PART 6

# Application Layer

## Objectives

The application layer enables the user, whether human or software, to access the network. It provides user interfaces and support for services such as electronic mail, file access and transfer, access to system resources, surfing the world wide web, and network management.

**The application layer is responsible for providing services to the user.**

In this part, we briefly discuss some applications that are designed as a client/server pair in the Internet. The client sends a request for a service to the server; the server responds to the client.

**Part 6 of the book is devoted to the application layer and the services provided by this layer.**

## Chapters

This part consists of five chapters: Chapters 25 to 29.

### *Chapter 25*

Chapter 25 discusses the Domain Name System (DNS). DNS is a client/server application that provides name services for other applications. It enables the use of application-layer addresses, such as an email address, instead of network layer logical addresses.

### *Chapter 26*

Chapter 26 discusses three common applications in the Internet: remote login, electronic mail, and file transfer. A remote login application allows the user to remotely access the resources of a system. An electronic mail application simulates the tasks of snail mail at a much higher speed. A file transfer application transfers files between remote systems.

### *Chapter 27*

Chapter 27 discuss the ideas and issues in the famous world wide web (WWW). It also briefly describes the client/server application program (HTTP) that is commonly used to access the web.

### *Chapter 28*

Chapter 28 is devoted to network management. We first discuss the general idea behind network management. We then introduce the client/server application, SNMP, that is used for this purpose in the Internet. Although network management can be implemented in every layer, the Internet has decided to use a client/server application.

### *Chapter 29*

Chapter 29 discusses multimedia and a set of wiely-used application programs. These programs have generated new issues such as the need for new protocols in other layers to handle the specific problems related to multimedia. We briefly discuss these issues in this chapter.

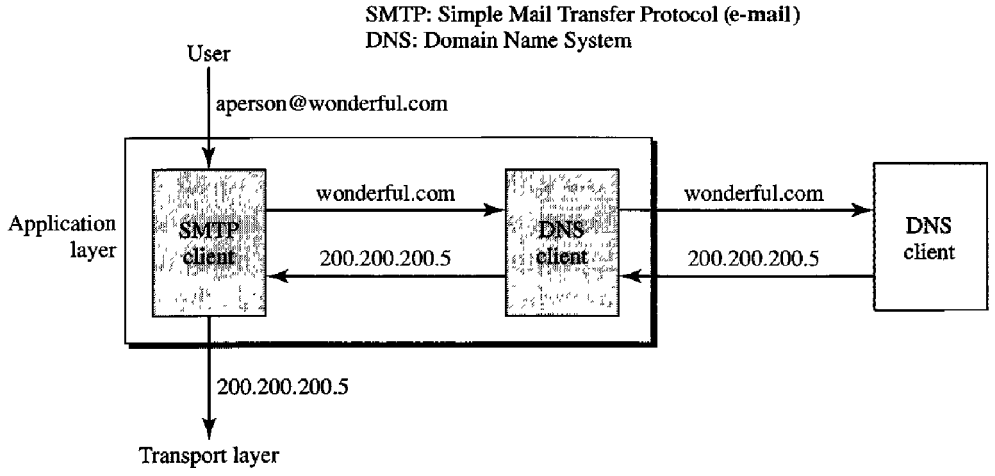
# CHAPTER 25

# *Domain Name System*

There are several applications in the application layer of the Internet model that follow the client/server paradigm. The client/server programs can be divided into two categories: those that can be directly used by the user, such as e-mail, and those that support other application programs. The Domain Name System (DNS) is a supporting program that is used by other programs such as e-mail.

Figure 25.1 shows an example of how a DNS client/server program can support an e-mail program to find the IP address of an e-mail recipient. A user of an e-mail program may know the e-mail address of the recipient; however, the IP protocol needs the IP address. The DNS client program sends a request to a DNS server to map the e-mail address to the corresponding IP address.

**Figure 25.1** *Example of using the DNS service*




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, we need a system that can map a name to an address or an address to a name.

When the Internet was small, mapping was done by using a **host file.** The host file had only two columns: name and address. Every host could store the host file on its disk and update it periodically from a master host file. When a program or a user wanted to map a name to an address, the host consulted the host file and found the mapping.

Today, however, it is impossible to have one single host file to relate every address with a name and vice versa. The host file would be too large to store in every host. In addition, it would be impossible to update all the host files every time there was a change.

One solution would be to store the entire host file in a single computer and allow access to this centralized information to every computer that needs mapping. But we know that this would create a huge amount of traffic on the Internet.

Another solution, the one used today, is to divide this huge amount of information into smaller parts and store each part on a different computer. 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).** In this chapter, we first discuss the concepts and ideas behind the DNS. We then describe the DNS protocol itself.

## 25.1 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.

### Flat Name Space

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.

