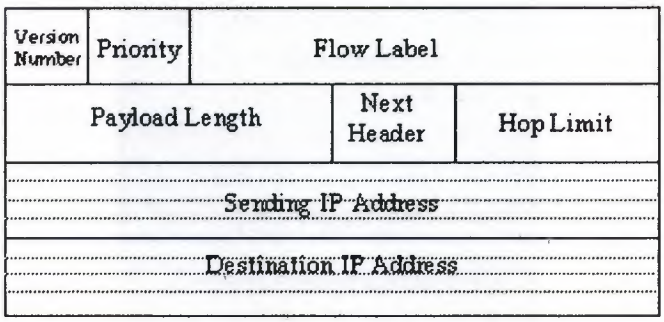

Figure 3.6. The IPng header layout.

The version number in the IP datagram header is four bits long and holds the release number (which is 6 with IPng). The Priority field is four bits long and holds a value indicating the datagram's priority. The priority is used to define the transmission

order. The priority is set first with a broad classification, then a narrower identifier within each class. I look at the priority classification in a little more detail in a moment. The Flow Label field is 24 bits long and is still in the development stage. It is likely to be used in combination with the source machine IP address to provide flow identification for the network. For example, if you are using a UNIX workstation on the network, the flow is different from another machine such as a Windows 95 PC. This field can be used to identify flow characteristics and provide some adjustment capabilities. The field can also be used to help identify target machines for large transfers, in which case a cache system becomes more efficient at routing between source and destination. Flow labels are discussed in more detail in the section titled "Flow Labels".

The Payload Length field is a 16-bit field used to specify the total length of the IP datagram, given in bytes. The total length is exclusive of the IP header itself. The use of a 16-bit field limits the maximum value in this field to 65,535, but there is a provision to send large datagrams using an extension header (see the section titled "IP Extension Headers" later today). The Next Header field is used to indicate which header follows the IP header when other applications want to piggy-back on the IP header. Several values have been defined for the Next Header field, as shown in Table 3.3.

Table 3.3. IP Next Header field values.

Value	*Description*
0	Hop-by-hop options
4	IP
6	TCP
17	UDP
43	Routing
44	Fragment
45	Interdomain Routine
46	Resource Reservation

50	Encapsulating Security
51	Authentication
58	ICMP
59	No Next Header
60	Destination Options

The Hop Limit field determines the number of hops the datagram can travel. With each forwarding, the number is decremented by 1. When the Hop Limit field reaches 0, the datagram is discarded, just as with IP version 4.

Finally, the Sending and Destination IP Addresses in 128-bit format are placed in the header. I look at the new IP address format in more detail in the section titled "128-Bit IP Addresses" later in this chapter.

2.3.1.1 Priority Classification

The Priority Classification field in the IPng header first divides the datagram into one of two categories: congestion controlled or noncongestion controlled. Noncongestion controlled datagrams are always routed as a priority over congestion controlled datagrams. There are subclassifications of noncongestion controlled datagram priorities available for use, but none of the categories have been accepted as standard yet. If the datagram is congestion controlled, it is sensitive to congestion problems on the network. If congestion occurs, the datagram can be slowed down and held temporarily in caches until the problem is alleviated. Beneath the broad congestion controlled category are several subclasses that further refine the priority of the datagram. The subcategories of congestion controlled priorities are given in Table 3.4. Noncongestion controlled traffic has priorities 8 through 15 available, but as mentioned earlier, they are not defined

Table 3.4. Priorities for congestion controlled datagrams.

Value	*Meaning*
0	No priority specified
1	Background traffic
2	Unattended data transfer
3	Unassigned
4	Attended bulk transfer
5	Unassigned
6	Interactive traffic
7	Control traffic

Examples of each of the primary subcategories might help us see how the datagrams are prioritized. Routing and network management traffic that is considered highest priority is assigned category 7. Interactive applications such as Telnet and remote X sessions are assigned as interactive traffic (category 6). Transfers that are not time-critical (such as Telnet sessions) but are still controlled by an interactive application such as FTP are assigned as category 4. E-mail is usually assigned as category 2, whereas low-priority material such as news is set to category 1.

2.3.1.2 Flow Labels

As mentioned earlier, the Flow Label field new to the IPng header can be used to help identify the sender and destination of many IP datagrams. By employing caches to handle flows, the datagrams can be routed more efficiently. Not all applications can handle flow labels, in which case the field is set to a value of 0.

A simple example might help show the usefulness of the flow label field. Suppose a PC running Windows 95 is connected to a UNIX server on another network and is sending a large number of datagrams. By setting a specific value of the flow label for all the datagrams in the transmission, the routers along the way to the server can

assemble entries in their routing caches that indicate which way to route each datagram with the same flow label. When subsequent datagrams with the same flow label arrive, the router doesn't have to recalculate the route; it can simply check the cache and extract the saved information from that. This speeds up the passage of the datagrams through each router. To prevent caches from growing too large or holding stale information, IPng stipulates that the cache maintained in a routing device cannot be kept for more than six seconds. If a new datagram with the same flow label is not received within six seconds, the cache entry is removed. To prevent repeated values from the sending machine, the sender must wait six seconds before using the same flow label value for another destination.

IPng allows flow labels to be used to reserve a route for time-critical applications. For example, a real-time application that has to send several datagrams along the same route and needs as rapid a transmission as possible (such as is needed for video or audio, for example) can establish the route by sending datagrams ahead of time, being careful not to exceed the six second time-out on the interim routers.

2.3.2 128-Bit IP Addresses

Probably the most important aspect of IPng is its capability to provide for longer IP addresses. IPng increases the IP address from 32 bits to 128 bits. This enables an incredible number of addresses to be assembled, probably more than can ever be used. The new IP addresses support three kinds of addresses: unicast, multicast, and anycast.

- Unicast addresses are meant to identify a particular machine's interface. This lets a PC, for example, have several different protocols in use, each with its own address. Thus, you could send messages specifically to a machine's IP interface address and not the NetBEUI interface address.
- A multicast address identifies a group of interfaces, enabling all machines in a group to receive the same packet. This is much like broadcasts in IP version 4, although with more flexibility for defining groups. Your machine's interfaces could belong to several multicast groups.
- An anycast address identifies a group of interfaces on a single multicast address. In other words, more than one interface can receive the datagram on the same machine.

The handling of fragmentation and reassembly is also changed with IPng to rovide more capabilities for IP. Also proposed for IPng is an authentication scheme nat can ensure that the data has not been corrupted between sender and receiver, as well s ensuring that the sending and receiving machines are who they claim they are.

.3.3 IP Extension Headers

IPng has the provision to enable additional headers to be tacked onto the IP eader. This might be necessary when a simple routing to the destination is not possible, r when special services such as authentication are required for the datagram. The dditional information required is packaged into an extension header and appended to he IP header. IPng defines several types of extension headers identified by a number laced in the Next Header field of the IP header. The currently accepted values and their neanings were shown in Table 3.3. Several extensions can be appended onto one IP eader, with each extension's Next Header field indicating the next extension. Normally, he extension headers are appended in ascending numerical order. This makes it easier or routers to analyze the extensions, stopping the examination when it gets past router-pecific extensions.

2.3.3.1 Hop-by-Hop Headers

Extension type 0 is hop-by-hop, which is used to provide IP options to every machine the datagram passes through. The options included in the hop-by-hop extension have a standard format of a Type value, a Length, and a Value (except for the Pad1 option, which has a single value set to 0 and no length or value field). Both the Type and Length fields are a single byte in length, whereas the Value field's length is variable and indicated by the length byte. There are three types of hop-by-hop extensions defined so far, called Pad1, PadN, and Jumbo Payload. The Pad1 option is a single byte with a value of 0, no length, and no value. It is used to alter the order and position of other options in the header when necessary, dictated usually by an application. The PadN option is similar except it has N zeros placed in the Value field and a calculated value for the length.

The Jumbo Payload extension option is used to handle datagram sizes in excess of 65,535 bytes. The Length field in the IP header is limited to 16 bits, hence the limit of 65,535 for the datagram size. To handle larger datagram lengths, the IP header's

Length field is set to 0, which redirects the routers to the extension to pick up a correct length value. The Length field can be defined in the extension header using 32 bits, which is in excess of 4 terabytes.

2.3.3.2 Routing Headers

A routing extension can be tacked onto the IP header when the sending machine wants to control the routing of the datagram instead of leaving it to the routers along the path. The routing extension can be used to give routes to the destination. The routing extension includes fields for each IP address along the desired route.

2.3.3.3 Fragment Headers

The fragment header can be appended to an IP datagram to enable a machine to fragment a large datagram into smaller parts. Part of the design of IPng was to prevent subsequent fragmentation, but in some cases fragmentation must be enabled in order to pass the datagram along the network.

2.3.3.4 Authentication Headers

The authentication header is used to ensure that no alteration was made to the contents of the datagram and that the datagram originated at the machine shown in the IP header. By default, IPng uses an authentication scheme called Message Digest 5 (MD5). Other authentication schemes can be used as long as both ends of the connection agree on the same scheme.

The authentication header consists of a security parameters index (SPI) that, when combined with the destination IP address, defines the authentication scheme. The SPI is followed by authentication data, which with MD5 is 16 bytes long. MD5 starts with a key (padded to 128 bits if it is shorter), then appends the entire datagram. The key is then tagged at the end, and the MD5 algorithm is run on the whole. To prevent problems with hop counters and the authentication header itself altering the values, they are zeroed for the purposes of calculating the authentication value. The MD5 algorithm generates a 128-bit value that is placed in the authentication header. The steps are repeated in reverse at the receiving end. Both ends must have the same key value, of course, for the scheme to work.

The datagram contents can be encrypted prior to generating authentication values using the default IPng encryption scheme, called Cipher Block Chaining (CBC), part of the Data Encryption Standard (DES).

2.4 Internet Protocol Support in Different Environments

The University of California at Berkeley was given a grant in the early 1980s to modify their UNIX operating system to included support for IP. The BSD4.2 UNIX release already offered support for TCP and IP, as well as the Simple Mail Transfer Protocol (SMTP) and Address Resolution Protocol (ARP), but with DARPA's funds, BSD4.3 was developed to provide more complete support. The BSD4.2 support for IP was quite good prior to this grant, but it was limited to use in small local area networks only. To increase the capabilities of BSD UNIX's IP support, BSD added retransmission capabilities, Time to Live information, and redirection messages. Other features were added, too, enabling BSD4.3 to work with larger networks, internetworks (connections between different networks), and wide area networks connected by leased lines. This process brought the BSD UNIX system (and its licensees, such as Sun's SunOS) in line with the IP standards used on AT&T UNIX and other UNIX-based platforms.

With the strong support for IP among the UNIX community, it was inevitable that manufacturers of other software operating systems would start to produce software that allowed their machines to interconnect to the UNIX IP system. Most of the drive to produce IP versions for non-UNIX operating systems was not because of the Internet (which hadn't started its phenomenal growth at the time) but the desire to integrate the other operating systems into local area networks that used UNIX servers. This section of today's material examines several hardware and software systems, focusing on the most widely used platforms, and shows the availability of IP (and entire TCP/IP suites) for those machines. Much of this is of interest only if we have the particular platform discussed (DEC VAX users tend not to care about interconnectivity to IBM SNA platforms, for example), so we can be selective about the sections we read.

