CHAPTER 26

Standard Client-Server Protocols

After introducing the application layer in the previous chapter, we discuss some standard application-layer protocols in this chapter. During the lifetime of the Internet, several client-server application programs have been developed. We do not have to redefine them, but we need to understand what they do. For each application, we also need to know the options available to us. The study of these applications and the ways they provide different services can help us to create customized applications in the future.

We have selected six standard application programs in this section. Some other applications have been or will be discussed in other chapters. Dynamic Host Configuration Protocol (DHCP) was discussed in Chapter 18 and Simple Network Management Protocol (SNMP) will be discussed in Chapter 27.

This chapter is made of six sections:

- The first section introduces the World Wide Web. It then discusses the HyperText Transfer Protocol, the most common client-server application program used in relation to the World Wide Web.
- The second section discusses the File Transfer Protocol, which is the standard protocol provided by TCP/IP for copying a file from one host to another.
- The third section discusses electronic mail, which involves two protocols: SMPT and POP. As we will see, the nature of this application is different from the other two previous applications. We need two different protocols to handle electronic mail.
- The fourth section discusses TELNET, a general client-server program that allows users to log in to a remote machine and use any application available on the remote host.
- The fifth section discusses Secure Shell, which can be used as a secured TELNET, but it can also provide a secure tunnel for other applications.
- The sixth section talks about the Domain Name System, which acts as the directory system in the Internet. It maps the name of an entity to its IP address.

26.1 WORLD WIDE WEB AND HTTP

In this section, we first introduce the **World Wide Web** (abbreviated WWW or Web). We then discuss the HyperText Transfer Protocol (HTTP), the most common client-server application program used in relation to the Web.

26.1.1 World Wide Web

The idea of the Web was first proposed by Tim Berners-Lee in 1989 at *CERN*†, the European Organization for Nuclear Research, to allow several researchers at different locations throughout Europe to access each others' researches. The commercial Web started in the early 1990s.

The Web today is a repository of information in which the documents, called ***web pages,*** are distributed all over the world and related documents are linked together. The popularity and growth of the Web can be related to two terms in the above statement: *distributed* and *linked*. Distribution allows the growth of the Web. Each web server in the world can add a new web page to the repository and announce it to all Internet users without overloading a few servers. Linking allows one web page to refer to another web page stored in another server somewhere else in the world. The linking of web pages was achieved using a concept called ***hypertext,*** which was introduced many years before the advent of the Internet. The idea was to use a machine that automatically retrieved another document stored in the system when a link to it appeared in the document. The Web implemented this idea electronically to allow the linked document to be retrieved when the link was clicked by the user. Today, the term *hypertext,* coined to mean linked text documents, has been changed to ***hypermedia,*** to show that a web page can be a text document, an image, an audio file, or a video file.

The purpose of the Web has gone beyond the simple retrieving of linked documents. Today, the Web is used to provide electronic shopping and gaming. One can use the Web to listen to radio programs or view television programs whenever one desires without being forced to listen to or view these programs when they are broadcast.

Architecture

The WWW today is a distributed client-server service, in which a client using a browser can access a service using a server. However, the service provided is distributed over many locations called *sites.* Each site holds one or more web pages. Each web page, however, can contain some links to other web pages in the same or other sites. In other words, a web page can be simple or composite. A simple web page has no links to other web pages; a composite web page has one or more links to other web pages. Each web page is a file with a name and address.

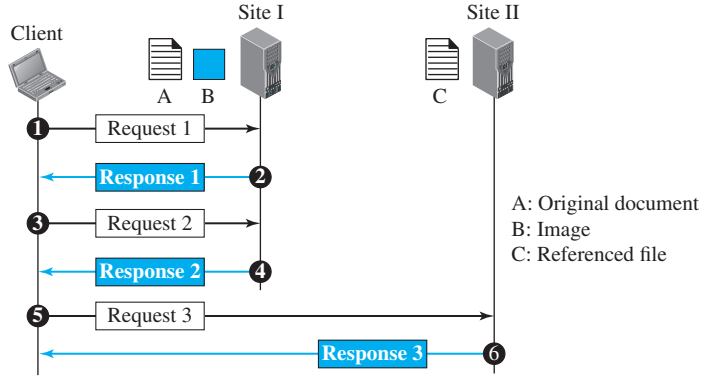
Example 26.1

Assume we need to retrieve a scientific document that contains one reference to another text file and one reference to a large image. Figure 26.1 shows the situation.

The main document and the image are stored in two separate files (file A and file B) in the same site; the referenced text file (file C) is stored in another site. Since we are dealing with three

† In French: *Conseil Européen pour la Recherche Nucléaire*

Figure 26.1 *Example 26.1*

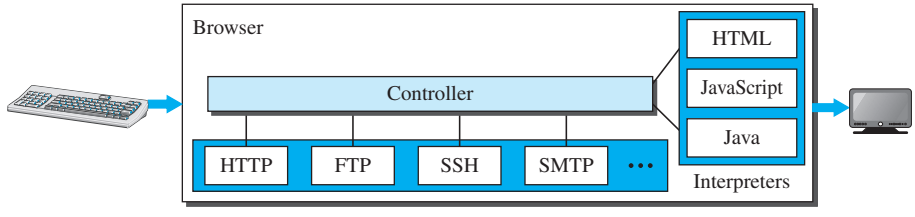


different files, we need three transactions if we want to see the whole document. The first transaction (request/response) retrieves a copy of the main document (file A), which has references (pointers) to the second and third files. When a copy of the main document is retrieved and browsed, the user can click on the reference to the image to invoke the second transaction and retrieve a copy of the image (file B). If the user needs to see the contents of the referenced text file, she can click on its reference (pointer) invoking the third transaction and retrieving a copy of file C. Note that although files A and B both are stored in site I, they are independent files with different names and addresses. Two transactions are needed to retrieve them. A very important point we need to remember is that file A, file B, and file C in Example 26.1 are independent web pages, each with independent names and addresses. Although references to file B or C are included in file A, it does not mean that each of these files cannot be retrieved independently. A second user can retrieve file B with one transaction. A third user can retrieve file C with one transaction.

Web Client (Browser)

A variety of vendors offer commercial **browsers** that interpret and display a web page, and all of them use nearly the same architecture. Each browser usually consists of three parts: a controller, client protocols, and interpreters. (see Figure 26.2).

Figure 26.2 *Browser*



The controller receives input from the keyboard or the mouse and uses the client programs to access the document. After the document has been accessed, the controller uses one of the interpreters to display the document on the screen. The client protocol can be one of the protocols described later, such as HTTP or FTP. The interpreter can be HTML, Java, or JavaScript, depending on the type of document. Some commercial browsers include Internet Explorer, Netscape Navigator, and Firefox.

Web Server

The web page is stored at the server. Each time a request arrives, the corresponding document is sent to the client. To improve efficiency, servers normally store requested files in a cache in memory; memory is faster to access than a disk. A server can also become more efficient through multithreading or multiprocessing. In this case, a server can answer more than one request at a time. Some popular web servers include Apache and Microsoft Internet Information Server.

Uniform Resource Locator (URL)

A web page, as a file, needs to have a unique identifier to distinguish it from other web pages. To define a web page, we need three identifiers: *host, port,* and *path*. However, before defining the web page, we need to tell the browser what client-server application we want to use, which is called the *protocol*. This means we need four identifiers to define the web page. The first is the type of vehicle to be used to fetch the web page; the last three make up the combination that defines the destination object (web page).

- ***Protocol.*** The first identifier is the abbreviation for the client-server program that we need in order to access the web page. Although most of the time the protocol is HTTP (HyperText Transfer Protocol), which we will discuss shortly, we can also use other protocols such as FTP (File Transfer Protocol).
- ***Host.*** The host identifier can be the IP address of the server or the unique name given to the server. IP addresses can be defined in dotted decimal notation, as described in Chapter 18 (such as 64.23.56.17); the name is normally the domain name that uniquely defines the host, such as *forouzan.com,* which we discuss in Domain Name System (DNS) later in this chapter.
- ***Port.*** The port, a 16-bit integer, is normally predefined for the client-server application. For example, if the HTTP protocol is used for accessing the web page, the well-known port number is 80. However, if a different port is used, the number can be explicitly given.
- ***Path.*** The path identifies the location and the name of the file in the underlying operating system. The format of this identifier normally depends on the operating system. In UNIX, a path is a set of directory names followed by the file name, all separated by a slash. For example, */top/next/last/myfile* is a path that uniquely defines a file named *myfile,* stored in the directory *last,* which itself is part of the directory *next,* which itself is under the directory *top*. In other words, the path lists the directories from the top to the bottom, followed by the file name.

To combine these four pieces together, the **uniform resource locator (URL)** has been designed; it uses three different separators between the four pieces as shown below:

protocol://host/path	**Used most of the time**
protocol://host:port/path	**Used when port number is needed**

Example 26.2

The URL ***http://www.mhhe.com/compsci/forouzan/*** defines the web page related to one of the authors of this book. The string ***www.mhhe.com*** is the name of the computer in the McGraw-Hill company (the three letters ***www*** are part of the host name and are added to the commercial host). The path is ***compsci/forouzan/,*** which defines Forouzan's web page under the directory ***compsci*** (computer science).

Web Documents

The documents in the WWW can be grouped into three broad categories: static, dynamic, and active.

Static Documents

Static documents are fixed-content documents that are created and stored in a server. The client can get a copy of the document only. In other words, the contents of the file are determined when the file is created, not when it is used. Of course, the contents in the server can be changed, but the user cannot change them. When a client accesses the document, a copy of the document is sent. The user can then use a browser to see the document. Static documents are prepared using one of several languages: *HyperText Markup Language* (HTML), *Extensible Markup Language* (XML), *Extensible Style Language* (XSL), and *Extensible Hypertext Markup Language* (XHTML). We discuss these languages in Appendix C.

Dynamic Documents

A **dynamic document** is created by a web server whenever a browser requests the document. When a request arrives, the web server runs an application program or a script that creates the dynamic document. The server returns the result of the program or script as a response to the browser that requested the document. Because a fresh document is created for each request, the contents of a dynamic document may vary from one request to another. A very simple example of a dynamic document is the retrieval of the time and date from a server. Time and date are kinds of information that are dynamic in that they change from moment to moment. The client can ask the server to run a program such as the *date* program in UNIX and send the result of the program to the client. Although the *Common Gateway Interface* (*CGI*) was used to retrieve a dynamic document in the past, today's options include one of the scripting languages such as *Java Server Pages* (*JSP*), which uses the Java language for scripting, or *Active Server Pages* (*ASP*), a Microsoft product that uses Visual Basic language for scripting, or *ColdFusion,* which embeds queries in a Structured Query Language (SQL) database in the HTML document.

Active Documents

For many applications, we need a program or a script to be run at the client site. These are called ***active documents*****.** For example, suppose we want to run a program that creates animated graphics on the screen or a program that interacts with the user. The program

definitely needs to be run at the client site where the animation or interaction takes place. When a browser requests an active document, the server sends a copy of the document or a script. The document is then run at the client (browser) site. One way to create an active document is to use *Java applets,* a program written in Java on the server. It is compiled and ready to be run. The document is in bytecode (binary) format. Another way is to use *JavaScripts* but download and run the script at the client site.

26.1.2 HyperText Transfer Protocol (HTTP)

The **HyperText Transfer Protocol (HTTP)** is used to define how the client-server programs can be written to retrieve web pages from the Web. An HTTP client sends a request; an HTTP server returns a response. The server uses the port number 80; the client uses a temporary port number. HTTP uses the services of TCP, which, as discussed before, is a connection-oriented and reliable protocol. This means that, before any transaction between the client and the server can take place, a connection needs to be established between them. After the transaction, the connection should be terminated. The client and server, however, do not need to worry about errors in messages exchanged or loss of any message, because the TCP is reliable and will take care of this matter, as we saw in Chapter 24.

Nonpersistent versus Persistent Connections

As we discussed in the previous section, the hypertext concept embedded in web page documents may require several requests and responses. If the web pages, objects to be retrieved, are located on different servers, we do not have any other choice than to create a new TCP connection for retrieving each object. However, if some of the objects are located on the same server, we have two choices: to retrieve each object using a new TCP connection or to make a TCP connection and retrieve them all. The first method is referred to as a *nonpersistent connection,* the second as a *persistent connection*. HTTP, prior to version 1.1, specified *nonpersistent* connections, while *persistent* connections are the default in version 1.1, but it can be changed by the user.

Nonpersistent Connections

In a **nonpersistent connection**, one TCP connection is made for each request/response. The following lists the steps in this strategy:

1. The client opens a TCP connection and sends a request.
2. The server sends the response and closes the connection.
3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

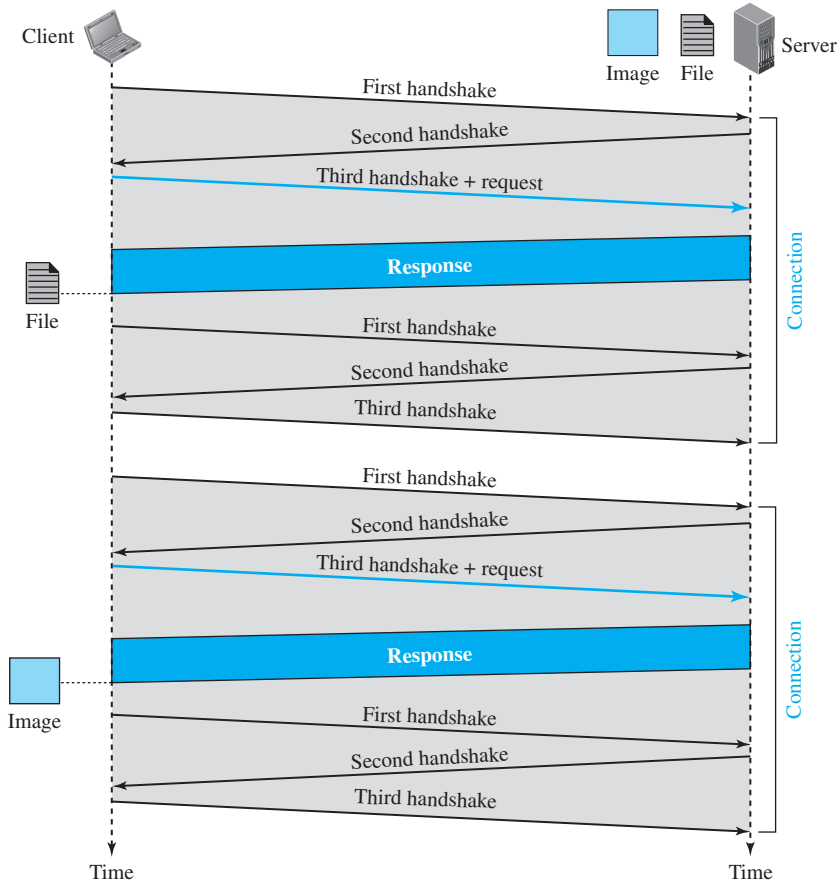
In this strategy, if a file contains links to N different pictures in different files (all located on the same server), the connection must be opened and closed $N + 1$ times. The nonpersistent strategy imposes high overhead on the server because the server needs $N + 1$ different buffers each time a connection is opened.

Example 26.3

Figure 26.3 shows an example of a nonpersistent connection. The client needs to access a file that contains one link to an image. The text file and image are located on the same server. Here we need two connections. For each connection, TCP requires at least three handshake messages to

establish the connection, but the request can be sent with the third one. After the connection is established, the object can be transferred. After receiving an object, another three handshake messages are needed to terminate the connection, as we saw in Chapter 24. This means that the

Figure 26.3 *Example 26.3*



client and server are involved in two connection establishments and two connection terminations. If the transaction involves retrieving 10 or 20 objects, the round trip times spent for these handshakes add up to a big overhead. When we describe the client-server programming at the end of the chapter, we will show that for each connection the client and server need to allocate extra resources such as buffers and variables. This is another burden on both sites, but especially on the server site.

