CHAPTER 2

Applications and Layered Architectures

> ***architecture,*** *n. Any design or orderly arrangement perceived by man.*
> ***design,*** *n. The invention and disposition of the forms, parts, or details of something according to a plan.*[1]

Communication networks can be called upon to support an extremely wide range of services. We routinely use networks to talk to people, to send e-mail, to transfer files, and to retrieve information. Business and industry use networks to carry out critical functions, such as the transfer of funds and the automated processing of transactions, and to query or update database information. Increasingly, the Internet is also being used to provide "broadcast" services along the lines of traditional radio and television. It is clear then that the network must be designed so that it has the flexibility to provide support for current services and to accommodate future services. To achieve this flexibility, *an overall network architecture or plan is necessary*.

The overall process of enabling two or more devices to communicate effectively across a network is extremely complex. In Chapter 1 we identified the many elements of a network that are required to enable effective communication. Early network designers recognized the need to develop architectures that would provide a structure to organize these functions into a coherent form. As a result, in the early 1970s various computer companies developed proprietary network architectures. A common feature to all of these was the grouping of the communication functions into related and manageable sets called **layers.** We saw in Chapter 1 that communication functions can be grouped according to the following tasks:

- The transport across a network of data from a process in one machine to the process at another machine.

[1] Definitions are from *The American Heritage Dictionary of the English Language,* Houghton Mifflin Co., 1978.

- The routing and forwarding of packets across multiple hops in a network.
- The transfer of a frame of data from one physical interface to another.

These layers of functions build on top of each other to enable communications. We use the term **network architecture** to refer to a set of protocols that specify how every layer is to function.

The decomposition of the overall communications problem into a set of layers is a first step to simplifying the design of the overall network. In addition the interaction between layers needs to be defined precisely. This is done through the *definition of the service* provided by each layer to the layer above, and through the definition of the *interface* between layers through which a service is requested and through which results are conveyed. A clearly defined service and interface allows a layer to invoke a service from the layer below without regard to how the service is implemented by any of the layers below. As long as the service is provided as specified, the implementation of the underlying layers can be changed. Also, new services that build on existing services can be introduced at any time, and in turn enable other new services at layers above. This provides flexibility in modifying and evolving the network. In contrast, a monolithic network design that uses a single large body of hardware and software to meet all the network requirements can quickly become obsolete and also is extremely difficult and expensive to modify. The layered approach accommodates incremental changes much more readily.

In this chapter we develop the notion of a layered architecture, and we provide examples from TCP/IP, the most important current network architecture. The discussion is organized as follows:

1. Web-browsing and e-mail applications are used to demonstrate the operation of a protocol within a layer and how it makes use of the communication services of the layer below. We introduce the HTTP, DNS, and SMTP application layer protocols in these examples.
2. The Open Systems Interconnection (OSI) reference model is discussed to show how the overall communication process can be organized into functions that are carried out in seven layers.
3. The TCP/IP architecture is introduced and compared to the OSI reference model. We present a detailed end-to-end example in a typical TCP/IP Internet. We use a network protocol analyzer to show the exchange of messages and packets in real networks. This section is key to seeing the big picture because it shows how all the layers work together.

Two optional sections present material that is useful in developing lab exercises and experiments involving TCP/IP:

4. We introduce Berkeley sockets, which allow the student to write applications that use the services provided by the TCP/IP protocols. We develop example programs that show the use of UDP and TCP sockets.
5. We introduce several important TCP/IP application layer protocols: Telnet, FTP, and HTTP. We also introduce several utilities and a network protocol analyzer that can be used as tools to study the operation of the Internet.

2.1 EXAMPLES OF PROTOCOLS, SERVICES, AND LAYERING

A **protocol** is a set of rules that governs how two or more communicating parties are to interact. When dealing with networks we run into a multiplicity of protocols, such as HTTP, FTP, and TCP. The purpose of a protocol is to provide some type of communication service. For example, the HTTP protocol enables the retrieval of web pages, and the TCP protocol enables the reliable transfer of streams of information between computers. In this chapter, we will see that the overall communications process can be organized into a stack of layers. Each layer carries out a specific set of communication functions using its own protocol, and each layer builds on the services of the layer below it.

This section uses concrete examples to illustrate what is meant by a protocol and to show how two adjacent layers interact. Together the examples also show the advantages of layering. The examples use two familiar applications, namely, e-mail and Web browsing. We present a simplified discussion of the associated protocols. Our purpose here is to relate familiar applications to the underlying network services that are the focus of this textbook.

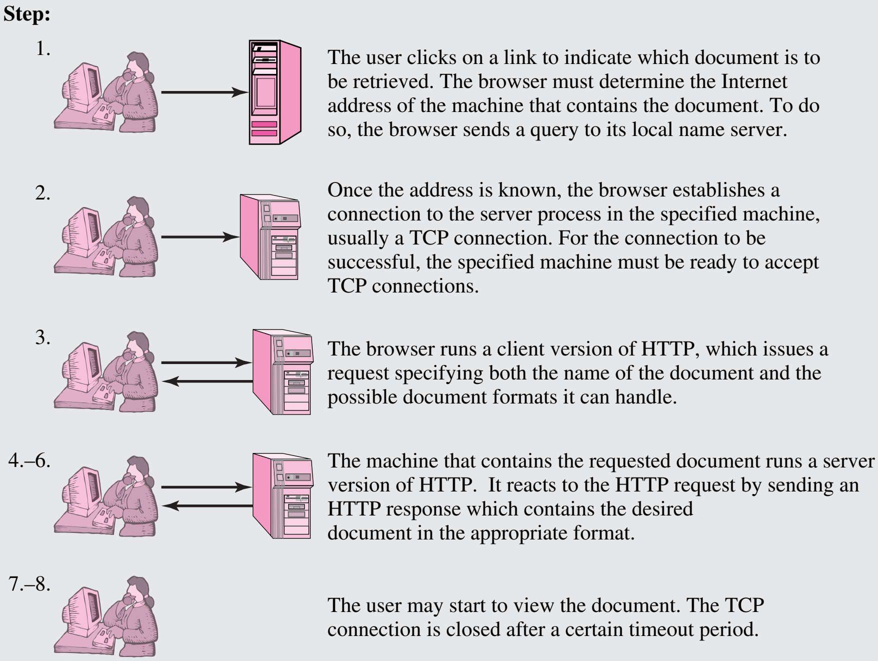
2.1.1 HTTP, DNS, and SMTP

All the examples discussed in this section involve a **client/server** application. A server process in a computer waits for incoming requests by listening to a **port.** A port is an address that identifies which process is to receive a message that is delivered to a given machine. Widely used applications have **well-known port numbers** assigned to their servers, so that client processes in other computers can readily make *requests* as required. The servers provide *responses* to those requests. The server software usually runs in the background and is referred to as a **daemon.** For example, `httpd` refers to the server daemon for HTTP.

EXAMPLE **HTTP and Web Browsing**

Let us consider an example of browsing through the World Wide Web (WWW). The WWW consists of a framework for accessing documents that are located in computers connected to the Internet. These documents are prepared using the *HyperText Markup Language (HTML)* and may consist of text, graphics, and other media and are interconnected by links that appear within the documents. The WWW is accessed through a browser program that displays the documents and allows the user to access other documents by clicking one of these links. Each link provides the browser with a uniform resource locator (URL) that specifies the name of the machine where the document is located as well as the name of the file that contains the requested document.

The *HyperText Transfer Protocol (HTTP)* specifies rules by which the client and server interact so as to retrieve a document. The rules also specify how the request and response are phrased. The protocol assumes that the client and server can exchange



Step:

1. The user clicks on a link to indicate which document is to be retrieved. The browser must determine the Internet address of the machine that contains the document. To do so, the browser sends a query to its local name server.

2. Once the address is known, the browser establishes a connection to the server process in the specified machine, usually a TCP connection. For the connection to be successful, the specified machine must be ready to accept TCP connections.

3. The browser runs a client version of HTTP, which issues a request specifying both the name of the document and the possible document formats it can handle.

4.–6. The machine that contains the requested document runs a server version of HTTP. It reacts to the HTTP request by sending an HTTP response which contains the desired document in the appropriate format.

7.–8. The user may start to view the document. The TCP connection is closed after a certain timeout period.

FIGURE 2.1 Retrieving a document from the web.

messages directly. In general, the client software needs to set up a two-way connection prior to the HTTP request.

Figure 2.1 and Table 2.1 show the sequence of events and messages that are involved in retrieving a document. In step 1 a user selects a document by clicking on its corresponding link. For example, the browser may extract the URL associated with the following link:

http://www.comm.utoronto.ca/comm.html

The client software must usually carry out a Domain Name System (DNS) query to determine the IP address corresponding to the host name, www.comm.utoronto.ca. (We discuss how this query is done in the next example.) The client software then sets up a TCP connection with the WWW server (the default is port 80) at the given IP address (step 2). The client end identifies itself by an **ephemeral port number** that is used only for the duration of the connection. The TCP protocol provides a reliable byte-stream transfer service that can be used to transmit files across the Internet.

After the connection is established, the client uses HTTP to request a document (step 3). The request message specifies the method or command (GET), the document (comm.html), and the protocol version that the browser is using (HTTP/1.1). The server daemon identifies the three components of the message and attempts to locate the file (step 4).

TABLE 2.1 Retrieving a document from the web: HTTP message exchange.

Event	Message Content
1. User selects document.	
2. Network software of client locates the server host and establishes a two-way connection.	
3. HTTP client sends message requesting document.	GET /comm.html HTTP/1.1
4. HTTP daemon listening on TCP port 80 interprets message.	
5. HTTP daemon sends a result code and a description of the information that the client will receive.	HTTP/1.1 200 OK Date: Mon, 06 Jan 2003 23:56:44 GMT Server: Apache/1.3.23 (Unix) Last Modified: 03 Sep 2002 02:58:36 GMT Content-Length: 8218 Content-Type: text/html
6. HTTP daemon reads the file and sends requested file through the TCP port.	<html> <head><title></title>... <font face="Arial" >What is Communications?</font>
7. Text is displayed by client browser, which interprets the HTML format.	
8. HTTP daemon disconnects the connection after the connection is idle for some timeout period.	

In step 5 the daemon sends a status line and a description of the information that it will send. Result code 200 indicates that the client request was successful and that the document is to follow. The message also contains information about the server software, the length of the document (8218 bytes), and the content type of the document (text/html). If the request was for an image, the type might be image/gif. If the request is not successful, the server sends a different result code, which usually indicates the type of failure, for example, 404 when a document is not found.

In step 6 the HTTP daemon sends the file over the TCP connection. In the meantime, the client receives the file and displays it (step 7). The server maintains the TCP connection open so it can accept additional requests from the client. The server closes the TCP connection if it remains idle for some timeout period (step 8).

The HTTP example clearly indicates that a protocol is solely concerned with the interaction between the two peer processes, that is, the client and the server. The protocol assumes that the message exchange between peer processes occurs directly as shown in Figure 2.2. Because the client and server machines are not usually connected directly, a connection needs to be set up between them. In the case of HTTP, we require a two-way connection that transfers a stream of bytes in correct sequential order and without errors. The TCP protocol provides this type of *communication service* between two processes in two machines connected to a network. Each HTTP process inserts its

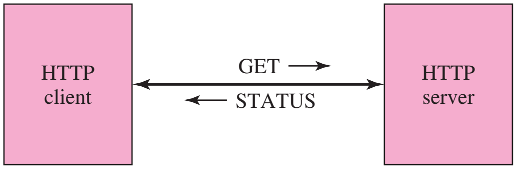


FIGURE 2.2 HTTP client/server interaction.

messages into a buffer, and TCP transmits the contents of the buffer to the other TCP in blocks of information called segments, as shown in Figure 2.3. Each segment contains port number information in addition to the HTTP message information. *HTTP is said to use the service provided by TCP in the layer below.* Thus the transfer of messages between HTTP client and server in fact is *virtual* and occurs *indirectly* via the TCP connection as shown in Figure 2.3. Later you will see that TCP, in turn, uses the service provided by IP.

It is worth noting exactly how the HTTP application protocol invokes the service provided by TCP. When the HTTP client software first needs to set up the TCP connection, the client does so by making a series of *socket system calls.* These calls are similar to function calls except that control is passed to the operating system kernel when a socket system call is made. A socket system call specifies a certain action and may contain parameters such as socket type, for example, TCP or UDP, and address information. Thus the interaction between the HTTP layer and the TCP layer takes place through these socket system calls.[2]

EXAMPLE **DNS Query**

The HTTP example notes that the client first needs to perform a DNS query to obtain the IP address corresponding to the domain name. This step is done by sending a

[2] Sockets are explained in detail in Section 2.4.

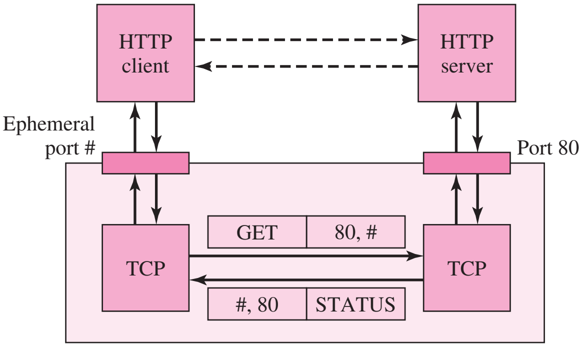


FIGURE 2.3 TCP provides a pipe between the HTTP client and HTTP server.

message to a DNS server. The **Domain Name System (DNS)** is a distributed database that resides in multiple machines on the Internet and is used to convert between names and addresses and to provide e-mail routing information. Each DNS machine maintains its own database and acts as a DNS server that other machines can query. Typically the requesting machine accesses a local name server, which, for example, may reside in a university department or at an ISP. These local name servers are able to resolve frequently used domain names into the corresponding IP addresses by caching recent information. When unable to resolve a name, the local name server may sometimes send a query to a root name server, of which there are currently 13 distributed globally. When a root server is unable to determine an IP address, it sends a query to an authoritative name server. Every machine on the Internet is required to register with at least two authoritative name servers. If a given name server cannot resolve the domain name, the queried name server will refer to another name server, and this process continues until a name server that can resolve the domain name is found.

We now consider a simple case where the resolution takes place in the first server. Table 2.2 shows the basic steps required for this example. After receiving the address request, a process in the host, called the resolver, composes the short message shown in step 2. The OPCODE value in the DNS message header indicates that the message is a standard query. The question portion of the query contains the following information: QNAME identifies the domain name that is to be translated. The DNS server can handle a variety of queries, and the type is specified by QTYPE. In the example, QTYPE = A requests a translation of a name to an IP address. QCLASS requests an Internet address (some name servers handle non-IP addresses). In step 3 the resolver sends the message to the local server using the datagram communication service UDP.

TABLE 2.2 DNS query and response.

Event	Message content
1. Application requests name to address translation.	

2. Resolver composes query message.	`Header: OPCODE=SQUERY`
	`Question:`
	`QNAME=tesla.comm.toronto.edu.,`
	`QCLASS=IN, QTYPE=A`

3. Resolver sends UDP datagram encapsulating the query message.	

4. DNS server looks up address and prepares response.	`Header: OPCODE=SQUERY,`
	`RESPONSE, AA`
	`Question: QNAME=`
	`tesla.comm.toronto.edu.,`
	`QCLASS=IN, QTYPE=A`
	`Answer: tesla.comm.toronto.edu.`
	`86400 IN A 128.100.11.1`

5. DNS sends UDP datagram encapsulating the response message.	

The short message returned by the server in step 4 has the Response and Authoritative Answer bits set in the header. This setting indicates that the response comes from an authority that manages the domain name. The question portion is identical to that of the query. The answer portion contains the domain name for which the address is provided. This portion is followed by the Time-to-Live field, which specifies the time in units of seconds that this information is to be cached by the client. Next are the two values for QCLASS and QTYPE. IN again indicates that it is an Internet address. Finally, the IP address of the domain name is given (128.100.11.1).

In this example the DNS query and response messages are transmitted by using the communication service provided by the **User Datagram Protocol (UDP).** The UDP client attaches a header to the user information to provide port information (port 53 for DNS) and encapsulates the resulting block in an IP packet. The UDP service is connectionless; no connection setup is required, and the datagram can be sent immediately. Because DNS queries and responses consist of short messages, UDP is ideally suited for conveying them.

The DNS example shows again how a protocol, in this case the DNS query protocol, is solely concerned with the interaction between the client and server processes. The example also shows how the transfer of messages between client and server, in fact, is virtual and occurs indirectly via UDP datagrams.

EXAMPLE SMTP and E-mail

Finally, we consider an e-mail example, using the **Simple Mail Transfer Protocol (SMTP).** Here a mail client application interacts with a local SMTP server to initiate the delivery of an e-mail message. The user prepares an e-mail message that includes the recipient's e-mail address, a subject line, and a body. When the user clicks Send, the mail application prepares a file with the above information and additional information specifying format, for example, plain ASCII or Multipurpose Internet Mail Extensions (MIME) to encode non-ASCII information. The mail application has the name of the local SMTP server and may issue a DNS query for the IP address. Table 2.3 shows the remaining steps involved in completing the transfer of the e-mail message to the local SMTP server.

Before the e-mail message can be transferred, the application process must set up a TCP connection to the local SMTP server (step 1). Thereafter, the SMTP protocol is used in a series of exchanges in which the client identifies itself, the sender of the e-mail, and the recipient (steps 2–8). The client then transfers the message that the SMTP server accepts for delivery (steps 9–12) and ends the mail session. The local SMTP server then repeats this process with the destination SMTP server. To locate the destination SMTP server, the local server may have to perform a DNS query of type MX (mail exchange). SMTP works best when the destination machine is always available. For this reason, users in a PC environment usually retrieve their e-mail from a mail server using the **Post Office Protocol version 3 (POP3)** instead.

TABLE 2.3 Sending e-mail.

Event	Message content
1. The mail application establishes a TCP connection (port 25) to its local SMTP server.	
2. SMTP daemon issues the following message to the client, indicating that it is ready to receive mail.	220 tesla.comm.toronto.edu ESMTP Sendmail 8.9.0/8.9.0; Thu, 2 Jul 1998 05:07:59 -0400 (EDT)
3. Client sends a HELO message and identifies itself.	HELO bhaskara.comm.utoronto.ca
4. SMTP daemon issues a 250 message, indicating the client may proceed.	250 tesla.comm.toronto.edu Hello bhaskara.comm [128.100.10.9], pleased to meet you
5. Client sends sender's address.	MAIL FROM: <banerjea@comm.utoronto.ca>
6. If successful, SMTP daemon replies with a 250 message.	250 <banerjea@comm.utoronto.ca> ... Sender ok
7. Client sends recipient's address.	RCPT TO: <alg@nal.utoronto.ca>
8. A 250 message is returned.	250 <alg@nal.utoronto.ca> ... Recipient ok
9. Client sends a DATA message requesting permission to send the mail message.	DATA
10. The daemon sends a message giving the client permission to send.	354 Enter mail, end with "." on a line by itself
11. Client sends the actual text.	Hi Al, This section on email sure needs a lot of work...
12. Daemon indicates that the message is accepted for delivery. A message ID is returned.	250 FAA00803 Message accepted for delivery
13. Client indicates that the mail session is over.	QUIT
14. Daemon confirms the end of the session.	221 tesla.comm.toronto.edu closing connection

2.1.2 TCP and UDP Transport Layer Services

The e-mail, DNS query, and HTTP examples show how multiple protocols can operate by using the communication services provided by the TCP and UDP protocols. Both the TCP and UDP protocols operate by using the connectionless packet network service provided by IP.

UDP provides connectionless transfer of datagrams between processes in hosts attached to the Internet. UDP provides port numbering to identify the source and destination processes in each host. UDP is simple and fast but provides no guarantees in terms of delivery or sequence addressing.

TCP provides for reliable transfer of a byte stream between processes in hosts attached to the Internet. The processes write bytes into a buffer for transfer across the Internet by TCP. TCP is considerably more complex than UDP. TCP involves the establishment of a connection between the two processes. To provide their service, the TCP entities implement error detection and retransmission as well as flow control algorithms (discussed in Chapters 5 and 8). In addition, TCP also implements congestion control, which regulates the flow of segments into the network. This topic is discussed in Chapters 7 and 8.

Indeed, an entire suite of protocols has been developed to operate on top of TCP and UDP, thereby demonstrating the usefulness of the layering concept. New services can be quickly developed by building on the services provided by existing layer protocols.