2.4.1 MS-DOS

PCs came onto the scene when TCP/IP was already in common use, so it was not surprising to find interconnection software rapidly introduced. In many ways, the PC was a perfect platform as a stand-alone machine with access through a

communications package to other larger systems. The PC was perfect for a client/server environment. There are many PC-based versions of TCP/IP. The most widely used packages come from FTP Software, The Wollongong Group, and Beame and Whiteside Software Inc. All the packages feature interconnection capabilities to other machines using TCP/IP, and most add other useful features such as FTP and mail routing.

FTP Software's PC/TCP is one of the most widely used. PC/TCP supports the major network interfaces: Packet Driver, IBM's Adapter Support Interface (ASI), Novell's Open Data Link Interface (ODI), and Microsoft/3Com's Network Driver Interface Specification (NDIS). All four LAN interfaces are discussed in more detail in the section titled "Local Area Networks" later today. The design of PC/TCP covers all seven layers of the OSI model, developed in such a manner that components can be configured as required to support different transport mechanisms and applications. Typically, the Packet Driver, ASI, ODI, or NDIS module has a generic PC/TCP kernel on top of it, with the PC/TCP application on top of that. PC/TCP enables the software to run both TCP/IP and another protocol, such as DECnet, Novell NetWare, or LAN Manager, simultaneously. This can be useful for enabling a PC to work within a small LAN workgroup, as well as within a larger network, without switching software.

2.4.2 Microsoft Windows

There are several TCP/IP products appearing for Microsoft's Windows 3.*x*, Windows for Workgroups, and Windows 95. Most of the early packages for Windows 3.*x* were ports of DOS products. Although these tend to work well, a totally Windows-designed product tends to have a slight edge in terms of integration with the Windows environment. Windows for Workgroups 3.11 has no inherent TCP/IP drives, but several products are available to add TCP/IP suites for this GUI, as well as Windows 3.1 and Windows 3.11. One Windows 3.*x*-designed product is NetManage's Chameleon TCP/IP for Windows. Chameleon offers a complete port of TCP/IP and additional software utilities to enable a PC running Windows 3.*x* to integrate into a TCP/IP network. Chameleon offers terminal emulation, Telnet, FTP, electronic mail, DNS directory services, and NFS capabilities. There are several versions of Chameleon, depending on whether NFS is required.

Windows 95 has TCP/IP drivers included with the distribution software, but they are not loaded by default (NetWare's IPX/SPX is the default protocol for Windows 95). You must install and configure the TCP/IP product as a separate step after installing

Windows 95 if you want to use IP on your network. You can see how this is done on Day 10, "Setting Up a Sample TCP/IP Network: DOS and Windows Clients."

2.4.3 Windows NT

Windows NT is ideally suited for TCP/IP because it is designed to act as a server and gateway. Although Windows NT is not inherently multiuser, it does work well as a TCP/IP access device. Windows NT includes support for the TCP/IP protocols as a network transport, although the implementation does not include all the utilities usually associated with TCP/IP. TCP/IP can be chosen as the default protocol on a Windows NT machine when the operating system is installed. Among the add-on products available for Windows NT, NetManage's Chameleon32 is a popular package. Similar to the Microsoft Windows version, Chameleon32 offers versions for NFS.

2.4.4 OS/2

IBM's OS/2 platform has a strong presence in corporations because of the IBM reputation and OS/2's solid performance. Not surprisingly, TCP/IP products are popular in these installations, as well. Although OS/2 differs from DOS in many ways, it is possible to run DOS-based ports of TCP/IP software under OS/2. A better solution is to run a native OS/2 application. Several TCP/IP OS/2-native implementations are available, including a TCP/IP product from IBM itself.

2.4.5 Macintosh

Except for versions of UNIX that run on the Macintosh, the Macintosh and UNIX worlds have depended on several different versions of TCP/IP to keep them connected. With many corporations now wanting their investment in Macintosh computers to serve double duty as X terminals onto UNIX workstation, TCP/IP for the Mac has become even more important. Macintosh TCP products are available in several forms, usually as an add-on application or device driver for the Macintosh operating system. An alternative is Tenon Intersystems' MachTen product line, which enables a UNIX kernel and the Macintosh operating system to coexist on the same machine, providing compatibility between UNIX and the Macintosh file system and Apple events. TCP/IP is part of the MachTen product.

The AppleTalk networking system enables Macs and UNIX machines to interconnect to a limited extent, although this requires installation of AppleTalk software on the UNIX host—something many system administrators are reluctant to do. Also, because AppleTalk is not as fast and versatile as Ethernet and other network transports, this solution is seldom favored. A better solution is simply to install TCP/IP on the Macintosh using one of several commercial packages available. Apple's own MacTCP software product can perform the basic services but must be coupled with software from other vendors for the higher layer applications. MacTCP also requires a Datagram Delivery Protocol to Internet Protocol (DDP-to-IP) router to handle the sending and receiving of DDP and IP datagrams.

Apple's MacTCP functions by providing the physical through transport layers of the architecture. MacTCP allows for both LocalTalk and Ethernet hardware and supports both IP and TCP, as well as several other protocols. Running on top of MacTCP is the third-party application, which uses MacTCP's function calls to provide the final application for the user. Functions such as Telnet and FTP protocols are supported with add-on software, too.

2.4.6 DEC

Digital Equipment Corporation's minicomputers were for many years a mainstay in scientific and educational research, so an obvious development for DEC and third-party software companies was to introduce IP software. Most DEC machines run either VMS or Ultrix (DEC's licensed version of UNIX). Providing IP capabilities to Ultrix was a matter of duplicating the code developed at Berkeley, but VMS was not designed for IP-type communications, relying instead on DEC's proprietary network software. DEC's networking software is the Digital Network Architecture (DECnet). The first widely used version was DECnet Phase IV (introduced in 1982), which used industry-standard protocols for the lower layers but was proprietary in the upper layers. The 1987 release of DECnet Phase V provided a combined DECnet IV and OSI system that allowed new OSI protocols to be used within the DECnet environment.

DEC announced the ADVANTAGE-NETWORKS in 1991 as an enhancement of DECnet Phase V, adding support for the Internet Protocols. With the ADVANTAGE-NETWORKS, users could choose between the older, DEC-specific DECnet, OSI, or IP schemes. ADVANTAGE-NETWORKS is DEC's attempt to provide interoperability,

providing the DEC-exclusive DECnet system for LAN use, and the TCP/IP and OSI systems for WANs and system interconnection between different hardware types.

Users of VMS systems can connect to the UNIX environment in several ways. The easiest is to use a software gateway between the VMS machine and a UNIX machine. DEC's TCP/IP Services for VMS performs this function, as do several third-party software solutions, such as the Kermit protocol from Columbia University, Wollongong Group's WIN/TCP, and TGV's MultiNet. The advantage of the third-party communications protocol products such as Kermit is that they don't have to be connected to a UNIX machine, because any operating system that supports the communications protocol will work. ADVANTAGE-NETWORKS users have more options available, many from DEC. Because the protocol is already embedded in the network software, it makes the most sense simply to use it as it comes, if it fits into the existing system architecture. Because of internal conversion software, ADVANTAGE-NETWORKS can connect from a DECnet machine using either the DECnet or the OSI protocols.

2.4.7 IBM's SNA

IBM's Systems Network Architecture (SNA) is in widespread use for both mainframes and minicomputers. Essentially all IBM equipment provides full support for IP and TCP, as well as many other popular protocols. Native IBM software is available for each machine, and several third-party products have appeared (usually at a lower cost than those offered by IBM). The IBM UNIX version, AIX (which few people know stands for Advanced Interactive Executive), has the TCP/IP software built in, enabling any machine that can run AIX (from workstations to large minicomputers) to interconnect through IP with no additional software. The different versions of AIX have slightly different support, so users should check before blindly trying to connect AIX machines.

For large systems such as mainframes, IBM has the 3172 Interconnect Controller, which sits between the mainframe and a network. The 3172 is a hefty box that handles high-speed traffic between a mainframe channel and the network, off-loading the processing for the communications aspect from the mainframe processor. It can connect to Ethernet or token ring networks and through additional software to DEC's DECnet. IBM mainframes running either MVS or VM can run software appropriately called TCP/IP for MVS and TCP/IP for VM. These products provide

access from other machines running TCP/IP to access the mainframe operating system remotely, usually over a LAN. The software enables the calling machine (the client in a client/server scheme) to act as a 3270-series terminal to MVS or VM. FTP is provided for file transfers with automatic conversion from EBCDIC to ASCII. An interface to PROFS is available. Both TCP/IP software products support SMTP for electronic mail.

2.4.8 Local Area Networks

LANs are an obvious target for TCP/IP, because TCP/IP helps solve many interconnection problems between different hardware and software platforms. To run TCP/IP over a network, the existing network and transport layer software must be replaced with TCP/IP, or the two must be merged together in some manner so that the LAN protocol can carry TCP/IP information within its existing protocol (encapsulation). Whichever solution is taken for the lower layer, a higher layer interface also must be developed, which resides in the equivalent of the data link layer, communicating between the higher layer applications and the hardware. This interface enables the higher layers to be independent of the hardware when using TCP/IP, which many popular LAN operating systems are not currently able to claim.

Three interfaces (which have been mentioned earlier today) are currently in common use. The Packet Driver interface was the first interface developed to meet these needs. 3Com Corporation and Microsoft developed the Network Driver Interface Specification (NDIS) for OS/2 and 3Com's networking software. NDIS provides a driver to communicate with the networking hardware and a protocol driver that acts as the interface to the higher layers. Novell's Open Data Link Interface (ODI) is similar to NDIS. For single-vendor, PC-based networks, several dedicated TCP/IP packages are available, such as Novell's LAN WorkPlace, designed to enable any NetWare system to connect to a LAN using an interface hardware card and a software driver.

CHAPTER 3

TCP AND UDP

The Internet Protocol handles the lower-layer functionality. We will look at the transport layer, where the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) come into play.

TCP is one of the most widely used transport layer protocols, expanding from its original implementation on the ARPANET to connecting commercial sites all over the world. On Day 1, "Open Systems, Standards, and Protocols," you looked at the OSI seven-layer model, which bears a striking resemblance to TCP/IP's layered model, so it is not surprising that many of the features of the OSI transport layer were based on TCP.

In theory, a transport layer protocol could be a very simple software routine, but TCP cannot be called simple. Why use a transport layer that is as complex as TCP? The most important reason depends on IP's unreliability. IP does not guarantee delivery of a datagram; it is a connectionless system with no reliability. IP simply handles the routing of datagrams, and if problems occur, IP discards the packet without a second thought (generating an ICMP error message back to the sender in the process). The task of ascertaining the status of the datagrams sent over a network and handling the resending of information if parts have been discarded falls to TCP, which can be thought of as riding shotgun over IP. Most users think of TCP and IP as a tightly knit pair, but TCP can be (and frequently is) used with other protocols without IP. For example, TCP or parts of it are used in the File Transfer Protocol (FTP) and the Simple Mail Transfer Protocol (SMTP), both of which do not use IP.