Persistent Connections

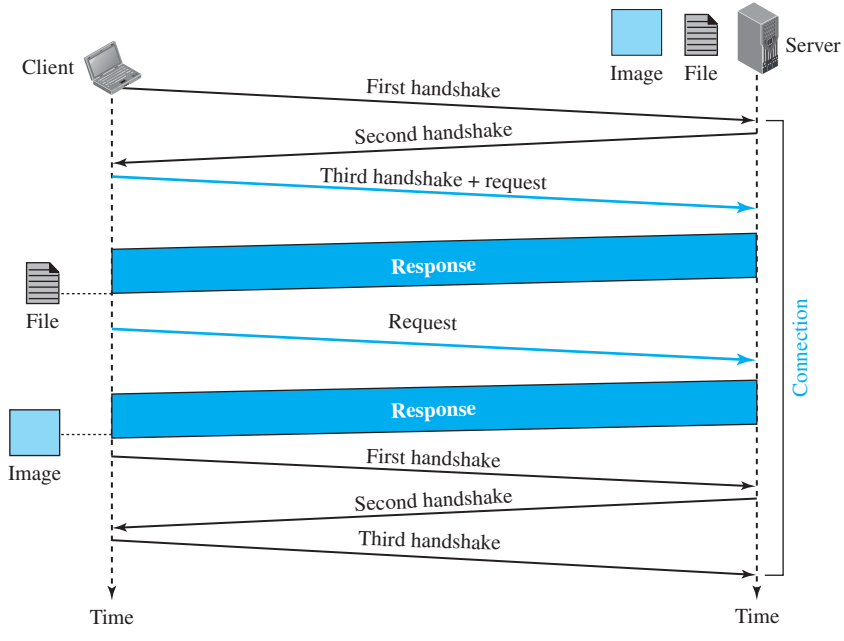
HTTP version 1.1 specifies a **persistent connection** by default. In a persistent connection, the server leaves the connection open for more requests after sending a response.

The server can close the connection at the request of a client or if a time-out has been reached. The sender usually sends the length of the data with each response. However, there are some occasions when the sender does not know the length of the data. This is the case when a document is created dynamically or actively. In these cases, the server informs the client that the length is not known and closes the connection after sending the data so the client knows that the end of the data has been reached. Time and resources are saved using persistent connections. Only one set of buffers and variables needs to be set for the connection at each site. The round trip time for connection establishment and connection termination is saved.

Example 26.4

Figure 26.4 shows the same scenario as in Example 26.3, but using a persistent connection. Only one connection establishment and connection termination is used, but the request for the image is sent separately.

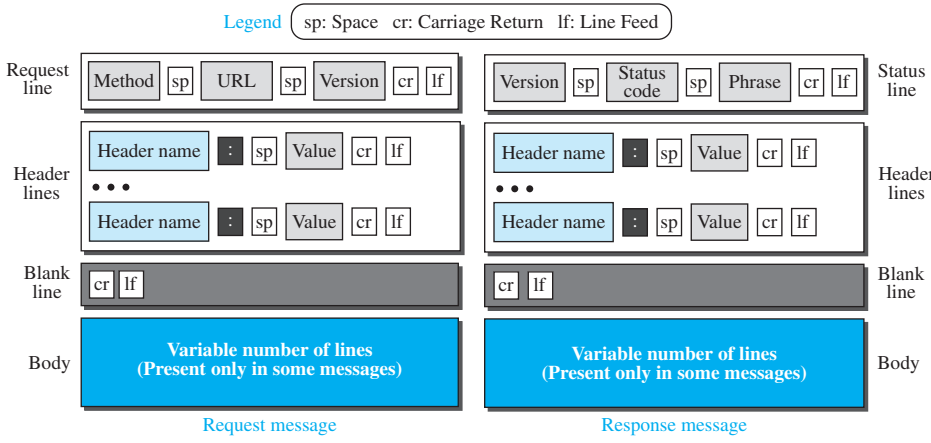
Figure 26.4 *Example 26.4*



Message Formats

The HTTP protocol defines the format of the request and response messages, as shown in Figure 26.5. We have put the two formats next to each other for comparison. Each message is made of four sections. The first section in the request message is called the *request line;* the first section in the response message is called the *status line*. The other three sections have the same names in the request and response messages. However, the

Figure 26.5 *Formats of the request and response messages*



similarities between these sections are only in the names; they may have different contents. We discuss each message type separately.

Request Message

As we said before, the first line in a request message is called a request line. There are three fields in this line separated by one space and terminated by two characters (carriage return and line feed) as shown in Figure 26.5. The fields are called *method, URL*, and *version*.

The method field defines the request types. In version 1.1 of HTTP, several methods are defined, as shown in Table 26.1. Most of the time, the client uses the GET method to send a request. In this case, the body of the message is empty. The HEAD method is used when the client needs only some information about the web page from the server, such as the last time it was modified. It can also be used to test the validity of a URL. The response message in this case has only the header section; the body section is empty. The PUT method is the inverse of the GET method; it allows the client to post a new web page on the server (if permitted). The POST method is similar to the PUT method, but it is used to send some information to the server to be added to the web page or to modify the web page. The TRACE method is used for debugging; the client asks the server to echo back the request to check whether the server is getting the requests. The DELETE method allows the client to delete a web page on the server if the client has permission to do so. The CONNECT method was originally made as a reserve method; it may be used by proxy servers, as discussed later. Finally, the OPTIONS method allows the client to ask about the properties of a web page.

The second field, URL, was discussed earlier in the chapter. It defines the address and name of the corresponding web page. The third field, version, gives the version of the protocol; the most current version of HTTP is 1.1.

Table 26.1 *Methods*

Method	*Action*
GET	Requests a document from the server
HEAD	Requests information about a document but not the document itself
PUT	Sends a document from the client to the server
POST	Sends some information from the client to the server
TRACE	Echoes the incoming request
DELETE	Removes the web page
CONNECT	Reserved
OPTIONS	Inquires about available options

After the request line, we can have zero or more *request header* lines. Each header line sends additional information from the client to the server. For example, the client can request that the document be sent in a special format. Each header line has a header name, a colon, a space, and a header value (see Figure 26.5). Table 26.2 shows some header names commonly used in a request. The value field defines the values associated with each header name. The list of values can be found in the corresponding RFCs.

The body can be present in a request message. Usually, it contains the comment to be sent or the file to be published on the website when the method is PUT or POST.

Table 26.2 *Request header names*

Header	*Description*
User-agent	Identifies the client program
Accept	Shows the media format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
Host	Shows the host and port number of the client
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Cookie	Returns the cookie to the server (explained later)
If-Modified-Since	If the file is modified since a specific date

Response Message

The format of the response message is also shown in Figure 26.5. A response message consists of a status line, header lines, a blank line, and sometimes a body. The first line in a response message is called the *status line*. There are three fields in this line separated by spaces and terminated by a carriage return and line feed. The first field defines the version of HTTP protocol, currently 1.1. The status code field defines the status of the request. It consists of three digits. Whereas the codes in the 100 range are only informational, the codes in the 200 range indicate a successful request. The codes in the 300 range redirect the client to another URL, and the codes

in the 400 range indicate an error at the client site. Finally, the codes in the 500 range indicate an error at the server site. The status phrase explains the status code in text form.

After the status line, we can have zero or more *response header* lines. Each header line sends additional information from the server to the client. For example, the sender can send extra information about the document. Each header line has a header name, a colon, a space, and a header value. We will show some header lines in the examples at the end of this section. Table 26.3 shows some header names commonly used in a response message.

Table 26.3 *Response header names*

Header	*Description*
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Server	Gives information about the server
Set-Cookie	The server asks the client to save a cookie
Content-Encoding	Specifies the encoding scheme
Content-Language	Specifies the language
Content-Length	Shows the length of the document
Content-Type	Specifies the media type
Location	To ask the client to send the request to another site
Accept-Ranges	The server will accept the requested byte-ranges
Last-modified	Gives the date and time of the last change

The body contains the document to be sent from the server to the client. The body is present unless the response is an error message.

Example 26.5

This example retrieves a document (see Figure 26.6). We use the GET method to retrieve an image with the path ***/usr/bin/image1*.** The request line shows the method (GET), the URL, and the HTTP version (1.1). The header has two lines that show that the client can accept images in the GIF or JPEG format. The request does not have a body. The response message contains the status line and four lines of header. The header lines define the date, server, content encoding (MIME version, which will be described in electronic mail), and length of the document. The body of the document follows the header.

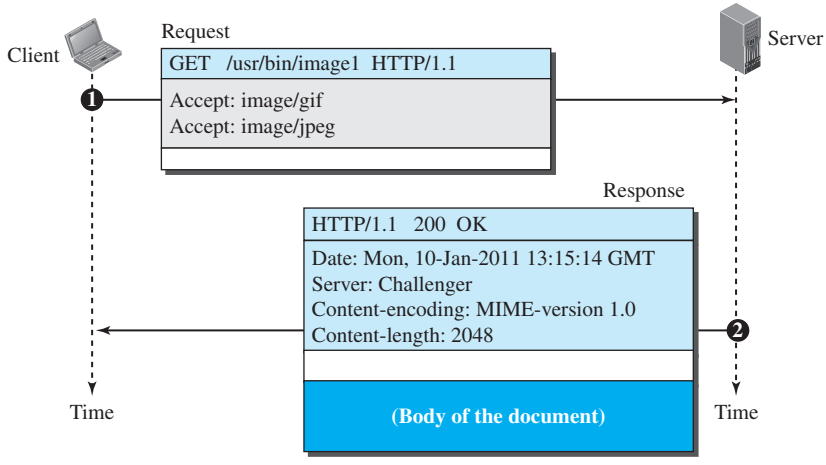
Example 26.6

In this example, the client wants to send a web page to be posted on the server. We use the PUT method. The request line shows the method (PUT), URL, and HTTP version (1.1). There are four lines of headers. The request body contains the web page to be posted. The response message contains the status line and four lines of headers. The created document, which is a CGI document, is included as the body (see Figure 26.7).

Conditional Request

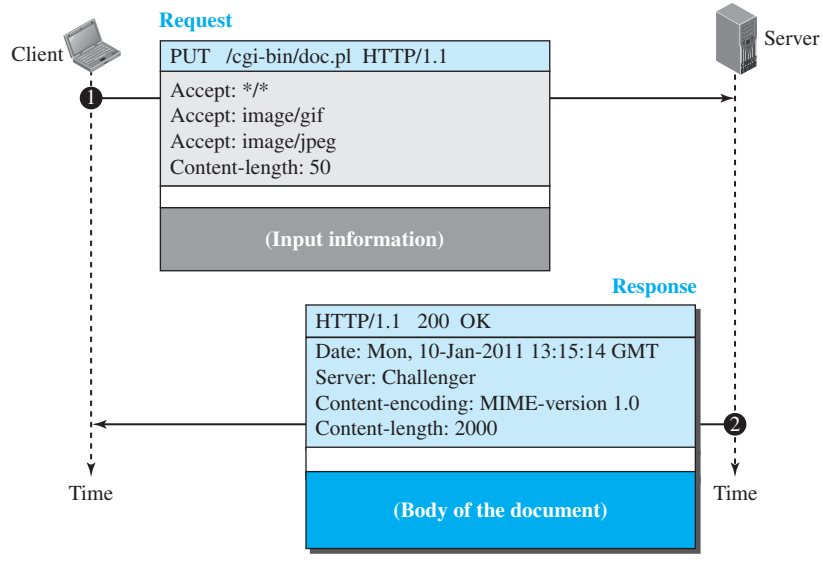
A client can add a condition in its request. In this case, the server will send the requested web page if the condition is met or inform the client otherwise. One of the most common conditions imposed by the client is the time and date the web

Figure 26.6 *Example 26.5*



page is modified. The client can send the header line *If-Modified-Since* with the request to tell the server that it needs the page only if it is modified after a certain point in time.

Figure 26.7 *Example 26.6*



Example 26.7

The following shows how a client imposes the modification data and time condition on a request.

GET http://www.commonServer.com/information/file1 HTTP/1.1	**Request line**
If-Modified-Since: Thu, Sept 04 00:00:00 GMT	**Header line**
	Blank line

The status line in the response shows the file was not modified after the defined point in time. The body of the response message is also empty.

HTTP/1.1 304 Not Modified	**Status line**
Date: Sat, Sept 06 08 16:22:46 GMT	**First header line**
Server: commonServer.com	**Second header line**
	Blank line
(Empty Body)	**Empty body**

Cookies

The World Wide Web was originally designed as a stateless entity. A client sends a request; a server responds. Their relationship is over. The original purpose of the Web, retrieving publicly available documents, exactly fits this design. Today the Web has other functions that need to remember some information about the clients; some are listed below:

- Websites are being used as *electronic stores* that allow users to browse through the store, select wanted items, put them in an electronic cart, and pay at the end with a credit card.
- Some websites need to allow access to *registered clients* only.
- Some websites are used as *portals:* the user selects the web pages he wants to see.
- Some websites are just *advertising* agencies.

For these purposes, the **cookie** mechanism was devised.

Creating and Storing Cookies

The creation and storing of cookies depend on the implementation; however, the principle is the same.

1. When a server receives a request from a client, it stores information about the client in a file or a string. The information may include the domain name of the client, the contents of the cookie (information the server has gathered about the client such as name, registration number, and so on), a timestamp, and other information depending on the implementation.
2. The server includes the cookie in the response that it sends to the client.
3. When the client receives the response, the browser stores the cookie in the cookie directory, which is sorted by the server domain name.

Using Cookies

When a client sends a request to a server, the browser looks in the cookie directory to see if it can find a cookie sent by that server. If found, the cookie is included in the

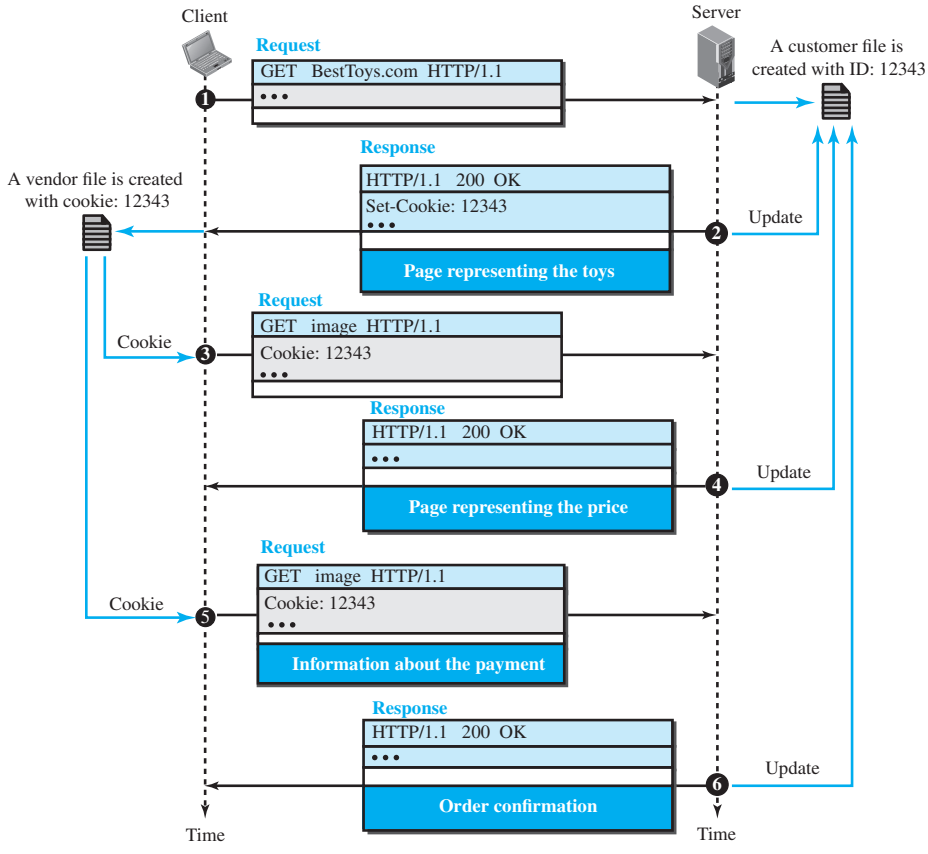
request. When the server receives the request, it knows that this is an old client, not a new one. Note that the contents of the cookie are never read by the browser or disclosed to the user. It is a cookie *made* by the server and *eaten* by the server. Now let us see how a cookie is used for the four previously mentioned purposes:

- An *electronic store* (e-commerce) can use a cookie for its client shoppers. When a client selects an item and inserts it in a cart, a cookie that contains information about the item, such as its number and unit price, is sent to the browser. If the client selects a second item, the cookie is updated with the new selection information, and so on. When the client finishes shopping and wants to check out, the last cookie is retrieved and the total charge is calculated.
- The site that restricts access to *registered clients* only sends a cookie to the client when the client registers for the first time. For any repeated access, only those clients that send the appropriate cookie are allowed.
- A web *portal* uses the cookie in a similar way. When a user selects her favorite pages, a cookie is made and sent. If the site is accessed again, the cookie is sent to the server to show what the client is looking for.
- A cookie is also used by *advertising* agencies. An advertising agency can place banner ads on some main website that is often visited by users. The advertising agency supplies only a URL that gives the advertising agency's address instead of the banner itself. When a user visits the main website and clicks the icon of a corporation, a request is sent to the advertising agency. The advertising agency sends the requested banner, but it also includes a cookie with the ID of the user. Any future use of the banners adds to the database that profiles the Web behavior of the user. The advertising agency has compiled the interests of the user and can sell this information to other parties. This use of cookies has made them very controversial. Hopefully, some new regulations will be devised to preserve the privacy of users.