### Hierarchical Name Space

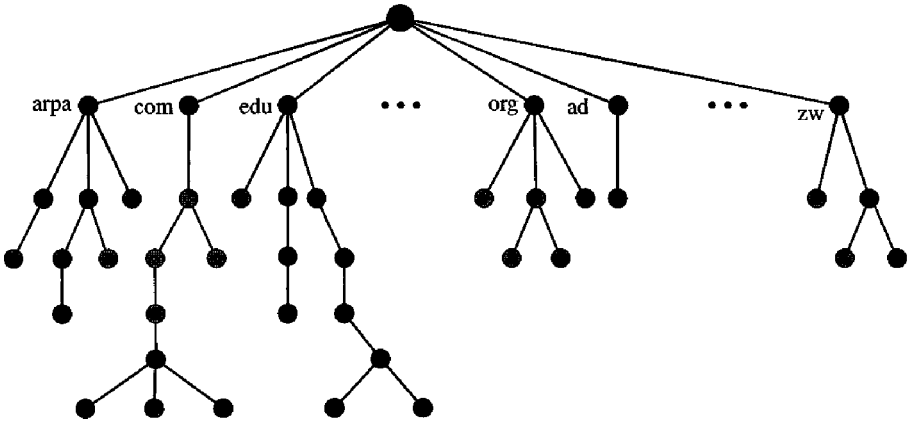
In a **hierarchical name space,** each name is made of several parts. The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on. In this case, the authority to assign and control 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 of 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 colleges and a company call one of their computers *challenger.* The first college is given a name by the central authority such as *fhda.edu,* the second college is given the name *berkeley.edu,* and the company is given the name *smart.com.* When these organizations add the name *challenger* to the name they have already been given, the end result is three distinguishable names: *challenger.fhda.edu, challenger.berkeley.edu,* and *challenger.smart.com.* The names are unique without the need for assignment by a central authority. The central authority controls only part of the name, not the whole.

## 25.2 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 25.2).

**Figure 25.2** *Domain name space*



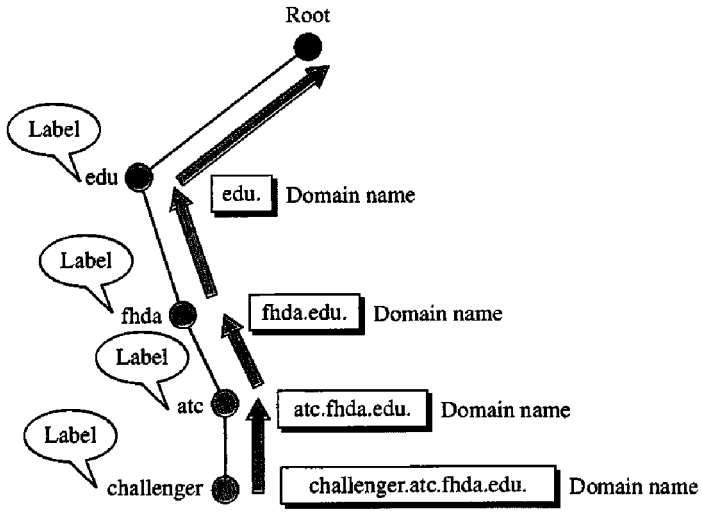

### Label

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 25.3 shows some domain names.

**Figure 25.3** *Domain names and labels*




### *Fully Qualified Domain Name*

If a label is terminated by a null string, it is called a **fully qualified domain name (FQDN).** An FQDN is a domain name that contains the full name of a host. It contains all labels, from the most specific to the most general, that uniquely define the name of the host. For example, the domain name

challenger.atc.fhda.edu.

is the FQDN of a computer named *challenger* installed at the Advanced Technology Center (ATC) at De Anza College. A DNS server can only match an FQDN to an address. Note that the name must end with a null label, but because null means nothing, the label ends with a dot (.).

### *Partially Qualified Domain Name*

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. For example, if a user at the *fhda.edu.* site wants to get the IP address of the challenger computer, he or she can define the partial name

challenger

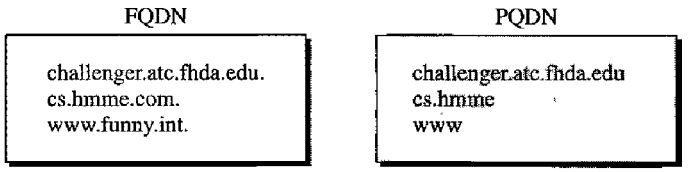
The DNS client adds the suffix *atc.fhda.edu.* before passing the address to the DNS server.

The DNS client normally holds a list of suffixes. The following can be the list of suffixes at De Anza College. The null suffix defines nothing. This suffix is added when the user defines an FQDN.

atc.fhda.edu
fhda.edu
*null*

Figure 25.4 shows some FQDNs and PQDNs.

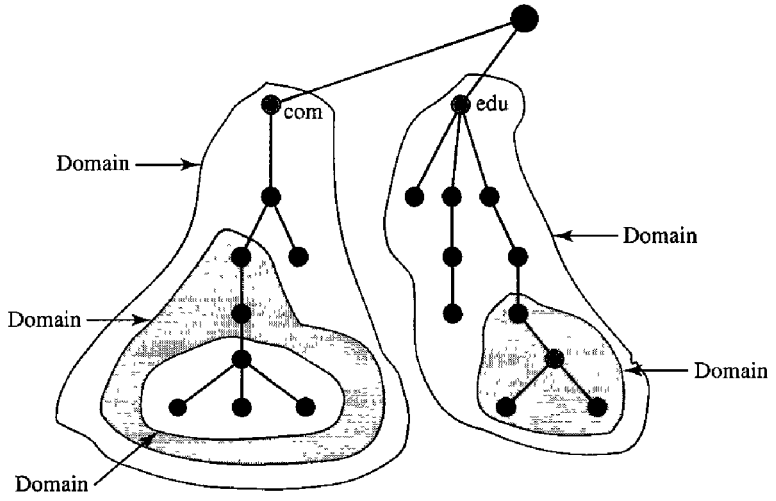
**Figure 25.4** *FQDN and PQDN*




## Domain

A **domain** is a subtree of the domain name space. The name of the domain is the domain name of the node at the top of the subtree. Figure 25.5 shows some domains. Note that a domain may itself be divided into domains (or **subdomains** as they are sometimes called).