PEER-TO-PEER FILE SHARING

File-sharing applications such as Napster and Gnutella became extremely popular as a means of exchanging MP3 audio and other files. The essence of these peer-to-peer applications is that ordinary PCs ("peers") attached to the Internet can act not only as clients, but also as transient file servers while the applications are activated. When a peer is interested in finding a certain file, it sends a query. The response provides a list of peers that have the file and additional information such as the speed of each peer's connection to the Internet. The requesting peer can then set up a TCP connection to one of the peers in the list and proceed to retrieve the file.

The technically difficult part in peer-to-peer file sharing is maintaining the database of peers that are connected at a given point in time and the files that they have available for sharing. The Napster approach used a centralized database that peers could contact when they became available for file sharing and/or when they needed to make a query. The Gnutella approach uses a distributed approach where the peers organize themselves into an overlay network by keeping track of peers that are assigned to be adjacent to them. A query from a given peer is then broadcast by sending the query to each neighbor, their neighbors' neighbors, and so on up to some maximum number of hops.

Peer-to-peer file sharing provides another example of how new services and applications can be deployed very quickly over the Internet. Peer-to-peer file sharing also brings up many legal, commercial, and cultural issues that will require many years to resolve.

2.2 THE OSI REFERENCE MODEL

The early network architectures developed by various computer vendors were not compatible with each other. This situation had the effect of locking in customers with a single vendor. As a result, there was pressure in the 1970s for an open systems architecture that would eventually lead to the design of computer network equipment that could communicate with each other. This desire led to an effort in the International Organization for Standardization (ISO) first to develop a reference model for open systems interconnection (OSI) and later to develop associated standard protocols. *The OSI reference model partitioned the communications process into seven layers and* *provided a framework for talking about the overall communications process* and hence was intended to facilitate the development of standards. The OSI work also provided a unified view of layers, protocols and services. This unified view has provided the basis for the development of networking standards to the present day.

2.2.1 The Seven-Layer OSI Reference Model

Consider an application that involves communications between a process in computer A and a process in computer B. The **OSI reference model** divides the basic communication functions required for computers A and B to communicate into the seven layers shown in Figure 2.4. In this section, we will discuss the functions of the seven layers starting from the bottom (physical layer) to the top (application layer). The reader should compare the definition of the OSI layers to the elements described for the telegraph, telephone, and computer network architectures discussed in Chapter 1.

The **physical layer** deals with the transfer of *bits* over a communication channel, for example, the digital transmission system and the transmission media such as copper wire pairs, coaxial cable, radio, or optical fiber. The layer is concerned with the particular choice of system parameters such as voltage levels and signal durations. The layer is also concerned with the procedures to set up and release the physical connection, as well as with mechanical aspects such as socket type and number of pins. For example, an Ethernet physical layer standard defines the connector and signal interfaces in a PC.

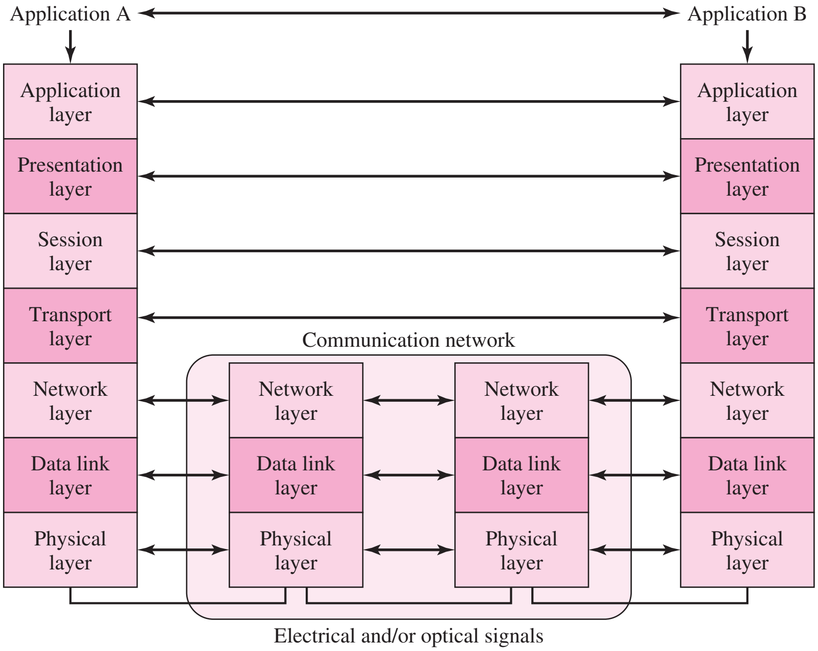


FIGURE 2.4 The seven-layer OSI reference model.

The **data link layer** provides for the transfer of **frames** (blocks of information) across a transmission link that *directly* connects two nodes. The data link layer inserts *framing* information in the sequence of transmitted bits to indicate the boundaries of the frames. It also inserts control and address information in the header and check bits to enable recovery from transmission errors, as well as flow control. The data link control is particularly important when the transmission link is prone to transmission errors. Historically, the data link layer has included the case where multiple terminals are connected to a host computer in point-to-multipoint fashion. In Chapter 5 we will discuss High-level Data Link Control (HDLC) protocol and Point-to-Point Protocol (PPP), which are two standard data link controls that are in wide use.

The OSI data link layer was defined so that it included the functions of *LANs,* which are characterized by the use of broadcast transmissions. The notion of a "link," then, includes the case where multiple nodes are connected to a broadcast medium. As before, frames flow directly between nodes. A *medium access control* procedure is required to coordinate the transmissions from the machines into the medium. A flat addressing space is used to enable machines to listen and recognize frames that are destined to them. Later in this chapter we will discuss the Ethernet LAN standard.

The **network layer** provides for the transfer of data in the form of **packets** across a communication *network.* One key aspect of the network layer is the use of a hierarchical addressing scheme that identifies the point of attachment to the network and that can accommodate a large number of network users. A key aspect of the packet transfer service is the routing of the packets from the source machine to the destination machine, typically traversing a number of transmission links and network nodes where routing is carried out. By *routing protocol* we mean the procedure that is used to select paths across a network. The nodes in the network must work together to perform the routing effectively. This function makes the network layer the most complex in the reference model. The network layer is also responsible for dealing with the *congestion* that occurs from time to time due to temporary surges in packet traffic.

When the two machines are connected to the same packet-switching network as in Figure 2.5, a single address space and routing procedure are used. However, when the two machines are connected to different networks, the transfer of data must traverse two or more networks that possibly differ in their internal routing and addressing

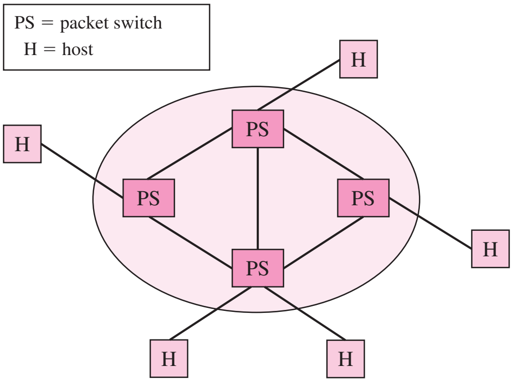


FIGURE 2.5 A packet-switching network using a uniform routing procedure.

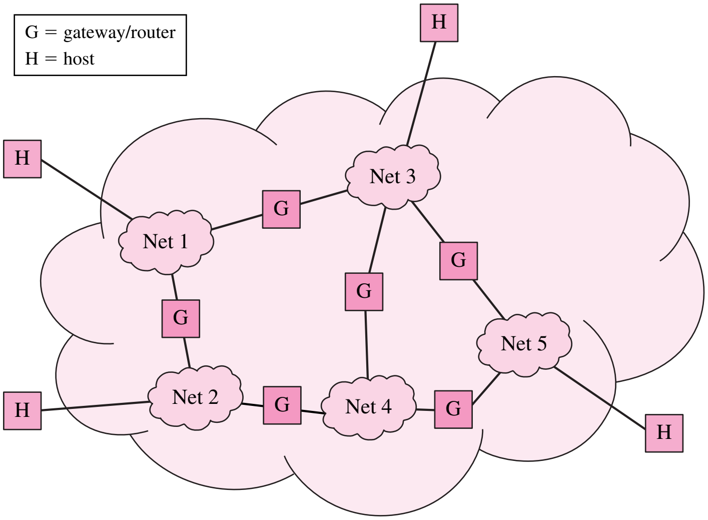


FIGURE 2.6 An internetwork.

scheme. In this case **internetworking** protocols are necessary to route the data between gateways/routers that connect the intermediate networks, as shown in Figure 2.6. The internetworking protocols must also deal with differences in addressing and differences in the size of the packets that are handled within each network. This *internet sublayer* of the network layer assumes the responsibility for hiding the details of the underlying network(s) from the upper layers. This function is particularly important given the large and increasing number of available network technologies for accomplishing packet transfer.

As shown in Figure 2.4, each intermediate node in the network must implement the lower three layers. Thus one pair of network layer entities exists for each hop of the path required through the network. Note that the network layer entities in the source and destination machines are *not* peer processes, that is, if there are intermediate nodes between them, they do not talk directly to each other.

The **transport layer** is responsible for the end-to-end transfer of messages from a process in the source machine to a process in the destination machine. The transport layer protocol accepts messages from its higher layers and prepares blocks of information called **segments** or datagrams for transfer between end machines. The transport layer uses the services offered by the underlying network or internetwork to provide the session layer with a transfer of messages that meets a certain quality of service. The transport layer can provide a variety of services. At one extreme the transport layer may provide a connection-oriented service that involves the error-free transfer of a sequence of bytes or messages. The associated protocol carries out error detection and recovery, and sequence and flow control. At the other extreme the transport layer may instead provide an unconfirmed connectionless service that involves the transfer of individual messages. In this case the role of the transport layer is to provide the appropriate address information so that the messages can be delivered to the appropriate destination process. The transport layer may be called upon to segment messages that are too long into shorter blocks of information for transfer across a network and to reassemble these messages at the destination.

In TCP/IP networks, processes typically access the transport layer through **socket** interfaces that are identified by a port number. We discuss the socket interface in the Berkeley UNIX application programming interface (API) in an optional section later in this chapter.

The transport layer can be responsible for setting up and releasing connections across the network. To optimize the use of network services, the transport layer may multiplex several transport layer connections onto a single network layer connection. On the other hand, to meet the requirements of a high throughput transport layer connection, the transport layer may use splitting to support its connection over several network layer connections.

Note from Figure 2.4 that the top four layers are end to end and involve the interaction of peer processes across the network. In contrast the lower two layers of the OSI reference model involve interaction of peer-to-peer processes across a single hop.

The **session layer** can be used to control the manner in which data are exchanged. For example, certain applications require a half-duplex dialog where the two parties take turns transmitting information. Other applications require the introduction of synchronization points that can be used to mark the progress of an interaction and can serve as points from which error recovery can be initiated. For example, this type of service may be useful in the transfer of very long files over connections that have short times between failures.

The **presentation layer** is intended to provide the application layer with independence from differences in the representation of data. In principle, the presentation layer should first convert the machine-dependent information provided by application A into a machine-independent form, and later convert the machine-independent form into a machine-dependent form suitable for application B. For example, different computers use different codes for representing characters and integers, and also different conventions as to whether the first or last bit is the most significant bit.

Finally, the purpose of the **application layer** is to provide services that are frequently required by applications that involve communications. In the WWW example the browser application uses the HTTP application-layer protocol to access a WWW document. Application layer protocols have been developed for file transfer, virtual terminal (remote log-in), electronic mail, name service, network management, and other applications.

In general each layer adds a header, and possibly a trailer, to the block of information it accepts from the layer above. Figure 2.7 shows the headers and trailers that are added as a block of application data works its way down the seven layers. At the destination each layer reads its corresponding header to determine what action to take and it eventually passes the block of information to the layer above after removing the header and trailer.

In addition to defining a reference model, an objective of the ISO activity was the development of *standards* for computer networks. This objective entailed *specifying the particular protocols* that were to be used in various layers of the OSI reference model. However, in the time that it took to develop the OSI protocol standards, the *TCP/IP network architecture* emerged as an alternative for open systems interconnection. The free distribution of TCP/IP as part of the Berkeley UNIX® ensured the development of

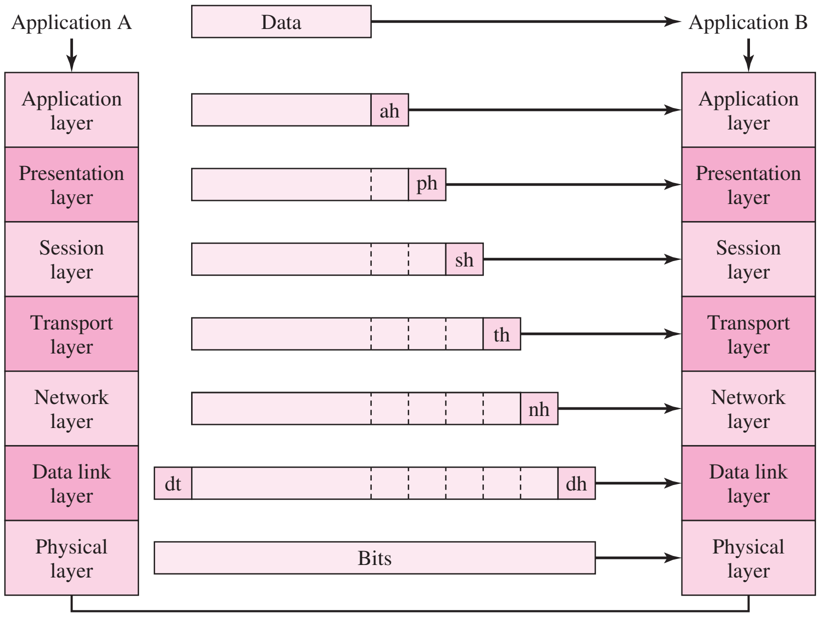


FIGURE 2.7 Headers and trailers are added to a block of data as it moves down the layers.

numerous applications at various academic institutions and the emergence of a market for networking software. This situation eventually led to the development of the global Internet and to the dominance of the TCP/IP network architecture.

2.2.2 Unified View of Layers, Protocols, and Services

A lasting contribution from the development of the OSI reference model was the development of a unified view of layers, protocols, and services. Similar requirements occur at different layers in a network architecture, for example, in terms of addressing, multiplexing, and error and flow control. This unified view enables a common understanding of the protocols that are found in different layers. In each layer a process on one machine carries out a conversation with a **peer process** on the other machine across a *peer interface,* as shown in Figure 2.8.[3] In OSI terminology the processes at layer n are referred to as **layer n entities.** Layer n entities communicate by exchanging **protocol data units (PDUs).** Each PDU contains a **header,** which contains protocol control information, and usually user information. The behavior of the layer n entities is governed by a set of rules or conventions called the **layer n protocol.** In the HTTP example the HTTP client and server applications acted as peer processes. The processes that carry

[3] Peer-to-peer protocols are present in every layer of a network architecture. In Chapter 5 we present detailed examples of peer-to-peer protocols.

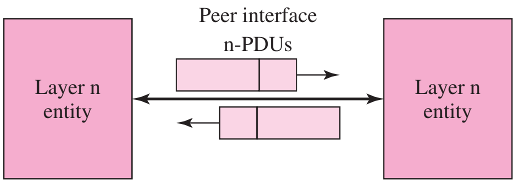


FIGURE 2.8 Peer-to-peer communication.

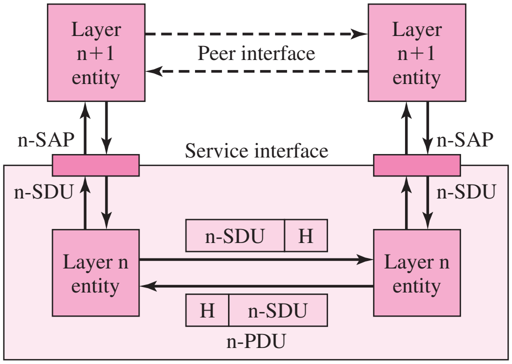


FIGURE 2.9 Layer services: SDUs are exchanged between layers while PDUs are exchanged within a layer.

out the transmitter and receiver functions of TCP also constitute peer processes at the layer below.

The communication between peer processes is usually virtual in the sense that no direct communication link exists between them. *For communication to take place, the layer n + 1 entities make use of the services provided by layer n.* The transmission of the layer n + 1 PDU is accomplished by passing a block of information from layer n + 1 to layer n through a software port called the layer n **service access point (SAP)** across a *service interface,* as shown in Figure 2.9. Each SAP is identified by a unique identifier (for example, recall that a WWW server process passes information to a TCP process through a Transport-SAP or port number 80). The block of information passed between layer n and layer n + 1 entities consists of control information and a layer n **service data unit (SDU),** which is the layer n + 1 PDU itself. The layer n entity uses the control information to form a header that is attached to the SDU to produce the layer n PDU. Upon receiving the layer n PDU, the layer n peer process uses the header to execute the layer n protocol and, if appropriate, to deliver the SDU to the corresponding layer n + 1 entity. The communication process is completed when the SDU (layer n + 1 PDU) is passed to the layer n + 1 peer process.[4]

In principle, the layer n protocol does not interpret or make use of the information contained in the SDU.[5] We say that the layer n SDU, which is the layer n + 1 PDU, is *encapsulated* in the layer n PDU. This process of **encapsulation** narrows the scope of the dependencies between adjacent layers to the service definition only. In other

[4] It may be instructive to reread this paragraph where a DNS query message constitutes the layer n + 1 PDU and a UDP datagram constitutes the layer n PDU.

[5] On the other hand, accessing some of the information "hidden" inside the SDU can sometimes be useful.

words, *layer n + 1,* as a user of the service provided by layer n, *is only interested in the correct execution of the service required to transfer its PDUs. The details of the implementation of the layers below layer n + 1 are irrelevant.*

The service provided by layer n typically involves accepting a block of information from layer n + 1, transferring the information to its peer process, which in turn delivers the block to the user at layer n + 1. The service provided by a layer can be connection oriented or connectionless. A **connection-oriented service** has three phases.

1. Establishing a connection between two layer n SAPs. The setup involves negotiating connection parameters as well as initializing "state information" such as the sequence numbers, flow control variables, and buffer allocations.
2. Transferring n-SDUs using the layer n protocol.
3. Tearing down the connection and releasing the various resources allocated to the connection.

In the HTTP example in Section 2.1, the HTTP client process uses the connection services provided by TCP to transfer the HTTP PDU, which consists of the request message. A TCP connection is set up between the HTTP client and server processes, and the TCP transmitter/receiver entities carry out the TCP protocol to provide a reliable message stream service for the exchange of HTTP PDUs. The TCP connection is later released after one or more HTTP responses have been received.

Connectionless service does not require a connection setup, and each SDU is transmitted directly through the SAP. In this case the control information that is passed from layer n + 1 to layer n must contain all the address information required to transfer the SDU. In the DNS example in Section 2.1, UDP provides a connectionless service for the exchange of DNS PDUs. No connection is established between the DNS client and server processes.

In general, it is not necessary for the layers to operate in the same connection mode. Thus for example, TCP provides a connection-oriented service but builds on the connectionless service provided by IP.

The services provided by a layer can be **confirmed** or **unconfirmed** depending on whether the sender must eventually be informed of the outcome. For example, connection setup is usually a confirmed service. Note that a connectionless service can be confirmed or unconfirmed depending on whether the sending entity needs to receive an acknowledgment.

Information exchanged between entities can range from a few bytes to multi-megabyte blocks or continuous byte streams. Many transmission systems impose a limit on the maximum number of bytes that can be transmitted as a unit. For example, Ethernet LANs have a maximum transmission size of approximately 1500 bytes. Consequently, when the number of bytes that needs to be transmitted exceeds the maximum transmission size of a given layer, it is necessary to divide the bytes into appropriate-sized blocks.

In Figure 2.10a a layer n SDU is too large to be handled by the layer n − 1, and so **segmentation** and **reassembly** are applied. The layer n SDU is *segmented* into multiple layer n PDUs that are then transmitted using the services of layer n − 1. The layer n entity at the other side must *reassemble* the original layer n SDU from the sequence of layer n PDUs it receives.

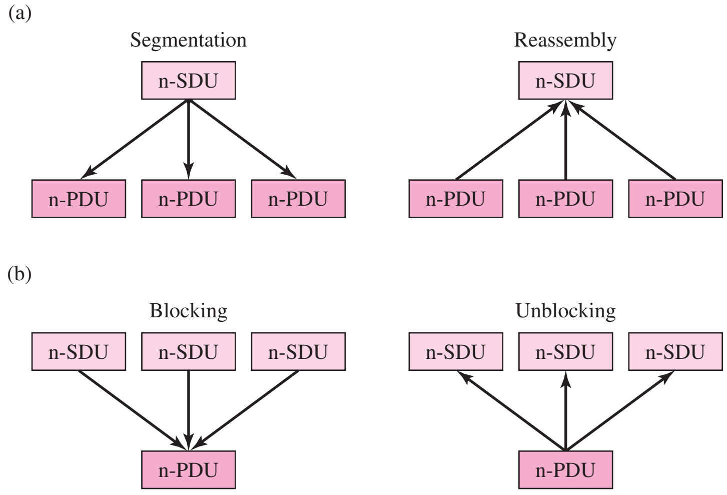


FIGURE 2.10 Segmentation/reassembly and blocking/unblocking.