Example 26.8

Figure 26.8 shows a scenario in which an electronic store can benefit from the use of cookies. Assume a shopper wants to buy a toy from an electronic store named BestToys. The shopper browser (client) sends a request to the BestToys server. The server creates an empty shopping cart (a list) for the client and assigns an ID to the cart (for example, 12343). The server then sends a response message, which contains the images of all toys available, with a link under each toy that selects the toy if it is being clicked. This response message also includes the Set-Cookie header line whose value is 12343. The client displays the images and stores the cookie value in a file named BestToys. The cookie is not revealed to the shopper. Now the shopper selects one of the toys and clicks on it. The client sends a request, but includes the ID 12343 in the Cookie header line. Although the server may have been busy and forgotten about this shopper, when it receives the request and checks the header, it finds the value 12343 as the cookie. The server knows that the customer is not new; it searches for a shopping cart with ID 12343. The shopping cart (list) is opened and the selected toy is inserted in the list. The server now sends another response to the shopper to tell her the total price and ask her to provide payment. The shopper provides information about her credit card and sends a new request with the ID 12343 as the cookie value. When the request arrives at the server, it again sees the ID 12343, and accepts the order and the payment and sends a confirmation in a response. Other information about the client is stored in

Figure 26.8 *Example 26.8*



the server. If the shopper accesses the store sometime in the future, the client sends the cookie again; the store retrieves the file and has all the information about the client.

Web Caching: Proxy Servers

HTTP supports **proxy servers**. A proxy server is a computer that keeps copies of responses to recent requests. The HTTP client sends a request to the proxy server. The proxy server checks its cache. If the response is not stored in the cache, the proxy server sends the request to the corresponding server. Incoming responses are sent to the proxy server and stored for future requests from other clients.

The proxy server reduces the load on the original server, decreases traffic, and improves latency. However, to use the proxy server, the client must be configured to access the proxy instead of the target server.

Note that the proxy server acts as both server and client. When it receives a request from a client for which it has a response, it acts as a server and sends the response to the client. When it receives a request from a client for which it does not have a response, it first acts as a client and sends a request to the target server. When the response has been received, it acts again as a server and sends the response to the client.

Proxy Server Location

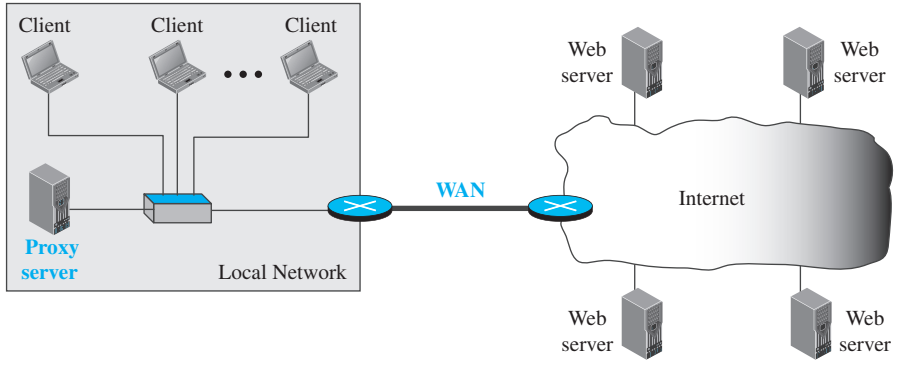
The proxy servers are normally located at the client site. This means that we can have a hierarchy of proxy servers, as shown below:

1. A client computer can also be used as a proxy server, in a small capacity, that stores responses to requests often invoked by the client.
2. In a company, a proxy server may be installed on the computer LAN to reduce the load going out of and coming into the LAN.
3. An ISP with many customers can install a proxy server to reduce the load going out of and coming into the ISP network.

Example 26.9

Figure 26.9 shows an example of a use of a proxy server in a local network, such as the network

Figure 26.9 *Example of a proxy server*



on a campus or in a company. The proxy server is installed in the local network. When an HTTP request is created by any of the clients (browsers), the request is first directed to the proxy server. If the proxy server already has the corresponding web page, it sends the response to the client. Otherwise, the proxy server acts as a client and sends the request to the web server in the Internet. When the response is returned, the proxy server makes a copy and stores it in its cache before sending it to the requesting client.

Cache Update

A very important question is how long a response should remain in the proxy server before being deleted and replaced. Several different strategies are used for this purpose. One solution is to store the list of sites whose information remains the same for a while. For example, a news agency may change its news page every morning. This means that

a proxy server can get the news early in the morning and keep it until the next day. Another recommendation is to add some headers to show the last modification time of the information. The proxy server can then use the information in this header to guess how long the information would be valid.

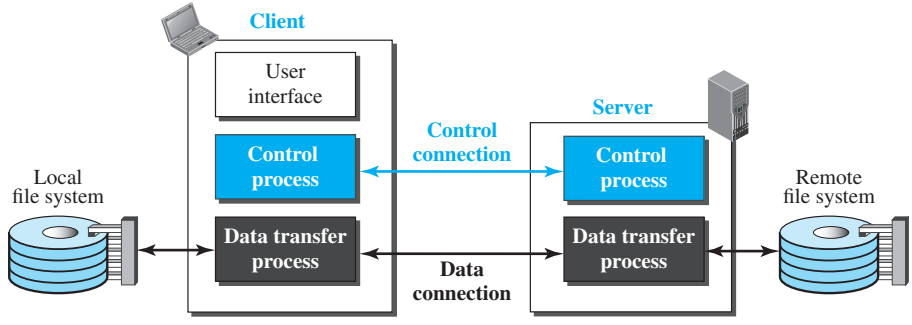
HTTP Security

HTTP per se does not provide security. However, as we show in Chapter 32, HTTP can be run over the Secure Socket Layer (SSL). In this case, HTTP is referred to as HTTPS. HTTPS provides confidentiality, client and server authentication, and data integrity.

26.2 FTP

File Transfer Protocol (FTP) is the standard protocol provided by TCP/IP for copying a file from one host to another. Although transferring files from one system to another seems simple and straightforward, some problems must be dealt with first. For example, two systems may use different file name conventions. Two systems may have different ways to represent data. Two systems may have different directory structures. All of these problems have been solved by FTP in a very simple and elegant approach. Although we can transfer files using HTTP, FTP is a better choice to transfer large files or to transfer files using different formats. Figure 26.10 shows the

Figure 26.10 *FTP*



basic model of FTP. The client has three components: the user interface, the client control process, and the client data transfer process. The server has two components: the server control process and the server data transfer process. The control connection is made between the control processes. The data connection is made between the data transfer processes.

Separation of commands and data transfer makes FTP more efficient. The control connection uses very simple rules of communication. We need to transfer only a line of command or a line of response at a time. The data connection, on the other hand, needs more complex rules due to the variety of data types transferred.

26.2.1 Two Connections

The two connections in FTP have different lifetimes. The control connection remains connected during the entire interactive FTP session. The data connection is opened and then closed for each file transfer activity. It opens each time commands that involve transferring files are used, and it closes when the file is transferred. In other words, when a user starts an FTP session, the control connection opens. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred. FTP uses two well-known TCP ports: port 21 is used for the control connection, and port 20 is used for the data connection.

26.2.2 Control Connection

For control communication, FTP uses the same approach as TELNET (discussed later). It uses the NVT ASCII character set as used by TELNET. Communication is achieved through commands and responses. This simple method is adequate for the control connection because we send one command (or response) at a time. Each line is terminated with a two-character (carriage return and line feed) end-of-line token.

During this control connection, commands are sent from the client to the server and responses are sent from the server to the client. Commands, which are sent from the FTP client control process, are in the form of ASCII uppercase, which may or may not be followed by an argument. Some of the most common commands are shown in Table 26.4.

Table 26.4 *Some FTP commands*

Command	*Argument(s)*	*Description*
ABOR		Abort the previous command
CDUP		Change to parent directory
CWD	Directory name	Change to another directory
DELE	File name	Delete a file
LIST	Directory name	List subdirectories or files
MKD	Directory name	Create a new directory
PASS	User password	Password
PASV		Server chooses a port
PORT	Port identifier	Client chooses a port
PWD		Display name of current directory
QUIT		Log out of the system
RETR	File name(s)	Retrieve files; files are transferred from server to client
RMD	Directory name	Delete a directory
RNFR	File name (old)	Identify a file to be renamed
RNTO	File name (new)	Rename the file
STOR	File name(s)	Store files; file(s) are transferred from client to server
STRU	**F**, **R**, or **P**	Define data organization (**F**: file, **R**: record, or **P**: page)
TYPE	**A**, **E**, **I**	Default file type (**A**: ASCII, **E**: EBCDIC, **I**: image)
USER	User ID	User information
MODE	**S**, **B**, or **C**	Define transmission mode (**S**: stream, **B**: block, or **C**: compressed

Every FTP command generates at least one response. A response has two parts: a three-digit number followed by text. The numeric part defines the code; the text part defines needed parameters or further explanations. The first digit defines the status of the command. The second digit defines the area in which the status applies. The third digit provides additional information. Table 26.5 shows some common responses.

Table 26.5 *Some responses in FTP*

Code	*Description*	*Code*	*Description*
125	Data connection open	**250**	Request file action OK
150	File status OK	**331**	User name OK; password is needed
200	Command OK	**425**	Cannot open data connection
220	Service ready	**450**	File action not taken; file not available
221	Service closing	**452**	Action aborted; insufficient storage
225	Data connection open	**500**	Syntax error; unrecognized command
226	Closing data connection	**501**	Syntax error in parameters or arguments
230	User login OK	**530**	User not logged in

26.2.3 Data Connection

The data connection uses the well-known port 20 at the server site. However, the creation of a data connection is different from the control connection. The following shows the steps:

1. The client, not the server, issues a passive open using an ephemeral port. This must be done by the client because it is the client that issues the commands for transferring files.
2. Using the PORT command the client sends this port number to the server.
3. The server receives the port number and issues an active open using the well-known port 20 and the received ephemeral port number.

Communication over Data Connection

The purpose and implementation of the data connection are different from those of the control connection. We want to transfer files through the data connection. The client must define the type of file to be transferred, the structure of the data, and the transmission mode. Before sending the file through the data connection, we prepare for transmission through the control connection. The heterogeneity problem is resolved by defining three attributes of communication: file type, data structure, and transmission mode.

File Type

FTP can transfer one of the following file types across the data connection: ASCII file, EBCDIC file, or image file.

Data Structure

FTP can transfer a file across the data connection using one of the following interpretations of the structure of the data: *file structure, record structure,* or *page structure*. The file structure format (used by default) has no structure. It is a continuous stream of bytes. In the record structure, the file is divided into *records*. This can be used only with text files. In the page structure, the file is divided into pages, with each page having a page number and a page header. The pages can be stored and accessed randomly or sequentially.

Transmission Mode

FTP can transfer a file across the data connection using one of the following three transmission modes: *stream mode, block mode,* or *compressed mode*. The stream mode is the default mode; data are delivered from FTP to TCP as a continuous stream of bytes. In the block mode, data can be delivered from FTP to TCP in blocks. In this case, each block is preceded by a 3-byte header. The first byte is called the *block descriptor*; the next two bytes define the size of the block in bytes.

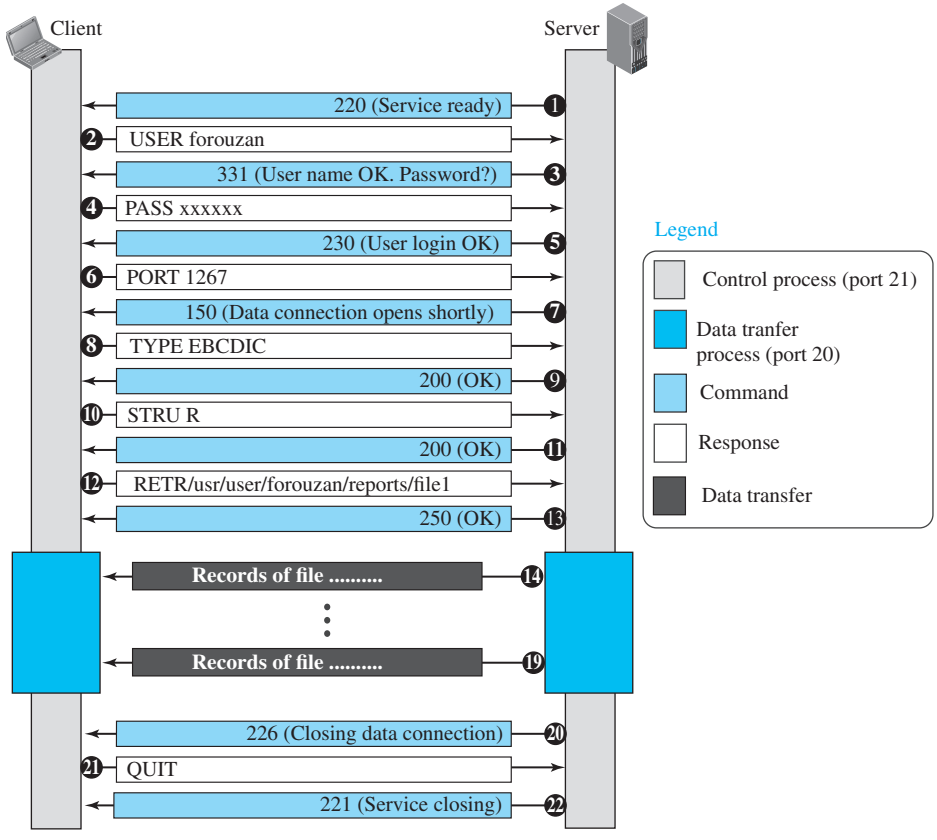
File Transfer

File transfer occurs over the data connection under the control of the commands sent over the control connection. However, we should remember that file transfer in FTP means one of three things: *retrieving a file* (server to client), *storing a file* (client to server), and *directory listing* (server to client).

Example 26.10

Figure 26.11 shows an example of using FTP for retrieving a file. The figure shows only one file to be transferred. The control connection remains open all the time, but the data connection is

Figure 26.11 *Example 26.10*



opened and closed repeatedly. We assume the file is transferred in six sections. After all records have been transferred, the server control process announces that the file transfer is done. Since the client control process has no file to retrieve, it issues the QUIT command, which causes the service connection to be closed.

Example 26.11

The following shows an actual FTP session that lists the directories. The colored lines show the responses from the server control connection; the black lines show the commands sent by the client. The lines in white with black background show data transfer.

```
$ ftp voyager.deanza.fhda.edu
Connected to voyager.deanza.fhda.edu.
220 (vsFTPd 1.2.1)
530 Please login with USER and PASS.
Name (voyager.deanza.fhda.edu:forouzan): forouzan
331 Please specify the password.
Password:*********
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
227 Entering Passive Mode (153,18,17,11,238,169)
150 Here comes the directory listing.
drwxr-xr-x    2    3027    411    4096    Sep 24    2002    business
drwxr-xr-x    2    3027    411    4096    Sep 24    2002    personal
drwxr-xr-x    2    3027    411    4096    Sep 24    2002    school
226 Directory send OK.
ftp> quit
221 Goodbye.
```

26.2.4 Security for FTP

The FTP protocol was designed when security was not a big issue. Although FTP requires a password, the password is sent in plaintext (unencrypted), which means it can be intercepted and used by an attacker. The data transfer connection also transfers data in plaintext, which is insecure. To be secure, one can add a Secure Socket Layer between the FTP application layer and the TCP layer. In this case FTP is called SSL-FTP. We also explore some secure file transfer applications when we discuss SSH later in the chapter.