**Figure 25.5** *Domains*



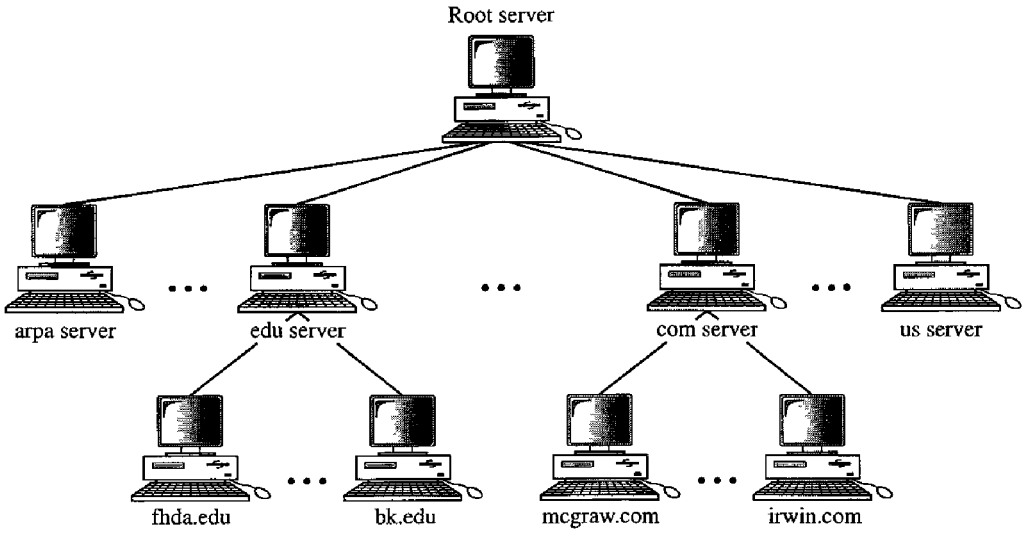

# 25.3 DISTRIBUTION OF NAME SPACE

The information contained in the domain name space must be stored. However, it is very inefficient and also unreliable to have just one computer store such a huge 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 unreliable 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 in 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 a small domain. In other words, we have a hierarchy of servers in the same way that we have a hierarchy of names (see Figure 25.6).

**Figure 25.6** *Hierarchy of name servers*

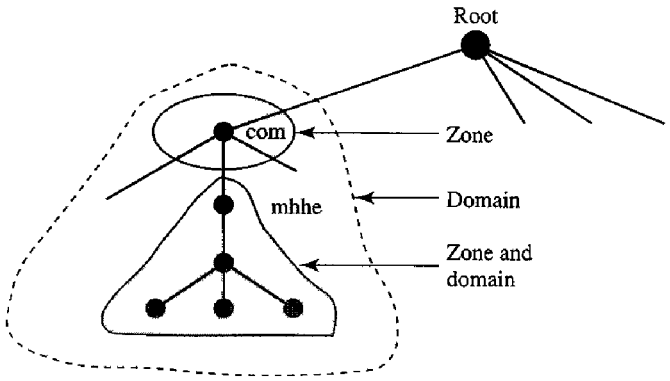



## Zone

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 25.7).

A server can also divide part of its domain and delegate responsibility but still keep part of the domain for itself. In this case, its zone is made of detailed information for the part of the domain that is not delegated and references to those parts that are delegated.

**Figure 25.7** *Zones and domains*




## 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 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.

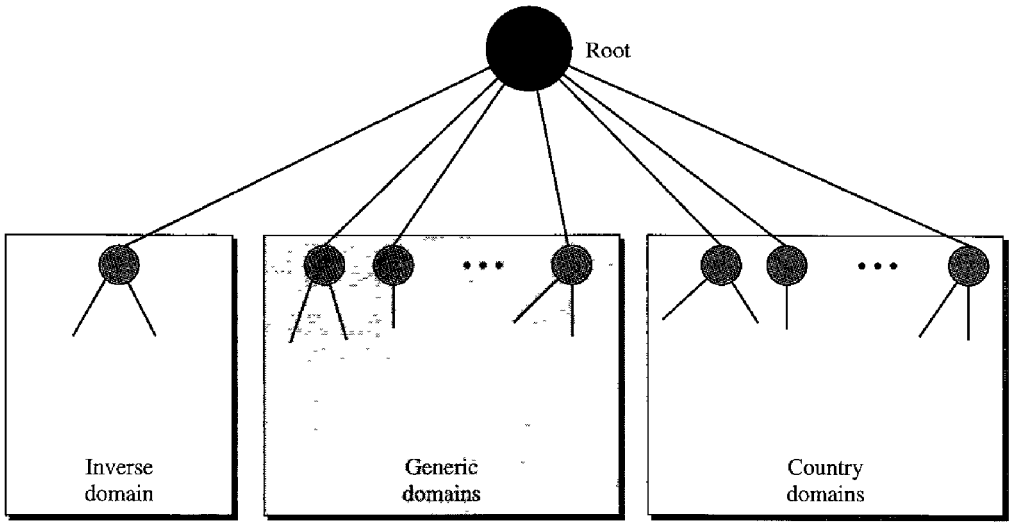
The primary and secondary servers are both authoritative for the zones they serve. The idea is not to put the secondary server at a lower level of authority but to create redundancy for the data so that if one server fails, the other can continue serving clients. Note also that a server can be a primary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

**A primary server loads all information from the disk file; the secondary server loads all information from the primary server. When the secondary downloads information from the primary, it is called zone transfer.**

# 25.4 DNS IN THE INTERNET

DNS is a protocol that can be used in different platforms. In the Internet, the domain name space (tree) is divided into three different sections: generic domains, country domains, and the inverse domain (see Figure 25.8).