3.1 What Is TCP?

The Transmission Control Protocol provides a considerable number of services to the IP layer and the upper layers. Most importantly, it provides a connection-oriented protocol to the upper layers that enable an application to be sure that a datagram sent out over the network was received in its entirety. In this role, TCP acts as a message-validation protocol providing reliable communications. If a datagram is corrupted or lost, TCP usually handles the retransmission, rather than the applications in the higher layers.

TCP manages the flow of datagrams from the higher layers to the IP layer, as well as incoming datagrams from the IP layer up to the higher level protocols. TCP has to ensure that priorities and security are properly respected. TCP must be capable of handling the termination of an application above it that was expecting incoming datagrams, as well as failures in the lower layers. TCP also must maintain a state table of all data streams in and out of the TCP layer. The isolation of all these services in a separate layer enables applications to be designed without regard to flow control or message reliability. Without the TCP layer, each application would have to implement the services themselves, which is a waste of resources. TCP resides in the transport layer, positioned above IP but below the upper layers and their applications, as shown in Figure 4.1. TCP resides only on devices that actually process datagrams, ensuring that the datagram has gone from the source to the target machine. It does not reside on a device that simply routes datagrams, so there is usually no TCP layer in a gateway. This makes sense, because on a gateway the datagram has no need to go higher in the layered model than the IP layer.

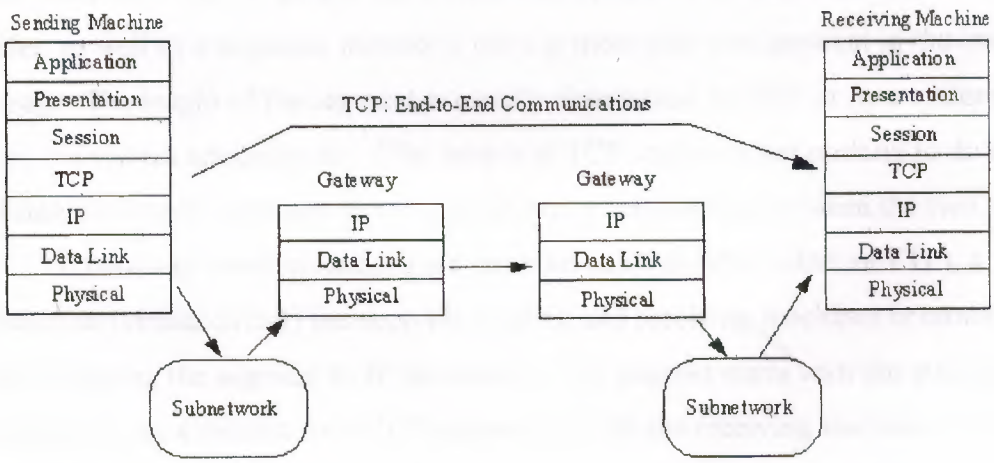

Figure 4.1. TCP provides end-to-end communications.

Because TCP is a connection-oriented protocol responsible for ensuring the transfer of a datagram from the source to destination machine (end-to-end communications), TCP must receive communications messages from the destination machine to acknowledge receipt of the datagram. The term *virtual circuit* is usually

used to refer to the communications between the two end machines, most of which are simple acknowledgment messages (either confirmation of receipt or a failure code) and datagram sequence numbers.

3.2 Following a Message

To illustrate the role of TCP, it is instructive to follow a sample message between two machines. The processes are simplified at this stage, to be expanded on later today. The message originates from an application in an upper layer and is passed to TCP from the next higher layer in the architecture through some protocol (often referred to as an upper-layer protocol, or ULP, to indicate that it resides above TCP). The message is passed as a *stream*—a sequence of individual characters sent asynchronously. This is in contrast to most protocols, which use fixed blocks of data. This can pose some conversion problems with applications that handle only formally constructed blocks of data or insist on fixed-size messages.

TCP receives the stream of bytes and assembles them into TCP *segments*, or packets. In the process of assembling the segment, header information is attached at the front of the data. Each segment has a checksum calculated and embedded within the header, as well as a sequence number if there is more than one segment in the entire message. The length of the segment is usually determined by TCP or by a system value set by the system administrator. (The length of TCP segments has nothing to do with the IP datagram length, although there is sometimes a relationship between the two.)

If two-way communications are required (such as with Telnet or FTP), a connection (virtual circuit) between the sending and receiving machines is established prior to passing the segment to IP for routing. This process starts with the sending TCP software issuing a request for a TCP connection with the receiving machine. In the message is a unique number (called a socket number) that identifies the sending machine's connection. The receiving TCP software assigns its own unique socket number and sends it back to the original machine. The two unique numbers then define the connection between the two machines until the virtual circuit is terminated. (I look at sockets in a little more detail in a moment.) After the virtual circuit is established, TCP sends the segment to the IP software, which then issues the message over the network as a datagram. IP can perform any of the changes to the segment that you saw in yesterday's material, such as fragmenting it and reassembling it at the destination machine. These steps are completely transparent to the TCP layers, however. After

winding its way over the network, the receiving machine's IP passes the received segment up to the recipient machine's TCP layer, where it is processed and passed up to the applications above it using an upper-layer protocol.

If the message was more than one TCP segment long (not IP datagrams), the receiving TCP software reassembles the message using the sequence numbers contained in each segment's header. If a segment is missing or corrupt (which can be determined from the checksum), TCP returns a message with the faulty sequence number in the body. The originating TCP software can then resend the bad segment. If only one segment is used for the entire message, after comparing the segment's checksum with a newly calculated value, the receiving TCP software can generate either a positive acknowledgment (ACK) or a request to resend the segment and route the request back to the sending layer.

The receiving machine's TCP implementation can perform a simple flow control to prevent buffer overload. It does this by sending a buffer size called a window value to the sending machine, following which the sender can send only enough bytes to fill the window. After that, the sender must wait for another window value to be received. This provides a handshaking protocol between the two machines, although it slows down the transmission time and slightly increases network traffic. As with most connection-based protocols, timers are an important aspect of TCP. The use of a timer ensures that an undue wait is not involved while waiting for an ACK or an error message. If the timers expire, an incomplete transmission is assumed. Usually an expiring timer before the sending of an acknowledgment message causes a retransmission of the datagram from the originating machine.

Timers can cause some problems with TCP. The specifications for TCP provide for the acknowledgment of only the highest datagram number that has been received without error, but this cannot properly handle fragmentary reception. If a message is composed of several datagrams that arrive out of order, the specification states that TCP cannot acknowledge the reception of the message until all the datagrams have been received. So even if all but one datagram in the middle of the sequence have been successfully received, a timer might expire and cause all the datagrams to be resent. With large messages, this can cause an increase in network traffic.

If the receiving TCP software receives duplicate datagrams (as can occur with a retransmission after a timeout or due to a duplicate transmission from IP), the receiving version of TCP discards any duplicate datagrams, without bothering with an error

message. After all, the sending system cares only that the message was received—not how many copies were received. TCP does not have a negative acknowledgment (NAK) function; it relies on a timer to indicate lack of acknowledgment. If the timer has expired after sending the datagram without receiving an acknowledgment of receipt, the datagram is assumed to have been lost and is retransmitted. The sending TCP software keeps copies of all unacknowledged datagrams in a buffer until they have been properly acknowledged. When this happens, the retransmission timer is stopped, and the datagram is removed from the buffer. TCP supports a push function from the upper-layer protocols. A push is used when an application wants to send data immediately and confirm that a message passed to TCP has been successfully transmitted. To do this, a push flag is set in the ULP connection, instructing TCP to forward any buffered information from the application to the destination as soon as possible (as opposed to holding it in the buffer until it is ready to transmit it).

3.3 Ports and Sockets

All upper-layer applications that use TCP (or UDP) have a port number that identifies the application. In theory, port numbers can be assigned on individual machines, or however the administrator desires, but some conventions have been adopted to enable better communications between TCP implementations. This enables the port number to identify the type of service that one TCP system is requesting from another. Port numbers can be changed, although this can cause difficulties. Most systems maintain a file of port numbers and their corresponding service.

Typically, port numbers above 255 are reserved for private use of the local machine, but numbers below 255 are used for frequently used processes. A list of frequently used port numbers is published by the Internet Assigned Numbers Authority and is available through an RFC or from many sites that offer Internet summary files for downloading. The commonly used port numbers on this list are shown in Table 4.1. The numbers 0 and 255 are reserved.

Table 4.1. Frequently used TCP port numbers.

Port Number	*Process Name*	*Description*
1	TCPMUX	TCP Port Service Multiplexer
5	RJE	Remote Job Entry
7	ECHO	Echo
9	DISCARD	Discard
11	USERS	Active Users
13	DAYTIME	Daytime
17	Quote	Quotation of the Day
19	CHARGEN	Character generator
20	FTP-DATA	File Transfer Protocol•Data
21	FTP	File Transfer Protocol•Control
23	TELNET	Telnet
25	SMTP	Simple Mail Transfer Protocol
27	NSW-FE	NSW User System Front End
29	MSG-ICP	MSG-ICP
31	MSG-AUTH	MSG Authentication
33	DSP	Display Support Protocol
35		Private Print Servers
37	TIME	Time
39	RLP	Resource Location Protocol
41	GRAPHICS	Graphics
42	NAMESERV	Host Name Server
43	NICNAME	Who Is
49	LOGIN	Login Host Protocol

53	DOMAIN	Domain Name Server
67	BOOTPS	Bootstrap Protocol Server
68	BOOTPC	Bootstrap Protocol Client
69	TFTP	Trivial File Transfer Protocol
79	FINGER	Finger
101	HOSTNAME	NIC Host Name Server
102	ISO-TSAP	ISO TSAP
103	X400	X.400
104	X400SND	X.400 SND
105	CSNET-NS	CSNET Mailbox Name Server
109	POP2	Post Office Protocol v2
110	POP3	Post Office Protocol v3
111	RPC	Sun RPC Portmap
137	NETBIOS-NS	NETBIOS Name Service
138	NETBIOS-DG	NETBIOS Datagram Service
139	NETBIOS-SS	NETBIOS Session Service
146	ISO-TP0	ISO TP0
147	ISO-IP	ISO IP
150	SQL-NET	SQL NET
153	SGMP	SGMP
156	SQLSRV	SQL Service
160	SGMP-TRAPS	SGMP TRAPS
161	SNMP	SNMP
162	SNMPTRAP	SNMPTRAP

163	CMIP-MANAGE	CMIP/TCP Manager
164	CMIP-AGENT	CMIP/TCP Agent
165	XNS-Courier	Xerox
179	BGP	Border Gateway Protocol

Each communication circuit into and out of the TCP layer is uniquely identified by a combination of two numbers, which together are called a socket. The socket is composed of the IP address of the machine and the port number used by the TCP software. Both the sending and receiving machines have sockets. Because the IP address is unique across the internetwork, and the port numbers are unique to the individual machine, the socket numbers are also unique across the entire internetwork. This enables a process to talk to another process across the network, based entirely on the socket number.