26.3 ELECTRONIC MAIL

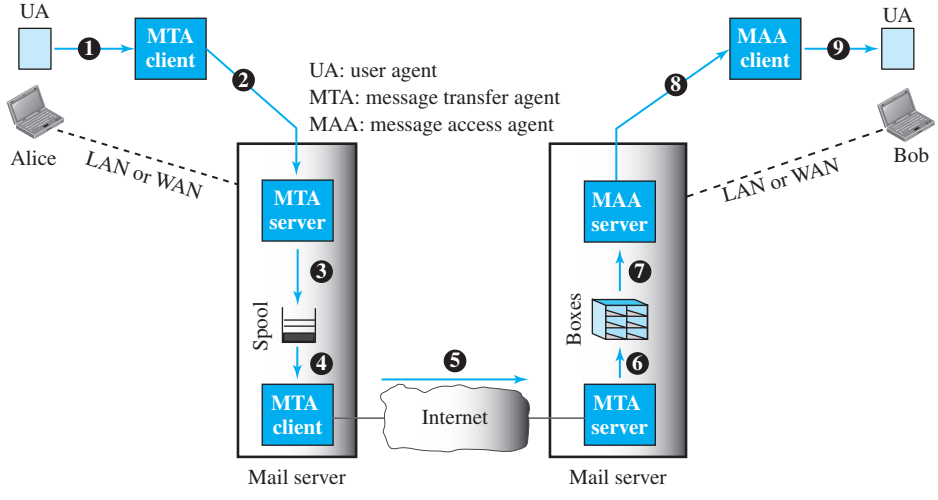
Electronic mail (or e-mail) allows users to exchange messages. The nature of this application, however, is different from other applications discussed so far. In an application such as HTTP or FTP, the server program is running all the time, waiting for a request from a client. When the request arrives, the server provides the service. There is a request and there is a response. In the case of electronic mail, the situation is

different. First, e-mail is considered a one-way transaction. When Alice sends an e-mail to Bob, she may expect a response, but this is not a mandate. Bob may or may not respond. If he does respond, it is another one-way transaction. Second, it is neither feasible nor logical for Bob to run a server program and wait until someone sends an e-mail to him. Bob may turn off his computer when he is not using it. This means that the idea of client/server programming should be implemented in another way: using some intermediate computers (servers). The users run only client programs when they want and the intermediate servers apply the client/server paradigm, as we discuss in the next section.

26.3.1 Architecture

To explain the architecture of e-mail, we give a common scenario, as shown in Figure 26.12. Another possibility is the case in which Alice or Bob is directly connected to the corresponding mail server, in which LAN or WAN connection is not required, but this variation in the scenario does not affect our discussion.

Figure 26.12 *Common scenario*



In the common scenario, the sender and the receiver of the e-mail, Alice and Bob respectively, are connected via a LAN or a WAN to two mail servers. The administrator has created one mailbox for each user where the received messages are stored. A *mailbox* is part of a server hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it. The administrator has also created a queue (spool) to store messages waiting to be sent.

A simple e-mail from Alice to Bob takes nine different steps, as shown in the figure. Alice and Bob use three different *agents*: a **user agent (UA),** a **message transfer agent (MTA),** and a **message access agent (MAA).** When Alice needs to send a message to

Bob, she runs a UA program to prepare the message and send it to her mail server. The mail server at her site uses a queue (spool) to store messages waiting to be sent. The message, however, needs to be sent through the Internet from Alice's site to Bob's site using an MTA. Here two message transfer agents are needed: one client and one server. Like most client-server programs on the Internet, the server needs to run all the time because it does not know when a client will ask for a connection. The client, on the other hand, can be triggered by the system when there is a message in the queue to be sent. The user agent at the Bob site allows Bob to read the received message. Bob later uses an MAA client to retrieve the message from an MAA server running on the second server.

There are two important points we need to emphasize here. First, Bob cannot bypass the mail server and use the MTA server directly. To use the MTA server directly, Bob would need to run the MTA server all the time because he does not know when a message will arrive. This implies that Bob must keep his computer on all the time if he is connected to his system through a LAN. If he is connected through a WAN, he must keep the connection up all the time. Neither of these situations is feasible today.

Second, note that Bob needs another pair of client-server programs: message access programs. This is because an MTA client-server program is a *push* program: the client pushes the message to the server. Bob needs a *pull* program. The client needs to pull the message from the server. We discuss more about MAAs shortly.

The electronic mail system needs two UAs, two pairs of MTAs (client and server), and a pair of MAAs (client and server).

User Agent

The first component of an electronic mail system is the **user agent (UA).** It provides service to the user to make the process of sending and receiving a message easier. A user agent is a software package (program) that composes, reads, replies to, and forwards messages. It also handles local mailboxes on the user computers.

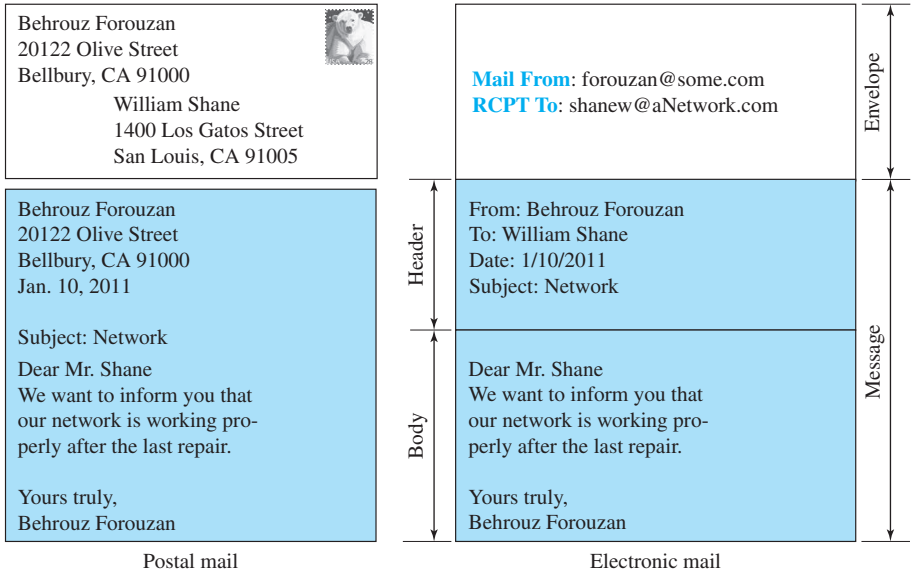
There are two types of user agents: command-driven and GUI-based. Command-driven user agents belong to the early days of electronic mail. They are still present as the underlying user agents. A command-driven user agent normally accepts a one-character command from the keyboard to perform its task. For example, a user can type the character *r*, at the command prompt, to reply to the sender of the message, or type the character *R* to reply to the sender and all recipients. Some examples of command-driven user agents are *mail, pine,* and *elm*.

Modern user agents are GUI-based. They contain graphical user interface (GUI) components that allow the user to interact with the software by using both the keyboard and the mouse. They have graphical components such as icons, menu bars, and windows that make the services easy to access. Some examples of GUI-based user agents are *Eudora* and *Outlook*.

Sending Mail

To send mail, the user, through the UA, creates mail that looks very similar to postal mail. It has an *envelope* and a *message* (see Figure 26.13). The envelope usually contains the sender address, the receiver address, and other information. The message

Figure 26.13 *Format of an e-mail*



contains the *header* and the *body*. The header of the message defines the sender, the receiver, the subject of the message, and some other information. The body of the message contains the actual information to be read by the recipient.

Receiving Mail

The user agent is triggered by the user (or a timer). If a user has mail, the UA informs the user with a notice. If the user is ready to read the mail, a list is displayed in which each line contains a summary of the information about a particular message in the mailbox. The summary usually includes the sender mail address, the subject, and the time the mail was sent or received. The user can select any of the messages and display its contents on the screen.

Addresses

To deliver mail, a mail handling system must use an addressing system with unique addresses. In the Internet, the address consists of two parts: a *local part* and a *domain name,* separated by an @ sign (see Figure 26.14).

Figure 26.14 *E-mail address*

The local part defines the name of a special file, called the user mailbox, where all the mail received for a user is stored for retrieval by the message access agent. The second part of the address is the domain name. An organization usually selects one or more hosts to receive and send e-mail; they are sometimes called *mail servers* or *exchangers*. The domain name assigned to each mail exchanger either comes from the DNS database or is a logical name (for example, the name of the organization).

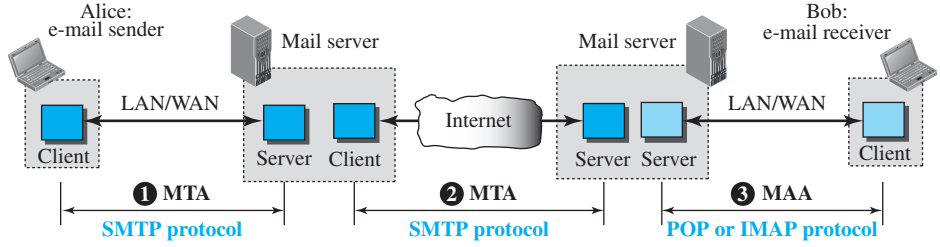
Mailing List or Group List

Electronic mail allows one name, an *alias,* to represent several different e-mail addresses; this is called a mailing list. Every time a message is to be sent, the system checks the recipient's name against the alias database; if there is a mailing list for the defined alias, separate messages, one for each entry in the list, must be prepared and handed to the MTA.

Message Transfer Agent: SMTP

Based on the common scenario (Figure 26.12), we can say that the e-mail is one of those applications that needs three uses of client-server paradigms to accomplish its task. It is important that we distinguish these three when we are dealing with e-mail. Figure 26.15 shows these three client-server applications. We refer to the first and the second as Message Transfer Agents (MTAs), the third as Message Access Agent (MAA).

Figure 26.15 *Protocols used in electronic mail*



The formal protocol that defines the MTA client and server in the Internet is called ***Simple Mail Transfer Protocol (SMTP).*** SMTP is used two times, between the sender and the sender's mail server and between the two mail servers. As we will see shortly, another protocol is needed between the mail server and the receiver. SMTP simply defines how commands and responses must be sent back and forth.

Commands and Responses

SMTP uses commands and responses to transfer messages between an MTA client and an MTA server. The command is from an MTA client to an MTA server; the response is from an MTA server to the MTA client. Each command or reply is terminated by a two-character (carriage return and line feed) end-of-line token.

Commands Commands are sent from the client to the server. The format of a command is shown below:

Keyword: argument(s)

It consists of a keyword followed by zero or more arguments. SMTP defines 14 commands, listed in Table 26.6.

Table 26.6 *SMTP commands*

Keyword	*Argument(s)*	*Description*
HELO	Sender's host name	Identifies itself
MAIL FROM	Sender of the message	Identifies the sender of the message
RCPT TO	Intended recipient	Identifies the recipient of the message
DATA	Body of the mail	Sends the actual message
QUIT		Terminates the message
RSET		Aborts the current mail transaction
VRFY	Name of recipient	Verifies the address of the recipient
NOOP		Checks the status of the recipient
TURN		Switches the sender and the recipient
EXPN	Mailing list	Asks the recipient to expand the mailing list
HELP	Command name	Asks the recipient to send information about the command sent as the argument
SEND FROM	Intended recipient	Specifies that the mail be delivered only to the terminal of the recipient, and not to the mailbox
SMOL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *or* the mailbox of the recipient
SMAL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *and* the mailbox of the recipient

Responses Responses are sent from the server to the client. A response is a three-digit code that may be followed by additional textual information. Table 26.7 shows the most common response types.

Table 26.7 *Responses*

Code	*Description*
Positive Completion Reply	
211	System status or help reply
214	Help message
220	Service ready
221	Service closing transmission channel
250	Request command completed
251	User not local; the message will be forwarded
Positive Intermediate Reply	
354	Start mail input
Transient Negative Completion Reply	
421	Service not available
450	Mailbox not available
451	Command aborted: local error
452	Command aborted; insufficient storage
Permanent Negative Completion Reply	
500	Syntax error; unrecognized command

Table 26.7 *Responses (continued)*

Code	*Description*
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command temporarily not implemented
550	Command is not executed; mailbox unavailable
551	User not local
552	Requested action aborted; exceeded storage location
553	Requested action not taken; mailbox name not allowed
554	Transaction failed

Mail Transfer Phases

The process of transferring a mail message occurs in three phases: connection establishment, mail transfer, and connection termination.

Connection Establishment After a client has made a TCP connection to the well-known port 25, the SMTP server starts the connection phase. This phase involves the following three steps:

1. The server sends code 220 (service ready) to tell the client that it is ready to receive mail. If the server is not ready, it sends code 421 (service not available).
2. The client sends the HELO message to identify itself, using its domain name address. This step is necessary to inform the server of the domain name of the client.
3. The server responds with code 250 (request command completed) or some other code depending on the situation.

Message Transfer After connection has been established between the SMTP client and server, a single message between a sender and one or more recipients can be exchanged. This phase involves eight steps. Steps 3 and 4 are repeated if there is more than one recipient.

1. The client sends the MAIL FROM message to introduce the sender of the message. It includes the mail address of the sender (mailbox and the domain name). This step is needed to give the server the return mail address for returning errors and reporting messages.
2. The server responds with code 250 or some other appropriate code.
3. The client sends the RCPT TO (recipient) message, which includes the mail address of the recipient.
4. The server responds with code 250 or some other appropriate code.
5. The client sends the DATA message to initialize the message transfer.
6. The server responds with code 354 (start mail input) or some other appropriate message.
7. The client sends the contents of the message in consecutive lines. Each line is terminated by a two-character end-of-line token (carriage return and line feed). The message is terminated by a line containing just one period.
8. The server responds with code 250 (OK) or some other appropriate code.

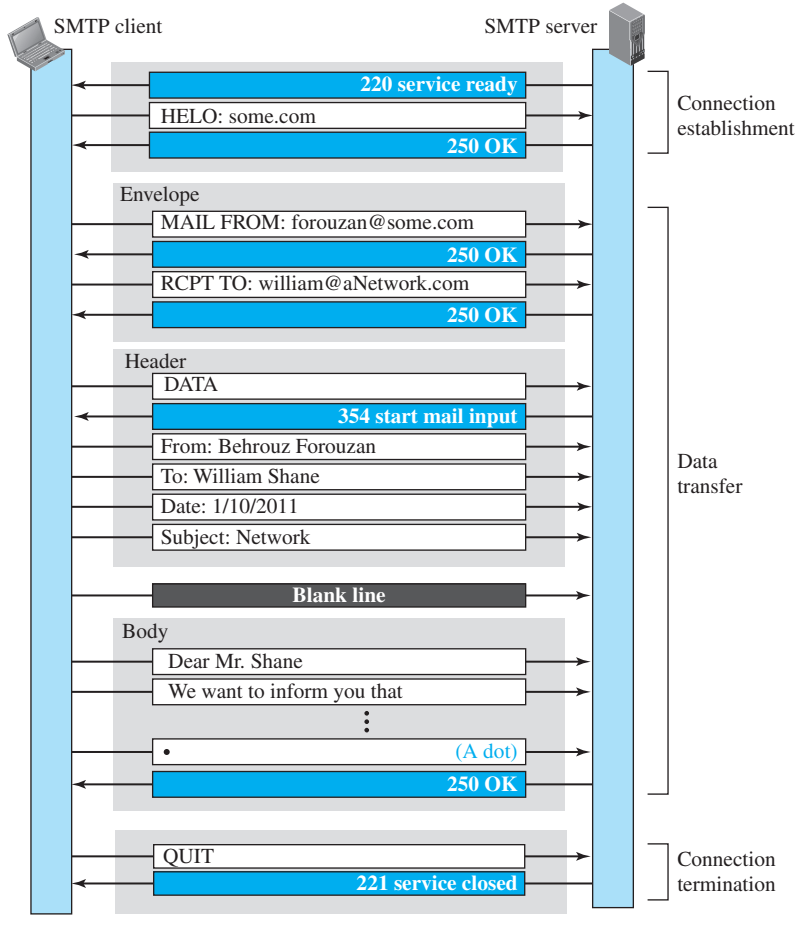
Connection Termination After the message is transferred successfully, the client terminates the connection. This phase involves two steps.

1. The client sends the QUIT command.
2. The server responds with code 221 or some other appropriate code.

Example 26.12

To show the three mail transfer phases, we show all of the steps described above using the information depicted in Figure 26.16. In the figure, we have separated the messages related to the envelope, header, and body in the data transfer section. Note that the steps in this figure are repeated two times in each e-mail transfer: once from the e-mail sender to the local mail server and once from the local mail server to the remote mail server. The local mail server, after receiving the whole e-mail message, may spool it and send it to the remote mail server at another time.