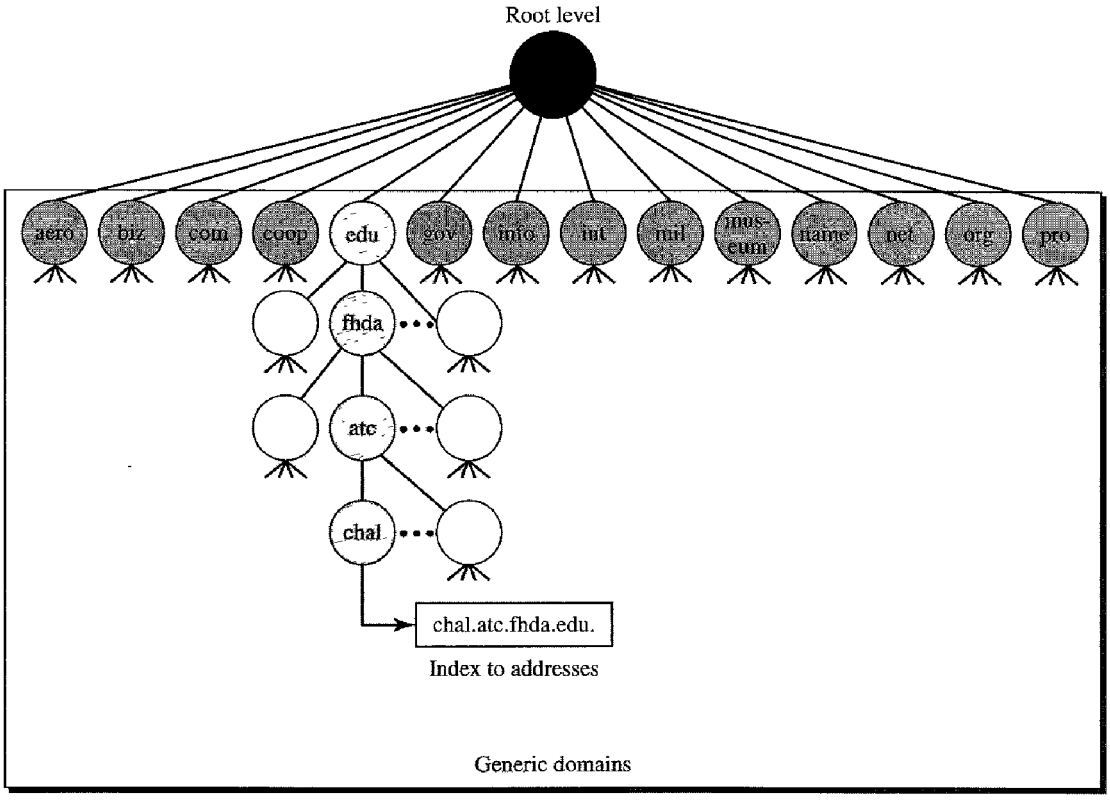
**Figure 25.8** *DNS used in the Internet*




## 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 25.9).

**Figure 25.9** *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 25.1.

**Table 25.1** *Generic domain labels*

| *Label* | *Description* |
|---|---|
| **aero** | Airlines and aerospace companies |
| **biz** | Businesses or firms (similar to "com") |
| **com** | Commercial organizations |
| **coop** | Cooperative business organizations |
| **edu** | Educational institutions |
| **gov** | Government institutions |
| **info** | Information service providers |
| **int** | International organizations |
| **mil** | Military groups |
| **museum** | Museums and other nonprofit organizations |
| **name** | Personal names (individuals) |
| **net** | Network support centers |
| **org** | Nonprofit organizations |
| **pro** | Professional individual 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 25.10 shows the country domains section. The address *anza.cup.ca.us* can be translated to De Anza College in Cupertino, California, in the United States.

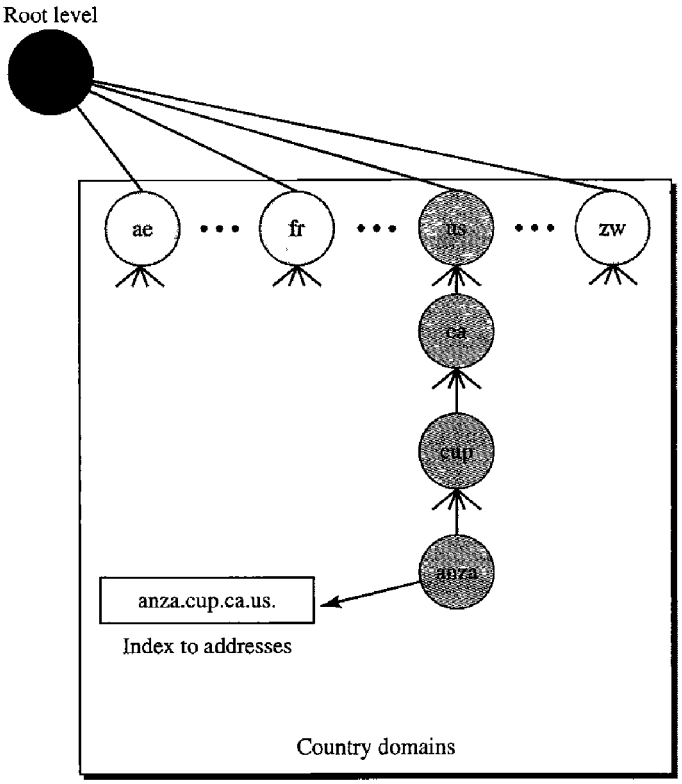
## Inverse Domain

The **inverse domain** is used to map an address to a name. This may happen, for example, when a server has received a request from a client to do a task. Although the server has a file that contains a list of authorized clients, only the IP address of the client (extracted from the received IP packet) is listed. The server asks its resolver to send a query to the DNS server to map an address to a name to determine if the client is on the authorized list.