On the other hand, it is also possible that the layer n SDUs are so small as to result in inefficient use of the layer n − 1 services, and so **blocking** and **unblocking** may be applied. In this case, the layer n entity may *block* several layer n SDUs into a single layer n PDU as shown in Figure 2.10b. The layer n entity on the other side must then *unblock* the received PDU into the individual SDUs.

Multiplexing involves the sharing of a layer n service by *multiple* layer n + 1 users. Figure 2.11 shows the case where each layer n + 1 user passes its SDUs for transfer using the service of a single layer n entity. **Demultiplexing** is carried out by the layer n entity at the other end. When the layer n PDUs arrive at the other end of the connection, the SDUs are recovered and must then be delivered to the appropriate layer n + 1 user. Clearly a *multiplexing tag* is needed in each PDU to determine which user an SDU belongs to. As an example consider the case where several application layer processes share the

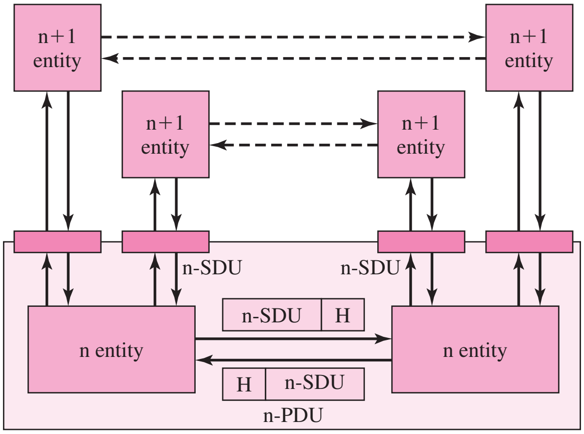


FIGURE 2.11 Multiplexing involves sharing of layer n service by multiple layer n + 1 users.

datagram services of UDP. Each application layer process passes its SDU through its socket to the UDP entity. UDP prepares a datagram that includes the source port number, the destination port number, as well as the IP address of the source and destination machines. The server-side port number is a *well-known port number* that unambiguously identifies the process that is to receive the SDU at the server end. The client-side port number is an ephemeral number that is selected when the socket for the application is established. Demultiplexing can then be carried out unambiguously at each UDP entity by directing an arriving SDU to the port number indicated in the datagram.

Splitting involves the use of several layer n services to support a single layer n + 1 user. The SDUs from the single user are directed to one of several layer n entities, which in turn transfer the given SDU to a peer entity at the destination end. **Recombining** takes place at the destination where the SDUs recovered from each of the layer n entities are passed to the layer n + 1 user. Sequence numbers may be required to reorder the received SDUs.

Multiplexing is used to achieve more efficient use of communications services. Multiplexing is also necessary when only a single connection is available between two points. Splitting can be used to increase reliability in situations where the underlying transfer mechanism is unreliable. Splitting is also useful when the transfer rate required by the user is greater than the transfer rate available from individual services.

In closing we re-iterate: Similar needs occur at different layers and these can be met by a common set of services such as those introduced here.

2.3 OVERVIEW OF TCP/IP ARCHITECTURE

The TCP/IP network architecture is a set of protocols that allows communication across multiple diverse networks. The architecture evolved out of research that had the original objective of transferring packets across three different packet networks: the ARPANET packet-switching network, a packet radio network, and a packet satellite network. The military orientation of the research placed a premium on robustness with regard to failures in the network and on flexibility in operating over diverse networks. This environment led to a set of protocols that are highly effective in enabling communications among the many different types of computer systems and networks. Indeed, the Internet has become the primary fabric for interconnecting the world's computers. In this section we introduce the TCP/IP network architecture. The details of specific protocols that constitute the TCP/IP network architecture are discussed in later chapters.

2.3.1 TCP/IP Architecture

Figure 2.12a shows the **TCP/IP network architecture,** which consists of four layers. The application layer provides services that can be used by other applications. For example, protocols have been developed for remote login, for e-mail, for file transfer, and for network management. The TCP/IP application layer incorporates the functions of the

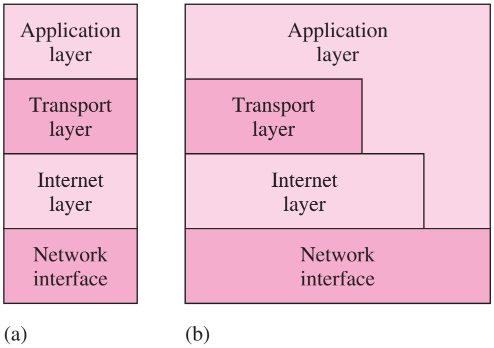


FIGURE 2.12 TCP/IP network architecture.

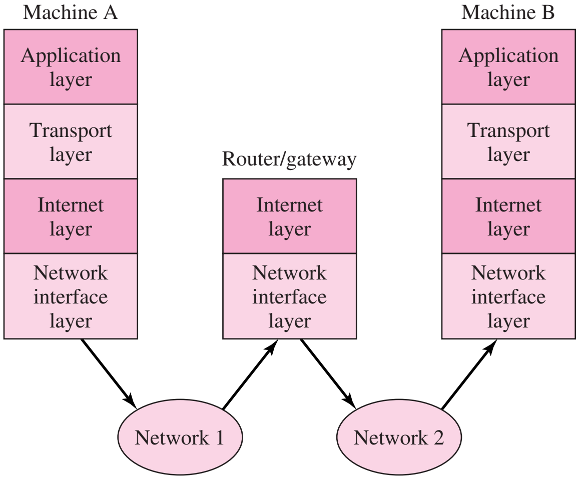


FIGURE 2.13 The internet layer and network interface layers.

top three OSI layers. The HTTP protocol discussed in Section 2.1 is actually a TCP/IP application layer protocol. Recall that the HTTP request message included format information and the HTTP protocol defined the dialogue between the client and server.

The TCP/IP application layer programs are intended to run directly over the transport layer. Two basic types of services are offered in the transport layer. The first service consists of reliable connection-oriented transfer of a byte stream, which is provided by the *Transmission Control Protocol (TCP).* The second service consists of best-effort connectionless transfer of individual messages, which is provided by the *User Datagram Protocol (UDP).* This service provides no mechanisms for error recovery or flow control. UDP is used for applications that require quick but not necessarily reliable delivery.

The TCP/IP model does not require strict layering, as shown in Figure 2.12b. In other words, the application layer has the option of bypassing intermediate layers. For example, an application layer may run directly over the internet layer.

The **internet layer** handles the transfer of information across multiple networks through the use of gateways/routers, as shown in Figure 2.13. The internet layer corresponds to the part of the OSI network layer that is concerned with the transfer of packets between machines that are connected to different networks. It must therefore deal with the routing of packets from router to router across these networks. A key aspect of routing in the internet layer is the definition of *globally unique addresses* for machines that are attached to the Internet. The internet layer provides a single service, namely, *best-effort connectionless packet transfer.* IP packets are exchanged between routers without a connection setup; the packets are routed independently, and so they may traverse different paths. For this reason, IP packets are also called **datagrams.** The connectionless approach makes the system robust; that is, if failures occur in the network, the packets are routed around the points of failure; there is no need to set up the connections again. The gateways that interconnect the intermediate networks may discard packets when congestion occurs. The responsibility for recovery from these losses is passed on to the transport layer.

Finally, the **network interface layer** is concerned with the network-specific aspects of the transfer of packets. As such, it must deal with part of the OSI network layer and data link layer. Various interfaces are available for connecting end computer systems to specific networks such as ATM, frame relay, Ethernet, and token ring. These networks are described in later chapters.

The network interface layer is particularly concerned with the protocols that access the intermediate networks. At each gateway the network access protocol encapsulates the IP packet into a packet or frame of the underlying network or link. The IP packet is recovered at the exit gateway of the given network. This gateway must then encapsulate the IP packet into a packet or frame of the type of the next network or link. This approach provides a clear separation of the internet layer from the technology-dependent network interface layer. This approach also allows the internet layer to provide a data transfer service that is transparent in the sense of not depending on the details of the underlying networks. The next section provides a detailed example of how IP operates over the underlying networks.

Figure 2.14 shows some of the protocols of the TCP/IP protocol suite. The figure shows two of the many protocols that operate over TCP, namely, HTTP and SMTP.

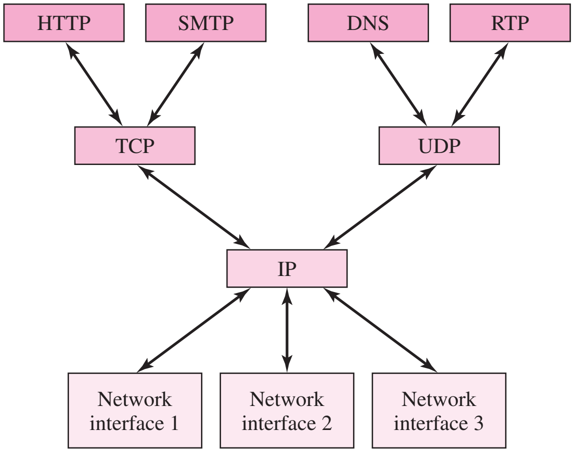


FIGURE 2.14 TCP/IP protocol graph.

The figure also shows DNS and Real-Time Protocol (RTP), which operate over UDP. The transport layer protocols TCP and UDP, on the other hand, operate over IP. Many network interfaces are defined to support IP. The salient part of Figure 2.14 is that all higher-layer protocols access the network interfaces through IP. This feature provides the capability to operate over multiple networks. The IP protocol is complemented by additional protocols (ICMP, IGMP, ARP, RARP) that are required to operate an internet. These protocols are discussed in Chapter 8.

The hourglass shape of the TCP/IP protocol graph underscores the features that make TCP/IP so powerful. The operation of the single IP protocol over various networks provides independence from the underlying network technologies. The communication services of TCP and UDP provide a network-independent platform on which applications can be developed. By allowing multiple network technologies to coexist, the Internet is able to provide ubiquitous connectivity and to achieve enormous economies of scale.

2.3.2 TCP/IP Protocol: How the Layers Work Together

We now provide a detailed example of how the layering concepts discussed in the previous sections are put into practice in a typical TCP/IP network scenario. We show

- Examples of each of the layers.
- How the layers interact across the interfaces between them.
- How the PDUs of a layer are built and what key information is in the header.
- The relationship between physical addresses and IP addresses.
- How an IP packet or datagram is routed across several networks.

We first consider a simplified example, and then we present an example showing PDUs captured in a live network by a network protocol analyzer. These examples will complete our goal of providing the big picture of networking. In the remainder of the book we systematically examine the details of the various components and aspects of networks.

Consider the network configuration shown in Figure 2.15a. A server, a workstation, and a router are connected to an Ethernet LAN, and a remote PC is connected to the router through a point-to-point link. From the point of view of IP, the Ethernet LAN and the point-to-point link constitute two different networks as shown in Figure 2.15b.

IP ADDRESSES AND PHYSICAL ADDRESSES

Each host in the Internet is identified by a **globally unique IP address.** Strictly speaking, the IP address identifies the host's network interface rather than the host itself. A node that is attached to two or more physical networks is called a router. In this example the router attaches to two networks with each network interface assigned to a unique IP address. An IP address is divided into two parts: a *network id* and a *host id*. The network id must be obtained from an organization authorized to issue IP addresses. In this example we use simplified notation and assume that the Ethernet has net id 1 and that the point-to-point link has a net id 2. In the Ethernet we suppose that the server has IP address (1,1), the workstation has IP address (1,2), and the router has address (1,3). In the point-to-point link, the PC has address (2,2), and the router has address (2,1).

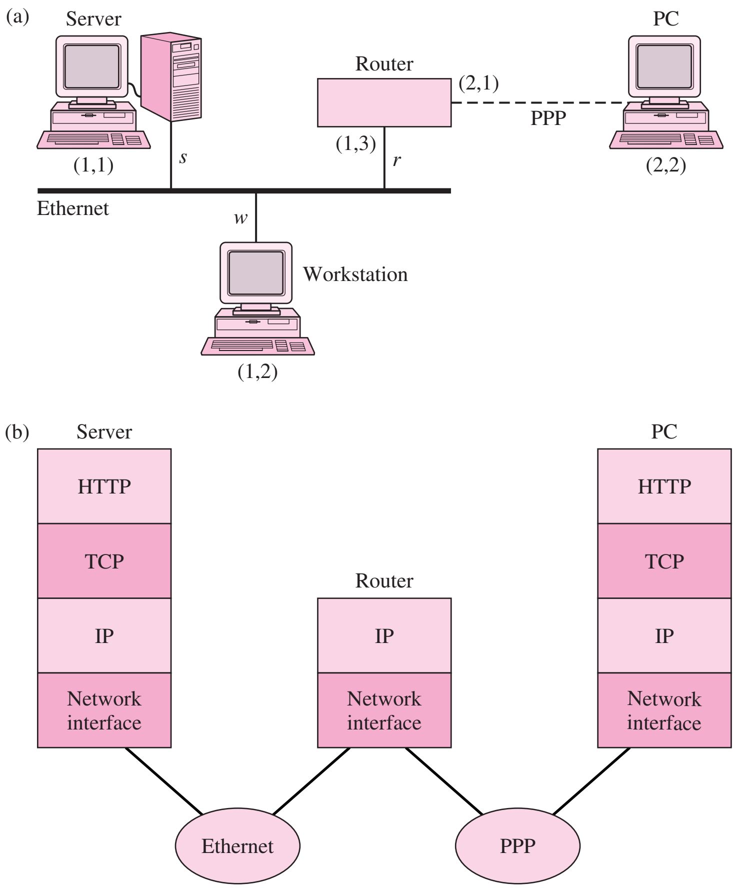


FIGURE 2.15 An example of an internet consisting of an Ethernet LAN and a point-to-point link: (a) physical configuration view and (b) IP network view.

On a LAN the attachment of a device to the network is often identified by a **physical address.** The format of the physical address depends on the particular type of network. For example, Ethernet LANs use 48-bit addresses. Each Ethernet network interface card (NIC) is issued a globally unique medium access control (MAC) or physical address. When a NIC is used to connect a machine to any Ethernet LAN, all machines in the LAN are automatically guaranteed to have unique addresses. Thus the router, server, and workstation also have physical addresses designated by r, s, and w, respectively.

SENDING AND RECEIVING IP DATAGRAMS

First, let us consider the case in which the workstation wants to send an IP datagram to the server. The IP datagram has the workstation's IP address and the server's IP address in the IP packet header. We suppose that the IP address of the server is known. The IP entity in the workstation looks at its routing table to see whether it has an entry for the

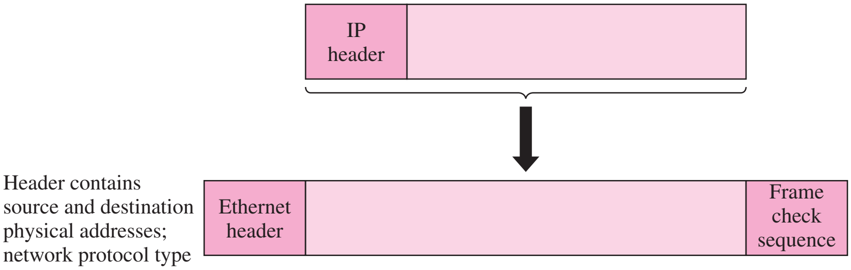

IP header
Header contains source and destination physical addresses; network protocol type
Ethernet header
Frame check sequence

FIGURE 2.16 IP datagram is encapsulated in an Ethernet frame.

complete IP address. It finds that the server is directly connected to the same network and that the server has physical address s.[6] The IP datagram is passed to the Ethernet device driver, which prepares an Ethernet frame as shown in Figure 2.16. The header in the frame contains the source physical address, w, and the destination physical address, s. The header also contains a protocol type field that is set to the value that corresponds to IP. The type field is required because the Ethernet may be carrying packets for other non-IP protocols. The Ethernet frame is then broadcast over the LAN. The server's NIC recognizes that the frame is intended for its host, so the card captures the frame and examines it. The NIC finds that the protocol type field is set to IP and therefore passes the IP datagram up to the IP entity.

Next let us consider the case in which the server wants to send an IP datagram to the personal computer. The PC is connected to the router through a point-to-point link that we assume is running PPP as the data link control.[7] We suppose that the server knows the IP address of the PC and that the IP addresses on either side of the link were negotiated when the link was set up. The IP entity in the server looks at its routing table to see whether it has an entry for the complete IP address of the PC. We suppose that it doesn't. The IP entity then checks to see whether it has a routing table entry that matches the network id portion of the IP portion of the IP address of the PC. Again we suppose that the IP entity does not find such an entry. The IP entity then checks to see whether it has an entry that specifies a default router that is to be used when no other entries are found. We suppose that such an entry exists and that it specifies the router with address (1,3).

The IP datagram is passed to the Ethernet device driver, which prepares an Ethernet frame. The header in the frame contains the source physical address, s, and the destination physical address, r. However, the IP datagram in the frame contains the destination IP address of the PC, (2,2), not the destination IP address of the router. The Ethernet frame is then broadcast over the LAN. The router's NIC captures the frame and examines it. The card passes the IP datagram up to its IP entity, which discovers that the IP datagram is not for itself but is to be routed on.

[6] If the IP entity does not know the physical address corresponding to the IP address of the server, the entity uses the Address Resolution Protocol (ARP) to find it. ARP is discussed in Chapter 8.
[7] PPP is discussed in Chapter 5.

The routing tables at the router show that the machine with address (2,2) is connected directly on the other side of the point-to-point link. The router encapsulates the IP datagram in a PPP frame that is similar to that of the Ethernet frame shown in Figure 2.16. However, the frame does not require physical address information, since there is only one "other side" of the link. The PPP receiver at the PC receives the frame, checks the protocol type field, and passes the IP datagram to its IP entity.

HOW THE LAYERS WORK TOGETHER

The preceding discussion shows how IP datagrams are sent across an internet. Next let's complete the picture by seeing how things work at the higher layers. Consider the browser application discussed in the beginning of the chapter. We suppose that the user at the PC has clicked on a web link of a document contained in the server and that a TCP connection has already been established between the PC and the server.[8] Consider what happens when the TCP connection is confirmed at the PC. The HTTP request message GET is passed to the TCP layer, which encapsulates the message into a TCP segment as shown in Figure 2.17. The TCP segment contains an ephemeral port number for the client process, say, *c*, and a well-known port number for the server process, 80 for HTTP.

The TCP segment is passed to the IP layer, which in turn encapsulates the segment into an Internet packet. The IP packet header contains the IP addresses of the sender, (2,2), and the destination, (1,1). The header also contains a protocol field, which designates the layer that is operating above IP, in this case TCP. The IP datagram is then encapsulated using PPP and sent to the router, which routes the datagram to the server using the procedures discussed above. Note that the router encapsulates the IP datagram for the server in an Ethernet frame.

Eventually the server NIC captures the Ethernet frame and extracts the IP datagram and passes it to the IP entity. The protocol field in the IP header indicates that a TCP segment is to be extracted and passed on to the TCP layer. The TCP layer, in turn, uses the port number to find out that the message is to be passed to the HTTP server process. A problem arises at this point: The server process is likely to be simultaneously handling multiple connections to multiple clients. All these connections have the same destination IP address; the same destination port number, 80; and the same protocol type, TCP. How does the server know which connection the message corresponds to? The answer is in how an end-to-end process-to-process connection is specified.

The source port number, the source IP address, and the protocol type are said to define the sender's **socket address.** Similarly, the destination port number, the destination IP address, and the protocol type define the destination's socket address. Together the source socket address and the destination socket address uniquely specify the connection between the HTTP client process and the HTTP server process. For example, in the earlier HTTP example the sender's socket is (TCP, (2,2), *c*), and the destination's socket is (TCP, (1,1), 80). The combination of these five parameters (TCP, (2,2), *c*, (1,1), 80) uniquely specify the process-to-process connection.

[8] The details of how a TCP connection is set up are described in Chapter 8.