Figure 26.16 *Example 26.12*



Message Access Agent: POP and IMAP

The first and second stages of mail delivery use SMTP. However, SMTP is not involved in the third stage because SMTP is a *push* protocol; it pushes the message from the client to the server. In other words, the direction of the bulk data (messages) is from the client to the server. On the other hand, the third stage needs a *pull* protocol; the client must pull messages from the server. The direction of the bulk data is from the server to the client. The third stage uses a message access agent.

Currently two message access protocols are available: Post Office Protocol, version 3 (POP3) and Internet Mail Access Protocol, version 4 (IMAP4). Figure 26.15 shows the position of these two protocols.

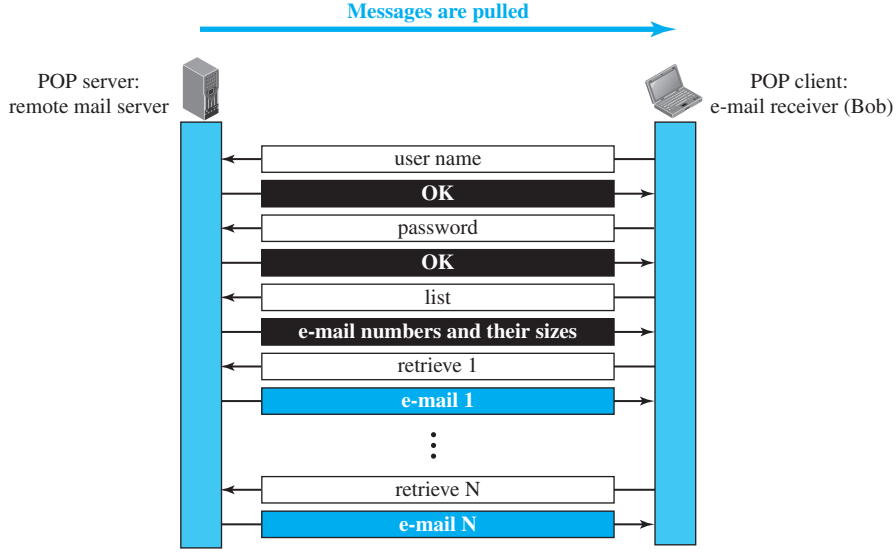
POP3

Post Office Protocol, version 3 (POP3) is simple but limited in functionality. The client POP3 software is installed on the recipient computer; the server POP3 software is installed on the mail server.

Mail access starts with the client when the user needs to download its e-mail from the mailbox on the mail server. The client opens a connection to the server on TCP port 110.

It then sends its user name and password to access the mailbox. The user can then list and retrieve the mail messages, one by one. Figure 26.17 shows an example of downloading using POP3. Unlike other figures in this chapter, we have put the client on the right hand side because the e-mail receiver (Bob) is running the client process to pull messages from the remote mail server.

Figure 26.17 *POP3*



POP3 has two modes: the *delete* mode and the *keep* mode. In the delete mode, the mail is deleted from the mailbox after each retrieval. In the keep mode, the mail remains in the mailbox after retrieval. The delete mode is normally used when the user

is working at her permanent computer and can save and organize the received mail after reading or replying. The keep mode is normally used when the user accesses her mail away from her primary computer (for example, from a laptop). The mail is read but kept in the system for later retrieval and organizing.

IMAP4

Another mail access protocol is **Internet Mail Access Protocol, version 4 (IMAP4).** IMAP4 is similar to POP3, but it has more features; IMAP4 is more powerful and more complex.

POP3 is deficient in several ways. It does not allow the user to organize her mail on the server; the user cannot have different folders on the server. In addition, POP3 does not allow the user to partially check the contents of the mail before downloading. IMAP4 provides the following extra functions:

- A user can check the e-mail header prior to downloading.
- A user can search the contents of the e-mail for a specific string of characters prior to downloading.
- A user can partially download e-mail. This is especially useful if bandwidth is limited and the e-mail contains multimedia with high bandwidth requirements.
- A user can create, delete, or rename mailboxes on the mail server.
- A user can create a hierarchy of mailboxes in a folder for e-mail storage.

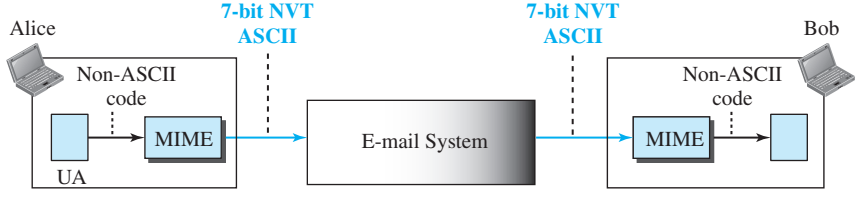
MIME

Electronic mail has a simple structure. Its simplicity, however, comes with a price. It can send messages only in NVT 7-bit ASCII format. In other words, it has some limitations. It cannot be used for languages other than English (such as French, German, Hebrew, Russian, Chinese, and Japanese). Also, it cannot be used to send binary files or video or audio data.

Multipurpose Internet Mail Extensions (MIME) is a supplementary protocol that allows non-ASCII data to be sent through e-mail. MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the Internet. The message at the receiving site is transformed back to the original data.

We can think of MIME as a set of software functions that transforms non-ASCII data to ASCII data and vice versa, as shown in Figure 26.18.

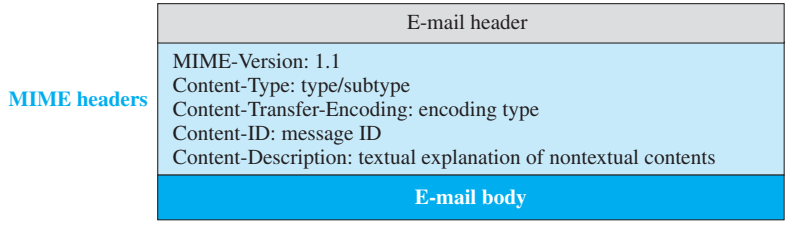
Figure 26.18 *MIME*



MIME Headers

MIME defines five headers, as shown in Figure 26.19, which can be added to the original e-mail header section to define the transformation parameters:

Figure 26.19 *MIME header*



MIME-Version This header defines the version of MIME used. The current version is 1.1.

Content-Type This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters. MIME allows seven different types of data, listed in Table 26.8.

Table 26.8 *Data types and subtypes in MIME*

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix C)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

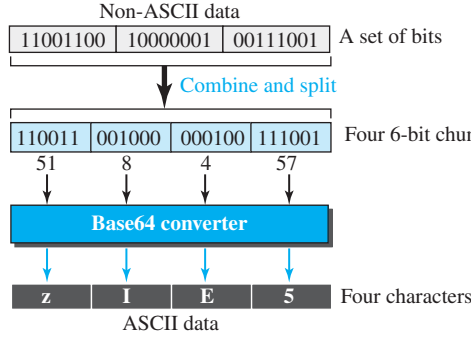
Content-Transfer-Encoding This header defines the method used to encode the messages into 0s and 1s for transport. The five types of encoding methods are listed in Table 26.9.

Table 26.9 *Methods for Content-Transfer-Encoding*

Type	*Description*
7-bit	NVT ASCII characters with each line less than 1000 characters
8-bit	Non-ASCII characters with each line less than 1000 characters
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters encoded as an equal sign plus an ASCII code

The last two encoding methods are interesting. In the Base64 encoding, data, as a string of bits, is first divided into 6-bit chunks as shown in Figure 26.20.

Figure 26.20 *Base64 conversion*



Each 6-bit section is then converted into an ASCII character according to Table 26.10.

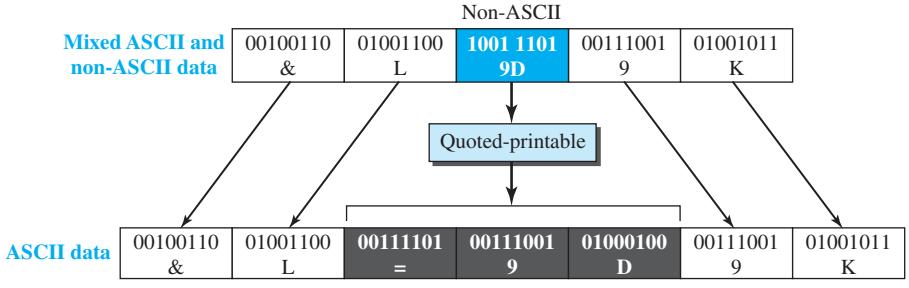
Table 26.10 *Base64 converting table*

Value	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*
0	**A**	11	**L**	22	**W**	33	**h**	44	**s**	55	**3**
1	**B**	12	**M**	23	**X**	34	**i**	45	**t**	56	**4**
2	**C**	13	**N**	24	**Y**	35	**j**	46	**u**	57	**5**
3	**D**	14	**O**	25	**Z**	36	**k**	47	**v**	58	**6**
4	**E**	15	**P**	26	**a**	37	**l**	48	**w**	59	**7**
5	**F**	16	**Q**	27	**b**	38	**m**	49	**x**	60	**8**
6	**G**	17	**R**	28	**c**	39	**n**	50	**y**	61	**9**
7	**H**	18	**S**	29	**d**	40	**o**	51	**z**	62	+
8	**I**	19	**T**	30	**e**	41	**p**	52	**0**	63	/
9	**J**	20	**U**	31	**f**	42	**q**	53	**1**		
10	**K**	21	**V**	32	**g**	43	**r**	54	**2**		

Base64 is a redundant encoding scheme; that is, every six bits become one ASCII character and are sent as eight bits. We have an overhead of 25 percent. If the data consist mostly of ASCII characters with a small non-ASCII portion, we can use quoted-printable encoding. In quoted-printable, if a character is ASCII, it is sent as is.

If a character is not ASCII, it is sent as three characters. The first character is the equal sign (=). The next two characters are the hexadecimal representations of the byte. Figure 26.21 shows an example. In the example, the third character is a non-ASCII because it starts with bit 1. It is interpreted as two hexadecimal digits ($9D_{16}$), which is replaced by three ASCII characters (=, 9, and D).

Figure 26.21 *Quoted-printable*



Content-ID This header uniquely identifies the whole message in a multiple message environment.

Content-Description This header defines whether the body is image, audio, or video.

26.3.2 Web-Based Mail

E-mail is such a common application that some websites today provide this service to anyone who accesses the site. Three common sites are Hotmail, Yahoo, and Google mail. The idea is very simple. Figure 26.22 shows two cases:

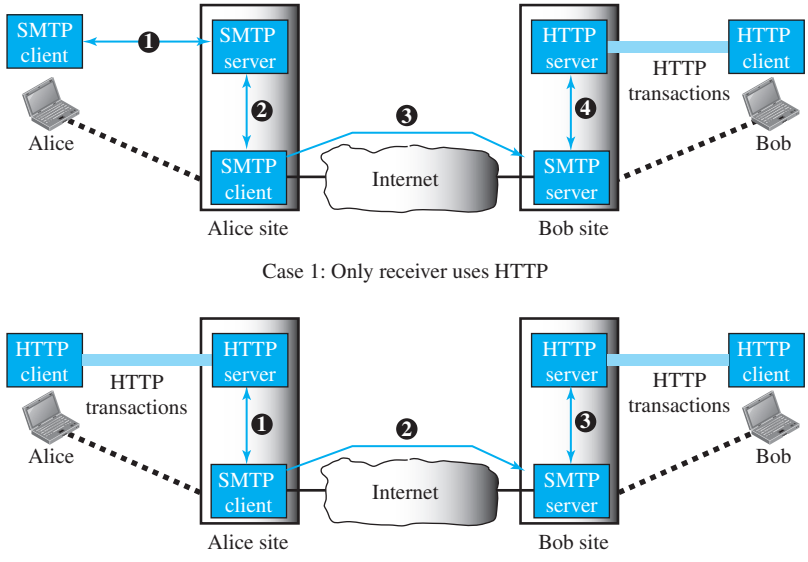
Case I

In the first case, Alice, the sender, uses a traditional mail server; Bob, the receiver, has an account on a web-based server. Mail transfer from Alice's browser to her mail server is done through SMTP. The transfer of the message from the sending mail server to the receiving mail server is still through SMTP. However, the message from the receiving server (the web server) to Bob's browser is done through HTTP. In other words, instead of using POP3 or IMAP4, HTTP is normally used. When Bob needs to retrieve his e-mails, he sends a request HTTP message to the website (Hotmail, for example). The website sends a form to be filled in by Bob, which includes the log-in name and the password. If the log-in name and password match, the list of e-mails is transferred from the web server to Bob's browser in HTML format. Now Bob can browse through his received e-mails and then, using more HTTP transactions, can get his e-mails one by one.

Case II

In the second case, both Alice and Bob use web servers, but not necessarily the same server. Alice sends the message to the web server using HTTP transactions. Alice sends an HTTP request message to her web server using the name and address of Bob's mailbox as the URL. The server at the Alice site passes the message to the SMTP client and

Figure 26.22 *Web-based e-mail, cases I and II*



sends it to the server at the Bob site using SMTP protocol. Bob receives the message using HTTP transactions. However, the message from the server at the Alice site to the server at the Bob site still takes place using SMTP protocol.

26.3.3 E-Mail Security

The protocol discussed in this chapter does not provide any security provisions per se. However, e-mail exchanges can be secured using two application-layer securities designed in particular for e-mail systems. Two of these protocols, *Pretty Good Privacy* (PGP) and *Secure/Multipurpose Internet Mail Extensions* (S/MIME), are discussed in Chapter 32 after we have discussed basic network security.

26.4 TELNET

A server program can provide a specific service to its corresponding client program. For example, the FTP server is designed to let the FTP client store or retrieve files on the server site. However, it is impossible to have a client/server pair for each type of service we need; the number of servers soon becomes intractable. The idea is not scalable. Another solution is to have a specific client/server program for a set of common scenarios, but to have some generic client/server programs that allow a user on the client site to log into the computer at the server site and use the services available there. For example, if a student needs to use the Java compiler program at her university lab, there is no need for a Java compiler client and a Java compiler server. The student can use a client

logging program to log into the university server and use the compiler program at the university. We refer to these generic client/server pairs as ***remote logging*** applications.

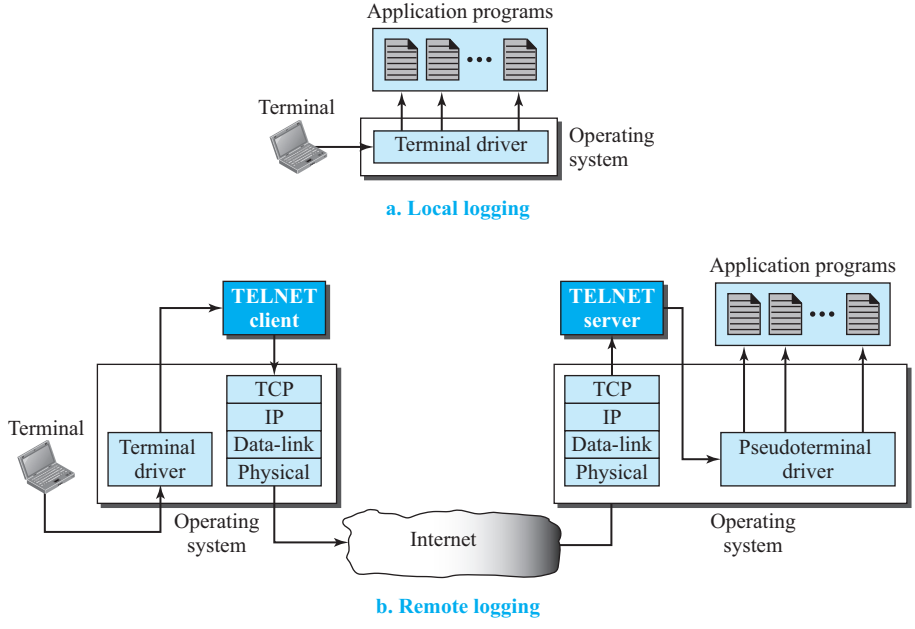
One of the original remote logging protocols is **TELNET,** which is an abbreviation for *TErminaL NETwork.* Although TELNET requires a logging name and password, it is vulnerable to hacking because it sends all data including the password in plaintext (not encrypted). A hacker can eavesdrop and obtain the logging name and password. Because of this security issue, the use of TELNET has diminished in favor of another protocol, Secure Shell (SSH), which we describe in the next section. Although TELNET is almost replaced by SSH, we briefly discuss TELNET here for two reasons:

1. The simple plaintext architecture of TELNET allows us to explain the issues and challenges related to the concept of remote logging, which is also used in SSH when it serves as a remote logging protocol.
2. Network administrators often use TELNET for diagnostic and debugging purposes.