This type of query is called an inverse or pointer (PTR) query. To handle a pointer query, the inverse domain is added to the domain name space with the first-level node called *arpa* (for historical reasons). The second level is also one single node named *in-addr* (for inverse address). The rest of the domain defines IP addresses.

The servers that handle the inverse domain are also hierarchical. This means the netid part of the address should be at a higher level than the subnetid part, and the subnetid part

**Figure 25.10** *Country domains*




higher than the hostid part. In this way, a server serving the whole site is at a higher level than the servers serving each subnet. This configuration makes the domain look inverted when compared to a generic or country domain. To follow the convention of reading the domain labels from the bottom to the top, an IP address such as 132.34.45.121 (a class B address with netid 132.34) is read as 121.45.34.132.in-addr. arpa. See Figure 25.11 for an illustration of the inverse domain configuration.

## 25.5 RESOLUTION

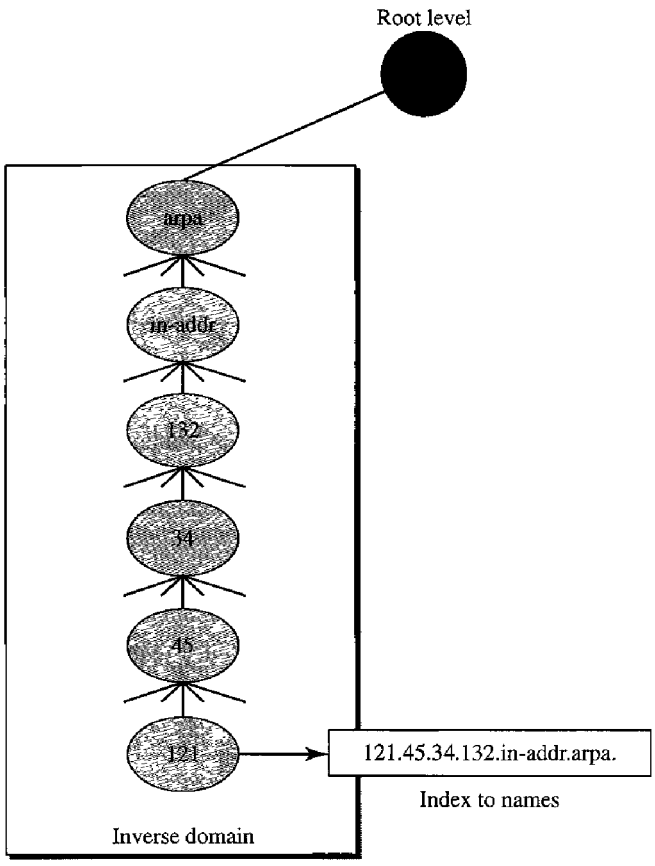
Mapping a name to an address or an address to a name is called *name-address resolution.*

### Resolver

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.

**Figure 25.11** *Inverse domain*




## Mapping Names to Addresses

Most of the time, the resolver gives a domain name to the server and asks for the corresponding address. In this case, the server checks the generic domains or the country domains to find the mapping.

If the domain name is from the generic domains section, the resolver receives a domain name such as "*chal.atc.fhda.edu.*". The query is sent by the resolver to the local DNS server for resolution. If the local server cannot resolve the query, it either refers the resolver to other servers or asks other servers directly.

If the domain name is from the country domains section, the resolver receives a domain name such as "*ch.fhda.cu.ca.us.*". The procedure is the same.

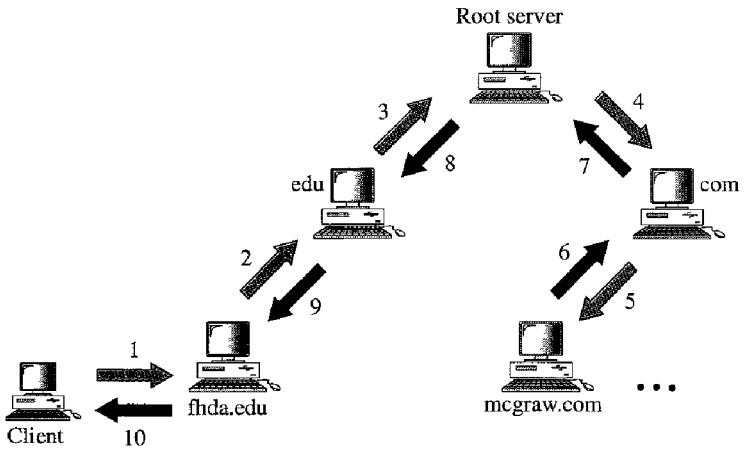
## Mapping Addresses to Names

A client can send an IP address to a server to be mapped to a domain name. As mentioned before, this is called a PTR query. To answer queries of this kind, DNS uses the inverse domain. However, in the request, the IP address is reversed and the two labels *in-addr* and *arpa* are appended to create a domain acceptable by the inverse domain section. For example, if the resolver receives the IP address 132.34.45.121, the resolver first inverts the address and then adds the two labels before sending. The domain name sent is "*121.45.34.132.in-addr.arpa.*" which is received by the local DNS and resolved.