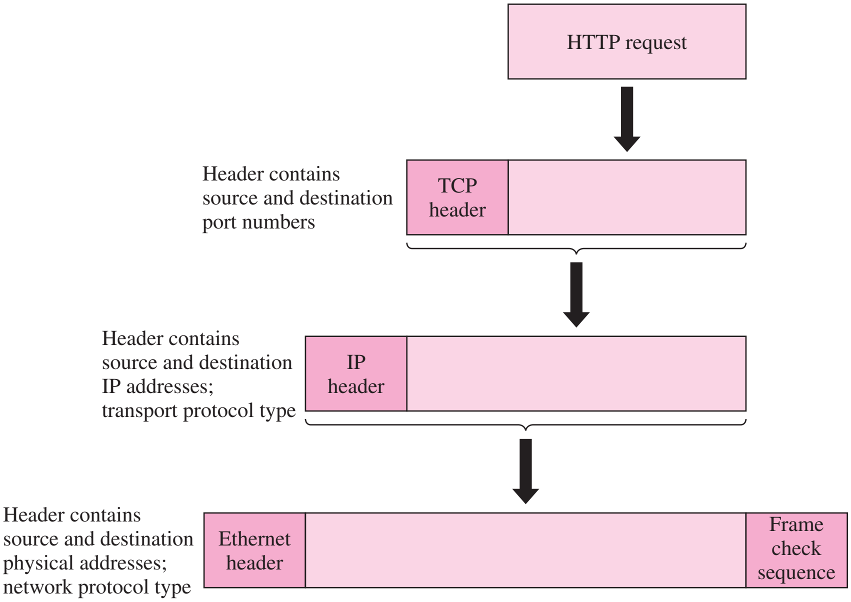

HTTP request
Header contains source and destination port numbers
TCP header
Header contains source and destination IP addresses; transport protocol type
IP header
Header contains source and destination physical addresses; network protocol type
Ethernet header
Frame check sequence

FIGURE 2.17 Encapsulation of PDUs in TCP/IP and addressing information in the headers. (Ethernet header is replaced with PPP header on a PPP link.)

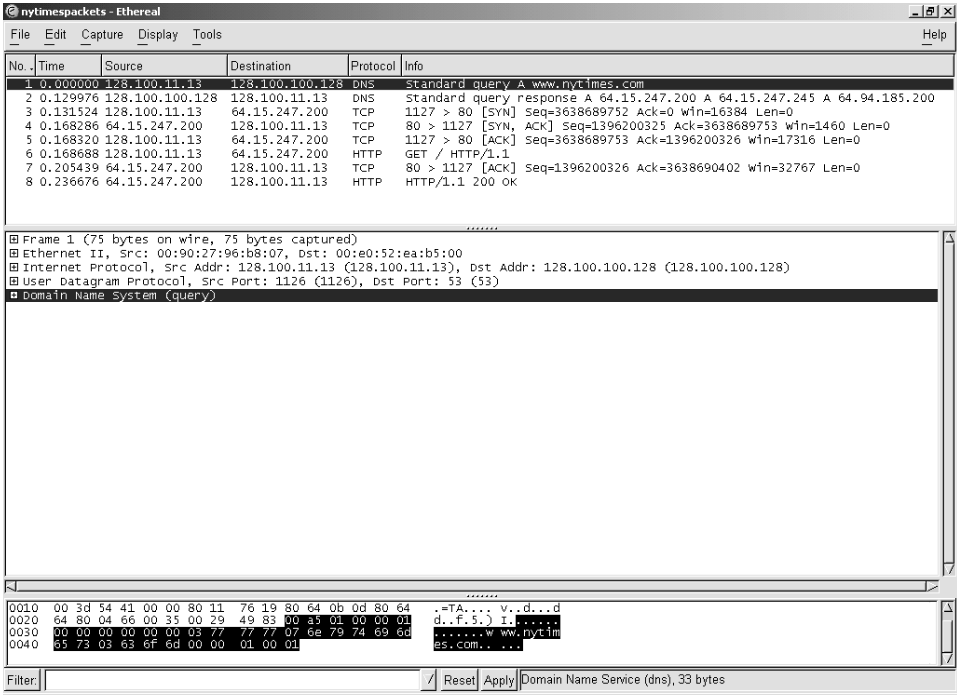
VIEWING THE LAYERS USING A NETWORK PROTOCOL ANALYZER

A network protocol analyzer is a tool that can capture, display, and analyze the PDUs that are exchanged between peer processes. Protocol analyzers are extremely useful in troubleshooting network problems and also as an educational tool. Network protocol analyzers are discussed in the last section of this chapter. In our examples we will use the *Ethereal* open source package. In this section we use a sequence of captured packets to show how the layers work together in a simple web interaction.

Figure 2.18 shows the Ethereal display after capturing packets that are transmitted after clicking on the URL of the *New York Times*. The top pane in the display shows the first eight packets that are transmitted during the interaction:

1. The first packet carries a DNS query from the machine with IP address 128.100.100.13 for the IP address of www.nytimes.com. It can be seen from the first packet that the IP address of the local DNS server is 128.100.100.128. The second packet carries the DNS response that provides three IP addresses, 64.15.347.200, 64.15.347.245, and 64.94.185.200 for the URL.
2. The next three packets correspond to the *three-way handshake* that is used to establish a TCP connection between the client and the server.[9] In the first packet the client (with IP address 128.100.100.128 and port address 1127) makes a TCP connection setup request by sending an IP packet to 64.15.347.200 and well-known port number 80. In the first packet, the client also includes an initial sequence number to keep

[9] TCP is discussed in detail in Chapter 8.



```
No. Time     Source          Destination     Protocol Info
  1 0.000000 128.100.11.13   128.100.100.128 DNS      Standard query A www.nytimes.com
  2 0.129976 128.100.100.128 128.100.11.13   DNS      Standard query response A 64.15.247.200 A 64.15.247.245 A 64.94.185.200
  3 0.131524 128.100.11.13   64.15.247.200   TCP      1127 > 80 [SYN] Seq=3638689752 Ack=0 Win=16384 Len=0
  4 0.168286 64.15.247.200   128.100.11.13   TCP      80 > 1127 [SYN, ACK] Seq=1396200325 Ack=3638689753 Win=1460 Len=0
  5 0.168320 128.100.11.13   64.15.247.200   TCP      1127 > 80 [ACK] Seq=3638689753 Ack=1396200326 Win=17316 Len=0
  6 0.168688 128.100.11.13   64.15.247.200   HTTP     GET / HTTP/1.1
  7 0.205439 64.15.247.200   128.100.11.13   TCP      80 > 1127 [ACK] Seq=1396200326 Ack=3638690402 Win=32767 Len=0
  8 0.236676 64.15.247.200   128.100.11.13   HTTP     HTTP/1.1 200 OK
```

```
⊞ Frame 1 (75 bytes on wire, 75 bytes captured)
⊞ Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
⊞ Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 128.100.100.128 (128.100.100.128)
⊞ User Datagram Protocol, Src Port: 1126 (1126), Dst Port: 53 (53)
⊞ Domain Name System (query)
```

```
0010  00 3d 54 41 00 00 80 11  76 19 80 64 0b 0d 80 64   .=TA.... v..d...d
0020  64 80 04 66 00 35 00 29  49 83 00 a5 01 00 00 01   d..f.5.) I.......
0030  00 00 00 00 00 00 03 77  77 77 07 6e 79 74 69 6d   .......w ww.nytim
0040  65 73 03 63 6f 6d 00 00  01 00 01                  es.com.. ...
```

Domain Name Service (dns), 33 bytes

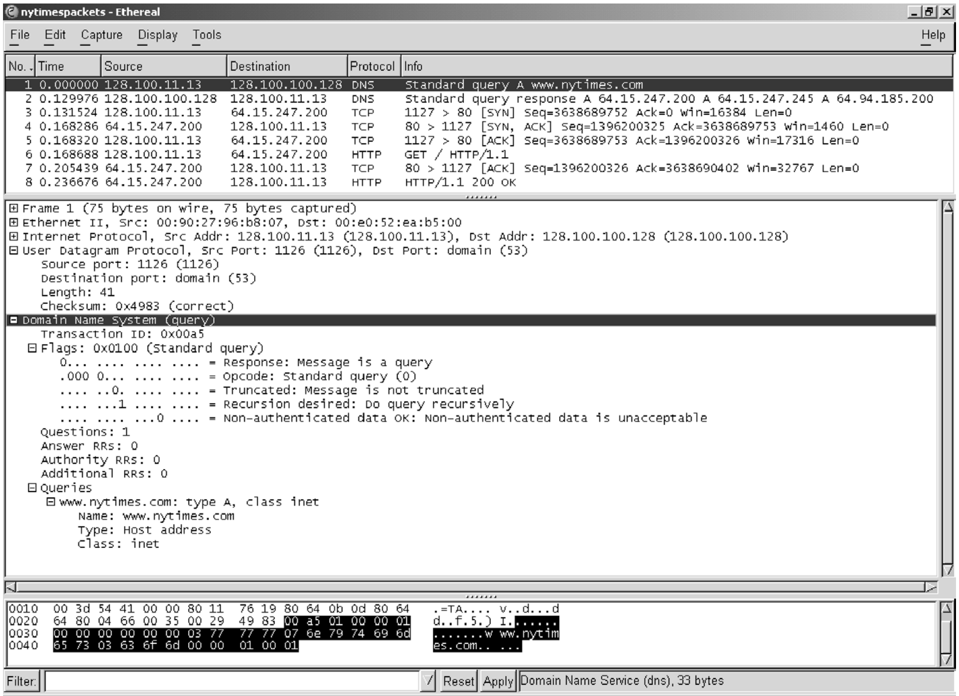
FIGURE 2.18 Viewing packet exchanges and protocol layers using Ethereal; the display has three panes (from top to bottom): packet capture list, details of selected packet, and data from the selected packet.

count of the bytes it transmits. In the second packet the server acknowledges the connection request and proposes its own initial sequence number. With the third packet, the client confirms the TCP connection setup and the initial sequence numbers. The TCP connection is now ready.

3. The sixth packet carries the HTTP "GET" request from the client to the server.
4. The seventh packet carries an acknowledgment message that is part of the TCP protocol.
5. The final packet carries the HTTP status response from the server. The response code 200 confirms that the request was successful and that the document will follow.

The process of encapsulation (see Figure 2.17) means that a given captured frame carries information from multiple layers. Figure 2.18 illustrates this point quite clearly. The middle pane displays information about the highlighted packet (a DNS query) of the top pane. By looking down the list in the middle pane, one can see the protocol stack that the DNS query traversed, UDP over IP over Ethernet.

Figure 2.19 provides more information about the same DNS packet (obtained by clicking on the "+" to the left of the desired entry in the middle pane). In the figure the UDP and DNS entries have been expanded. In the UDP entry, it can be seen that the first packet carries source port number 1126 and well-known destination port number 53. The DNS entry shows the contents of the DNS query. Finally the third pane in Figure 2.19



```
No. . Time     Source          Destination     Protocol Info
  1 0.000000 128.100.11.13   128.100.100.128 DNS      Standard query A www.nytimes.com
  2 0.129976 128.100.100.128 128.100.11.13   DNS      Standard query response A 64.15.247.200 A 64.15.247.245 A 64.94.185.200
  3 0.131524 128.100.11.13   64.15.247.200   TCP      1127 > 80 [SYN] Seq=3638689752 Ack=0 Win=16384 Len=0
  4 0.168286 64.15.247.200   128.100.11.13   TCP      80 > 1127 [SYN, ACK] Seq=1396200325 Ack=3638689753 Win=1460 Len=0
  5 0.168320 128.100.11.13   64.15.247.200   TCP      1127 > 80 [ACK] Seq=3638689753 Ack=1396200326 Win=17316 Len=0
  6 0.168688 128.100.11.13   64.15.247.200   HTTP     GET / HTTP/1.1
  7 0.205439 64.15.247.200   128.100.11.13   TCP      80 > 1127 [ACK] Seq=1396200326 Ack=3638690402 Win=32767 Len=0
  8 0.236676 64.15.247.200   128.100.11.13   HTTP     HTTP/1.1 200 OK
```

```
⊞ Frame 1 (75 bytes on wire, 75 bytes captured)
⊞ Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
⊞ Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 128.100.100.128 (128.100.100.128)
⊟ User Datagram Protocol, Src Port: 1126 (1126), Dst Port: domain (53)
    Source port: 1126 (1126)
    Destination port: domain (53)
    Length: 41
    Checksum: 0x4983 (correct)
⊟ Domain Name System (query)
    Transaction ID: 0x00a5
  ⊟ Flags: 0x0100 (Standard query)
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... ...0 .... = Non-authenticated data OK: Non-authenticated data is unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ www.nytimes.com: type A, class inet
        Name: www.nytimes.com
        Type: Host address
        Class: inet
```

```
0010  00 3d 54 41 00 00 80 11  76 19 80 64 0b 0d 80 64   .=TA.... v..d...d
0020  64 80 04 66 00 35 00 29  49 83 00 a5 01 00 00 01   d..f.5.) I.......
0030  00 00 00 00 00 00 03 77  77 77 07 6e 79 74 69 6d   .......w ww.nytim
0040  65 73 03 63 6f 6d 00 00  01 00 01                  es.com.. ...
```

Domain Name Service (dns), 33 bytes

FIGURE 2.19 More detailed protocol layer information for selected captured packet.

shows the actual raw data of the captured packet. The highlighted data in the third pane corresponds to the fields that are highlighted in the middle pane. Thus the highlighted area in the figure contains the data relating to the DNS PDU. From the third line in the middle pane we can see the source and destination IP addresses of the IP datagram that carries the given UDP packet. The second line shows the Ethernet physical addresses of this frame that carries the given UDP packet. By expanding the IP and Ethernet entries we can obtain the details of the IP datagram and the Ethernet frame. We will explore the details of these and other protocols in the remainder of the book.

2.3.3 Protocol Overview

This completes the discussion on how the different layers work together in a TCP/IP Internet. In the remaining chapters we examine the details of the operation of the various layers. In Chapters 3 and 4 we consider various aspects of physical layers. In Chapter 5 we discuss peer-to-peer protocols that allow protocols such as TCP to provide reliable service. We also discuss data link control protocols. In Chapter 6 we discuss LANs and their medium access controls. In Chapter 7 we return to the network layer and examine the operation of routers and packet switches as well as issues relating to addressing, routing, and congestion control. Chapter 8 presents a detailed discussion of the TCP and IP protocols. In Chapter 9 we introduce ATM, a connection-oriented packet network architecture. In Chapter 10 we discuss advanced topics, such as connection-oriented IP networks realized through MPLS, new developments in TCP/IP architecture and the support of real-time multimedia services over IP. In Chapter 11 we introduce enhancements to IP that provide security. From time to time it may be worthwhile to return to this example to place the discussion of details in the subsequent chapters into the big picture presented here.

◆ 2.4 THE BERKELEY API[10]

An Application Programming Interface (API) allows application programs (such as Telnet, web browsers, etc.) to access certain resources through a predefined and preferably consistent interface. One of the most popular of the APIs that provide access to network resources is the Berkeley socket interface, which was developed by a group at the University of California at Berkeley in the early 1980s. The socket interface is now widely available on many UNIX machines. Another popular socket interface, which was derived from the Berkeley socket interface, is called the Windows sockets or Winsock and was designed to operate in a Microsoft® Windows environment.

By hiding the details of the underlying communication technologies as much as possible, the socket mechanism allows programmers to write application programs easily without worrying about the underlying networking details. Figure 2.20 shows how two applications talk to each other across a communication network through the socket interface. In a typical communication session, one application operates as a server and the other as a client. The server is the provider of a particular service while the client is the consumer. A server waits passively most of the time until a client requires a service.

This section explains how the socket mechanism can provide services to the applications. Two modes of services are available through the socket interface: *connection-oriented* and *connectionless.* With the connection-oriented mode, an application must first establish a connection to the other end before the actual communication (i.e., data transfer) can take place. The connection is established if the other end agrees to accept the connection. Once the connection is established, data will be delivered through the connection to the destination in sequence. The connection-oriented mode provides a reliable delivery service. With the connectionless mode an application sends its data immediately without waiting for the connection to get established at all. This mode avoids the setup overhead found in the connection-oriented mode. However, the price to pay is that an application may waste its time sending data when the other end is not ready to accept it. Moreover, data may not arrive at the other end if the network decides to discard it. Worse yet, even if data arrives at the destination, it may not arrive in the same order as it was transmitted. The connectionless mode is said to provide *best-effort service,* since the network would try its best to deliver the information but cannot guarantee delivery.

Figure 2.21 shows a typical diagram of the sequence of socket calls for the connection-oriented mode. The server begins by carrying out a *passive open* as

[10] This section is optional and is not required for later sections. A knowledge of C programming is assumed.

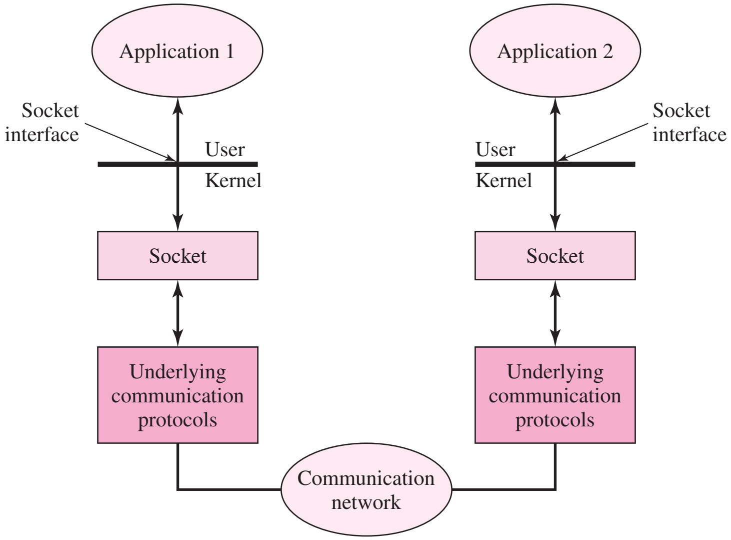


FIGURE 2.20 Communications through the socket interface.

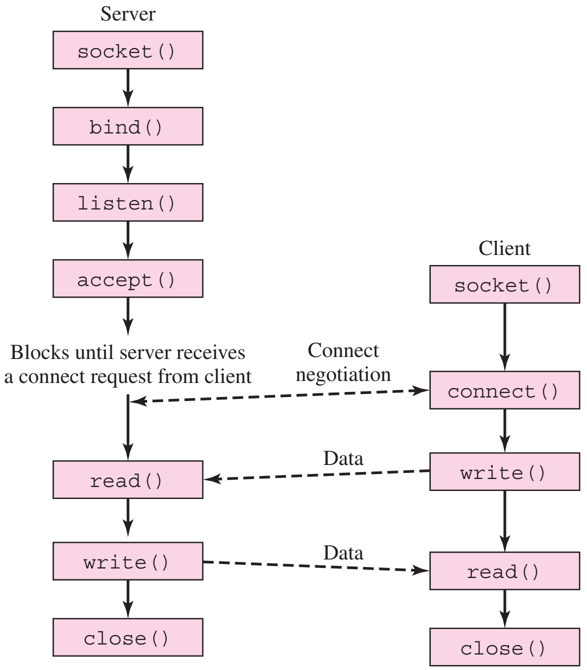


FIGURE 2.21 Socket calls for connection-oriented mode.

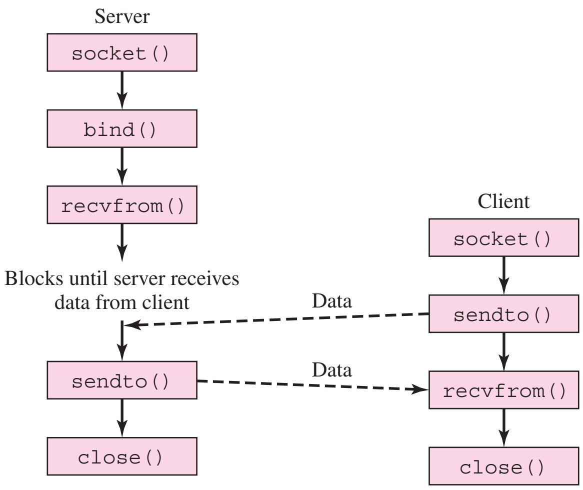


FIGURE 2.22 Socket calls for connectionless mode.

follows. The `socket` call creates a TCP socket. The `bind` call then binds the well-known port number of the server to the socket. The `listen` call turns the socket into a listening socket that can accept incoming connections from clients. Finally, the `accept` call puts the server process to sleep until the arrival of a client connection request. The client does an *active open*. The `socket` call creates a socket on the client side, and the `connect` call attempts to establish the TCP connection to the server with the specified destination socket address. When the TCP connection is established, the `accept` function at the server wakes up and returns the descriptor for the given connection, namely, the source IP address, source port number, destination IP address, and destination port number. The client and server are now ready to exchange information.