The last section examined the process of establishing a message. During the process, the sending TCP requests a connection with the receiving TCP, using the unique socket numbers. This process is shown in Figure 4.2. If the sending TCP wants to establish a Telnet session from its port number 350, the socket number would be composed of the source machine's IP address and the port number (350), and the message would have a destination port number of 23 (Telnet's port number). The receiving TCP has a source port of 23 (Telnet) and a destination port of 350 (the sending machine's port).

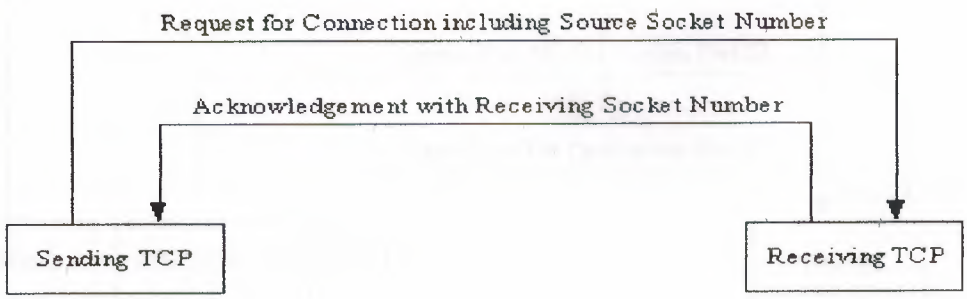

Figure 4.2. Setting up a virtual circuit with socket numbers.

The sending and receiving machines maintain a port table, which lists all active port numbers. The two machines involved have reversed entries for each session between the two. This is called binding and is shown in Figure 4.3. The source and destination numbers are simply reversed for each connection in the port table. Of course, the IP addresses, and hence the socket numbers, are different.

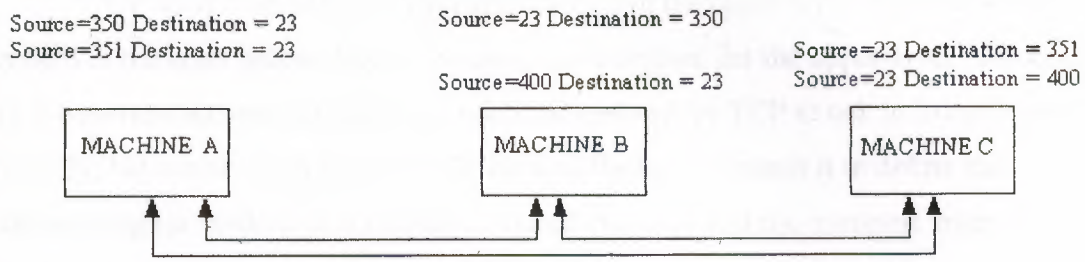

Figure 4.3. Binding entries in port tables.

If the sending machine is requesting more than one connection, the source port numbers are different, even though the destination port numbers might be the same. For example, if the sending machine were trying to establish three Telnet sessions simultaneously, the source machine port numbers might be 350, 351, and 352, and the destination port numbers would all be 23. It is possible for more than one machine to share the same destination socket—a process called multiplexing. In Figure 4.4, three machines are establishing Telnet sessions with a destination. They all use destination port 23, which is port multiplexing. Because the datagrams emerging from the port have the full socket information (with unique IP addresses), there is no confusion as to which machine a datagram is destined for.

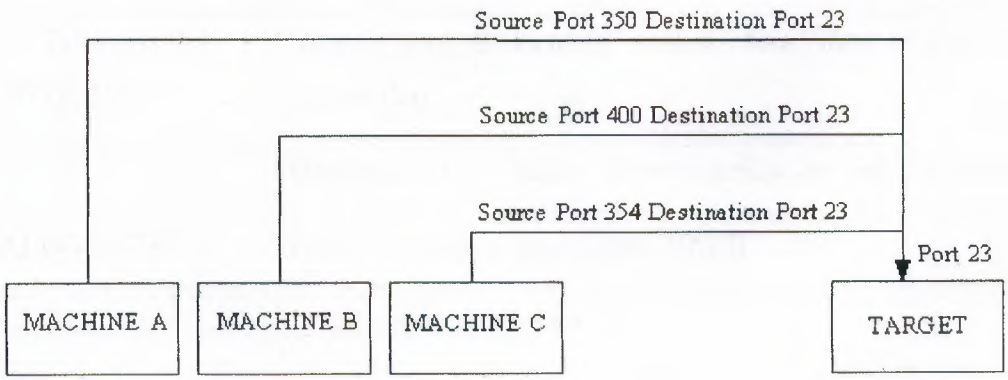

Figure 4.4. Multiplexing one destination port.

When multiple sockets are established, it is conceivable that more than one machine might send a connection request with the same source and destination ports. However, the IP addresses for the two machines are different, so the sockets are still uniquely identified despite identical source and destination port numbers.

3.4 TCP Communications with the Upper Layers

TCP must communicate with applications in the upper layer and a network system in the layer below. Several messages are defined for the upper-layer protocol to TCP communications, but there is no defined method for TCP to talk to lower layers (usually, but not necessarily, IP). TCP expects the layer beneath it to define the communication method. It is usually assumed that TCP and the transport layer communicate asynchronously. The TCP to upper-layer protocol (ULP) communication method is well-defined, consisting of a set of service request primitives. The primitives involved in ULP to TCP communications are shown in Table 4.2.

Table 4.2. ULP-TCP service primitives.

Command	*Parameters Expected*
ULP to TCP Service Request Primitives	
ABORT	Local connection name
ACTIVE-OPEN	Local port, remote socket
	Optional: ULP timeout, timeout action, precedence, security, options
ACTIVE-OPEN-WITH-DATA	Source port, destination socket, data, data length, push flag, urgent flag
	Optional: ULP timeout, timeout action, precedence, security
ALLOCATE	Local connection name, data length
CLOSE	Local connection name
FULL-PASSIVE-OPEN	Local port, destination socket

	Optional: ULP timeout, timeout action, precedence, security, options
RECEIVE	Local connection name, buffer address, byte count, push flag, urgent flag
SEND	Local connection name, buffer address, data length, push flag, urgent flag
	Optional: ULP timeout, timeout action
STATUS	Local connection name
UNSPECIFIED-PASSIVE-OPEN	Local port
	Optional: ULP timeout, timeout action, precedence, security, options
TCP to ULP Service Request Primitives	
CLOSING	Local connection name
DELIVER	Local connection name, buffer address, data length, urgent flag
ERROR	Local connection name, error description
OPEN-FAILURE	Local connection name
OPEN-ID	Local connection name, remote socket, destination address
OPEN-SUCCESS	Local connection name
STATUS RESPONSE	Local connection name, source port, source address, remote socket, connection state, receive window, send window, amount waiting ACK, amount waiting receipt, urgent mode, precedence, security, timeout, timeout action
TERMINATE	Local connection name, description

3.5 Passive and Active Ports

TCP enables two methods to establish a connection: active and passive. An active connection establishment happens when TCP issues a request for the connection, based on an instruction from an upper-level protocol that provides the socket number. A passive approach takes place when the upper-level protocol instructs TCP to wait for the arrival of connection requests from a remote system (usually from an active open instruction). When TCP receives the request, it assigns a port number. This enables a connection to proceed rapidly, without waiting for the active process.

There are two passive open primitives. A specified passive open creates a connection when the precedence level and security level are acceptable. An unspecified passive open opens the port to any request. The latter is used by servers that are waiting for clients of an unknown type to connect to them. TCP has strict rules about the use of passive and active connection processes. Usually a passive open is performed on one machine, while an active open is performed on the other, with specific information about the socket number, precedence (priority), and security levels. Although most TCP connections are established by an active request to a passive port, it is possible to open a connection without a passive port waiting. In this case, the TCP that sends a request for a connection includes both the local socket number and the remote socket number. If the receiving TCP is configured to enable the request (based on the precedence and security settings, as well as application-based criteria), the connection can be opened. This process is looked at again in the section titled "TCP and Connections."

3.6 TCP Timers

TCP uses several timers to ensure that excessive delays are not encountered during communications. Several of these timers are elegant, handling problems that are not immediately obvious at first analysis. The timers used by TCP are examined in the following sections, which reveal their roles in ensuring that data is properly sent from one connection to another.

3.6.1 The Retransmission Timer

The retransmission timer manages retransmission timeouts (RTOs), which occur when a preset interval between the sending of a datagram and the returning acknowledgment is exceeded. The value of the timeout tends to vary, depending on the

network type, to compensate for speed differences. If the timer expires, the datagram is retransmitted with an adjusted RTO, which is usually increased exponentially to a maximum preset limit. If the maximum limit is exceeded, connection failure is assumed, and error messages are passed back to the upper-layer application.

Values for the timeout are determined by measuring the average time that data takes to be transmitted to another machine and the acknowledgment received back, which is called the round-trip time, or RTT. From experiments, these RTTs are averaged by a formula that develops an expected value, called the smoothed round-trip time, or SRTT. This value is then increased to account for unforeseen delays.

3.6.2 The Quiet Timer

After a TCP connection is closed, it is possible for datagrams that are still making their way through the network to attempt to access the closed port. The quiet timer is intended to prevent the just-closed port from reopening again quickly and receiving these last datagrams. The quiet timer is usually set to twice the maximum segment lifetime (the same value as the Time to Live field in an IP header), ensuring that all segments still heading for the port have been discarded. Typically, this can result in a port being unavailable for up to 30 seconds, prompting error messages when other applications attempt to access the port during this interval.

3.6.3 The Persistence Timer

The persistence timer handles a fairly rare occurrence. It is conceivable that a receive window might have a value of 0, causing the sending machine to pause transmission. The message to restart sending might be lost, causing an infinite delay. The persistence timer waits a preset time and then sends a one-byte segment at predetermined intervals to ensure that the receiving machine is still clogged. The receiving machine resends the zero window-size message after receiving one of these status segments, if it is still backlogged. If the window is open, a message giving the new value is returned, and communications are resumed.

3.6.4 The Keep-Alive Timer and the Idle Timer

Both the keep-alive timer and the idle timer were added to the TCP specifications after their original definition. The keep-alive timer sends an empty packet

at regular intervals to ensure that the connection to the other machine is still active. If no response has been received after sending the message by the time the idle timer has expired, the connection is assumed to be broken. The keep-alive timer value is usually set by an application, with values ranging from 5 to 45 seconds. The idle timer is usually set to 360 seconds.

3.7 Transmission Control Blocks and Flow Control

TCP has to keep track of a lot of information about each connection. It does this through a Transmission Control Block (TCB), which contains information about the local and remote socket numbers, the send and receive buffers, security and priority values, and the current segment in the queue. The TCB also manages send and receive sequence numbers. The TCB uses several variables to keep track of the send and receive status and to control the flow of information. These variables are shown in Table 4.3.

Table 4.3. TCP send and receive variables.