## Recursive Resolution

The client (resolver) can ask for a recursive answer from a name server. This means that the resolver expects the server to supply the final answer. If the server is the authority for the domain name, it checks its database and responds. If the server is not the authority, it sends the request to another server (the parent usually) and waits for the response. If the parent is the authority, it responds; otherwise, it sends the query to yet another server. When the query is finally resolved, the response travels back until it finally reaches the requesting client. This is called **recursive resolution** and is shown in Figure 25.12.

**Figure 25.12** *Recursive resolution*



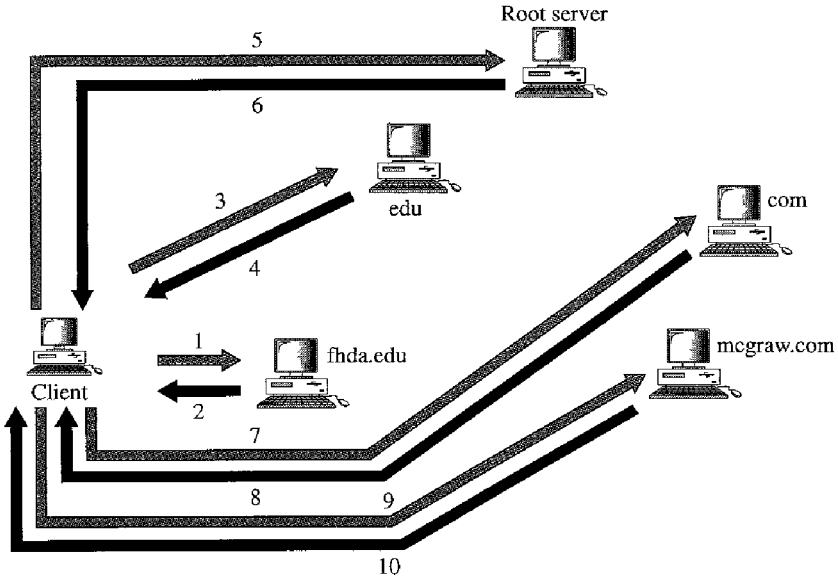

## Iterative Resolution

If the client does not ask for a recursive answer, the mapping can be done iteratively. If the server is an authority for the name, it sends the answer. If it is not, it returns (to the client) the IP address of the server that it thinks can resolve the query. The client is responsible for repeating the query to this second server. If the newly addressed server can resolve the problem, it answers the query with the IP address; otherwise, it returns the IP address of a new server to the client. Now the client must repeat the query to the third server. This process is called **iterative resolution** because the client repeats the same query to multiple servers. In Figure 25.13 the client queries four servers before it gets an answer from the mcgraw.com server.

## 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 solve the problem. However, to

**Figure 25.13** *Iterative resolution*




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.

## 25.6 DNS MESSAGES

DNS has two types of messages: query and response. Both types have the same format. The **query message** consists of a header and question records; the **response message** consists of a header, question records, answer records, authoritative records, and additional records (see Figure 25.14).

### Header

Both query and response messages have the same header format with some fields set to zero for the query messages. The header is 12 bytes, and its format is shown in Figure 25.15.

The *identification* subfield is used by the client to match the response with the query. The client uses a different identification number each time it sends a query. The server duplicates this number in the corresponding response. The *flags* subfield is a collection of

**Figure 25.14** *Query and response messages*

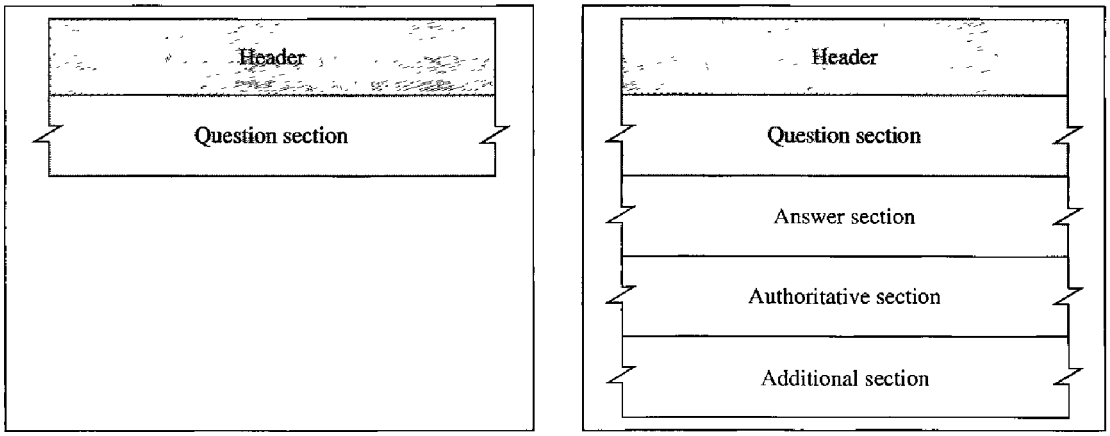



**Figure 25.15** *Header format*

| Identification | Flags |
|---|---|
| Number of question records | Number of answer records (all 0s in query message) |
| Number of authoritative records (all 0s in query message) | Number of additional records (all 0s in query message) |

subfields that define the type of the message, the type of answer requested, the type of desired resolution (recursive or iterative), and so on. The *number of question records* subfield contains the number of queries in the question section of the message. The *number of answer records* subfield contains the number of answer records in the answer section of the response message. Its value is zero in the query message. The *number of authoritative records* subfield contains the number of authoritative records in the authoritative section of a response message. Its value is zero in the query message. Finally, the *number of additional records* subfield contains the number additional records in the additional section of a response message. Its value is zero in the query message.