Figure 2.22 shows the sequence of socket calls for the connectionless mode. Note that no connection is established prior to data transfer. The `recvfrom` call returns when a complete UDP datagram has been received. For both types of communication, the data transfer phase may occur in an arbitrary number of exchanges.

2.4.1 Socket System Calls

Socket facilities are provided to programmers through C system calls that are similar to function calls except that control is transferred to the operating system kernel once a call is entered. To use these facilities, the header files `<sys/types.h>` and `<sys/socket.h>` must be included in the program.

CREATING A SOCKET

Before an application program (client or server) can transfer any data, it must first create an endpoint for communication by calling `socket`. Its prototype is

```
int socket(int family, int type, int protocol);
```

where family identifies the family by address or protocol. The address family identifies a collection of protocols with the same address format, while the protocol family identifies a collection of protocols having the same architecture. Although it may be possible to classify the family based on addresses or protocols, these two families are currently equivalent. Some examples of the address family that are defined in <sys/socket.h> include AF_UNIX, which is used for communication on the local UNIX machine, and AF_INET, which is used for Internet communication using TCP/IP protocols. The protocol family is identified by the prefix PF_. The value of PF_XXX is equal to that of AF_XXX, indicating that the two families are equivalent. We are concerned only with AF_INET in this book.

The type identifies the semantics of communication. Some of the types include SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW. A SOCK_STREAM type provides data delivery service as a sequence of bytes and does not preserve message boundaries. A SOCK_DGRAM type provides data delivery service in blocks of bytes called datagrams. A SOCK_RAW type provides access to internal network interfaces and is available only to superuser.

The protocol identifies the specific protocol to be used. Normally, only one protocol is available for each family and type, so the value for the protocol argument is usually set to 0 to indicate the default protocol. The default protocol of SOCK_STREAM type with AF_INET family is TCP, which is a connection-oriented protocol providing a reliable service with in-sequence data delivery. The default protocol of SOCK_DGRAM type with AF_INET family is UDP, which is a connectionless protocol with unreliable service.

The socket call returns a nonnegative integer value called the socket descriptor or handle (just like a file descriptor) on success. On failure, socket returns −1.

ASSIGNING AN ADDRESS TO THE SOCKET

After a socket is created, the bind system call can be used to assign an address to the socket. Its prototype is

```
int bind(int sd, struct sockaddr *name, int namelen);
```

where sd is the socket descriptor returned by the socket call, name is a pointer to an address structure that contains the local IP address and port number, and namelen is the size of the address structure in bytes. The bind system call returns 0 on success and −1 on failure. The sockaddr structure is a generic address structure and has the following definition:

```
struct sockaddr {
        u_short  sa_family;            /* address family  */
        char     sa_data[14];          /* address   */
};
```

where sa_family holds the address family and sa_data holds up to 14 bytes of address information that varies from one family to another. For the Internet family the address information consists of the port number that is two bytes long and an IP address that is four bytes long. The appropriate structures to use for the Internet family are defined in <netinet/in.h>:

```
struct in addr {
        u_long    s_addr;                /* 32-bit IP address */
};
struct sockaddr_in {
        u_short    sin_family;           /* AF_INET */
        u_short    sin_port;             /* TCP or UDP port */
        struct     in_addr sin_addr;     /* 32-bit IP address */
        char       sin_zero[8];          /* unused */
};
```

An application program using the Internet family should use the sockaddr_in structure to assign member values and should use the sockaddr structure only for casting purposes in function arguments. For this family sin_family holds the value of the identifier AF_INET. The structure member sin_port holds the local port number. Port numbers 1 to 1023 are normally reserved for system use. For a server, sin_port contains a well-known port number that clients must know in advance to establish a connection. Specifying a port number 0 to bind asks the system to assign an available port number. The structure member sin_addr holds the local IP address. For a host with multiple IP addresses, sin_addr is typically set to INADDR_ANY to indicate that the server is willing to accept communication through any of its IP addresses. This setting is useful for a host with multiple IP addresses. The structure member sin_zero is used to fill out struct sockaddr_in to 16 bytes.

Different computers may store a multibyte word in different orders. If the least significant byte is stored first (has lower address), it is known as *little endian*. If the most significant byte is stored first, it is known as *big endian*. For any two computers to be able to communicate, they must agree on a common data format while transferring multibyte words. The Internet adopts the big-endian format. This representation is known as *network byte order* in contrast to the representation adopted by the host, which is called *host byte order*. It is important to remember that the values of sin_port and sin_addr must be in the network byte order, since these values are communicated across the network. Four functions are available to convert between the host and network byte order conveniently. Functions htons and htonl convert an unsigned short and an unsigned long, respectively, from the host to network byte order. Functions ntohs and ntohl convert an unsigned short and an unsigned long, respectively, from the network to host byte order. We need to use these functions so that programs will be portable to any machine. To use these functions, we should include the header files <sys/types.h> and <netinet/in.h>. The appropriate prototypes are

```
u_long htonl(u_long hostlong);
u_short htons(u_short hostshort);
u_long ntohl(u_long netlong);
u_short ntohs(u_short netshort);
```

ESTABLISHING AND ACCEPTING CONNECTIONS

A client establishes a connection on a socket by calling connect. The prototype is

```
int connect(int sd, struct sockaddr *name, int namelen);
```

where sd is the socket descriptor returned by the socket call, name points to the server address structure, and namelen specifies the amount of space in bytes pointed to by name. For the connection-oriented mode, connect attempts to establish a connection between a client and a server. For the connectionless mode, connect stores the server's address so that the client can use a mode socket descriptor when sending datagrams, instead of specifying the server's address each time a datagram is sent. The connect system call returns 0 on success and −1 on failure.

A connection-oriented server indicates its willingness to receive connection requests by calling listen. The prototype is

```
int listen(int sd, int backlog);
```

where sd is the socket descriptor returned by the socket call and backlog specifies the maximum number of connection requests that the system should queue while it waits for the server to accept them (the maximum value is usually 5). This mechanism allows pending connection requests to be saved while the server is busy processing other tasks. The listen system call returns 0 on success and −1 on failure.

After a server calls listen, it can accept the connection request by calling accept with the prototype

```
int accept(int sd, struct sockaddr *addr, int *addrlen);
```

where sd is the socket descriptor returned by the socket call, addr is a pointer to an address structure that accept fills in with the client's IP address and port number, and addrlen is a pointer to an integer specifying the amount of space pointed to by addr before the call. On return, the value pointed to by addrlen specifies the number of bytes of the client address information.

If no connection requests are pending, accept will block the caller until a connection request arrives. The accept system call returns a new socket descriptor having nonnegative value on success and −1 on failure. The new socket descriptor inherits the properties of sd. The server uses the new socket descriptor to perform data transfer for the new connection. While data transfer occurs on an existing connection, a *concurrent* server can accept further connection requests using the original socket descriptor sd, allowing multiple clients to be served simultaneously.

TRANSMITTING AND RECEIVING DATA

Clients and servers may transmit data using write or sendto. The write call is usually used for the connection-oriented mode. However, a connectionless client may also call write if it has a connected socket (that is, the client has executed connect). On the other hand, the sendto call is usually used for the connectionless mode. Their prototypes are

```
int write(int sd, char *buf, int buflen);
int sendto(int sd, char *buf, int buflen, int flags,
   struct sockaddr *addrp, int addrlen);
```

where sd is the socket descriptor, buf is a pointer to a buffer containing the data to transmitted, buflen is the length of the data in bytes, flags can be used to control transmission behavior such as handling out-of-band (high priority) data but is usually set to 0 for normal operation, addrp is a pointer to the sockaddr structure containing the address information of the remote hosts, and addrlen is the length of the address information. Both write and sendto return the number of bytes transmitted on success or −1 on failure.

The corresponding system calls to receive data read and recvfrom. Their prototypes are

```
int read(int sd, char *buf, int buflen);
int recvfrom(int sd, char * buf, int buflen, int flags,
   struct sockaddr *addrp, int *addrlen);
```

The parameters are similar to the ones discussed above except buf is now a pointer to a buffer that is used to store the received data and buflen is the length of the buffer in bytes. Both read and recvfrom return the number of bytes received on success or −1 on failure. Both calls will block if no data arrives at the local host.

CLOSING A CONNECTION

If a socket is no longer in use, the application can call close to terminate a connection and return system resources to the operating system. The prototype is

```
int close(int sd);
```

where sd is the socket descriptor to be closed. The close call returns 0 on success and −1 on failure.

2.4.2 Network Utility Functions

Library routines are available to convert a human-friendly domain name such as tesla.comm.utoronto.ca into a 32-bit machine-friendly IP as 10000000 01100100 00001011 00000001 and vice versa. To perform the conversion we should include the header files <sys/socket.h>, <sys/types.h>, and <netdb.h>. The appropriate structure that stores the host information defined in the <netdb.h> file is

```
struct hostent {
    char *h_name;              /* official name of host */
    char **h_aliases;          /* alias name this host uses */
    int h_addrtype;            /* address type */
    int h_length;              /* length of address */
    char **h_addr_list;        /* list of addresses from name
                                  server */
};
```

The h_name element points to the official name of the host. If the host has name aliases, these aliases are pointed to by h_aliases, which is terminated by a NULL. Thus h_aliases[0] points to the first alias, h_aliases[1] points to the second alias, and so on. Currently, the h_addrtype element always takes on the value of AF_INET, and the h_length element always contains a value of 4. The h_addr_list points to the list of network addresses in network byte order and is terminated by a NULL.

NAME-TO-ADDRESS CONVERSION FUNCTIONS

Two functions are used for routines performing a name-to-address-conversion: gethostbyname and gethostbyaddr.

```
struct hostent *gethostbyname (char *name);
```

The function gethostbyname takes a domain name at the input and returns the host information as a pointer to struct hostent. The function returns a NULL on error. The parameter name is a pointer to a domain name of a host whose information we would like to obtain. The function gethostbyname obtains the host information either from the file /etc/hosts or from a name server. Recall that the host information includes the desired address.

```
struct hostent *gethostbyaddr (char *addr, int len, int type);
```

The function gethostbyaddr takes a host address at the input in network byte order, its length in bytes, and type, which should be AF_INET. The function returns the same information as gethostbyname. This information includes the desired host name.

The IP address is usually communicated by people using a notation called the *dotted-decimal notation.* As an example, the dotted-decimal notation of the IP address 10000000 01100100 00001011 00000001 is 128.100.11.1. To convert between these two formats, we could use the functions inet_addr and inet_ntoa. The header files that must be included are <sys/types.h>, <sys/socket.h>, <netinet/in.h>, and <arpa/inet.h>.

IP ADDRESS MANIPULATION FUNCTIONS

Two functions are used for routines converting addresses between a 32-bit format and the dotted-decimal notation: inet_nota and inet_addr.

```
char *inet_ntoa(struct in_addr in);
```

The function inet_ntoa takes a 32-bit IP address in network byte order and returns the corresponding address in dotted-decimal notation.

```
unsigned long inet_addr(char *cp);
```

The function inet_addr takes a host address in dotted-decimal notation and returns the corresponding 32-bit IP address in network byte order.

EXAMPLE Communicating with TCP

As an illustration of the use of the system calls and functions described previously, let us show two application programs that communicate via TCP. The client prompts a user to type a line of text, sends it to the server, reads the data back from the server, and prints it out. The server acts as a simple echo server. After responding to a client, the server closes the connection and then waits for the next new connection. In this example each application (client and server) expects a fixed number of bytes from the other end, specified by `BUFLEN`. Because TCP is stream oriented, the received data may come in multiple pieces of byte streams independent of how the data was sent at the other end. For example, when a transmitter sends 100 bytes of data in a single `write` call, the receiver may receive the data in two pieces—80 bytes and 20 bytes—or in three pieces—10 bytes, 50 bytes, and 40 bytes—or in any other combination. Thus the program has to make repeated calls to `read` until all the data has been received. The following program is the server.

```
/* A simple echo server using TCP */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_TCP_PORT   3000       /* well-known port */
#define BUFLEN         256           /* buffer length */

int main(int argc, char **argv)
{
    int      n, bytes_to_read;
    int      sd, new_sd, client_len, port;
    struct   sockaddr_in server, client;
    char     *bp, buf[BUFLEN];

    switch(argc) {
    case 1:
        port = SERVER_TCP_PORT;
        break;
    case 2:
        port = atoi(argv[1]);
        break;
    default:
        fprintf(stderr, "Usage: %s [port]\n", argv[0]);
        exit(1);
    }

    /* Create a stream socket */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        fprintf(stderr, "Can't create a socket\n");
        exit(1);
    }
    /* Bind an address to the socket */
    bzero((char *)&server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind(sd, (struct sockaddr *)&server,
    sizeof(server)) == -1) {
      fprintf(stderr, "Can’t bind name to socket\n");
      exit(1);
    }

    /* queue up to 5 connect requests */
    listen(sd, 5);

     while (1) {
      client_len = sizeof(client);
      if ((new_sd = accept(sd, (struct sockaddr *)
      &client, &client_len)) == -1) {
        fprintf(stderr, "Can’t accept client\n");
        exit(1);
      }

      bp = buf;
        bytes_to_read = BUFLEN;
        while ((n = read(new_sd, bp, bytes_to_read)) > 0) {
          bp += n;
          bytes_to_read -= n;
        }

        write(new_sd, buf, BUFLEN);
        close(new_sd);
      }
      close(sd);
      return(0);
}
```

The client program allows the user to identify the server by its domain name. Conversion to the IP address is done by the `gethostbyname` function. Again, the client makes repeated calls to `read` until no more data is expected to arrive. The following program is the client.

```
/* A simple TCP client */
#include <stdio.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_TCP_PORT   3000
#define BUFLEN         256     /* buffer length */
```

```
int main(int argc, char **argv)
{
   int      n, bytes_to_read;
   int      sd, port;
   struct   hostent     *hp;
   struct   sockaddr_in  server;
   char     *host, *bp, rbuf[BUFLEN], sbuf[BUFLEN];

   switch(argc) {
   case 2:
       host = argv[1];
       port = SERVER_TCP_PORT;
       break;
   case 3:
       host = argv[1];
       port = atoi(argv[2]);
       break;
   default:
       fprintf(stderr, "Usage: %s host[port]\n", argv[0]);
       exit(1);
   }

   /* Create a stream socket */
   if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        fprintf(stderr, "Can’t create a socket\n");
        exit(1);
   }

   bzero((char *)&server, sizeof(struct sockaddr_in));
   server.sin_family = AF_INET;
   server.sin_port = htons(port);
   if ((hp = gethostbyname(host)) == NULL) {
        fprintf(stderr, "Can’t get server’s address\n");
        exit(1);
    }
    bcopy(hp->h_addr, (char *)&server.sin_addr,
        hp->h_length);

    /* Connecting to the server */
    if (connect(sd, (struct sockaddr *)&server,
    sizeof(server)) == -1) {
        fprintf(stderr, "Can’t connect\n");
        exit(1);
    }
    printf("Connected: server’s address is %s\n",
        hp->h_name);
    printf("Transmit:\n");
    gets(sbuf);          /* get user’s text */
    write(sd, sbuf, BUFLEN); /* send it out */

    printf("Receive:\n");
    bp = rbuf;
    bytes_to_read = BUFLEN;
    while ((n = read (sd, bp, bytes_to_read)) > 0) {
        bp += n;
        bytes_to_read -= n;
    }
    printf("%s\n", rbuf);

    close(sd);
    return(0);
}
```

The student is encouraged to verify the sequence of socket calls in the above client and server programs with those shown in Figure 2.21. Further, the student may trace the sequence of calls by inserting a print statement after each call and verify that the `accept` call in the TCP server blocks until the `connect` call in the TCP client returns.

EXAMPLE Using the UDP Protocol

Let us now take a look at client/server programs using the UDP protocol. The following source code is a program that uses the UDP server as an echo server as before. Note that data receipt can be done in a single call with `recvfrom`, since UDP is blocked oriented.

```
/* Echo server using UDP */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_UDP_PORT  5000    /* well-known port */
#define MAXLEN       4096        /* maximum data length */

int main(int argc, char **argv)
{
   int      sd, client_len, port, n;
   char     buf[MAXLEN];
   struct   sockaddr_in server, client;
   switch(argc) {
   case 1:
        port = SERVER_UDP_PORT;
        break;
   case 2:
        port = atoi(argv[1]);
        break;
   default:
        fprintf(stderr, "Usage: %s [port]\n", argv[0]);
        exit(1);
   }

   /* Create a datagram socket */
   if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        fprintf(stderr, "Can't create a socket\n");
        exit(1);
   }

   /* Bind an address to the socket */
   bzero((char *)&server, sizeof(server));
   server.sin_family = AF_INET;
   server.sin_port = htons(port);
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   if (bind(sd, (struct sockaddr *)&server,
   sizeof(server)) == -1) {
        fprintf(stderr, "Can't bind name to socket\n");
        exit(1);
   }

   while (1) {
        client_len = sizeof(client);
        if ((n = recvfrom(sd, buf, MAXLEN, 0,
        (struct sockaddr *)&client, &client_len)) < 0) {
            fprintf(stderr, "Can't receive datagram\n");
            exit(1);
        }

        if (sendto(sd, buf, n, 0,
        (struct sockaddr *)&client, client_len) != n) {
             fprintf(stderr, "Can't send datagram\n");
             exit(1);
        }
   }
   close(sd);
   return(0);
}
```

The following client program first constructs a simple message of a predetermined length containing a string of characters a, b, c, . . . , z, a, b, c, . . . , z, . . . The client then gets the `start` time from the system using `gettimeofday` and sends the message to the echo server. After the message travels back, the client records the `end` time and measures the difference that represents the round-trip latency between the client and the server. The unit of time is recorded in milliseconds. This simple example shows how we can use sockets to gather important network statistics such as latencies and jitter.

```
/* A simple UDP client which measures round trip delay */
#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_UDP_PORT  5000
#define MAXLEN       4096     /* maximum data length */
#define DEFLEN       64       /* default length */

long delay(struct timeval t1, struct timeval t2);

int main(int argc, char **argv)
{
      int       data_size = DEFLEN, port = SERVER_UDP_PORT;
      int       i, j, sd, server_len;
      char      *pname, *host, rbuf[MAXLEN], sbuf[MAXLEN];
      struct    hostent    *hp;
      struct    sockaddr_in     server;
      struct    timeval      start, end;

      pname = argv[0];
      argc--;
      argv++;
      if (argc > 0 && (strcmp(*argv, "-s") == 0)) {
          if (--argc > 0 && (data_size = atoi(*++argv))) {
               argc--;
               argv++;
          }
          else {
               fprintf (stderr,
               "Usage: %s [-s data_size] host [port]\n",
               pname);
               exit(1);
          }
      }
      if (argc > 0) {
          host = *argv;
          if (--argc > 0)
              port = atoi(*++argv);
      }
```

```
else {
    fprintf(stderr,
    "Usage: %s [-s data_size] host [port]\n", pname);
    exit(1);
}

/* Create a datagram socket */
if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    fprintf(stderr, "Can’t create a socket\n");
    exit(1);
}

/* Store server’s information */
bzero((char *)&server, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(port);
if ((hp = gethostbyname(host)) == NULL) {
    fprintf(stderr, "Can’t get server’s IP address\n");
    exit(1);
}
bcopy(hp->h_addr, (char *)&server.sin_addr,
     hp->h_length);
if (data_size > MAXLEN) {
     fprintf(stderr, "Data is too big\n");
     exit(1);
}
/* data is a, b, c,..., z, a, b,... */
for (i = 0; i < data_size; i++) {
     j = (i < 26) ? i : i % 26;
     sbuf[i] = ’a’ + j;
}

gettimeofday(&start, NULL); /* start delay measure */

/* transmit data */
server_len = sizeof(server);
if (sendto(sd, sbuf, data_size, 0, (struct sockaddr *)
     &server, server_len) == -1) {
     fprintf(stderr, "sendto error\n");
     exit(1);
}

/* receive data */
if (recvfrom(sd, rbuf, MAXLEN, 0, (struct sockaddr *)
     &server, &server_len) < 0) {
     fprintf(stderr, "recvfrom error\n");
     exit(1);
}
```

```
     gettimeofday(&end, NULL); /* end delay measure */

     printf ("Round-trip delay = %ld ms.\n",
          delay(start, end));

     if (strncmp(sbuf, rbuf, data_size) != 0)
         printf("Data is corrupted\n");

     close(sd);
     return(0);
}

/*
 * Compute the delay between t1 and t2 in milliseconds
 */
long delay (struct timeval t1, struct timeval t2)
{
      long d;

      d = (t2.tv_sec - t1.tv_sec) * 1000;
      d += ((t2.tv_usec - t1.tv_usec + 500) / 1000);
      return(d);
}
```

It is important to remember that datagram communication using UDP is unreliable. If the communication is restricted to a local area network environment, say within a building, then datagram losses are extremely rare in practice, and the above client program should work well. However, in a wide area network environment, datagrams may be frequently discarded by the network. If the reply from the server does not reach the client, the client will wait forever! In this situation, the client must provide a timeout mechanism and retransmit the message. Also, further reliability may be provided to reorder the datagram at the receiver and to ensure that duplicated datagrams are discarded.