Variable Name	***Description***
Send Variables	
SND.UNA	Send Unacknowledged
SND.NXT	Send Next
SND.WND	Send Window
SND.UP	Sequence number of last urgent set
SND.WL1	Sequence number for last window update
SND.WL2	Acknowledgment number for last window update
SND.PUSH	Sequence number of last pushed set
ISS	Initial send sequence number
Receive Variables	
RCV.NXT	Sequence number of next received set
RCV.WND	Number of sets that can be received

RCV.UP	Sequence number of last urgent data
RCV.IRS	Initial receive sequence number

Using these variables, TCP controls the flow of information between two sockets. A sample connection session helps illustrate the use of the variables. It begins with Machine A wanting to send five blocks of data to Machine B. If the window limit is seven blocks, a maximum of seven blocks can be sent without acknowledgment. The SND.UNA variable on Machine A indicates how many blocks have been sent but are unacknowledged (5), and the SND.NXT variable has the value of the next block in the sequence (6). The value of the SND.WND variable is 2 (seven blocks possible, minus five sent), so only two more blocks could be sent without overloading the window. Machine B returns a message with the number of blocks received, and the window limit is adjusted accordingly.

The passage of messages back and forth can become quite complex as the sending machine forwards blocks unacknowledged up to the window limit, waiting for acknowledgment of earlier blocks that have been removed from the incoming cue, and then sending more blocks to fill the window again. The tracking of the blocks becomes a matter of bookkeeping, but with large window limits and traffic across internetworks that sometimes cause blocks to go astray, the process is, in many ways, remarkable.

3.8 TCP Protocol Data Units

As mentioned earlier, TCP must communicate with IP in the layer below (using an IP-defined method) and applications in the upper layer (using the TCP-ULP primitives). TCP also must communicate with other TCP implementations across networks. To do this, it uses Protocol Data Units (PDUs), which are called segments in TCP parlance. The layout of the TCP PDU (commonly called the header) is shown in Figure 4.5.

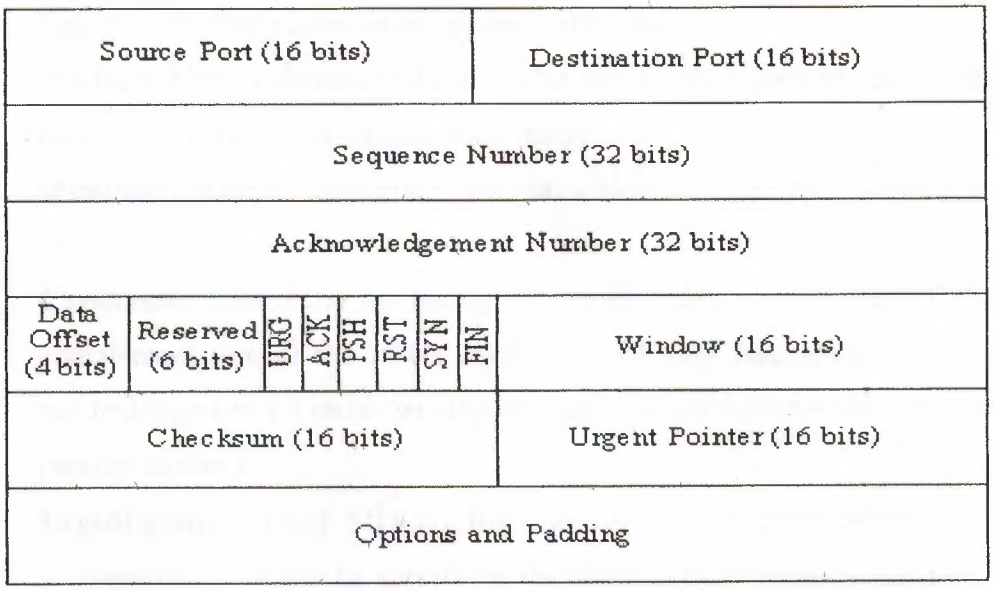

Figure 4.5. The TCP Protocol Data Unit.

The different fields are as follows:

- **Source port:** A 16-bit field that identifies the local TCP user (usually an upper-layer application program).
- **Destination port:** A 16-bit field that identifies the remote machine's TCP user.
- **Sequence number:** A number indicating the current block's position in the overall message. This number is also used between two TCP implementations to provide the initial send sequence (ISS) number.
- **Acknowledgment number:** A number that indicates the next sequence number expected. In a backhanded manner, this also shows the sequence number of the last data received; it shows the last sequence number received plus 1.
- **Data offset:** The number of 32-bit words that are in the TCP header. This field is used to identify the start of the data field.
- **Reserved:** A 6-bit field reserved for future use. The six bits must be set to 0.
- **Urg flag:** If on (a value of 1), indicates that the urgent pointer field is significant.
- **Ack flag:** If on, indicates that the Acknowledgment field is significant.
- **Psh flag:** If on, indicates that the push function is to be performed.
- **Rst flag:** If on, indicates that the connection is to be reset.

- **Syn flag:** If on, indicates that the sequence numbers are to be synchronized. This flag is used when a connection is being established.
- **Fin flag:** If on, indicates that the sender has no more data to send. This is the equivalent of an end-of-transmission marker.
- **Window:** A number indicating how many blocks of data the receiving machine can accept.
- **Checksum:** Calculated by taking the 16-bit one's complement of the one's complement sum of the 16-bit words in the header (including pseudo-header) and text together. (A rather lengthy process required to fit the checksum properly into the header.)
- **Urgent pointer:** Used if the urg flag was set; it indicates the portion of the data message that is urgent by specifying the offset from the sequence number in the header. No specific action is taken by TCP with respect to urgent data; the action is determined by the application.
- **Options:** Similar to the IP header option field, this is used for specifying TCP options. Each option consists of an option number (one byte), the number of bytes in the option, and the option values. Only three options are currently defined for TCP:

  0 End of option list

  1 No operation

  2 Maximum segment size
- **Padding:** Filled to ensure that the header is a 32-bit multiple.

Following the PDU or header is the data. The Options field has one useful function: to specify the maximum buffer size a receiving TCP implementation can accommodate. Because TCP uses variable-length data areas, it is possible for a sending machine to create a segment that is longer than the receiving software can handle. The Checksum field calculates the checksum based on the entire segment size, including a 96-bit pseudoheader that is prefixed to the TCP header during the calculation. The pseudoheader contains the source address, destination address, protocol identifier, and segment length. These are the parameters that are passed to IP when a send instruction is passed, and also the ones read by IP when delivery is attempted.

3.9 TCP and Connections

TCP has many rules imposed on how it communicates. These rules and the processes that TCP follows to establish a connection, transfer data, and terminate a connection are usually presented in state diagrams. (Because TCP is a state-driven protocol, its actions depend on the state of a flag or similar construct.) Avoiding overly complex state diagrams is difficult, so flow diagrams can be used as a useful method for understanding TCP.

3.9.1 Establishing a Connection

A connection can be established between two machines only if a connection between the two sockets does not exist, both machines agree to the connection, and both machines have adequate TCP resources to service the connection. If any of these conditions are not met, the connection cannot be made. The acceptance of connections can be triggered by an application or a system administration routine.

When a connection is established, it is given certain properties that are valid until the connection is closed. Typically, these are a precedence value and a security value. These settings are agreed upon by the two applications when the connection is in the process of being established. In most cases, a connection is expected by two applications, so they issue either active or passive open requests. Figure 4.6 shows a flow diagram for a TCP open. The process begins with Machine A's TCP receiving a request for a connection from its ULP, to which it sends an active open primitive to Machine B. (Refer back to Table 4.2 for the TCP primitives.) The segment that is constructed has the SYN flag set on (set to 1) and has a sequence number assigned. The diagram shows this with the notation "SYN SEQ 50," indicating that the SYN flag is on and the sequence number (Initial Send Sequence number or ISS) is 50. (Any number could have been chosen.) The application on Machine B has issued a passive open instruction to its TCP. When the SYN SEQ 50 segment is received, Machine B's TCP sends an acknowledgment back to Machine A with the sequence number of 51. Machine B also sets an ISS number of its own. The diagram shows this message as "ACK 51; SYN 200," indicating that the message is an acknowledgment with sequence number 51, it has the SYN flag set, and it has an ISS of 200.

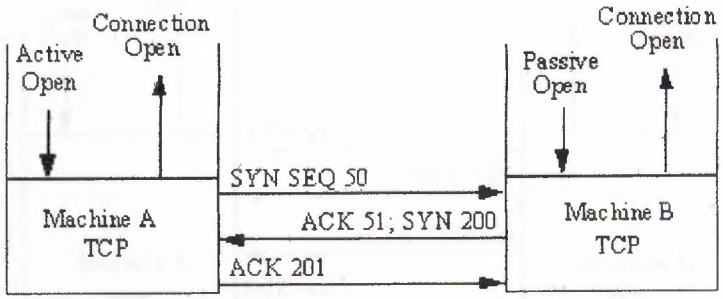

Figure 4.6. Establishing a connection.

Upon receipt, Machine A sends back its own acknowledgment message with the sequence number set to 201. This is "ACK 201" in the diagram. Then, having opened and acknowledged the connection, Machine A and Machine B both send connection open messages through the ULP to the requesting applications. It is not necessary for the remote machine to have a passive open instruction, as mentioned earlier. In this case, the sending machine provides both the sending and receiving socket numbers, as well as precedence, security, and timeout values. It is common for two applications to request an active open at the same time. This is resolved quite easily, although it does involve a little more network traffic.

3.9.2 Data Transfer

Transferring information is straightforward, as shown in Figure 4.7. For each block of data received by Machine A's TCP from the ULP, TCP encapsulates it and sends it to Machine B with an increasing sequence number. After Machine B receives the message, it acknowledges it with a segment acknowledgment that increments the next sequence number (and hence indicates that it has received everything up to that sequence number). Figure 4.7 shows the transfer of two segments of information—one each way.

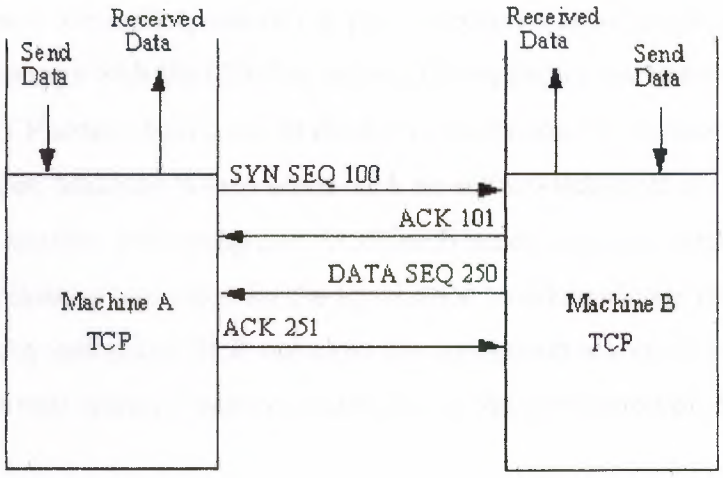

Figure 4.7. Data transfers.

The TCP data transport service actually embodies six subservices:

- **Full duplex:** Enables both ends of a connection to transmit at any time, even simultaneously.
- **Timeliness:** The use of timers ensures that data is transmitted within a reasonable amount of time.
- **Ordered:** Data sent from one application is received in the same order at the other end. This occurs despite the fact that the datagrams might be received out of order through IP, because TCP reassembles the message in the correct order before passing it up to the higher layers.
- **Labeled:** All connections have an agreed-upon precedence and security value.
- **Controlled flow:** TCP can regulate the flow of information through the use of buffers and window limits.
- **Error correction:** Checksums ensure that data is free of errors (within the checksum algorithm's limits).