### *Question Section*

This is a section consisting of one or more question records. It is present on both query and response messages. We will discuss the question records in a following section.

### *Answer Section*

This is a section consisting of one or more resource records. It is present only on response messages. This section includes the answer from the server to the client (resolver). We will discuss resource records in a following section.

*Authoritative Section*

This is a section consisting of one or more resource records. It is present only on response messages. This section gives information (domain name) about one or more authoritative servers for the query.

*Additional Information Section*

This is a section consisting of one or more resource records. It is present only on response messages. This section provides additional information that may help the resolver. For example, a server may give the domain name of an authoritative server to the resolver in the authoritative section, and include the IP address of the same authoritative server in the additional information section.

## 25.7 TYPES OF RECORDS

As we saw in Section 25.6, two types of records are used in DNS. The question records are used in the question section of the query and response messages. The resource records are used in the answer, authoritative, and additional information sections of the response message.

### Question Record

A **question record** is used by the client to get information from a server. This contains the domain name.

### Resource Record

Each domain name (each node on the tree) is associated with a record called the **resource record.** The server database consists of resource records. Resource records are also what is returned by the server to the client.

## 25.8 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

## 25.9 DYNAMIC DOMAIN NAME SYSTEM (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 (see Chapter 21) 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 requests information about the entire zone (zone transfer).

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

## 25.10 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.

**DNS can use the services of UDP or TCP using the well-known port 53.**

## 25.11 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [. . .] refer to the reference list at the end of the text.

### Books

DNS is discussed in [AL98], Chapter 17 of [For06], Section 9.1 of [PD03], and Section 7.1 of [Tan03].

### Sites

The following sites are related to topics discussed in this chapter.

- ❑ www.intenic.net/ Information about registrars
- ❑ www.ietf.org/rfc.html Information about RFCs

### RFCs

The following RFCs are related to DNS:

799, 811, 819, 830, 881, 882, 883, 897, 920, 921, 1034, 1035, 1386, 1480, 1535, 1536, 1537, 1591, 1637, 1664, 1706, 1712, 1713, 1982, 2065, 2137, 2317, 2535, 2671

## 25.12 KEY TERMS

caching
country domain
DNS server
domain
domain name
domain name space
Domain Name System (DNS)
Dynamic Domain Name System (DDNS)
flat name space
fully qualified domain name (FQDN)
generic domain
hierarchical name space
host file
inverse domain
iterative resolution
label
name space
partially qualified domain name (PQDN)
primary server
query message
question record
recursive resolution
registrar
resolver
resource record
response message
root server
secondary server
subdomain
suffix
zone

## 25.13 SUMMARY

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

- DNS can be pictured as an inverted hierarchical tree structure with one root node at the top and a maximum of 128 levels.
- Each node in the tree has a domain name.
- A domain is defined as any subtree of the domain name space.
- The name space information is distributed among DNS servers. Each server has jurisdiction over its zone.
- A root server's zone is the entire DNS tree.
- A primary server creates, maintains, and updates information about its zone.
- A secondary server gets its information from a primary server.
- The domain name space is divided into three sections: generic domains, country domains, and inverse domain.
- There are 14 generic domains, each specifying an organization type.
- Each country domain specifies a country.
- The inverse domain finds a domain name for a given IP address. This is called address-to-name resolution.
- Name servers, computers that run the DNS server program, are organized in a hierarchy.
- The DNS client, called a resolver, maps a name to an address or an address to a name.
- In recursive resolution, the client sends its request to a server that eventually returns a response.
- In iterative resolution, the client may send its request to multiple servers before getting an answer.
- Caching is a method whereby an answer to a query is stored in memory (for a limited time) for easy access to future requests.
- A fully qualified domain name (FQDN) is a domain name consisting of labels beginning with the host and going back through each level to the root node.
- A partially qualified domain name (PQDN) is a domain name that does not include all the levels between the host and the root node.
- There are two types of DNS messages: queries and responses.
- There are two types of DNS records: question records and resource records.
- Dynamic DNS (DDNS) automatically updates the DNS master file.
- DNS uses the services of UDP for messages of less than 512 bytes; otherwise, TCP is used.

## 25.14 PRACTICE SET

### Review Questions

1. What is an advantage of a hierarchical name space over a flat name space for a system the size of the Internet?
2. What is the difference between a primary server and a secondary server?
3. What are the three domains of the domain name space?

4. What is the purpose of the inverse domain?
5. How does recursive resolution differ from iterative resolution?
6. What is an FQDN?
7. What is a PQDN?
8. What is a zone?
9. How does caching increase the efficiency of name resolution?
10. What are the two main categories of DNS messages?
11. Why was there a need for DDNS?