26.4.1 Local versus Remote Logging

We first discuss the concept of local and remote logging as shown in Figure 26.23.

Figure 26.23 *Local versus remote logging*



When a user logs into a local system, it is called *local logging*. As a user types at a terminal or at a workstation running a terminal emulator, the keystrokes are accepted by the terminal driver. The terminal driver passes the characters to the operating system. The operating system, in turn, interprets the combination of characters and invokes the desired application program or utility.

However, when a user wants to access an application program or utility located on a remote machine, she performs *remote logging*. Here the TELNET client and server programs come into use. The user sends the keystrokes to the terminal driver where the local operating system accepts the characters but does not interpret them. The characters are sent to the TELNET client, which transforms the characters into a universal character set called *Network Virtual Terminal* (NVT) characters (discussed below) and delivers them to the local TCP/IP stack.

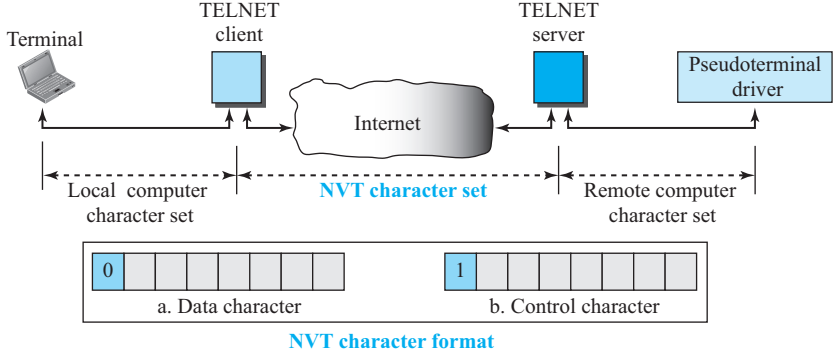
The commands or text, in NVT form, travel through the Internet and arrive at the TCP/IP stack at the remote machine. Here the characters are delivered to the operating system and passed to the TELNET server, which changes the characters to the corresponding characters understandable by the remote computer. However, the characters cannot be passed directly to the operating system because the remote operating system is not designed to receive characters from a TELNET server; it is designed to receive characters from a terminal driver. The solution is to add a piece of software called a *pseudoterminal driver*, which pretends that the characters are coming from a terminal. The operating system then passes the characters to the appropriate application program.

Network Virtual Terminal (NVT)

The mechanism to access a remote computer is complex. This is because every computer and its operating system accepts a special combination of characters as tokens. For example, the end-of-file token in a computer running the DOS operating system is Ctrl+z, while the UNIX operating system recognizes Ctrl+d.

We are dealing with heterogeneous systems. If we want to access any remote computer in the world, we must first know what type of computer we will be connected to, and we must also install the specific terminal emulator used by that computer. TELNET solves this problem by defining a universal interface called the *Network Virtual Terminal (NVT)* character set. Via this interface, the client TELNET translates characters (data or commands) that come from the local terminal into NVT form and delivers them to the network. The server TELNET, on the other hand, translates data and commands from NVT form into the form acceptable by the remote computer. Figure 26.24 shows the concept.

Figure 26.24 *Concept of NVT*



NVT uses two sets of characters, one for data and one for control. Both are 8-bit bytes as shown in Figure 26.24. For data, NVT normally uses what is called *NVT ASCII*. This is an 8-bit character set in which the seven lowest order bits are the same as US ASCII and the highest order bit is 0. To send control characters between computers (from client to server or vice versa), NVT uses an 8-bit character set in which the highest order bit is set to 1.

Options

TELNET lets the client and server negotiate options before or during the use of the service. Options are extra features available to a user with a more sophisticated terminal. Users with simpler terminals can use default features.

User Interface

The operating system (UNIX, for example) defines an interface with user-friendly commands. An example of such a set of commands can be found in Table 26.11.

Table 26.11 *Examples of interface commands*

Command	*Meaning*	*Command*	*Meaning*
open	Connect to a remote computer	**set**	Set the operating parameters
close	Close the connection	**status**	Display the status information
display	Show the operating parameters	**send**	Send special characters
mode	Change to line or character mode	**quit**	Exit TELNET

26.5 SECURE SHELL (SSH)

Although **Secure Shell (SSH)** is a secure application program that can be used today for several purposes such as remote logging and file transfer, it was originally designed to replace TELNET. There are two versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1, is now deprecated because of security flaws in it. In this section, we discuss only SSH-2.

26.5.1 Components

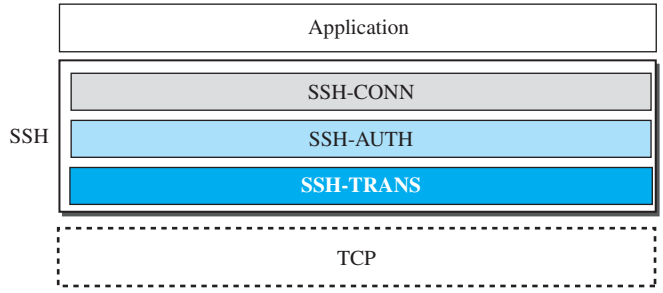
SSH is an application-layer protocol with three components, as shown in Figure 26.25.

SSH Transport-Layer Protocol (SSH-TRANS)

Since TCP is not a secured transport-layer protocol, SSH first uses a protocol that creates a secured channel on top of the TCP. This new layer is an independent protocol referred to as SSH-TRANS. When the procedure implementing this protocol is called, the client and server first use the TCP protocol to establish an insecure connection. Then they exchange several security parameters to establish a secure channel on top of the TCP. We discuss transport-layer security in Chapter 32, but here we briefly list the services provided by this protocol:

1. Privacy or confidentiality of the message exchanged
2. Data integrity, which means that it is guaranteed that the messages exchanged between the client and server are not changed by an intruder

Figure 26.25 *Components of SSH*



3. Server authentication, which means that the client is now sure that the server is the one that it claims to be
4. Compression of the messages, which improves the efficiency of the system and makes attack more difficult

SSH Authentication Protocol (SSH-AUTH)

After a secure channel is established between the client and the server and the server is authenticated for the client, SSH can call another procedure that can authenticate the client for the server. The client authentication process in SSH is very similar to what is done in Secure Socket Layer (SSL), which we discuss in Chapter 32. This layer defines a number of authentication tools similar to the ones used in SSL. Authentication starts with the client, which sends a request message to the server. The request includes the user name, server name, the method of authentication, and the required data. The server responds with either a success message, which confirms that the client is authenticated, or a failed message, which means that the process needs to be repeated with a new request message.

SSH Connection Protocol (SSH-CONN)

After the secured channel is established and both server and client are authenticated for each other, SSH can call a piece of software that implements the third protocol, SSH-CONN. One of the services provided by the SSH-CONN protocol is multiplexing. SSH-CONN takes the secure channel established by the two previous protocols and lets the client create multiple logical channels over it. Each channel can be used for a different purpose, such as remote logging, file transfer, and so on.

26.5.2 Applications

Although SSH is often thought of as a replacement for TELNET, SSH is, in fact, a general-purpose protocol that provides a secure connection between a client and server.

SSH for Remote Logging

Several free and commercial applications use SSH for remote logging. Among them, we can mention PuTTy, by Simon Tatham, which is a client SSH program that can be

used for remote logging. Another application program is Tectia, which can be used on several platforms.

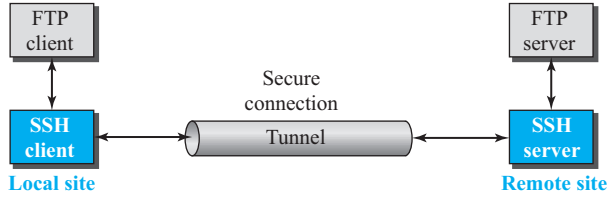
SSH for File Transfer

One of the application programs that is built on top of SSH for file transfer is the *Secure File Transfer Program* (*sftp*). The *sftp* application program uses one of the channels provided by the SSH to transfer files. Another common application is called *Secure Copy* (*scp*). This application uses the same format as the UNIX copy command, *cp,* to copy files.

Port Forwarding

One of the interesting services provided by the SSH protocol is **port forwarding**. We can use the secured channels available in SSH to access an application program that does not provide security services. Applications such as TELNET and Simple Mail Transfer Protocol (SMTP), which are discussed above, can use the services of the SSH port forwarding mechanism. The SSH port forwarding mechanism creates a tunnel through which the messages belonging to other protocols can travel. For this reason, this mechanism is sometimes referred to as SSH *tunneling*. Figure 26.26 shows the concept of port forwarding for securing the FTP application.

Figure 26.26 *Port forwarding*

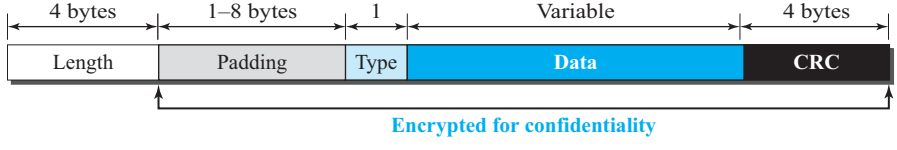


The FTP client can use the SSH client on the local site to make a secure connection with the SSH server on the remote site. Any request from the FTP client to the FTP server is carried through the tunnel provided by the SSH client and server. Any response from the FTP server to the FTP client is also carried through the tunnel provided by the SSH client and server.

Format of the SSH Packets

Figure 26.27 shows the format of packets used by the SSH protocols.

Figure 26.27 *SSH packet format*



The length field defines the length of the packet but does not include the padding. One to eight bytes of padding is added to the packet to make the attack on the security provision more difficult. The *cyclic redundancy check* (CRC) field is used for error detection. The type field designates the type of the packet used in different SSH protocols. The data field is the data transferred by the packet in different protocols.

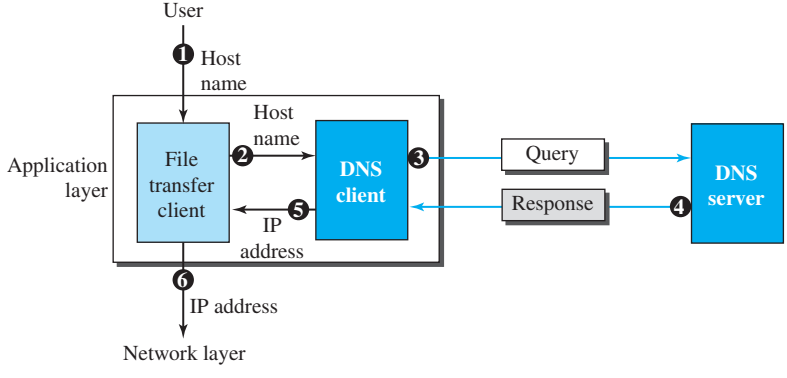
26.6 DOMAIN NAME SYSTEM (DNS)

The last client-server application program we discuss has been designed to help other application programs. To identify an entity, TCP/IP protocols use the IP address, which uniquely identifies the connection of a host to the Internet. However, people prefer to use names instead of numeric addresses. Therefore, the Internet needs to have a directory system that can map a name to an address. This is analogous to the telephone network. A telephone network is designed to use telephone numbers, not names. People can either keep a private file to map a name to the corresponding telephone number or can call the telephone directory to do so. We discuss how this directory system in the Internet can map names to IP addresses.

Since the Internet is so huge today, a central directory system cannot hold all the mapping. In addition, if the central computer fails, the whole communication network will collapse. A better solution is to distribute the information among many computers in the world. In this method, the host that needs mapping can contact the closest computer holding the needed information. This method is used by the **Domain Name System (DNS)**. We first discuss the concepts and ideas behind the DNS. We then describe the DNS protocol itself.

Figure 26.28 shows how TCP/IP uses a DNS client and a DNS server to map a name to an address. A user wants to use a file transfer client to access the corresponding file transfer server running on a remote host. The user knows only the file transfer

Figure 26.28 *Purpose of DNS*



server name, such as *afilesource.com.* However, the TCP/IP suite needs the IP address of the file transfer server to make the connection. The following six steps map the host name to an IP address:

1. The user passes the host name to the file transfer client.
2. The file transfer client passes the host name to the DNS client.
3. Each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server name using the known IP address of the DNS server.
4. The DNS server responds with the IP address of the desired file transfer server.
5. The DNS server passes the IP address to the file transfer client.
6. The file transfer client now uses the received IP address to access the file transfer server.

Note that the purpose of accessing the Internet is to make a connection between the file transfer client and server, but before this can happen, another connection needs to be made between the DNS client and DNS server. In other words, we need at least two connections in this case. The first is for mapping the name to an IP address; the second is for transferring files. We will see later that the mapping may need more than one connection.

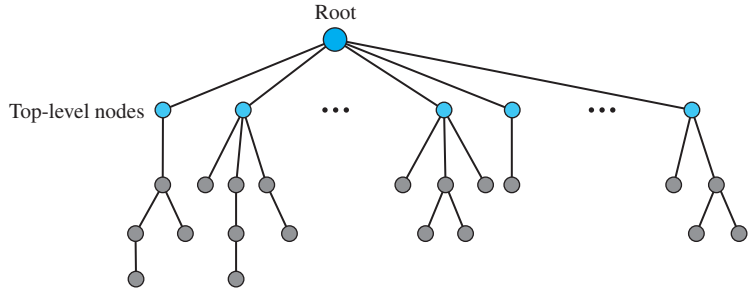
26.6.1 Name Space

To be unambiguous, the names assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP addresses. In other words, the names must be unique because the addresses are unique. A **name space** that maps each address to a unique name can be organized in two ways: flat or hierarchical. In a *flat name space,* a name is assigned to an address. A name in this space is a sequence of characters without structure. The names may or may not have a common section; if they do, it has no meaning. The main disadvantage of a flat name space is that it cannot be used in a large system such as the Internet because it must be centrally controlled to avoid ambiguity and duplication. In a *hierarchical name space,* each name is made of several parts. The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on. In this case, the authority to assign and control the name spaces can be decentralized. A central authority can assign the part of the name that defines the nature of the organization and the name of the organization. The responsibility for the rest of the name can be given to the organization itself. The organization can add suffixes (or prefixes) to the name to define its host or resources. The management of the organization need not worry that the prefix chosen for a host is taken by another organization because, even if part of an address is the same, the whole address is different. For example, assume two organizations call one of their computers *caesar.* The first organization is given a name by the central authority, such as *first.com,* the second organization is given the name *second.com.* When each of these organizations adds the name *caesar* to the name they have already been given, the end result is two distinguishable names: *ceasar.first.com* and *ceasar.second.com*. The names are unique.

Domain Name Space

To have a hierarchical name space, a **domain name space** was designed. In this design the names are defined in an inverted-tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127 (see Figure 26.29).

Figure 26.29 *Domain name space*



Label

Each node in the tree has a **label**, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.

Domain Name

Each node in the tree has a domain name. A full **domain name** is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null). This means that a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing. Figure 26.30 shows some domain names.

If a label is terminated by a null string, it is called a **fully qualified domain name (FQDN)**. The name must end with a null label, but because null means nothing, the label ends with a dot. If a label is not terminated by a null string, it is called a **partially qualified domain name (PQDN).** A PQDN starts from a node, but it does not reach the root. It is used when the name to be resolved belongs to the same site as the client. Here the resolver can supply the missing part, called the *suffix,* to create an FQDN.

Domain

A **domain** is a subtree of the domain name space. The name of the domain is the name of the node at the top of the subtree. Figure 26.31 shows some domains. Note that a domain may itself be divided into domains.