2.5 APPLICATION LAYER PROTOCOLS AND TCP/IP UTILITIES

Application layer protocols are high-level protocols that provide services to user applications. These protocols tend to be more visible to the user than other types of protocols. Furthermore, application protocols may be user written, or they may be standardized applications. Several standard application protocols form part of the TCP/IP protocol suite, the more common ones being Telnet, File Transfer Protocol (FTP), HTTP, and SMTP. Coverage of the various TCP/IP application layer protocols is beyond the scope of this textbook. The student is referred to “Internet Official Protocol Standards,” which provides a list of Internet protocols and standards [RFC 3000]. In this section the focus is on applications and utilities that can be used as tools to study the operation of the Internet. We also introduce network protocol analyzers and explain the basics of packet capture.

2.5.1 Telnet

Telnet is a TCP/IP protocol that provides a standardized means of accessing resources on a remote machine where the initiating machine is treated as local to the remote host. In many implementations Telnet can be used to connect to the port number of other servers and to interact with them using a command line. For example, the HTTP and SMTP examples in Section 2.1 were generated this way.

The Telnet protocol is based on the concept of a *network virtual terminal (NVT),* which is an imaginary device that represents a lowest common denominator terminal. By basing the protocol on this interface, the client and server machines do not have to obtain information about each other's terminal characteristics. Instead, each machine initially maps its characteristics to that of an NVT and *negotiates options* for changes to the NVT or other enhancements, such as changing the character set.

The NVT acts as a character-based terminal with a keyboard and printer. Data input by the client through the keyboard is sent to the server through the Telnet connection. This data is echoed back by the server to the client's printer. Other incoming data from the server is also printed.

Telnet commands use the seven-bit U.S. variant of the ASCII character set. A command consists minimally of a two-byte sequence: the Interpret as Command (IAC) escape character followed by the command code. If the command pertains to option negotiation, that is, one of WILL, WONT, DO, or DONT, then a third byte contains the option code. Table 2.4 lists the Telnet command names, their corresponding ASCII code, and their meaning.

A substantial number of Telnet options can be negotiated. Option negotiations begin once the connection is established and may occur at any time while connected. Negotiation is symmetric in the sense that either side can initiate a negotiation. A negotiation syntax is defined in RFC 854 to prevent acknowledgment loops from occurring.

Telnet uses one TCP connection. Because a TCP connection is identified by a pair of port numbers, a server is capable of supporting more than one Telnet connection at a time. Once the connection is established, the default is for the user, that is, the initiator of the connection, to enter a login name and password. By default the password is sent as clear text, although more recent versions of Telnet offer an authentication option.

2.5.2 File Transfer Protocol

File Transfer Protocol (FTP) is another commonly used application protocol. FTP provides for the transfer of a file from one machine to another. Like Telnet, FTP is intended to operate across different hosts, even when they are running different operating systems or have different file structures.

TABLE 2.4 Telnet commands.

Name	Code	Meaning
EOF	236	End of file.
SUSP	237	Suspend cursor process.
ABORT	238	Abort process.
EOR	239	End of record.
SE	240	End of subnegotiation parameters.
NOP	241	No operation.
Data mark	242	The data stream portion of a synch signal. This code should always be accompanied by a TCP urgent notification.
Break	243	NVT character BRK.
Interrupt process	244	The function IP.
Abort output	245	The function AO.
Are you there	246	The function AYT.
Erase character	247	The function EC.
Erase line	248	The function EL.
Go ahead	249	The GA signal.
SB	250	Indicates that what follows is subnegotiation of the indicated option.
WILL (option code)	251	Option negotiation.
WONT (option code)	252	Option negotiation.
DO (option code)	253	Option negotiation.
DONT (option code)	254	Option negotiation.
IAC	255	Data byte 255.

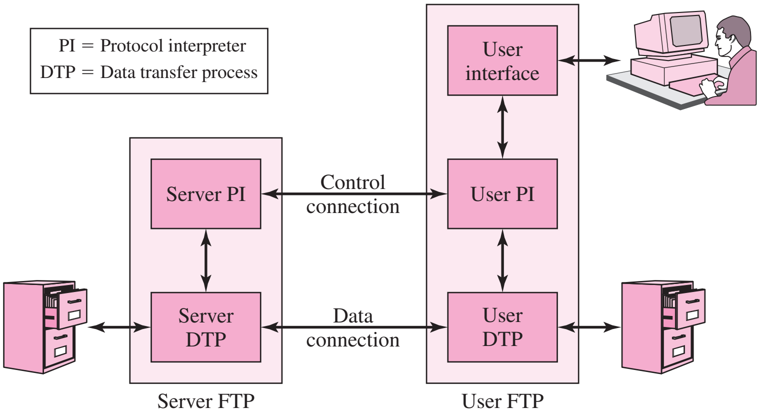


FIGURE 2.23 Transferring files using FTP.

FTP requires two TCP connections to transfer a file. One is the *control connection* that is established on port 21 at the server. The second TCP connection is a *data connection* used to perform a file transfer. A data connection must be established for each file transferred. Data connections are used for transferring a file in either direction or for obtaining lists of files or directories from the server to the client. Figure 2.23 shows the role of the two connections in FTP.

A control connection is established following the Telnet protocol from the user to the server port. FTP commands and replies are exchanged via the control connection. The user protocol interpreter (PI) is responsible for sending FTP commands and interpreting the replies. The server PI is responsible for interpreting commands, sending replies, and directing the server data transfer process (DTP) to establish a data connection and transfer. The commands are used to specify information about the data connection and about the particular file system operation being requested.

A data connection is established usually upon request from the user for some sort of file operation. The user PI usually chooses an ephemeral port number for its end of the operation and then issues a passive open from this port. The port number is then sent to the server PI using a PORT command. Upon receipt of the port number via the control connection, the server issues an active open to that same port. The server always uses port 20 for its end of the data connection. The user DTP then waits for the server to initiate and perform the file operation.

Note that the data connection may be used to send and receive simultaneously. Note also that the user may initiate a file transfer between two nonlocal machines, for example, between two servers. In this case there would be a control connection between the user and both servers but only one data connection, namely, the one between the two servers.

The user is responsible for requesting a close of the control connection, although the server performs the action. If the control connection is closed while the data connection is still open, then the server may terminate the data transfer. The data connection is usually closed by the server. The main exception is when the user DTP closes the data connection to indicate an end of file for a stream transmission. Note that FTP is not designed to detect lost or scrambled bits; the responsibility for error detection is left to TCP.

The Telnet protocol works across different systems because it specifies a common starting point for terminal emulation. FTP works across different systems because it can accommodate several different file types and structures. FTP commands are used to specify information about the file and how it will be transmitted. In general, three types of information must be specified. Note that the default specifications must be supported by every FTP implementation.

1. *File type.* FTP supports ASCII, EBCDIC, image (binary), or local. Local specifies that the data is to be transferred in logical bytes, where the size is specified in a separate parameter. *ASCII is the default type.* If the file is ASCII or EBCDIC, then a vertical format control may also be specified.
2. *Data structure.* FTP supports file structure (a continuous stream of bytes with no internal structure), record structure (used with text files), and page structure (file consists of independent indexed pages). *File structure is the default specification.*
3. *Transmission mode.* FTP supports stream, block, or compressed mode. When transmission is in stream mode, the user DTP closes the connection to indicate the end of file for data with file structure. If the data has block structure, then a special two-byte sequence indicates end of record and end of file. *The default is stream mode.*

An *FTP command* consists of three or four bytes of uppercase ASCII characters followed by a space if parameters follow, or by a Telnet end of option list (EOL) otherwise.

FTP commands fall into one of the following categories: access control identification, data transfer parameters, and FTP service requests. Table 2.5 lists some of the common FTP commands encountered.

Every command must produce at least one *FTP reply.* The replies are used to synchronize requests and actions and to keep the client informed of the state of the server. A reply consists of a three-digit number (in alphanumeric representation) followed by some text. The numeric code is intended for the user PI; the text, if processed, is intended for the user. For example, the reply issued following a successful connection termination request is "`221 Goodbye`." The first digit indicates whether and to what extent the specified request has been completed. The second digit indicates the category of the reply, and the third digit provides additional information about the particular category. Table 2.6 lists the possible values of the first two digits and their meanings.

In this case of the goodbye message, the first 2 indicates a successful completion. The second digit is also 2 to indicate that the reply pertains to a connection request.

TABLE 2.5 Some common FTP commands.

Command	Meaning
`ABOR`	Abort the previous FTP command and any data transfer.
`LIST`	List files or directories.
`QUIT`	Log off from server.
`RETR filename`	Retrieve the specified file.
`STOR filename`	Store the specified file.

TABLE 2.6 FTP replies—the first and second digits.

Reply	Meaning
`1yz`	Positive preliminary reply (action has begun, but wait for another reply before sending a new command).
`2yz`	Positive completion reply (action completed successfully; new command may be sent).
`3yz`	Positive intermediary reply (command accepted, but action cannot be performed without additional information; user should send a command with the necessary information).
`4yz`	Transient negative completion reply (action currently cannot be performed; resend command later).
`5yz`	Permanent negative completion reply (action cannot be performed; do not resend it).
`x0z`	Syntax errors.
`x1z`	Information (replies to requests for status or help).
`x2z`	Connections (replies referring to the control and data connections).
`x3z`	Authentication and accounting (replies for the login process and accounting procedures).
`x4z`	Unspecified.
`x5z`	File system status.

2.5.3 Hypertext Transfer Protocol and the World Wide Web

The World Wide Web (WWW) provides a framework for accessing documents and resources that are located in computers connected to the Internet. These documents consist of text, graphics and other media and are interconnected by *hyperlinks* or *links* that appear within the documents. The **Hypertext Markup Language (HTML)** is used to prepare these documents. The WWW is accessed through a browser program that interprets HTML, displays the documents, and allows the user to access other documents by clicking on these links.

Each link provides the browser with a uniform resource locator (URL) that specifies the name of the machine where the document is located as well as the name of the file that contains the requested document. For example, the sequence of packet exchanges captured in Figure 2.18 and Figure 2.19 result after clicking on the URL http://www.nytimes.com/. The first term ‘http’ specifies the retrieval mechanism to be used, in this case, the HTTP protocol. The next term specifies the name of the host machine, namely, www.nytimes.com. The remaining term gives the path component, that is, it identifies the file on that server containing the desired article. In this example, the final slash (/) refers to the server root. By clicking a highlighted item in a browser page the user begins an interaction to obtain the desired file from the server where it is stored. The **Hypertext Transfer Protocol (HTTP)** is the application layer protocol that defines the interaction between the web client and the web server.

HTTP

HTTP is a client/server application defined in RFC 1945 and RFC 2616 to support communications between web browsers and web servers. HTTP defines how the client makes the request for an object (typically a document) and how the server replies with a response message. The typical interaction is shown in Figure 2.18. After the user clicks on a link, the browser program must first resolve the URL to an IP address by invoking the DNS protocol (frame 1 in the figure). Once the IP address is returned (frame 2), the HTTP client must set up a TCP connection to the desired server over well-known port 80 (frames 2, 3, and 4). The HTTP client and server are then ready to exchange messages: the client sends a GET message requesting the document, and the server replies with a response followed by the desired document.

HTTP is a *stateless* protocol in that it does not maintain any information (“state”) about its clients. In other words, the HTTP server handles each request independently of all other requests. Thus if a client sends a request multiple times, the server handles each request in the same manner. HTTP was designed to be stateless in order to keep it simple. This design allows requests to be handled quickly and enables servers to handle large volumes of requests per second.

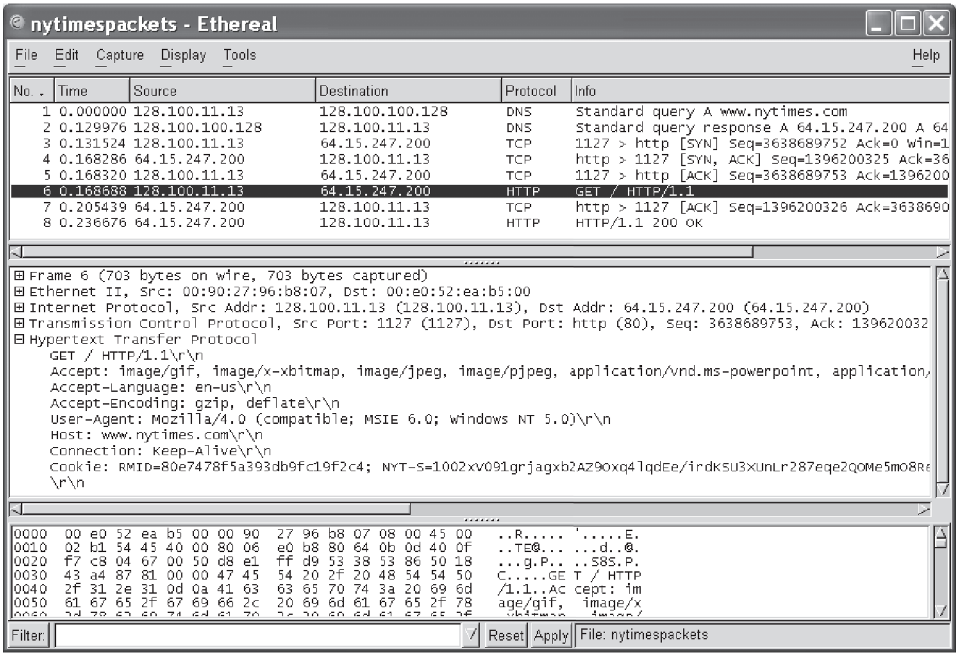
The initial design of HTTP/1.0 (and earlier versions) uses *nonpersistent connections*. The TCP connection is closed after each request-response interaction. Each subsequent request from the same client to the same server involves the setting up and tearing down of an additional TCP connection. From the example in Figure 2.18, we can see that each TCP connection setup involves the exchange of three segments between the client and server machines and hence the sending of the request is delayed by multiple round-trip times.[11] Another disadvantage of nonpersistent connections is that TCP processing and memory resources are wasted in the server and the client. HTTP/1.1 made *persistent connections* the default mode. The server now keeps the TCP connection open for a certain period of time after sending a response. This enables the client to make multiple requests over the same TCP connection and hence avoid the inefficiency and delay of the nonpersistent mode.

MESSAGE FORMATS

Like Telnet and FTP, HTTP messages are written in ASCII text and can be read and interpreted readily. Figure 2.24 (middle pane) shows a typical HTTP request message. The first line of a request message is the *request line,* and subsequent lines are called *header lines*. Each line is written in ASCII text and terminated by a carriage return followed by a line feed character. The last header line is followed by an extra carriage return and line feed. Some request messages include an *entity body* that follows the header section and provides additional information to the server.

The request line has the following form: *Method URL HTTP-Version* \r\n. The *Method* field specifies the action method or action that is applied to the object. The second field identifies the object, and the remaining field is the HTTP version. In the first line of the HTTP section in the middle pane of Figure 2.24, the method is

[11] A round-trip time (RTT) is the time that elapses from when a message is sent from a transmitter to when a response is received back from the receiver.



```
No. .  Time     Source          Destination      Protocol  Info
     1 0.000000 128.100.11.13   128.100.100.128  DNS       Standard query A www.nytimes.com
     2 0.129976 128.100.100.128 128.100.11.13    DNS       Standard query response A 64.15.247.200 A 64
     3 0.131524 128.100.11.13   64.15.247.200    TCP       1127 > http [SYN] Seq=3638689752 Ack=0 Win=1
     4 0.168286 64.15.247.200   128.100.11.13    TCP       http > 1127 [SYN, ACK] Seq=1396200325 Ack=36
     5 0.168320 128.100.11.13   64.15.247.200    TCP       1127 > http [ACK] Seq=3638689753 Ack=1396200
     6 0.168688 128.100.11.13   64.15.247.200    HTTP      GET / HTTP/1.1
     7 0.205439 64.15.247.200   128.100.11.13    TCP       http > 1127 [ACK] Seq=1396200326 Ack=3638690
     8 0.236676 64.15.247.200   128.100.11.13    HTTP      HTTP/1.1 200 OK
```

```
⊞ Frame 6 (703 bytes on wire, 703 bytes captured)
⊞ Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
⊞ Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 64.15.247.200 (64.15.247.200)
⊞ Transmission Control Protocol, Src Port: 1127 (1127), Dst Port: http (80), Seq: 3638689753, Ack: 139620032
⊟ Hypertext Transfer Protocol
    GET / HTTP/1.1\r\n
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/
    Accept-Language: en-us\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)\r\n
    Host: www.nytimes.com\r\n
    Connection: Keep-Alive\r\n
    Cookie: RMID=80e7478f5a393db9fc19f2c4; NYT-S=1002xV091grjagxb2AZ9Oxq4lqdEe/irdKSU3XUnLr287eqe2QOMe5mO8Re
    \r\n
```

```
0000  00 e0 52 ea b5 00 00 90  27 96 b8 07 08 00 45 00   ..R..... '.....E.
0010  02 b1 54 45 40 00 80 06  e0 b8 80 64 0b 0d 40 0f   ..TE@... ...d..@.
0020  f7 c8 04 67 00 50 d8 e1  ff d9 53 38 53 86 50 18   ...g.P.. ..S8S.P.
0030  43 a4 87 81 00 00 47 45  54 20 2f 20 48 54 54 50   C.....GE T / HTTP
0040  2f 31 2e 31 0d 0a 41 63  63 65 70 74 3a 20 69 6d   /1.1..Ac cept: im
0050  61 67 65 2f 67 69 66 2c  20 69 6d 61 67 65 2f 78   age/gif,  image/x
```

FIGURE 2.24 Ethereal capture of an HTTP GET message.

TABLE 2.7 HTTP request methods.

Request method	Meaning
GET	Retrieve information (object) identified by the URL.
HEAD	Retrieve meta-information about the object, but do not transfer the object; Can be used to find out if a document has changed.
POST	Send information to a URL (using the entity body) and retrieve result; used when a user fills out a form in a browser.
PUT	Store information in location named by URL.
DELETE	Remove object identified by URL.
TRACE	Trace HTTP forwarding through proxies, tunnels, etc.
OPTIONS	Used to determine the capabilities of the server, or characteristics of a named resource.

GET and the *absolute* URL of the file requested is http://www.nytimes.com/. However HTTP/1.1 uses the *relative* URL which consists of the path only, in this case /. The HTTP version is 1.1.

The HTTP headers consist of a sequence of zero or more lines each consisting of an attribute name, followed by a colon, ":", and an attribute value. The client uses header lines to inform the server about: the type of the client, the kind of content it can accept, and the identity of the requester. In the example in Figure 2.24, the Accept header indicates a list of document format types that the client can accept. The Accept-language header indicates that U.S. English is accepted. The User-agent header indicates that the browser is Mozilla/4.0.

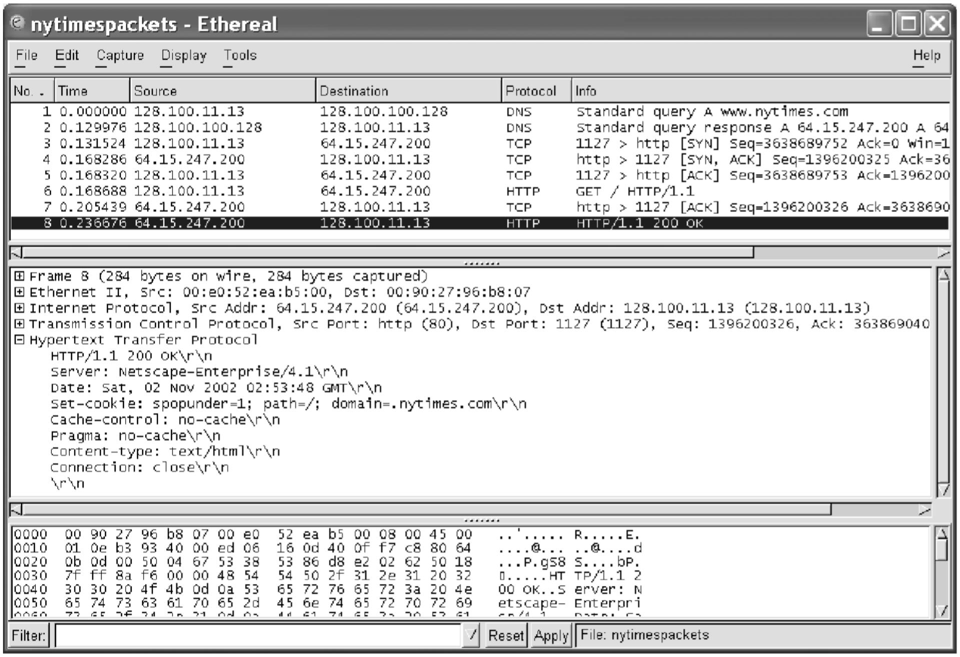
The current request methods available in HTTP/1.1 are given in Table 2.7. HTTP/1.0 only provides the methods GET, POST, and HEAD, and so these are the three methods that are supported widely.