3.9.3 Closing Connections

To close a connection, one of the TCPs receives a close primitive from the ULP and issues a message with the FIN flag set on. This is shown in Figure 4.8. In the figure, Machine A's TCP sends the request to close the connection to Machine B with the next sequence number. Machine B then sends back an acknowledgment of the request and its next sequence number. Following this, Machine B sends the close message through its ULP to the application and waits for the application to acknowledge the closure. This step is not strictly necessary; TCP can close the connection without the application's approval, but a well-behaved system would inform the application of the change in state.

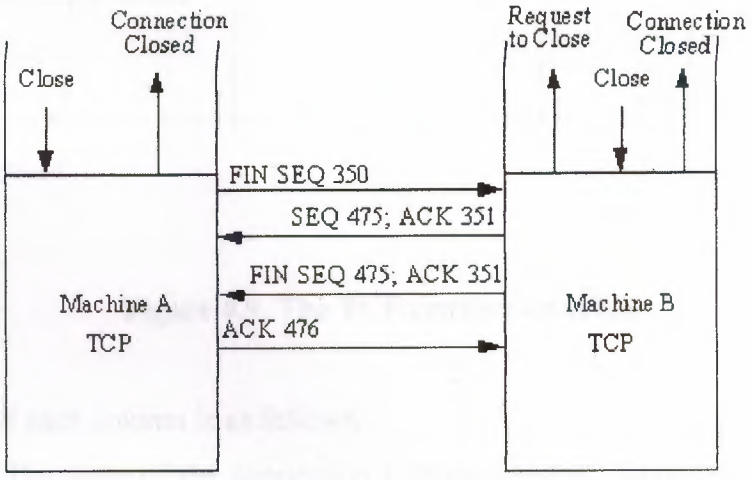

Figure 4.8. Closing a connection.

After receiving approval to close the connection from the application (or after the request has timed out), Machine B's TCP sends a segment back to Machine A with the FIN flag set. Finally, Machine A acknowledges the closure, and the connection is terminated. An abrupt termination of a connection can occur when one side shuts down the socket. This can be done without any notice to the other machine and without regard to any information in transit between the two. Aside from sudden shutdowns caused by malfunctions or power outages, abrupt termination can be initiated by a user, an application, or a system monitoring routine that judges the connection worthy of termination. The other end of the connection might not realize that an abrupt termination has occurred until it attempts to send a message and the timer expires.

To keep track of all the connections, TCP uses a connection table. Each existing connection has an entry in the table that shows information about the end-to-end connection. The layout of the TCP connection table is shown in Figure 4.9.

	STATE	LOCAL ADDRESS	LOCAL PORT	REMOTE ADDRESS	REMOTE PORT
Connection 1					
Connection 2					
Connection 3					
Connection n					

Figure 4.9. The TCP connection table.

The meaning of each column is as follows:

- **State:** The state of the connection (closed, closing, listening, waiting, and so on).
- **Local address:** The IP address for the connection. When in a listening state, this is set to 0.0.0.0.
- **Local port:** The local port number.
- **Remote address:** The remote machine's IP address.
- **Remote port:** The port number of the remote connection.

3.10 User Datagram Protocol (UDP)

TCP is a connection-based protocol. There are times when a connectionless protocol is required, so UDP is used. UDP is used with both the Trivial File Transfer Protocol (TFTP) and the Remote Call Procedure (RCP). Connectionless communications don't provide reliability, meaning there is no indication to the sending device that a message has been received correctly. Connectionless protocols also do not offer error-recovery capabilities—which must be either ignored or provided in the higher or lower layers. UDP is much simpler than TCP. It interfaces with IP (or other protocols) without the bother of flow control or error-recovery mechanisms, acting simply as a sender and receiver of datagrams.

The UDP message header is much simpler than TCP's. It is shown in Figure 4.10. Padding can be added to the datagram to ensure that the message is a multiple of 16 bits.

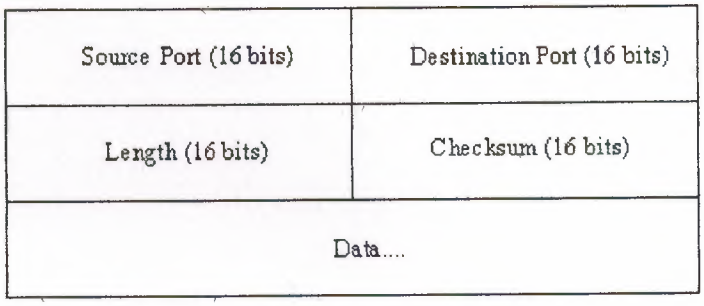

Figure 4.10. The UDP header.

The fields are as follows:

- **Source port:** An optional field with the port number. If a port number is not specified, the field is set to 0.
- **Destination port:** The port on the destination machine.
- **Length:** The length of the datagram, including header and data.
- **Checksum:** A 16-bit one's complement of the one's complement sum of the datagram, including a pseudoheader similar to that of TCP.

The UDP checksum field is optional, but if it isn't used, no checksum is applied to the data segment because IP's checksum applies only to the IP header. If the checksum is not used, the field should be set to 0.

CHAPTER 4

WINSOCK AND THE SOCKET PROGRAMMING INTERFACE

4.1 Winsock

For some Windows and Windows 95 users, Winsock is the easiest method to get into TCP/IP because it is available from many public domain, BBS, and online service sites. There are several versions of Winsock, some of which are public domain or shareware. We will look at two versions of Winsock, one for Windows 3.X and another for Windows 95. We have chosen the popular Trumpet Winsock implementations for both operating systems because they are shareware, readily available, and well supported. Winsock is short for Windows Sockets, originally developed by Microsoft. Released in 1993, Windows Sockets is an interface for network programming in the Windows environment. Microsoft has published the specifications for Windows Sockets, hence making it an open application programming interface (API). The Winsock API (called WSA) is a library of function calls, data structures, and programming procedures that provide this standardized interface for applications. The second release of Winsock, called Winsock version 2, was released in mid 1995.

4.1.1 Trumpet Winsock

Trumpet Winsock is a shareware implementation of Winsock produced by Trumpet Software International. Trumpet Winsock is available for Windows 3.X and Windows 95 systems. Registration of the Winsock package, developed in Australia, is $25 US. Trumpet Winsock lets you use several different protocols including PPP and SLIP for connection to the Internet or remote networks, direct connection using TCP/IP, and the BOOTP protocol. Trumpet Winsock allows dynamic IP addressing, which is necessary with many Internet Service Providers. The Trumpet Winsock files are usually provided in an archive ZIP file, and should be extracted into a new subdirectory on your system. The primary files in the Trumpet Winsock distribution are

WINSOCK.DLL: The primary protocol stack for Winsock
TCPMAN.EXE: Manages the communications between WINSOCK.DLL and the network

TRUMPWSK.INI: Contains Winsock variable settings

HOSTS: A list of hosts that Winsock is aware of

SERVICES: A list of services supported by Winsock

PROTOCOL: A list of protocols supported by Winsock

There are a number of sample configuration files included in the archive, as well as utilities such as PING and HOP. Some of the files in the Winsock archive, such as HOSTS, PROTOCOL, and SERVICES, mirror UNIX files of the same name.

4.1.2 Installing Trumpet Winsock

The installation process for Trumpet Winsock is the same whether you are using SLIP/PPP for connection or a packet driver for LAN-based operations. Begin the installation by adding the directory holding the Trumpet Winsock files to your PATH. The files should, of course, be extracted from the ZIP file they are usually supplied in. After the path has been modified, reboot your machine to effect the change.

You can create a Windows program group for the Trumpet Winsock system by adding a new program group from the Program Manager menus. (Select File menu, the New menu item, and then Program Group.) Create a title, such as Trumpet Winsock. for the new program group.

Next, create a Program Icon for the TCPMAN program (the primary Trumpet Winsock program) by either creating a new Program Item from the Program Manager or opening the File Manager and dragging the TCPMAN.EXE entry from its directory to the Trumpet Winsock program group. Windows will prompt you for any information it needs. The program icon is read from the distribution files if the path is properly set.

To test the installation of the path and the Windows icon, click the TCPMAN icon. If you receive error messages, either the PATH is not set properly or the program icon has not been properly defined. Because you are primarily interested in using Winsock on a TCP/IP network, ignore configuring PPP and SLIP and concentrate on the TCP/IP stack.

4.1.3 Configuring the TCP/IP Packet Driver

Trumpet Winsock relies on a program called WINPKT to provide TCP/IP packet capabilities under Windows. After you create a program group for Winsock, you need to set up the packet driver information in the network files.

You will need a packet driver for your system, which is not included with most Trumpet Winsock distributions. In many cases, the network card vendor includes a disk with a packet driver on it. If not, one of the best sources for a packet driver is the Crynwr Packet Driver collection, a library of different packet drivers available from many online, BBS, FTP, and WWW sites. The packet driver specifications are added to your network startup batch file, usually AUTOEXEC.BAT for DOS-based systems.

The process for configuring Trumpet Winsock for LAN operation is quite simple. Set the IRQ and I/O address of the packet driver and add the packet driver to your system. A typical entry in the network batch file looks like this:

```
ne2000 0x60 2 0x300

WINPKT 0x60
```

This sets the network to use an NE2000 (Novell) type card, with I/O address of 300H, IRQ of 2, and a vector of 60. Several configurations are usually provided with the Trumpet Winsock distribution, although it is easy to derive your own from the network interface card manufacturer's documentation.

To set up Trumpet Winsock for a packet driver, use the Setup screen that appears when TCPMAN is first launched, or use the menus within TCPMAN to display the setup screen. Deselect both Internal SLIP and Internal PPP settings. If either of them are checked, the packet driver will not launch properly. Enter the IP address, netmask, name server IP address, and domain name information. You may also modify the entries for Demand Load Time-out, MTU, TCP RWIN, TCP MSS, and TCP RTO MAX. See the section on SLIP/PPP configuration above for more details on any of these settings. The default values used for a packet driver are different than those for a SLIP/PPP setting. If you are using BOOTP or RARP to determine your machine IP address, enter the proper protocol name in the IP address field.

The Packet Vector field should be set to the vector you used in the network card description, or you can leave it as 00 to let Trumpet Winsock search for the packet

driver. After the configuration is saved, restart TCPMAN and the network will be available (if the configuration and packet drivers are properly set). A ping command or similar utility will verify the packet driver operation is correct.

4.2 The Socket Programming Interface

Because the original socket interface was developed for UNIX systems, today's text has a decidedly UNIX-based orientation. However, the same principles apply to most other operating systems that support TCP/IP.