Distribution of Name Space

The information contained in the domain name space must be stored. However, it is very inefficient and also not reliable to have just one computer store such a huge

Figure 26.30 *Domain names and labels*

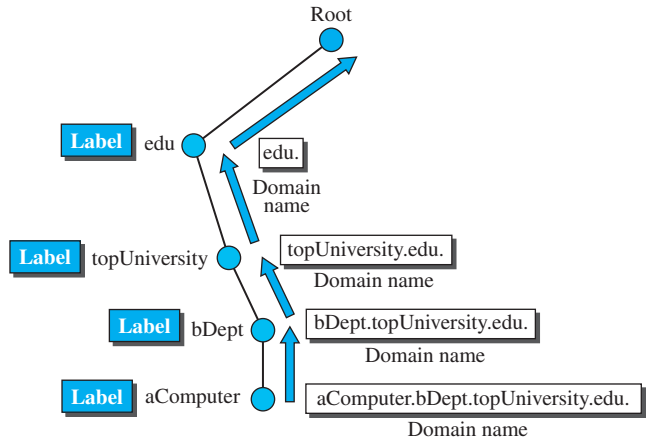
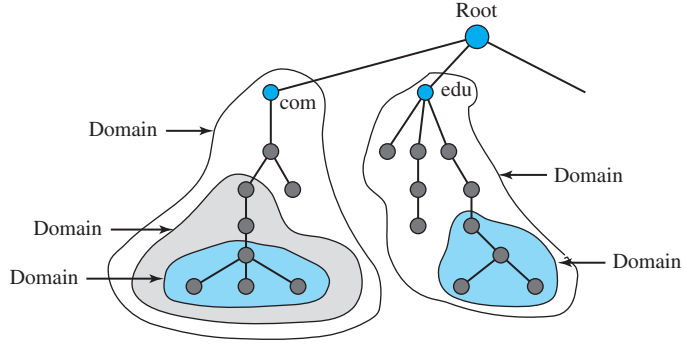


Figure 26.31 *Domains*

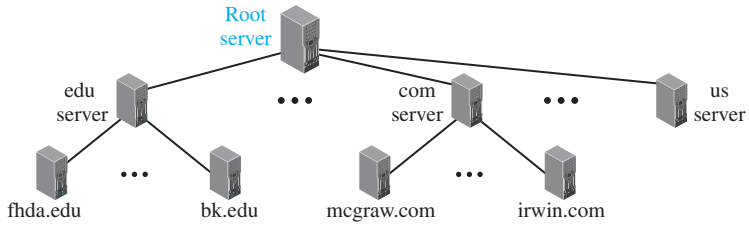


amount of information. It is inefficient because responding to requests from all over the world places a heavy load on the system. It is not reliable because any failure makes the data inaccessible.

Hierarchy of Name Servers

The solution to these problems is to distribute the information among many computers called ***DNS servers***. One way to do this is to divide the whole space into many domains based on the first level. In other words, we let the root stand alone and create as many domains (subtrees) as there are first-level nodes. Because a domain created this way could be very large, DNS allows domains to be divided further into smaller domains (subdomains). Each server can be responsible (authoritative) for either a large or small domain. In other words, we have a hierarchy of servers in the same way that we have a hierarchy of names (see Figure 26.32).

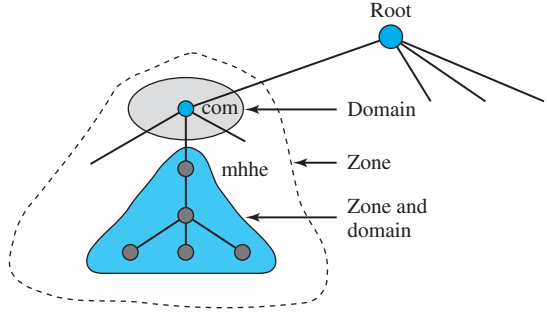
Figure 26.32 *Hierarchy of name servers*



Zone

Since the complete domain name hierarchy cannot be stored on a single server, it is divided among many servers. What a server is responsible for or has authority over is called a ***zone***. We can define a zone as a contiguous part of the entire tree. If a server accepts responsibility for a domain and does not divide the domain into smaller domains, the "domain" and the "zone" refer to the same thing. The server makes a database called a *zone file* and keeps all the information for every node under that domain. However, if a server divides its domain into subdomains and delegates part of its authority to other servers, "domain" and "zone" refer to different things. The information about the nodes in the subdomains is stored in the servers at the lower levels, with the original server keeping some sort of reference to these lower-level servers. Of course, the original server does not free itself from responsibility totally. It still has a zone, but the detailed information is kept by the lower-level servers (see Figure 26.33).

Figure 26.33 *Zone*



Root Server

A **root server** is a server whose zone consists of the whole tree. A root server usually does not store any information about domains but delegates its authority to other servers, keeping references to those servers. There are several root servers, each covering the whole domain name space. The root servers are distributed all around the world.

Primary and Secondary Servers

DNS defines two types of servers: primary and secondary. A *primary server* is a server that stores a file about the zone for which it is an authority. It is responsible for creating, maintaining, and updating the zone file. It stores the zone file on a local disk.

A *secondary server* is a server that transfers the complete information about a zone from another server (primary or secondary) and stores the file on its local disk. The secondary server neither creates nor updates the zone files. If updating is required, it must be done by the primary server, which sends the updated version to the secondary.

The primary and secondary servers are both authoritative for the zones they serve. The idea is not to put the secondary server at a lower level of authority but to create redundancy for the data so that if one server fails, the other can continue serving clients. Note also that a server can be a primary server for a specific zone and a secondary server for another zone. Therefore, when we refer to a server as a primary or secondary server, we should be careful about which zone we refer to.

A primary server loads all information from the disk file; the secondary server loads all information from the primary server.

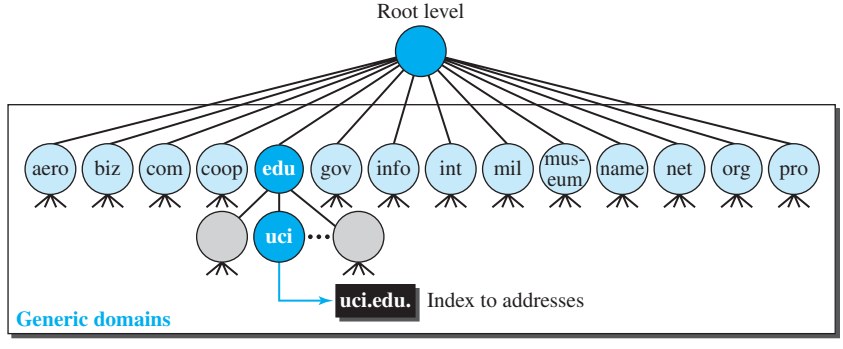
26.6.2 DNS in the Internet

DNS is a protocol that can be used in different platforms. In the Internet, the domain name space (tree) was originally divided into three different sections: generic domains, country domains, and the inverse domains. However, due to the rapid growth of the Internet, it became extremely difficult to keep track of the inverse domains, which could be used to find the name of a host when given the IP address. The inverse domains are now deprecated (see RFC 3425). We, therefore, concentrate on the first two.

Generic Domains

The **generic domains** define registered hosts according to their generic behavior. Each node in the tree defines a domain, which is an index to the domain name space database (see Figure 26.34).

Figure 26.34 *Generic domains*



Looking at the tree, we see that the first level in the **generic domains** section allows 14 possible labels. These labels describe the organization types as listed in Table 26.12.

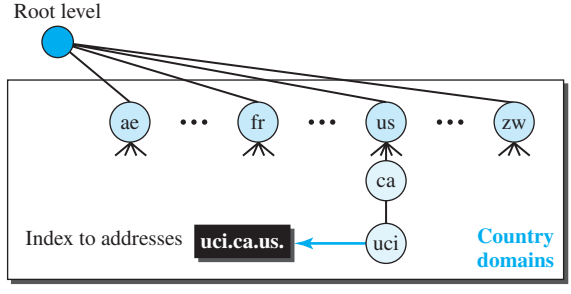
Table 26.12 *Generic domain labels*

Label	*Description*	*Label*	*Description*
aero	Airlines and aerospace	**int**	International organizations
biz	Businesses or firms	**mil**	Military groups
com	Commercial organizations	**museum**	Museums
coop	Cooperative organizations	**name**	Personal names (individuals)
edu	Educational institutions	**net**	Network support centers
gov	Government institutions	**org**	Nonprofit organizations
info	Information service providers	**pro**	Professional organizations

Country Domains

The **country domains** section uses two-character country abbreviations (e.g., us for United States). Second labels can be organizational, or they can be more specific national designations. The United States, for example, uses state abbreviations as a subdivision of us (e.g., ca.us.). Figure 26.35 shows the country domains section. The address ***uci.ca.us.*** can be translated to University of California, Irvine, in the state of California in the United States.

Figure 26.35 *Country domains*



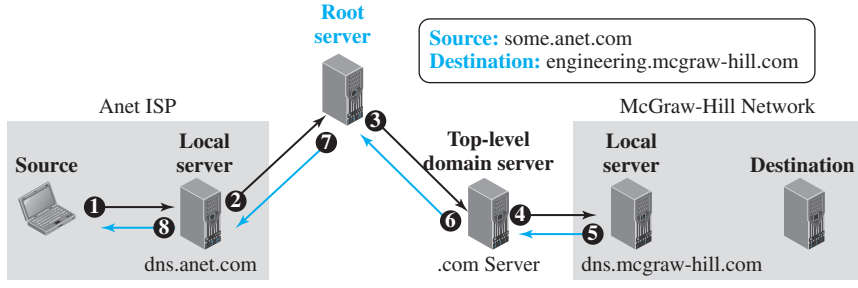
26.6.3 Resolution

Mapping a name to an address is called *name-address resolution.* DNS is designed as a client-server application. A host that needs to map an address to a name or a name to an address calls a DNS client called a ***resolver.*** The resolver accesses the closest DNS server with a mapping request. If the server has the information, it satisfies the resolver; otherwise, it either refers the resolver to other servers or asks other servers to provide the information. After the resolver receives the mapping, it interprets the response to see if it is a real resolution or an error, and finally delivers the result to the process that requested it. A resolution can be either recursive or iterative.

Recursive Resolution

Figure 26.36 shows a simple example of a **recursive resolution.** We assume that an application program running on a host named ***some.anet.com*** needs to find the IP address of another host named ***engineering.mcgraw-hill.com*** to send a message to. The source host is connected to the Anet ISP; the destination host is connected to the McGraw-Hill network.

Figure 26.36 *Recursive resolution*

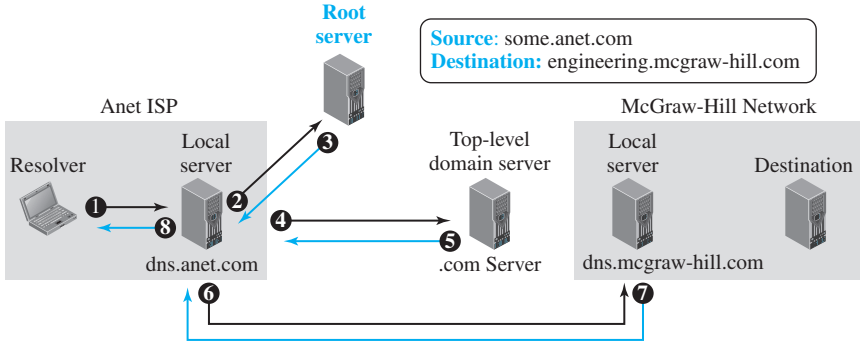


The application program on the source host calls the DNS resolver (client) to find the IP address of the destination host. The resolver, which does not know this address, sends the query to the local DNS server (for example, *dns.anet.com*) running at the Anet ISP site (event 1). We assume that this server does not know the IP address of the destination host either. It sends the query to a root DNS server, whose IP address is supposed to be known to this local DNS server (event 2). Root servers do not normally keep the mapping between names and IP addresses, but a root server should at least know about one server at each top level domain (in this case, a server responsible for *com* domain). The query is sent to this top-level-domain server (event 3). We assume that this server does not know the name-address mapping of this specific destination, but it knows the IP address of the local DNS server in the McGraw-Hill company (for example, *dns.mcgraw-hill.com*). The query is sent to this server (event 4), which knows the IP address of the destination host. The IP address is now sent back to the top-level DNS server (event 5), then back to the root server (event 6), then back to the ISP DNS server, which may cache it for the future queries (event 7), and finally back to the source host (event 8).

Iterative Resolution

In **iterative resolution,** each server that does not know the mapping sends the IP address of the next server back to the one that requested it. Figure 26.37 shows the flow of information in an iterative resolution in the same scenario as the one depicted in Figure 26.36. Normally the iterative resolution takes place between two local servers; the original resolver gets the final answer from the local server. Note that the messages shown by events 2, 4, and 6 contain the same query. However, the message shown by event 3 contains the IP address of the top-level domain server, the message shown by event 5 contains the IP address of the McGraw-Hill local DNS

Figure 26.37 *Iterative resolution*



server, and the message shown by event 7 contains the IP address of the destination. When the Anet local DNS server receives the IP address of the destination, it sends it to the resolver (event 8).

26.6.4 Caching

Each time a server receives a query for a name that is not in its domain, it needs to search its database for a server IP address. Reduction of this search time would increase efficiency. DNS handles this with a mechanism called *caching*. When a server asks for a mapping from another server and receives the response, it stores this information in its cache memory before sending it to the client. If the same or another client asks for the same mapping, it can check its cache memory and resolve the problem. However, to inform the client that the response is coming from the cache memory and not from an authoritative source, the server marks the response as *unauthoritative*.

Caching speeds up resolution, but it can also be problematic. If a server caches a mapping for a long time, it may send an outdated mapping to the client. To counter this, two techniques are used. First, the authoritative server always adds information to the mapping called *time to live* (TTL). It defines the time in seconds that the receiving server can cache the information. After that time, the mapping is invalid and any query must be sent again to the authoritative server. Second, DNS requires that each server keep a TTL counter for each mapping it caches. The cache memory must be searched periodically and those mappings with an expired TTL must be purged.

26.6.5 Resource Records

The zone information associated with a server is implemented as a set of *resource records*. In other words, a name server stores a database of resource records. A *resource record* is a 5-tuple structure, as shown below:

(Domain Name, Type, Class, TTL, Value)

The domain name field is what identifies the resource record. The value defines the information kept about the domain name. The TTL defines the number of

seconds for which the information is valid. The class defines the type of network; we are only interested in the class IN (Internet). The type defines how the value should be interpreted. Table 26.13 lists the common types and how the value is interpreted for each type.

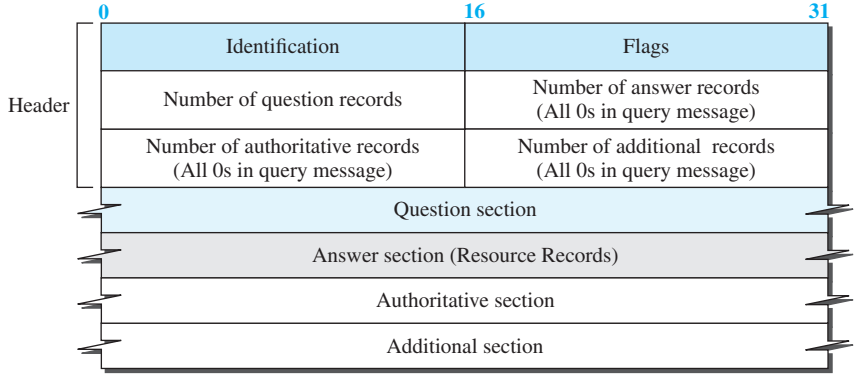
Table 26.13 *Types*

Type	*Interpretation of value*
A	A 32-bit IPv4 address (see Chapter 18)
NS	Identifies the authoritative servers for a zone
CNAME	Defines an alias for the official name of a host
SOA	Marks the beginning of a zone
MX	Redirects mail to a mail server
AAAA	An IPv6 address (see Chapter 22)

26.6.6 DNS Messages

To retrieve information about hosts, DNS uses two types of messages: *query* and *response*. Both types have the same format as shown in Figure 26.38.

Figure 26.38 *DNS message*



Note:
The query message contains only the question section.
The response message includes the question section,
the answer section, and possibly two other sections.

We briefly discuss the fields in a DNS message. The identification field is used by the client to match the response with the query. The flag field defines whether the message is a query or response. It also includes status of error. The next four fields in the header define the number of each record type in the message. The question section consists of one or more question records. It is present in both query and response messages. The answer section consists of one or more resource records. It is present only in response messages. The authoritative section gives information (domain name) about one or more authoritative servers for the query. The additional information section provides additional information that may help the resolver.