The HTTP response message begins with an ASCII status line, followed by a headers section, and then by content, which is usually either an image or an HTML document. Figure 2.25 shows an example of a captured HTTP response message.

The response status line has the form: *HTTP-Version Status-Code Message \r \n*. The status code is a 3-digit number that indicates the result of the request to the client. The message indicates the result of the request in text that can be interpreted by humans. Examples of commonly encountered status lines are:

- HTTP/1.0 200 OK
- HTTP/1.0 301 Moved Permanently
- HTTP/1.0 400 Bad Request
- HTTP/1.0 500 Internal Server Error

The response headers provide information about the object that is being transferred to the client. Header lines are used to indicate: the type of server; the date and time the HTTP response was prepared and sent; and the time and date when the object was created or last modified. A Content-length header line indicates the length in bytes



```
No. . Time     Source          Destination     Protocol Info
    1 0.000000 128.100.11.13   128.100.100.128 DNS      Standard query A www.nytimes.com
    2 0.129976 128.100.100.128 128.100.11.13   DNS      Standard query response A 64.15.247.200 A 64
    3 0.131524 128.100.11.13   64.15.247.200   TCP      1127 > http [SYN] Seq=3638689752 Ack=0 Win=1
    4 0.168286 64.15.247.200   128.100.11.13   TCP      http > 1127 [SYN, ACK] Seq=1396200325 Ack=36
    5 0.168320 128.100.11.13   64.15.247.200   TCP      1127 > http [ACK] Seq=3638689753 Ack=1396200
    6 0.168688 128.100.11.13   64.15.247.200   HTTP     GET / HTTP/1.1
    7 0.205439 64.15.247.200   128.100.11.13   TCP      http > 1127 [ACK] Seq=1396200326 Ack=3638690
    8 0.236676 64.15.247.200   128.100.11.13   HTTP     HTTP/1.1 200 OK
```

```
⊞ Frame 8 (284 bytes on wire, 284 bytes captured)
⊞ Ethernet II, Src: 00:e0:52:ea:b5:00, Dst: 00:90:27:96:b8:07
⊞ Internet Protocol, Src Addr: 64.15.247.200 (64.15.247.200), Dst Addr: 128.100.11.13 (128.100.11.13)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 1127 (1127), Seq: 1396200326, Ack: 363869040
⊟ Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Server: Netscape-Enterprise/4.1\r\n
    Date: Sat, 02 Nov 2002 02:53:48 GMT\r\n
    Set-cookie: spopunder=1; path=/; domain=.nytimes.com\r\n
    Cache-control: no-cache\r\n
    Pragma: no-cache\r\n
    Content-type: text/html\r\n
    Connection: close\r\n
    \r\n
```

```
0000  00 90 27 96 b8 07 00 e0  52 ea b5 00 08 00 45 00   ..'..... R.....E.
0010  01 0e b3 93 40 00 ed 06  16 0d 40 0f f7 c8 80 64   ....@... ..@....d
0020  0b 0d 00 50 04 67 53 38  53 86 d8 e2 02 62 50 18   ...P.gS8 S....bP.
0030  7f ff 8a f6 00 00 48 54  54 50 2f 31 2e 31 20 32   ▯.....HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 53  65 72 76 65 72 3a 20 4e   00 OK..S erver: N
0050  65 74 73 63 61 70 65 2d  45 6e 74 65 72 70 72 69   etscape- Enterpri
```

FIGURE 2.25 Ethereal capture showing HTTP response message.

of the object being transferred, and a Content-type header line indicates the type of the object (document) and how it is encoded. All these header lines are evident in the example in Figure 2.25. The response headers section ends with a blank line and may be followed by an entity body that carries the content.

HTTP PROXY SERVER AND CACHING

The simple-to-use graphical interface of web browsers made the web accessible to ordinary computer users and led to an explosion in the volume of traffic handled by the Internet. Web traffic is by far the largest component of all the traffic carried in the Internet. When the volume of requests for information from popular websites becomes sufficiently large, it makes sense to cache web information in servers closer to the user. By intercepting and responding to the HTTP request closer to the user, the volume of traffic that has to traverse the backbone of the Internet is reduced.

A *web proxy server* can be deployed to provide caching of web information. Typically proxy servers are deployed by Internet Service Providers to reduce the delay of web responses and to control the volume of web traffic. The user's browser must be configured to first access the proxy server when making a web request. If the proxy server has the desired object, it replies with an HTTP response. If it does not have the object, it sets up a TCP connection to the target URL and retrieves the desired object. It then replies to the client with the appropriate response, and caches the object for future requests.

COOKIES AND WEB SESSIONS

It was indicated that the HTTP protocol is stateless and does not maintain information about prior requests from a given client. The use of cookies makes it possible to have web sessions where a user interacts with a web site in a manner that takes into account the user's preferences. **Cookies** are data that are exchanged and stored by clients and servers and transferred as header lines in HTTP messages. These header lines provide context for each HTTP interaction.

When a client first accesses a web server that uses cookies, the server replies with Response message that includes a *Set-cookie* header line. This header line includes a unique ID number for the given client. If the client software accepts cookies, the cookie is added to the browser's cookie file. Each time the client makes a request to the given site, it includes a *Cookie* header line with the unique ID number in its requests. The server site maintains a separate cookie database where it can store which pages were accessed at what date and time by each client. In this manner, the server can prepare responses to HTTP requests that take into account the history of the given user. Cookies enable a website to keep track of a user's shopping cart during a session, as well as other longer term information such as address and credit card information. The example in Figure 2.24 can be seen to include a Cookie header line with an ID number that consists of 24 hexadecimal numerals.

Cookies are required to include an expiration date. Cookies that do not include an expiration date are deleted by the browser at the end of an interaction. Cookies are an indirect means for a user to identify itself to a server. If security and privacy are required, protocols such as SSL and TLS need to be used. These protocols are discussed in Chapter 12.

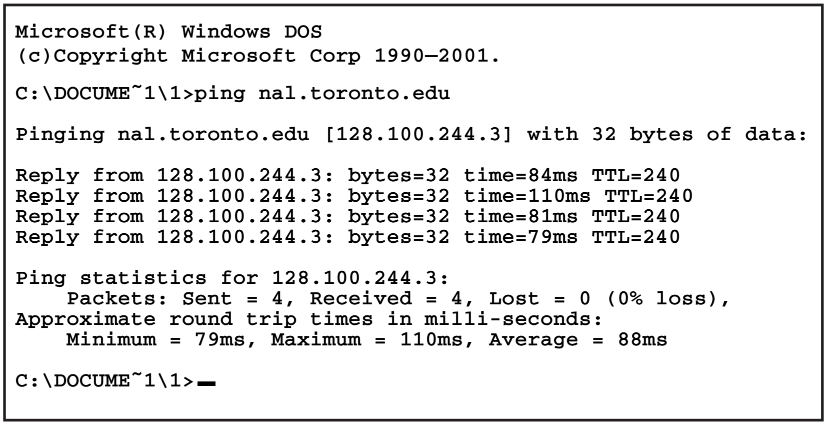
2.5.4 IP Utilities

A number of utilities are available to help in finding out about IP hosts and domains and to measure Internet performance. In this section we discuss PING, which can be used to determine whether a host is reachable; traceroute, a utility to determine the route that a packet will take to another host; netstat, which provides information about the network status of a local host; and tcpdump, which captures and observes packet exchanges in a link. We also discuss the use of Telnet with standard TCP/IP services as a troubleshooting and monitoring tool.

PING

PING is a fairly simple application used to determine whether a host is online and available. The name is said to derive from its analogous use in sonar operations to detect underwater objects.[12] PING makes use of Internet Control Message Protocol (ICMP) messages. The purpose of ICMP is to inform sending hosts about errors encountered in IP datagram processing or other control information by destination hosts or by routers. ICMP is discussed in Chapter 8. PING sends one or more ICMP Echo messages to a specified host requesting a reply. PING is often used to measure the round-trip delay

[12] PING is also reported to represent the acronym Packet Internet Groper [Murhammer 1998].



```
Microsoft(R) Windows DOS
(c)Copyright Microsoft Corp 1990–2001.

C:\DOCUME~1\1>ping nal.toronto.edu

Pinging nal.toronto.edu [128.100.244.3] with 32 bytes of data:

Reply from 128.100.244.3: bytes=32 time=84ms TTL=240
Reply from 128.100.244.3: bytes=32 time=110ms TTL=240
Reply from 128.100.244.3: bytes=32 time=81ms TTL=240
Reply from 128.100.244.3: bytes=32 time=79ms TTL=240

Ping statistics for 128.100.244.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 79ms, Maximum = 110ms, Average = 88ms

C:\DOCUME~1\1>
```

FIGURE 2.26 Using PING to determine host accessibility.

between two hosts. The sender sends a datagram with a type 8 Echo message and a sequence number to detect a lost, reordered, or duplicated message. The receiver changes the type to Echo Reply (type 0) and returns the datagram. Because the TCP/IP suite incorporates ICMP, any machine with TCP/IP installed can reply to PING. However, because of the increased presence of security measures such as firewalls, the tool is not always successful. Nonetheless, it is still the first test used to determine accessibility of a host.

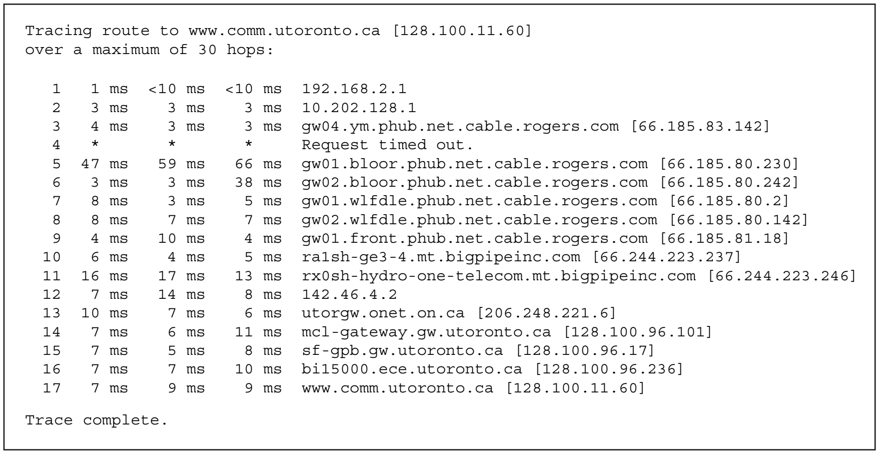
In Figure 2.26 PING is used to determine whether the NAL machine is available. In this example, the utility was run in an MS-DOS session under Windows XP. The command in its simplest form is `ping <hostname>`. The round-trip delay is indicated, as well as the time-to-live (TTL) value. The TTL is the maximum number of hops an IP packet is allowed to remain in the network. Each time an IP packet passes through a router, the TTL is decreased by 1. When the TTL reaches 0, the packet is discarded. See Chapter 8 for a PING and ICMP packet capture.

TELNET AND STANDARD SERVICES

Because ICMP operates at the IP level, PING tests the reachability of the IP layer only in the destination machine. PING does not test the layers above IP. A number of standard TCP/IP application layer services can be used to test the layers above IP. Telnet can be used to access these services for testing purposes. Examples of these services include Echo (port number 7), which echoes a character back to the sender, and Daytime (port number 13), which returns the time and date. A variety of utilities are becoming available for testing reachability and performance of HTTP and Web servers. The student is referred to the Cooperative Association for Internet Data Analysis (CAIDA) website, currently www.caida.org.

TRACEROUTE

A second TCP/IP utility that is commonly used is traceroute. This tool allows users to determine the route that a packet takes from the local host to a remote host, as well as latency and reachability from the source to each hop. Traceroute is generally used as a debugging tool by network managers.



```
Tracing route to www.comm.utoronto.ca [128.100.11.60]
over a maximum of 30 hops:

   1    1 ms  <10 ms  <10 ms  192.168.2.1
   2    3 ms    3 ms    3 ms  10.202.128.1
   3    4 ms    3 ms    3 ms  gw04.ym.phub.net.cable.rogers.com [66.185.83.142]
   4    *       *       *     Request timed out.
   5   47 ms   59 ms   66 ms  gw01.bloor.phub.net.cable.rogers.com [66.185.80.230]
   6    3 ms    3 ms   38 ms  gw02.bloor.phub.net.cable.rogers.com [66.185.80.242]
   7    8 ms    3 ms    5 ms  gw01.wlfdle.phub.net.cable.rogers.com [66.185.80.2]
   8    8 ms    7 ms    7 ms  gw02.wlfdle.phub.net.cable.rogers.com [66.185.80.142]
   9    4 ms   10 ms    4 ms  gw01.front.phub.net.cable.rogers.com [66.185.81.18]
  10    6 ms    4 ms    5 ms  ra1sh-ge3-4.mt.bigpipeinc.com [66.244.223.237]
  11   16 ms   17 ms   13 ms  rx0sh-hydro-one-telecom.mt.bigpipeinc.com [66.244.223.246]
  12    7 ms   14 ms    8 ms  142.46.4.2
  13   10 ms    7 ms    6 ms  utorgw.onet.on.ca [206.248.221.6]
  14    7 ms    6 ms   11 ms  mcl-gateway.gw.utoronto.ca [128.100.96.101]
  15    7 ms    5 ms    8 ms  sf-gpb.gw.utoronto.ca [128.100.96.17]
  16    7 ms    7 ms   10 ms  bi15000.ece.utoronto.ca [128.100.96.236]
  17    7 ms    9 ms    9 ms  www.comm.utoronto.ca [128.100.11.60]

Trace complete.
```

FIGURE 2.27 Output from traceroute (running from a home PC to a host at the University of Toronto).

Traceroute makes use of both ICMP and UDP. The sender first sends a UDP datagram with TTL = 1 as well as an invalid port number to the specified destination host. The first router to see the datagram sets the TTL field to zero, discards the datagram, and sends an ICMP Time Exceeded message to the sender. This information allows the sender to identify the first machine in the route. Traceroute continues to identify the remaining machines between the source and destination machines by sending datagrams with successively larger TTL fields. When the datagram finally reaches its destination, that host machine returns an ICMP Port Unreachable message to the sender because of the invalid port number deliberately set in the datagram.

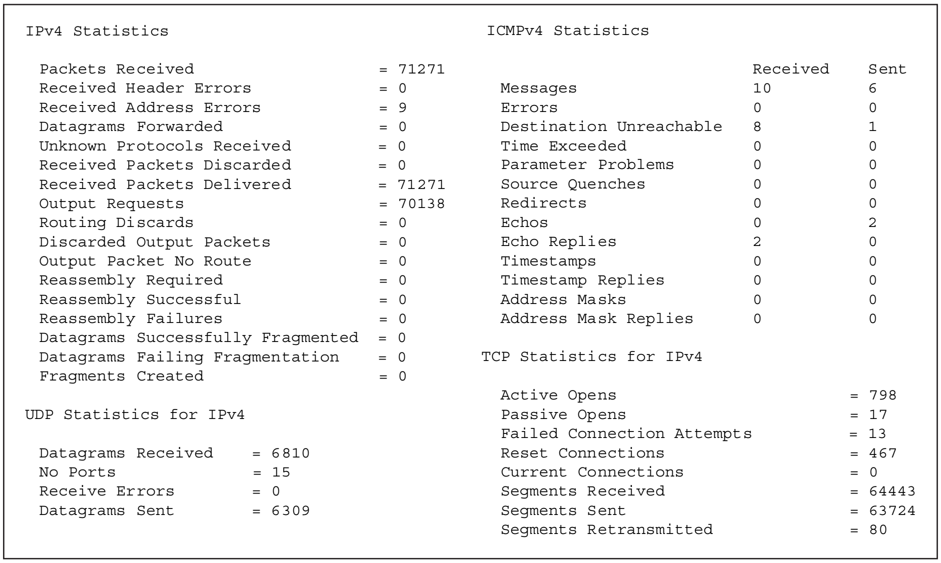
Figure 2.27 shows the result from running traceroute from a home PC to a host at the University of Toronto. The first line corresponds to the first hop in a home router. The next eight lines correspond to hops within the Internet Service Provider's network. The University of Toronto router gateway is reached in hop 13, and then various routers inside the university are traversed before arriving at the desired host.

IPCONFIG

The ipconfig utility, available on Microsoft® Windows operating systems, can be used to display the TCP/IP information about a host. In its simplest form the command returns the IP address, subnet mask (discussed in Chapter 8) and default gateway for the host. The utility can also be used to obtain information for each IP network interface for the host, for example, DNS hostname, IP addresses of DNS servers, physical address of the network card, IP address for the network interface, and whether DHCP is enabled for automatic configuration of the card's IP address. The ipconfig/renew command is used to renew an IP address with a DHCP server.

NETSTAT

The netstat queries a host about its TCP/IP network status. For example, netstat can be used to find the status of the network drivers and their interface cards, such as the



```
IPv4 Statistics                                 ICMPv4 Statistics

  Packets Received                   = 71271                            Received    Sent
  Received Header Errors             = 0          Messages              10          6
  Received Address Errors            = 9          Errors                0           0
  Datagrams Forwarded                = 0          Destination Unreachable  8        1
  Unknown Protocols Received         = 0          Time Exceeded         0           0
  Received Packets Discarded         = 0          Parameter Problems    0           0
  Received Packets Delivered         = 71271      Source Quenches       0           0
  Output Requests                    = 70138      Redirects             0           0
  Routing Discards                   = 0          Echos                 0           2
  Discarded Output Packets           = 0          Echo Replies          2           0
  Output Packet No Route             = 0          Timestamps            0           0
  Reassembly Required                = 0          Timestamp Replies     0           0
  Reassembly Successful              = 0          Address Masks         0           0
  Reassembly Failures                = 0          Address Mask Replies  0           0
  Datagrams Successfully Fragmented  = 0
  Datagrams Failing Fragmentation    = 0        TCP Statistics for IPv4
  Fragments Created                  = 0
                                                  Active Opens                   = 798
UDP Statistics for IPv4                           Passive Opens                  = 17
                                                  Failed Connection Attempts     = 13
  Datagrams Received    = 6810                    Reset Connections              = 467
  No Ports              = 15                      Current Connections            = 0
  Receive Errors        = 0                       Segments Received              = 64443
  Datagrams Sent        = 6309                    Segments Sent                  = 63724
                                                  Segments Retransmitted         = 80
```

FIGURE 2.28 Sample protocol statistics output from netstat.

number of in packets, out packets, errored packets, and so on. It can also find out the state of the routing table in a host, which TCP/IP server processes are active in the host, as well as which TCP connections are active. Figure 2.28 shows the result from running netstat with the protocol statistics option. Various counts for IP, ICMP, TCP, and UDP are displayed.

2.5.5 Tcpdump and Network Protocol Analyzers

The tcpdump program can capture and observe IP packet exchanges on a network interface. The program usually involves setting an Ethernet network interface card into a "promiscuous" mode so that the card listens and captures every frame that traverses the Ethernet broadcast network. A packet filter is used to select the IP packets that are of interest in a given situation. These IP packets and their higher-layer contents can then be observed and analyzed. Because of security concern, normal users typically cannot run the tcpdump program.

The tcpdump utility can be viewed as an early form of a protocol analyzer. A *network protocol analyzer* is a tool for capturing, displaying, and analyzing the PDUs that are exchanged in a network. Current analyzers cover a very broad range of protocols and are constantly being updated. Protocol analyzers are indispensable in troubleshooting network problems and in designing new network systems. Protocol analyzers are also extremely useful in teaching the operation of protocols by providing a means of examining traffic from a live network.

The first component for a protocol analyzer is hardware to capture the digital information from the physical medium. The most cost-effective means for capturing information is to use a LAN network interface card. Most LANs support operation in promiscuous mode where all frames on the LAN are captured for examination. Note that in most LAN protocols the frames can be seen by all devices attached to the medium, even if the frame is not intended for them. Since most computers are connected to Ethernet LANs, packet capture can be done readily by installing device driver software to control the network interface card.