4.2.1 Development of the Socket Programming Interface

TCP/IP is fortunate because it has a well-defined application programming interface (API), which dictates how an application uses TCP/IP. This solves a basic problem that has occurred on many other communications protocols, which have several approaches to the same problem, each incompatible with the other. The TCP/IP API is portable (it works across all operating systems and hardware that support TCP/IP), language-independent (it doesn't matter which language you use to write the application), and relatively uncomplicated. The Socket API was developed at the University of California at Berkeley as part of their BSD 4.1c UNIX version. Since then the API has been modified and enhanced but still retains its BSD flavor. Not to be outdone, AT&T (BSD's rival in the UNIX market) introduced the Transport Layer Interface (TLI) for TCP and several other protocols. One of the strengths of the Socket API and TLI is that they were not developed exclusively for TCP/IP but are intended for use with several communications protocols. The Socket interface remains the most widespread API in current use, although several newer interfaces are being developed.

The basic structure of all socket programming commands lies with the unique structure of UNIX I/O. With UNIX, both input and output are treated as simple pipelines, where the input can be from anything and the output can go anywhere. The UNIX I/O system is sometimes referred to as the *open-read-write-close* system, because those are the steps that are performed for each I/O operation, whether it involves a file, a device, or a communications port. Whenever a file is involved, the UNIX operating system gives the file a *file descriptor,* a small number that uniquely identifies the file. A program can use this file descriptor to identify the file at any time. (The same holds true for a device; the process is the same.) A file operation uses an open function to return

the file descriptor, which is used for the read (transfer data to the user's process) or write (transfer data from the user process to the file) functions, followed by a close function to terminate the file operation. The open function takes a filename as an argument. The read and write functions use the file descriptor number, the address of the buffer in which to read or write the information, and the number of bytes involved. The close function uses the file descriptor. The system is easy to use and simple to work with.

TCP/IP uses the same idea, relying on numbers to uniquely identify an end point for communications (a socket). Whenever the socket number is used, the operating system can resolve the socket number to the physical connector. An essential difference between a file descriptor and a socket number is that the socket requires some functions to be performed prior to the establishment of the socket (such as initialization). In techno-speak, "a file descriptor binds to a specific file or device when the open function is called, but the socket can be created without binding them to a specific destination at all (necessary for UDP), or bind them later (for TCP when the remote address is provided)." The same open-read-write-close procedure is used with sockets. The process was actually used literally with the first versions of TCP/IP. A special file called /dev/tcp was used as the device driver. The complexity added by networking made this approach awkward, though, so a library of special functions (the API) was developed. The essential steps of open, read, write, and close are still followed in the protocol API.

4.2.2 Socket Services

There are three types of socket interfaces defined in the TCP/IP API. A socket can be used for TCP *stream communications,* in which a connection between two machines is created. It can be used for *UDP datagram communications,* a connectionless method of passing information between machines using packets of a predefined format. Or it can be used as a *raw* datagram process, in which the datagrams bypass the TCP/UDP layer and go straight to IP. The latter type arises from the fact that the socket API was not developed exclusively for TCP/IP.

The presence of all three types of interfaces can lead to problems with some parameters that depend exclusively on the type of interface. You must always bear in mind whether TCP or UDP is used. There are six basic communications commands that the socket API addresses through the TCP layer:

open: Establishes a socket

send: Sends data to the socket

receive: Receives data from a socket

status: Obtains status information about a socket

close: Terminates a connection

abort: Cancels an operation and terminates the connection

All six operations are logical and used as you would expect. The details for each step can be quite involved, but the basic operation remains the same. Many of the functions have been seen in previous days when dealing with specific protocols in some detail. Some of the functions (such as open) comprise several other functions that are available if necessary (such as establishing each end of the connection instead of both ends at once). Despite the formal definition of the functions within the API specifications, no formal method is given for how to implement them. There are two logical choices: synchronous, or *blocking,* in which the application waits for the command to complete before continuing execution; and asynchronous, or *nonblocking,* in which the application continues executing while the API function is processed. In the latter case, a function call further in the application's execution can check the API functions' success and return codes. The problem with the synchronous or blocking method is that the application must wait for the function call to complete. If timeouts are involved, this can cause a noticeable delay for the user.

4.2.2.1 Transmission Control Block

The Transmission Control Block (TCB) is a complex data structure that contains details about a connection. The full TCB has over fifty fields in it. The exact layout and contents of the TCB are not necessary for today's material, but the existence of the TCB and the nature of the information it holds are key to the behavior of the socket interface.

4.2.2.2 Creating a Socket

The API lets a user create a socket whenever necessary with a simple function call. The function requires the family of the protocol to be used with the socket (so the operating system knows which type of socket to assign and how to decode information), the type of communication required, and the specific protocol. Such a function call is written as follows:

socket(*family*, *type*, *protocol*)

The *family* of the protocol actually specifies how the addresses are interpreted. Examples of families are TCP/IP (coded as AF_INET), Apple's AppleTalk (AF_APPLETALK), and UNIX filesystems (AF_UNIX). The exact protocol within the family is specified as the protocol parameter. When used, it specifically indicates the type of service that is to be used.

The *type* parameter indicates the type of communications used. It can be a connectionless datagram service (coded as SOCK_DGRAM), a stream delivery service (SOCK_STREAM), or a raw type (SOCK_RAW). The result from the function call is an integer that can be assigned to a variable for further checking.

4.2.2.3 Binding the Socket

Because a socket can be created without any binding to an address, there must be a function call to complete this process and establish the full connection. With the TCP/IP protocol, the socket function does not supply the local port number, the destination port, or the IP address of the destination. The bind function is called to establish the local port address for the connection. Some applications (especially on a server) want to use a specific port for a connection. Other applications are content to let the protocol software assign a port. A specific port can be requested in the bind function. If it is available, the software allocates it and returns the port information. If the port cannot be allocated (it might be in use), a return code indicates an error in port assignment.

The bind function has the following format:

bind(*socket*, *local_address*, *address_length*)

socket is the integer number of the socket to which the bind is completed; *local_address* is the local address to which the bind is performed; and *address_length* is an integer that gives the length of the address in bytes. The address is not returned as a simple number but has the structure shown in Figure 14.1.

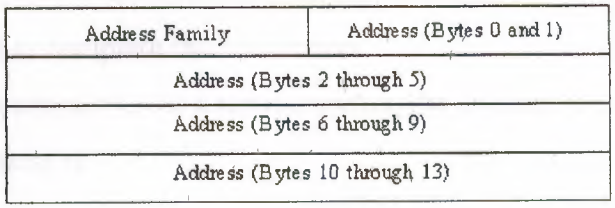

Figure 14.1. Address structure used by the socket API.

The address data structure (which is called usually called sockaddr for *socket address*) has a 16-bit Address Family field that identifies the protocol family of the address. The entry in this field determines the format of the address in the following field (which might contain other information than the address, depending on how the protocol has defined the field). The Address field can be up to 14 bytes in length, although most protocols do not need this amount of space.

TCP/IP has a family address of 2, following which the Address field contains both a protocol port number (16 bits) and the IP address (32 bits). The remaining eight bytes are unused. This is shown in Figure 14.2. Because the address family defines how the Address field is decoded, there should be no problem with TCP/IP applications understanding the two pieces of information in the Address field.

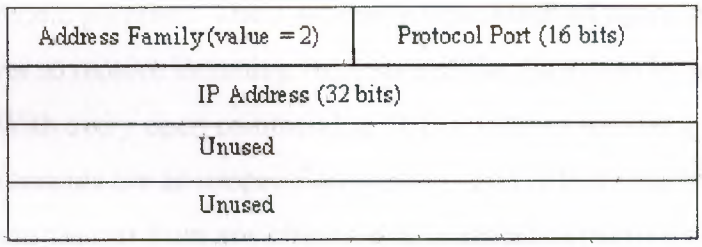

Figure 14.2. The address structure for TCP/IP.

4.2.2.4 Connecting to the Destination

After a local socket address and port number have been assigned, the destination socket can be connected. A one-ended connection is referred to as being in an

unconnected state, whereas a two-ended (complete) connection is in a *connected state.* After a bind function, an unconnected state exists. To become connected, the destination socket must be added to complete the connection.

To establish a connection to a remote socket, the connect function is used. The connect function's format is

connect(*socket, destination_address, address_length*)

The *socket* is the integer number of the socket to which to connect; the *destination_address* is the socket address data structure for the destination address (using the same format as shown in Figure 14.1); and the *address_length* is the length of the destination address in bytes.

The manner in which connect functions is protocol-dependent. For TCP, connect establishes the connection between the two endpoints and returns the information about the remote socket to the application. If a connection can't be established, an error message is generated. For a connectionless protocol such as UDP, the connect function is still necessary but stores only the destination address for the application.

4.2.2.5 The *open* Command

The open command prepares a communications port for communications. This is an alternative to the combination of the functions shown previously, used by applications for specific purposes. There are really three kinds of open commands, two of which set a server to receive incoming requests and the third used by a client to initiate a request. With every open command, a TCB is created for that connection. The three open commands are an unspecified passive open (which enables a server to wait for a connection request from any client), a fully specified passive open (which enables a server to wait for a connection request from a specific client), and an active open (which initiates a connection with a server). The input and output expected from each command are shown in Table 14.1.

Table 14.1. Open command parameters.

Type	*Input*	*Output*
Unspecified	local port	local connection name
passive open	Optional: timeout, precedence, security, maximum segment size	local connection name
Fully specified passive open	local port, remote IP address, remote port Optional: timeout, precedence, security, maximum segment size	local connection name
Active open	local port, destination IP address, destination port Optional: timeout, precedence, security, maximum segment size	local connection name

When an open command is issued by an application, a set of functions within the socket interface is executed to set up the TCB, initiate the socket number, and establish preliminary values for the variables used in the TCB and the application. The passive open command is issued by a server to wait for incoming requests. With the TCP (connection-based) protocol, the passive open issues the following function calls:

socket: Creates the sockets and identifies the type of communications.
bind: Establishes the server socket for the connection.
listen: Establishes a client queue.
accept: Waits for incoming connection requests on the socket.
The active open command is issued by a client. For TCP, it issues two functions:
socket: Creates the socket and identifies the communications type.
connect: Identifies the server's IP address and port; attempts to establish communications.

If the exact port to use is specified as part of the open command, a bind function call replaces the connect function.

4.2.2.6 Sending Data

There are five functions within the Socket API for sending data through a socket. These are send, sendto, sendmsg, write, and writev. Not surprisingly, all these functions send data from the application to TCP. They do this through a buffer created by the application (for example, it might be a memory address or a character string), passing the entire buffer to TCP. The send, write, and writev functions work only with a connected socket because they have no provision to specify a destination address within their function call. The format of the send function is simple. It takes the local socket connection number, the buffer address for the message to be sent, the length of the message in bytes, a Push flag, and an Urgent flag as parameters. An optional timeout might be specified. Nothing is returned as output from the send function. The format is

send(*socket, buffer_address, length, flags*)

The sendto and sendmsg functions are similar except they enable an application to send a message through an unconnected socket. They both require the destination address as part of their function call. The sendmsg function is simpler in format than the sendto function, primarily because another data structure is used to hold information. The sendmsg function is often used when the format of the sendto function would be awkward and inefficient in the application's code. Their formats are

sendto(*socket, buffer_address, length, flags, destination, address_length*)

sendmsg(*socket, message_structure, flags*)

The last two parameters in the sendto function are the destination address and the length of the destination address. The address is specified using the format shown in Figure 14.1. The *message_structure* of the sendmsg function contains the information left out of the sendto function call. The format of the message structure is shown in Figure 14.3.