Example 26.13

In UNIX and Windows, the *nslookup* utility can be used to retrieve address/name mapping. The following shows how we can retrieve an address when the domain name is given.

```
$nslookup www.forouzan.biz
Name:  www.forouzan.biz
Address: 198.170.240.179
```

Encapsulation

DNS can use either UDP or TCP. In both cases the well-known port used by the server is port 53. UDP is used when the size of the response message is less than 512 bytes because most UDP packages have a 512-byte packet size limit. If the size of the response message is more than 512 bytes, a TCP connection is used. In that case, one of two scenarios can occur:

- If the resolver has prior knowledge that the size of the response message is more than 512 bytes, it uses the TCP connection. For example, if a secondary name server (acting as a client) needs a zone transfer from a primary server, it uses the TCP connection because the size of the information being transferred usually exceeds 512 bytes.
- If the resolver does not know the size of the response message, it can use the UDP port. However, if the size of the response message is more than 512 bytes, the server truncates the message and turns on the TC bit. The resolver now opens a TCP connection and repeats the request to get a full response from the server.

26.6.7 Registrars

How are new domains added to DNS? This is done through a *registrar,* a commercial entity accredited by ICANN. A registrar first verifies that the requested domain name is unique and then enters it into the DNS database. A fee is charged. Today, there are many registrars; their names and addresses can be found at

http://www.intenic.net

To register, the organization needs to give the name of its server and the IP address of the server. For example, a new commercial organization named *wonderful* with a server named *ws* and IP address 200.200.200.5 needs to give the following information to one of the registrars:

Domain name: ws.wonderful.com **IP address:** 200.200.200.5

26.6.8 DDNS

When the DNS was designed, no one predicted that there would be so many address changes. In DNS, when there is a change, such as adding a new host, removing a host, or changing an IP address, the change must be made to the DNS master file. These types of changes involve a lot of manual updating. The size of today's Internet does not allow for this kind of manual operation.

The DNS master file must be updated dynamically. The **Dynamic Domain Name System (DDNS)** therefore was devised to respond to this need. In DDNS, when a binding between a name and an address is determined, the information is sent, usually by DHCP (discussed in Chapter 18) to a primary DNS server. The primary server updates the zone. The secondary servers are notified either actively or passively. In active notification, the primary server sends a message to the secondary servers about the change in the zone, whereas in passive notification, the secondary servers periodically check for any changes. In either case, after being notified about the change, the secondary server requests information about the entire zone (called the *zone transfer*).

To provide security and prevent unauthorized changes in the DNS records, DDNS can use an authentication mechanism.

26.6.9 Security of DNS

DNS is one of the most important systems in the Internet infrastructure; it provides crucial services to Internet users. Applications such as Web access or e-mail are heavily dependent on the proper operation of DNS. DNS can be attacked in several ways including:

1. The attacker may read the response of a DNS server to find the nature or names of sites the user mostly accesses. This type of information can be used to find the user's profile. To prevent this attack, DNS messages need to be confidential (see Chapters 31 and 32).
2. The attacker may intercept the response of a DNS server and change it or create a totally new bogus response to direct the user to the site or domain the attacker wishes the user to access. This type of attack can be prevented using message origin authentication and message integrity (see Chapters 31 and 32).
3. The attacker may flood the DNS server to overwhelm it or eventually crash it. This type of attack can be prevented using the provision against denial-of-service attack.

To protect DNS, IETF has devised a technology named *DNS Security* (DNSSEC) that provides message origin authentication and message integrity using a security service called *digital signature* (see Chapter 31). DNSSEC, however, does not provide confidentiality for the DNS messages. There is no specific protection against the denial-of-service attack in the specification of DNSSEC. However, the caching system protects the upper-level servers against this attack to some extent.

26.7 END-CHAPTER MATERIALS

26.7.1 Recommended Reading

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

Books

Several books give thorough coverage of materials discussed in this chapter including [Com 06], [Mir 07], [Ste 94], [Tan 03], [Bar et al. 05].

RFCs

HTTP is discussed in RFCs 2068, 2109, and 2616. FTP is discussed in RFCs 959, 2577, and 2585. TELNET is discussed in RFCs 854, 855, 856, 1041, 1091, 1372, and 1572. SSH is discussed in RFCs 4250, 4251, 4252, 4253, 4254, and 4344. DNS is discussed in RFCs 1034, 1035, 1996, 2535, 3008, 3342, 3396, 3658, 3755, 3757, and 3845. SMTP is discussed in RFCs 2821 and 2822. POP3 is explained in RFC 1939. MIME is discussed in RFCs 2046, 2047, 2048, and 2049.

26.7.2 Key Terms

active document
browser
cookie
country domain
DNS Server
domain
domain name
domain name space
Domain Name System (DNS)
dynamic document
Dynamic Domain Name System (DDNS)
File Transfer Protocol (FTP)
fully qualified domain name (FQDN)
generic domain
hypermedia
hypertext
HyperText Transfer Protocol (HTTP)
Internet Mail Access Protocol, version 4 (IMAP4)
iterative resolution
label
local logging
message access agent (MAA)
message transfer agent (MTA)
Multipurpose Internet Mail Extensions (MIME)
name space
Network Virtual Terminal (NVT)
nonpersistent connection
partially qualified domain name (PQDN)
persistent connection
port forwarding
Post Office Protocol, version 3 (POP3)
proxy server
recursive resolution
remote logging
resolver
root server
Secure Shell (SSH)
Simple Mail Transfer Protocol (SMTP)
static document
terminal network (TELNET)
uniform resource locator (URL)
user agent (UA)
web page
World Wide Web (WWW, the web)
zone

26.7.3 Summary

The World Wide Web (WWW) is a repository of information linked together from points all over the world. Hypertext and hypermedia documents are linked to one another through pointers. HyperText Transfer Protocol (HTTP) is the main protocol used to access data on the Web.

File Transfer Protocol (FTP) is a TCP/IP client-server application for copying files from one host to another. FTP requires two connections for data transfer: a control connection and a data connection. FTP employs NVT ASCII for communication between dissimilar systems.

Electronic mail is one of the most common applications on the Internet. The e-mail architecture consists of several components such as user agent (UA), message transfer agent (MTA), and message access agent (MAA). The protocol that implements MTA is called Simple Mail Transfer Protocol (SMTP). Two protocols are used to implement MAA: Post Office Protocol, version 3 (POP3) and Internet Mail Access Protocol, version 4 (IMAP4).

TELNET is a client-server application that allows a user to log into a remote machine, giving the user access to the remote system. When a user accesses a remote system via the TELNET process, this is comparable to a time-sharing environment. SSH is the secured version of TELNET that is very common today.

The Domain Name System (DNS) is a client-server application that identifies each host on the Internet with a unique name. DNS organizes the name space in a hierarchical structure to decentralize the responsibilities involved in naming.

26.8 PRACTICE SET

26.8.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

26.8.2 Questions

Q26-1. During the weekend, Alice often needs to access files stored on her office desktop from her home laptop. Last week, she installed a copy of the FTP server process on her desktop at her office and a copy of the FTP client process on her laptop at home. She was disappointed when she could not access her files during the weekend. What could have gone wrong?

Q26-2. Alice has a video clip that Bob is interested in getting; Bob has another video clip that Alice is interested in getting. Bob creates a web page and runs an HTTP server. How can Alice get Bob's clip? How can Bob get Alice's clip?

Q26-3. When an HTTP server receives a request message from an HTTP client, how does the server know when all headers have arrived and the body of the message is to follow?

Q26-4. In a nonpersistent HTTP connection, how can HTTP inform the TCP protocol that the end of the message has been reached?

Q26-5. Can you find an analogy in our daily life as to when we use two separate connections in communication similar to the control and data connections in FTP?

Q26-6. FTP uses two separate well-known port numbers for control and data connection. Does this mean that two separate TCP connections are created for exchanging control information and data?

Q26-7. FTP uses the services of TCP for exchanging control information and data transfer. Could FTP have used the services of UDP for either of these two connections? Explain.

Q26-8. In FTP, which entity (client or server) starts (actively opens) the control connection? Which entity starts (actively opens) the data transfer connection?

Q26-9. What do you think would happen if the control connection were severed before the end of an FTP session? Would it affect the data connection?

Q26-10. In FTP, if the client needs to retrieve one file from the server site and store one file on the server site, how many control connections and how many data-transfer connections are needed?

Q26-11. In FTP, can a server retrieve a file from the client site?

Q26-12. In FTP, can a server get the list of the files or directories from the client?

Q26-13. FTP can transfer files between two hosts using different operating systems with different file formats. What is the reason?

Q26-14. Does FTP have a message format for exchanging commands and responses during control connection?

Q26-15. Does FTP have a message format for exchanging files or a list of directories/files during the file-transfer connection?

Q26-16. Can we have a control connection without a data-transfer connection in FTP? Explain.

Q26-17. Can we have a data-transfer connection without a control connection in FTP? Explain.

Q26-18. Assume we need to download an audio using FTP. What file type should we specify in our command?

Q26-19. Both HTTP and FTP can retrieve a file from a server. Which protocol should we use to download a file?

Q26-20. Are the HELO and MAIL FROM commands both necessary in SMTP? Why or why not?

Q26-21. In Figure 26.13 in the text, what is the difference between the MAIL FROM in the envelope and the FROM in the header?

Q26-22. Alice has been on a long trip without checking her e-mail. She then finds out that she has lost some e-mails or attachments her friends claim they have sent to her. What can be the problem?

Q26-23. Assume a TELNET client uses ASCII to represent characters, but the TELNET server uses EBCDIC to represent characters. How can the client log into the server when character representations are different?

Q26-24. The TELNET application has no commands such as those found in FTP or HTTP to allow the user to do something such as transfer a file or access a web page. In what way can this application be useful?

Q26-25. Can a host use a TELNET client to get services provided by other client-server applications such as FTP or HTTP?

Q26-26. In DNS, which of the following are FQDNs and which are PQDNs?

a. xxx **b.** xxx.yyy.net **c.** zzz.yyy.xxx.edu.

26.8.3 Problems

P26-1. Assume there is a server with the domain name *www.common.com.*

a. Show an HTTP request that needs to retrieve the document **/usr/users/doc.** The client accepts MIME version 1, GIF or JPEG images, but the document should not be more than 4 days old.

b. Show the HTTP response to part *a* for a successful request.

P26-2. In HTTP, draw a figure to show the application of cookies in a scenario in which the server allows only the registered customer to access the server.

P26-3. In HTTP, draw a figure to show the application of cookies in a web portal using two sites.

P26-4. In HTTP, draw a figure to show the application of cookies in a scenario in which the server uses cookies for advertisement. Use only three sites.

P26-5. Draw a diagram to show the use of a proxy server that is part of the client network:

- **a.** Show the transactions between the client, proxy server, and the target server when the response is stored in the proxy server.
- **b.** Show the transactions between the client, proxy server, and the target server when the response is not stored in the proxy server.

P26-6. In Chapter 1, we mentioned that the TCP/IP suite, unlike the OSI model, has no presentation layer. But an application-layer protocol can include some of the features defined in this layer if needed. Does HTTP have any presentation-layer features?

P26-7. In Chapter 1, we mentioned that the TCP/IP suite, unlike the OSI model, has no session layer. But an application-layer protocol can include some of the features defined in this layer if needed. Does HTTP have any session-layer features?

P26-8. HTTP version 1.1 defines the persistent connection as the default connection. Using RFC 2616, find out how a client or server can change this default situation to nonpersistent.

P26-9. In SMTP, a sender sends unformatted text. Show the MIME header.

P26-10. Write concurrent TCP client-server programs to simulate a simplified version of HTTP using only a nonpersistent connection. The client sends an HTTP message; the server responds with the requested file. Use only two types of methods, GET and PUT, and only a few simple headers. Note that after testing, you should be able to test your program with a web browser.

P26-11. Write concurrent TCP client-server programs to simulate a simplified version of POP. The client sends a request to receive an e-mail in its mailbox; the server responds with the e-mail.

P26-12. In SMTP,

- **a.** a non-ASCII message of 1000 bytes is encoded using base64. How many bytes are in the encoded message? How many bytes are redundant? What is the ratio of redundant bytes to the total message?
- **b.** a message of 1000 bytes is encoded using quoted-printable. The message consists of 90 percent ASCII and 10 percent non-ASCII characters. How many bytes are in the encoded message? How many bytes are redundant? What is the ratio of redundant bytes to the total message?
- **c.** Compare the results of the two previous cases. How much is the efficiency improved if the message is a combination of ASCII and non-ASCII characters?

P26-13. Encode the following message in base64:

01010111 00001111 11110000

P26-14. Encode the following message in quoted-printable:

01001111 10101111 01110001

P26-15. According to RFC 1939, a POP3 session is in one of the following four states: closed, authorization, transaction, or update. Draw a diagram to show these four states and how POP3 moves between them.

P26-16. POP3 protocol has some basic commands (that each client/server needs to implement). Using the information in RFC 1939, find the meaning and the use of the following basic commands:

a. STAT **b.** LIST **c.** DELE 4

P26-17. POP3 protocol has some optional commands (that a client/server can implement). Using the information in RFC 1939, find the meaning and the use of the following optional commands:

a. UIDL **b.** TOP 1 15 **c.** USER **d.** PASS

P26-18. Using RFC 1939, assume a POP3 client is in the download-and-keep mode. Show the transaction between the client and the server if the client has only two messages of 192 and 300 bytes to download from the server.

P26-19. Using RFC 1939, assume a POP3 client is in the download-and-delete mode. Show the transaction between the client and the server if the client has only two messages of 230 and 400 bytes to download from the server.

P26-20. In Chapter 1, we mentioned that the TCP/IP suite, unlike the OSI model, has no presentation layer. But an application-layer protocol can include some of the features defined in this layer if needed. Does SMTP have any presentation-layer features?

P26-21. In Chapter 1, we mentioned that the TCP/IP suite, unlike the OSI model, has no session layer. But an application-layer protocol can include some of the features defined in this layer if needed. Does SMTP or POP3 have any session layer features?

P26-22. In FTP, assume a client with user name John needs to store a video clip called *video2* on the directory */top/videos/general* on the server. Show the commands and responses exchanged between the client and the server if the client chooses ephemeral port number 56002.

P26-23. In FTP, a user (Jane) wants to retrieve an EBCDIC file named *huge* from */usr/users/report* directory using the ephemeral port 61017. The file is so large that the user wants to compress it before it is transferred. Show all the commands and responses.

P26-24. In FTP, a user (Jan) wants to make a new directory called *Jan* under the directory */usr/usrs/letters*. Show all of the commands and responses.

P26-25. In FTP, a user (Maria) wants to move a file named *file1* from */usr/users/report* directory to the directory */usr/top/letters*. Note that this is a case of renaming a file. We first need to give the name of the old file and then define the new name. Show all of the commands and responses.

26.9 SIMULATION EXPERIMENTS

26.9.1 Applets

We have created some Java applets to show some of the main concepts discussed in this chapter. It is strongly recommended that the students activate these applets on the book website and carefully examine the protocols in action.

26.9.2 Lab Assignments

In Chapter 1, we downloaded and installed Wireshark and learned about its basic features. In this chapter, we use Wireshark to capture and investigate some application-layer protocols. We use Wireshark to simulate six protocols: HTTP, FTP, TELNET, SMTP, POP3, and DNS.

Lab26-1. In the first lab, we retrieve web pages using HTTP. We use Wireshark to capture packets for analysis. We learn about the most common HTTP messages. We also capture response messages and analyze them. During the lab session, some HTTP headers are also examined and analyzed.

Lab26-2. In the second lab, we use FTP to transfer some files. We use Wireshark to capture some packets. We show that FTP uses two separate connections: a control connection and a data-transfer connection. The data connection is opened and then closed for each file transfer activity. We also show that FTP is an insecure file transfer protocol because the transaction is done in plaintext.

Lab26-3. In the third lab, we use Wireshark to capture packets exchanged by the TELNET protocol. As in FTP, we are able to observe commands and responses in the captured packets during the session. Like FTP, TELNET is vulnerable to hacking because it sends all data including the password in plaintext.

Lab26-4. In the fourth lab, we investigate SMTP protocol in action. We send an e-mail and, using Wireshark, we investigate the contents and the format of the SMTP packet exchanged between the client and the server. We check that the three phases we discussed in the text exist in this SMTP session.

Lab26-5. In the fifth lab, we investigate the state and behavior of the POP3 protocol. We retrieve the mails stored in our mailbox at the POP3 server and observe and analyze the states of the POP3 and the type and the contents of the messages exchanged, by analyzing the packets through Wireshark.

Lab26-6. In the sixth lab, we analyze the behavior of the DNS protocol. In addition to Wireshark, several network utilities are available for finding some information stored in the DNS servers. In this lab, we use the *dig* utilities (which has replaced *nslookup*). We also use *ipconfig* to manage the cached DNS records in the host computer. When we use these utilities, we set Wireshark to capture the packets sent by these utilities.