Given the increasingly high speeds of LAN operation, the volume of information that can be captured can quickly become huge. The second component of a protocol analyzer is filtering software to select the frames that contain the desired information. Filtering can be done by frame address, by IP address, by protocol, and by many other combinations. The final component of a protocol analyzer consists of the utilities for the display and analysis of protocol exchanges. A number of commercial and open source network protocol analyzers packages are available. In this book we will use the Ethereal open source package. Several hundred developers have contributed to the development of Ethereal leading to a tool that supports an extensive set of protocols. The Ethereal package can be downloaded from www.ethereal.com. Their website also contains instructions and example screen captures.

Network protocol analyzers give the ability to capture all packets in a LAN and in doing so provide an opportunity to gain unauthorized access to network information. These tools should *always* be used in a responsible and ethical manner.

SUMMARY

This chapter describes how network architectures are based on the notion of layering. Layering involves combining network functions into groups that can be implemented together. Each layer provides a set of services to the layer above it; each layer builds its services using the services of the layer below. Thus applications are developed using application layer protocols, and application layer protocols are built on top of the communication services provided by TCP and UDP. These transport protocols in turn build on the datagram service provided by IP, which is designed to operate over various network technologies. IP allows the applications above it to be developed independently of specific underlying network technologies. The network technologies below IP range from full-fledged packet-switching networks, such as ATM, to LANs, and individual point-to-point links.

The Berkeley socket API allows the programmer to develop applications using the services provided by TCP and UDP. Examples of applications that run over TCP are HTTP, FTP, SMTP and Telnet. DNS and RTP are examples that run over UDP. The power of the TCP/IP architecture is that any new application that runs over TCP or UDP will run over the entire global Internet. Consequently, new services and applications can be deployed globally very quickly, a capability that no other network architecture can provide. We also introduced various TCP/IP utilities and tools that allow the programmer to determine the state and configuration of a TCP/IP network. Students can use these tools to get some hands on experience with the operation of TCP/IP.

CHECKLIST OF IMPORTANT TERMS

application layer
blocking/unblocking
client/server
confirmed/unconfirmed service
connectionless service
connection-oriented service
cookie
daemon
data link layer
datagram
Domain Name System (DNS)
encapsulation
ephemeral port number
frame
globally unique IP address
header
HyperText Markup Language (HTML)
Hypertext Transfer Protocol (HTTP)
internet layer
internetworking
layer
layer n entity
layer n protocol
multiplexing/demultiplexing
network architecture
network interface layer
network layer
OSI reference model
packet
peer process
physical address
physical layer
Point-to-Point Protocol (PPP)
port
Post Office Protocol version 3 (POP3)
presentation layer
protocol
protocol data unit (PDU)
segment
segmentation and reassembly
service access point (SAP)
service data unit (SDU)
session layer
Simple Mail Transfer Protocol (SMTP)
socket
socket address
splitting/recombining
TCP/IP network architecture
Transmission Control Protocol (TCP)
transport layer
User Datagram Protocol (UDP)
well-known port number

FURTHER READING

Comer, D. E. and D. L. Stevens, *Internetworking with TCP/IP, Vol. III: Client-Server Programming and Applications,* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

Murhammer, M. W., O. Atakan, S. Bretz, L. R. Pugh, K. Suzuki, and D. H. Wood, *TCP/IP Tutorial and Technical Overview,* Prentice Hall PTR, Upper Saddle River, New Jersey, 1998.

Perlman, R., *Interconnections: Bridges, Routers Switches and Internet Protocols,* Addison-Wesley, Reading, Massachusetts, 2000.

Piscitello, D. M. and A. L. Chapin, *Open Systems Networking: TCP/IP and OSI,* Addison-Wesley, Reading, Massachusetts, 1993.

Sechrest, S., "An Introductory 4.4 BSD Interprocess Communication Tutorial," *Computer Science Network Group,* UC Berkeley.

Stevens, W. R., *TCP/IP Illustrated, Volume 1: The Protocols,* Addison-Wesley, Reading, Massachusetts, 1994.

Stevens, W. R., *UNIX Network Programming,* Volume 1, 2nd edition, Prentice Hall, Englewood Cliffs, New Jersey, 1998. An excellent treatment of socket programming.

Yeager, N. J. and R. E. McGrath, *Web Server Technology: The Advanced Guide for World Wide Web Information Providers,* Morgan Kaufmann, San Francisco, 1996.

RFC 821, J. Postel, "Simple Mail Transfer Protocol," August 1982.

RFC 854, J. Postel and J. Reynolds, "Telnet Protocol Specification," May 1983.

RFC 959, J. Postel and J. Reynolds, "File Transfer Protocol," October 1985.

RFC 1034, Mockapetris, "Domain Names—Concepts and Facilities," November 1987.

RFC 1035, Mockapetris, "Domain Names—Implementation and Specification," November 1987.

RFC 1945, T. Berners-Lee, R. Fielding, and H. Frystik, "Hypertext Transfer Protocol/1.0," May 1996.

RFC 2068, R. Fielding, J. Geetys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol," January 1997.

RFC 2151, G. Kessler and S. Shepard, "Internet & TCP/IP Tools and Utilities," June 1997.

RFC 2616, R. Fielding, J. Geetys, J. Mogul, H. Frystyk, L. Masinder, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol/1.1," June 1999.

RFC 2821, J. Klensin, ed., "Simple Mail Transfer Protocol," April 2001.

RFC 3000, J. Reynolds, R. Braden, S. Ginoza, and L. Shiota, eds., "Internet Official Protocol Standards," November 2001.

See our website for additional references available through the Internet.

PROBLEMS

2.1. Explain how the notion of layering and internetworking make the rapid growth of applications such as the World Wide Web possible.

2.2. (a) What universal set of communication services is provided by TCP/IP?
(b) How is independence from underlying network technologies achieved?
(c) What economies of scale result from (a) and (b)?

2.3. What difference does it make to the network layer if the underlying data link layer provides a connection-oriented service versus a connectionless service?

2.4. Suppose transmission channels become virtually error free. Is the data link layer still needed?

2.5. Why is the transport layer not present inside the network?

2.6. Which OSI layer is responsible for the following?
(a) Determining the best path to route packets.
(b) Providing end-to-end communications with reliable service.
(c) Providing node-to-node communications with reliable service.

2.7. Should connection establishment be a confirmed service or an unconfirmed service? What about data transfer in a connection-oriented service? Connection release?

2.8. Does it make sense for a network to provide a confirmed, connectionless packet transfer service?

2.9. Explain how the notion of multiplexing can be applied at the data link, network, and transport layers. Draw a figure that shows the flow of PDUs in each multiplexing scheme.

2.10. Give two features that the data link layer and transport layer have in common. Give two features in which they differ. Hint: Compare what can go wrong to the PDUs that are handled by these layers.

2.11. (a) Can a connection-oriented, reliable message transfer service be provided across a connectionless packet network? Explain.

(b) Can a connectionless datagram transfer service be provided across a connection-oriented network?

2.12. An internet path between two hosts involves a hop across network A, a packet-switching network, to a router and then another hop across packet-switching network B. Suppose that packet-switching network A carries the packet between the first host and the router over a two-hop path involving one intermediate packet switch. Suppose also that the second network is an Ethernet LAN. Sketch the sequence of IP and non-IP packets and frames that are generated as an IP packet goes from host 1 to host 2.

2.13. Does Ethernet provide connection-oriented or connectionless service?

2.14. Ethernet is a LAN so it is placed in the data link layer of the OSI reference model.

(a) How is the transfer of frames in Ethernet similar to the transfer of frames across a wire? How is it different?

(b) How is the transfer of frames in Ethernet similar to the transfer of frames in a packet-switching network? How is it different?

2.15. Suppose that a group of workstations is connected to an Ethernet LAN. If the workstations communicate only with each other, does it make sense to use IP in the workstations? Should the workstations run TCP directly over Ethernet? How is addressing handled?

2.16. Suppose two Ethernet LANs are interconnected by a box that operates as follows. The box has a table that tells it the physical addresses of the machines in each LAN. The box listens to frame transmissions on each LAN. If a frame is destined to a station at the other LAN, the box retransmits the frame onto the other LAN; otherwise, the box does nothing.

(a) Is the resulting network still a LAN? Does it belong in the data link layer or the network layer?

(b) Can the approach be extended to connect more than two LANs? If so, what problems arise as the number of LANs becomes large?

2.17. Suppose all laptops in a large city are to communicate using radio transmissions from a high antenna tower. Is the data link layer or network layer more appropriate for this situation? Now suppose the city is covered by a large number of small antennas covering smaller areas. Which layer is more appropriate?

2.18. Suppose that a host is connected to a connection-oriented packet-switching network and that it transmits a packet to a server along a path that traverses two packet switches. Suppose that each hop in the path involves a point-to-point link, that is, a wire. Show the sequence of network layer and data link layer PDUs that is generated as the packet travels from the host to the server.

2.19. Suppose an application layer entity wants to send an L-byte message to its peer process, using an existing TCP connection. The TCP segment consists of the message plus 20 bytes of header. The segment is encapsulated into an IP packet that has an additional 20 bytes of header. The IP packet in turn goes inside an Ethernet frame that has 18 bytes of header and trailer. What percentage of the transmitted bits in the physical layer corresponds to message information if $L = 100$ bytes? 500 bytes? 1000 bytes?

2.20. Suppose that the TCP entity receives a 1.5-megabyte file from the application layer and that the IP layer is willing to carry blocks of maximum size 1500 bytes. Calculate the amount of overhead incurred from segmenting the file into packet-sized units.

2.21. Suppose a TCP entity receives a digital voice stream from the application layer. The voice stream arrives at a rate of 8000 bytes/second. Suppose that TCP arranges bytes into block sizes that result in a total TCP and IP header overhead of 50 percent. How much delay is incurred by the first byte in each block?

2.22. How does the network layer in a connection-oriented packet-switching network differ from the network layer in a connectionless packet-switching network?

2.23. Identify session layer and presentation layer functions in the HTTP protocol.

2.24. Suppose we need a communication service to transmit real-time voice over the Internet. What features of TCP and what features of UDP are appropriate?

2.25. Consider the end-to-end IP packet transfer examples in Figure 2.15. Sketch the sequences of IP packets and Ethernet and PPP frames that are generated by the three examples of packet transfers: from the workstation to the server, from the server to the PC, and from the PC to the server. Include all relevant header information in the sketch.

2.26. Suppose a user has two browser applications active at the same time and suppose that the two applications are accessing the same server to retrieve HTTP documents at the same time. How does the server tell the difference between the two applications?

2.27. Consider the operation of nonpersistent HTTP and persistent HTTP.

(a) In nonpersistent HTTP (version 1.0): Each client/server interaction involves setting up a TCP connection, carrying out the HTTP exchange, and closing the TCP connection. Let T be the time that elapses from when a packet is sent from client to server to when the response is received. Find the rate at which HTTP exchanges can be made using nonpersistent HTTP.

(b) In persistent HTTP (version 1.1) the TCP connection is kept alive. Find the rate at which HTTP exchanges can be made if the client cannot send an additional request until it receives a response for each request.

(c) Repeat part (b) if the client is allowed to pipeline requests, that is, it does not have to wait for a response before sending a new request.

2.28. What is the difference between a physical address, a network address, and a domain name?

2.29. Explain how a DNS query proceeds if the local name server does not have the IP address for a given host when the following approaches are used. Assume an example where four machines are involved in ultimately resolving a given query.

(a) When a machine B cannot resolve an address in response to a query from A, machine B sends the query to another machine in the chain. When B receives the response, it forwards the result to B.

(b) When a machine B cannot resolve an address in response to a query from A, machine B sends a DNS reply to A with the IP address of the next machine in the chain, and machine A contacts that machine.

2.30. Suppose that the DNS system used a single centralized database handle all queries. Compare this centralized approach to the distributed approach in terms of reliability, throughput (volume of queries/second that can be processed), query response delay, and maintainability.

2.31. What is wrong with the following methods of assigning host id addresses?
(a) Copy the address from the machine in the next office.
(b) Modify the address from the machine in the next office.
(c) Use an example from the vendor's brochure.

2.32. Suppose a machine is attached to several physical networks. Why does it need a different IP address for each attachment?

2.33. Suppose a computer is moved from one department to another. Does the physical address need to change? Does the IP address need to change? Does it make a difference if the computer is a laptop?

2.34. Suppose the population of the world is 6 billion people and that there is an average of 1000 communicating devices per person. How many bits are required to assign a unique host address to each communicating device? Suppose that each device attaches to a single network and that each network on average has 10,000 devices. How many bits are required to provide unique network ids to each network?

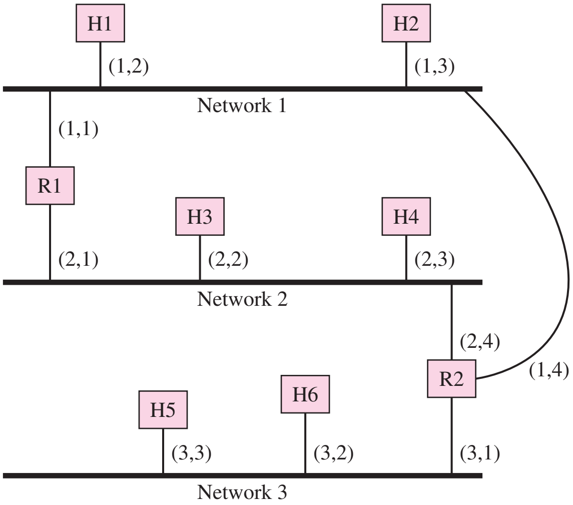
2.35. Can the Internet protocol be used to run a homogeneous packet-switching network, that is, a network with identical packet switches interconnected with point-to-point links?

2.36. Is it possible to build a homogeneous packet-switching network with Ethernet LANs interconnecting the packet switches? If so, can connection-oriented service be provided over such a network?

2.37. In telephone networks one basic network is used to provide worldwide communications. In the Internet a multiplicity of networks are interconnected to provide global connectivity. Compare these two approaches, namely, a single network versus an internetwork, in terms of the range of services that can be provided and the cost of establishing a worldwide network.

2.38. Consider an internetwork architecture that is defined using gateways/routers to communicate across networks but that uses a connection-oriented approach to packet switching? What functionality is required in the routers? Are any additional constraints imposed on the underlying networks?

2.39. The internet below consists of three LANs interconnected by two routers. Assume that the hosts and routers have the IP addresses as shown.
(a) Suppose that all traffic from network 3 that is destined to H1 is to be routed directly through router R2 and that all other traffic from network 3 is to go to network 2. What routing table entries should be present in the network 3 hosts and in R2?



(b) Suppose that all traffic from network 1 to network 3 is to be routed directly through R2. What routing table entries should be present in the network 1 hosts and in R2?

2.40. Explain why it is useful for application layer programs to have a "well-known" TCP port number?

2.41. Use a Web browser to connect to cnn.com. Explain what layers in the protocol stack are involved in the delivery of the video newscast.

2.42. Use a Web browser to connect to an audio program, say, www.rollingstone.com/radio/ (Rolling Stone Radio) or www.cbc.ca (CBC Radio). Explain what layers in the protocol stack are involved here. How does this situation differ from the delivery of video in problem 2.41?

2.43. Which of the TCP/IP transport protocol (UDP or TCP) would you select for the following applications: packet voice, file transfers, remote login, multicast communication (i.e., multiple destinations).

2.44. (a) Use the Telnet program to send an e-mail by directly interacting with your local mail server. The SMTP server has port 25. You can find the list of commands for the SMTP protocol in RFC 2821, which can be downloaded from www.ietf.org.

(b) Use Ethereal to capture and analyze the sequence of messages exchanged. Identify the various types of addresses for Ethernet, IP, and TCP PDUs. Examine the data in the Telnet messages to determine whether the login name and password are encrypted.

2.45. (a) Use the Telnet program to retrieve e-mail from your local mail server. The POP3 server has port 110. You can find the list of commands for the POP3 protocol in RFC 1939, which can be downloaded from www.ietf.org.

(b) Repeat Problem 2.44(b).

2.46. The `nslookup` program can be used to query the Internet domain name servers. Use this program to look up the IP address of www.utoronto.ca.

2.47. (a) Use PING to find the round-trip time to the home page of your university and to the home page of your department.
(b) Use Ethereal to capture the ICMP packets exchanged. Correlate the information in the packet capture with the information displayed by the PING result.

2.48. (a) Use netstat to find out the routing table for a host in your network.
(b) Use netstat to find the IP statistics for your host.

2.49. Suppose regularly spaced PING packets are sent to a remote host. What can you conclude from the following results?
(a) No replies arrive back.
(b) Some replies are lost.
(c) All replies arrive but with variable delays.
(d) What kind of statistics would be useful to calculate for the round-trip delays?

2.50. Suppose you want to test the response time of a specific web server. What attributes would such a measurement tool have? How would such a tool be designed?

2.51. A denial-of-service attack involves loading a network resource to the point where it becomes nonfunctional.
(a) Explain how PING can be used to carry out a denial-of-service attack.
(b) On October 21, 2002, the 13 DNS root servers were subject to a distributed denial-of-service attack. Explain the impact of the attack on the operation of the Internet if some of the servers are brought down; if all of the servers are brought down.

2.52. (a) Use a web browser to retrieve a file from a local web server.
(b) HTTP relies on ASCII characters. To verify the sequence of messages shown in Table 2.1, use the Telnet program to retrieve the same file from the local website.

2.53. Use Ethereal to capture the sequence of PDUs exchanged in problem 2.52 parts (a) and (b).
(a) Identify the Ethernet, IP, and TCP addresses of the machines involved in the exchange.
(b) Are there any DNS queries?
(c) Identify the TCP connection setup.
(d) Examine the contents of the HTTP GET and response messages.
(e) Examine how the TCP sequence numbers evolve over time.

2.54. Discuss the similarities and differences between the control connection in FTP and the remote control used to control a television. Can the FTP approach be used to provide VCR-type functionality to control the video from a video-on-demand service?

2.55. Use a Web browser to access the CAIDA Web page (http://www.caida.org/tools/taxonomy/) to retrieve the CAIDA measurement tool taxonomy document. You will find links there to many free Internet measurement tools and utilities.

2.56. Use traceroute to determine the path from your home PC to your university's main web page, while capturing the packets using Ethereal.
(a) Using the output from traceroute, try to identify how many different networks and service providers are traversed.
(b) Verify the operation of traceroute by examining the contents of the packets.

2.57. Run the UDP client and server programs from the Berkeley API section on different machines, record the round-trip delays with respect to the size of the data, and plot the results.

2.58. In the TCP example from the Berkeley API section, the message size communicated is fixed regardless of how many characters of actual information a user types. Even if the user wants to send only one character, the programs still sends 256 bytes of messages—clearly an inefficient method. One possible way to allow variable-length messages to be communicated is to indicate the end of a message by a unique character, called the sentinel. The receiver calls `read` for every character (or byte), compares each character with the sentinel value, and terminates after this special value is encountered. Modify the TCP client and server programs to handle variable-length messages using a sentinel value.

2.59. Another possible way to allow variable-length messages to be communicated is to precede the data to be transmitted by a header indicating the length of the data. After the header is decoded, the receiver knows how many more bytes it should read. Assuming the length of the header is two bytes, modify the TCP client and server programs to handle variable-length messages.

2.60. The UDP client program in the example from the Berkeley API section may wait forever if the datagram from the server never arrives. Modify the client program so that if the response from the server does not arrive after a certain timeout (say, 5 seconds), the `read` call is interrupted. The client then retransmits a datagram to the server and waits for a new response. If the client does not receive a response after a fixed number of trials (say, 10 trials), the client should print an error message and abandon the program. Hint: Use the `sigaction` and `alarm` functions.

2.61. Modify the UDP client to access a date-and-time server in a host local to your network. A date-and-time server provides client programs with the current day and time on demand. The system internal clock keeps the current day and time as a 32-bit integer. The time is incremented by the system (every second). When an application program (the server in this case) asks for the date or time, the system consults the internal clock and formats the date and time of day in human-readable format. Sending any datagram to a date-and-time server is equivalent to making a request for the current date and time; the server responds by returning a UDP message containing the current date and time. The date-and-time server can be accessed in UDP port 13.

2.62. Write a file transfer application that runs over UDP. Assume the transfer occurs over a local area network so reliable transfer is not a concern. Assume also that UDP will accept at most 500 bytes/datagram. Implement a server that opens a socket and listens for incoming data at a particular port number. Implement a client that reads a file from the file system and transfers the file to the server. When the server receives the client's data, it writes this data to a file. Include a means for the client to indicate the end of transmission.