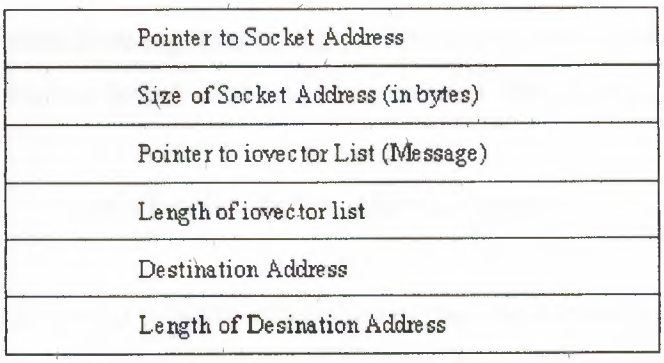

Figure 14.3. The message structure used by sendmsg.

The fields in the sendmsg message structure give the socket address, size of the socket address, a pointer to the iovector, which contains information about the message to be sent, the length of the iovector, the destination address, and the length of the destination address. The iovector is an address for an array that points to the message to be sent. The array is a set of pointers to the bytes that comprise the message. The format of the iovector is simple. For each 32-bit address to a memory location with a chunk of the message, a corresponding 32-bit field holds the length of the message in that memory location. This format is repeated until the entire message is specified. This is shown in Figure 14.4. The iovector format enables a noncontiguous message to be sent. In other words, the first part of the message can be in one location in memory, and the rest is separated by other information. This can be useful because it saves the application from copying long messages into a contiguous location.

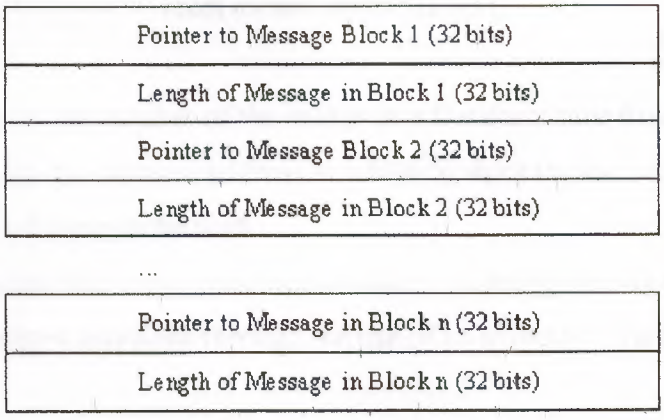

Figure 14.4. The iovector format.

The write function takes three arguments: the socket number, the buffer address of the message to be sent, and the length of the message to send. The format of the function call is

write(*socket*, *buffer_address*, *length*)

The writev function is similar to write except it uses the iovector to hold the message. This lets it send a message without copying it into another memory address. The format of writev is

writev(*socket*, *iovector*, *length*)

where *length* is the number of entries in iovector.

The type of function chosen to send data through a socket depends on the type of connection used and the level of complexity of the application. To a considerable degree, it is also a personal choice of the programmer.

4.2.2.7 Receiving Data

Not surprisingly, because there are five functions to send data through a socket, there are five corresponding functions to receive data: read, readv, recv, recvfrom, and recvmsg. They all accept incoming data from a socket into a reception buffer. The receive buffer can then be transferred from TCP to the application.

The read function is the simplest and can be used only when a socket is connected. Its format is

read(*socket*, *buffer*, *length*)

The first parameter is the number of the socket or a file descriptor from which to read the data, followed by the memory address in which to store the incoming data, and the maximum number of bytes to be read.

As with writev, the readv command enables incoming messages to be placed in noncontiguous memory locations through the use of an iovector. The format of readv is

readv(*socket*, *iovector*, *length*)

length is the number of entries in the iovector. The format of the iovector is the same as mentioned previously and shown in Figure 14.4.

The recv function also can be used with connected sockets. It has the format

recv(*socket*, *buffer_address*, *length*, *flags*)

which corresponds to the send function's arguments.

The recvfrom and recvmsg functions enable data to be read from an unconnected socket. Their formats include the sender's address:

recvfrom(*socket*, *buffer_address*, *length*, *flags*, *source_address*, *address_length*)

recvmsg(*socket*, *message_structure*, *flags*)

The message structure in the recvmsg function corresponds to the structure in sendmsg. (See Figure 14.3.)

4.2.2.8 Server Listening

A server application that expects clients to call in to it has to create a socket (using socket), bind it to a port (with bind), then wait for incoming requests for data. The listen function handles problems that could occur with this type of behavior by establishing a queue for incoming connection requests. The queue prevents bottlenecks and collisions, such as when a new request arrives before a previous one has been completely handled, or two requests arrive simultaneously.

The listen function establishes a buffer to queue incoming requests, thereby avoiding losses. The function lets the socket accept incoming connection requests, which are all sent to the queue for future processing. The function's format is

listen(*socket*, *queue_length*)

where *queue_length* is the size of the incoming buffer. If the buffer has room, incoming requests for connections are added to the buffer and the application can deal with them in the order of reception. If the buffer is full, the connection request is rejected.

After the server has used listen to set up the incoming connection request queue, the accept function is used to actually wait for a connection. The format of the function is

accept(*socket*, *address*, *length*)

socket is the socket on which to accept requests; *address* is a pointer to a structure similar to Figure 14.1; and *length* is a pointer to an integer showing the length of the address.

When a connection request is received, the protocol places the address of the client in the memory location indicated by the address parameter, and the length of that address in the length location. It then creates a new socket that has the client and server connected together, sending back the socket description to the client. The socket on which the request was received remains open for other connection requests. This enables multiple requests for a connection to be processed, whereas if that socket was closed down with each connection request, only one client/server process could be handled at a time.

One possible special occurrence must be handled on UNIX systems. It is possible for a single process to wait for a connection request on multiple sockets. This reduces the number of processes that monitor sockets, thereby lowering the amount of overhead the machine uses. To provide for this type of process, the select function is used. The format of the function is

select(*num_desc*, *in_desc*, *out_desc*, *excep_desc*, *timeout*)

num_desc is the number of sockets or descriptors that are monitored; *in_desc* and *out_desc* are pointers to a bit mask that indicates the sockets or file descriptors to monitor for input and output, respectively; *excep_desc* is a pointer to a bit mask that specifies the sockets or file descriptors to check for exception conditions; and *timeout* is a pointer to an integer that indicates how long to wait (a value of 0 indicates forever). To use the select function, a server creates all the necessary sockets first, then calls select to determine which ones are for input, output, and exceptions.

4.2.2.9 Getting Status Information

Several status functions are used to obtain information about a connection. They can be used at any time, although they are typically used to establish the integrity of a connection in case of problems or to control the behavior of the socket.

The status functions require the name of the local connection, and they return a set of information, which might include the local and remote socket names, local connection name, receive and send window states, number of buffers waiting for an acknowledgment, number of buffers waiting for data, and current values for the urgent state, precedence, security, and timeout variables. Most of this information is read from the Transmission Control Block (TCB). The format of the information and the exact contents vary slightly, depending on the implementation.

The function getsockopt enables an application to query the socket for information. The function format is

getsockopt(*socket*, *level*, *option_id*, *option_result*, *length*)

socket is the number of the socket; *level* indicates whether the function refers to the socket itself or the protocol that uses it; *option_id* is a single integer that identifies the type of information requested; *option_result* is a pointer to a memory location where the function should place the result of the query; and *length* is the length of the result.

The corresponding setsockopt function lets the application set a value for the socket. The function's format is the same as getsockopt except that *option_result* points to the value that is to be set, and *length* is the length of the value.

Two functions provide information about the local address of a socket. The getpeername function returns the address of the remote end. The getsockname function returns the local address of a socket. They have the following formats:

getpeername(*socket*, *destination_address*, *address_length*)

getsockname(*socket*, *local_address*, *address_length*)

The addresses in both functions are pointers to a structure of the format shown in Figure 14.1. Two host name functions for BSD UNIX are gethostname and sethostname, which

enable an application to obtain the name of the host and set the host name (if permissions allow). Their formats are as follows:

sethostname(*name*, *length*)

gethostname(*name*, *length*)

The *name* is the address of an array that holds the name, and the *length* is an integer that gives the name's length.

A similar set of functions provides for domain names. The functions setdomainname and getdomainname enable an application to obtain or set the domain names. Their formats are

setdomainname(*name*, *length*)

getdomainname(*name*, *length*)

The parameters are the same as with the sethostname and gethostname functions, except for the format of the name (which reflects domain name format).

4.2.2.10 Closing a Connection

The close function closes a connection. It requires only the local connection name to complete the process. It also takes care of the TCB and releases any variable created by the connection. No output is generated.

The close function is initiated with the call

close(*socket*)

where the socket name is required. If an application terminates abnormally, the operating system closes all sockets that were open prior to the termination.

4.2.2.11 Aborting a Connection

The abort function instructs TCP to discard all data that currently resides in send and receive buffers and close the connection. It takes the local connection name as input. No output is generated. This function can be used in case of emergency shutdown routines, or in case of a fatal failure of the connection or associated software.
The abort function is usually implemented by the close() call, although some special instructions might be available with different implementations.

4.2.2.12 UNIX Forks

UNIX has two system calls that can affect sockets: fork and exec. Both are frequently used by UNIX developers because of their power. (In fact, forks are one of the most powerful tools UNIX offers, and one that most other operating systems lack.) For simplicity, I deal with the two functions as though they perform the same task.
A fork call creates a copy of the existing application as a new process and starts executing it. The new process has all the original's file descriptors and socket information. This can cause a problem if the application programmer didn't take into account the fact that two (or more) processes try to use the same socket (or file) simultaneously. Therefore, applications that can fork have to take into account potential conflicts and code around them by checking the status of shared sockets.

The operating system itself keeps a table of each socket and how many processes have access to it. An internal counter is incremented or decremented with each process's open or close function call for the socket. When the last process using a socket is terminated, the socket is permanently closed. This prevents one forked process from closing a socket when its original is still using it.

CONCLUSION

Internet Protocols are the standard, routable entrise networking protocols. All modern operating systems offer TCP/IP support and most large networks rely on TCP/IP for much of their network traffic. This is a technology for connecting dissimilar systems. Many standard connectivity utilities are available to access and transfer data between dissimilar systems, including File Transfer Protocol and Telnet. It provides a robust, scalable, cross-platform client/server framework. TCP/IP offers the socket interface, which is ideal for developing client/server applications that can run on Sockets-compliant stacks from other venders. Sockets applications can also advantage of other networking protocols such as NWLink used in Novell Net Ware networks. Internet Protocols provide a method of gaining access to the Internet. The internet consists of thousands of network wordwide connecting research facilities, universities, liberaries, government agencies and private companies.

REFERENCES

[1] James Chellis, Charles Perkings, Matthew Strebe, "Networking Essentials", SYBEX Publishers 1999.

[2] James Chellis, "Windows 2000 Network Infra Structure" SYBEX Publishers 2000

[3] Charles W. "Understanding TCP/IP" BPB Publishers 1996

http://www.us-epanorama.net

http://www.microsoft.com

http://www.commweb.com

http://www.oreily.com