## Exercises

12. Determine which of the following is an FQDN and which is a PQDN.
    a. xxx
    b. xxx.yyy.
    c. xxx.yyy.net
    d. zzz.yyy.xxx.edu.
13. Determine which of the following is an FQDN and which is a PQDN.
    a. mil.
    b. edu.
    c. xxx.yyy.net
    d. zzz.yyy.xxx.edu
14. Which domain is used by your system, generic or country?
15. Why do we need a DNS system when we can directly use an IP address?
16. To find the IP address of a destination, we need the service of DNS. DNS needs the service of UDP or TCP. UDP or TCP needs the service of IP. IP needs an IP destination address. Is this a vicious cycle here?
17. If a DNS domain name is *voyager.fhda.edu,* how many labels are involved here? How many levels of hierarchy?
18. Is a PQDN necessarily shorter than the corresponding FQDN?
19. A domain name is *hello.customer.info*. Is this a generic domain or a country domain?
20. Do you think a recursive resolution is normally faster than an interactive one? Explain.
21. Can a query message have one question section but the corresponding response message have several answer sections?

# CHAPTER 26

# *Remote Logging, Electronic Mail, and File Transfer*

The main task of the Internet is to provide services for users. Among the most popular applications are remote logging, electronic mail, and file transfer. We discuss these three applications in this chapter; we discuss another popular use of the Internet, accessing the World Wide Web, in Chapter 27.

## 26.1 REMOTE LOGGING

In the Internet, users may want to run application programs at a remote site and create results that can be transferred to their local site. For example, students may want to connect to their university computer lab from their home to access application programs for doing homework assignments or projects. One way to satisfy that demand and others is to create a client/server application program for each desired service. Programs such as file transfer programs (FTPs), e-mail (SMTP), and so on are currently available. However, it would be impossible to write a specific client/server program for each demand.

The better solution is a general-purpose client/server program that lets a user access any application program on a remote computer; in other words, allow the user to log on to a remote computer. After logging on, a user can use the services available on the remote computer and transfer the results back to the local computer.

### TELNET

In this section, we discuss such a client/server application program: TELNET. **TELNET** is an abbreviation for *TErminaL NETwork*. It is the standard TCP/IP protocol for virtual terminal service as proposed by the International Organization for Standards (ISO). TELNET enables the establishment of a connection to a remote system in such a way that the local terminal appears to be a terminal at the remote system.

**TELNET is a general-purpose client/server application program.**

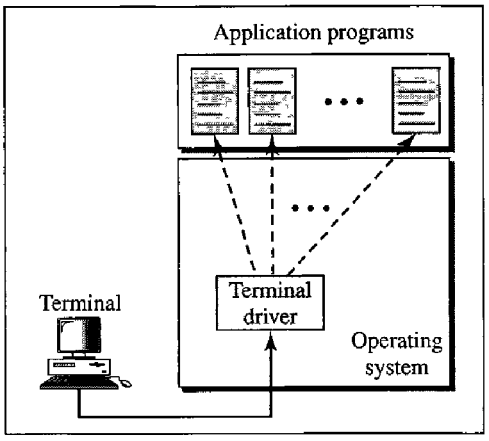
### *Timesharing Environment*

TELNET was designed at a time when most operating systems, such as UNIX, were operating in a **timesharing** environment. In such an environment, a large computer supports multiple users. The interaction between a user and the computer occurs through a terminal, which is usually a combination of keyboard, monitor, and mouse. Even a microcomputer can simulate a terminal with a terminal emulator. .

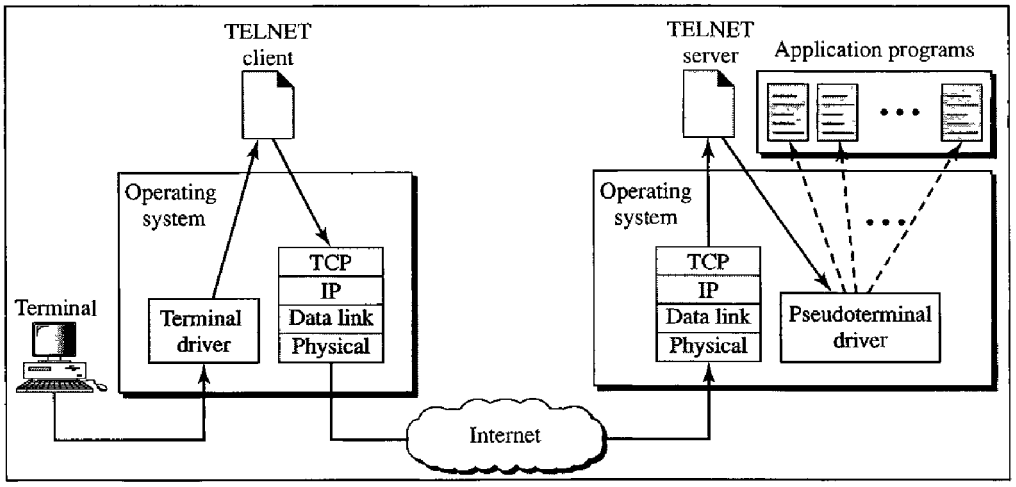
### *Logging*

In a timesharing environment, users are part of the system with some right to access resources. Each authorized user has an identification and probably a password. The user identification defines the user as part of the system. To access the system, the user logs into the system with a user id or log-in name. The system also includes password checking to prevent an unauthorized user from accessing the resources. Figure 26.1 shows the logging process.

**Figure 26.1** *Local and remote log-in*




**a. Local log-in**




**b. Remote log-in**

When a user logs into a local timesharing system, it is called **local log-in.** 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.

When a user wants to access an application program or utility located on a remote machine, she performs **remote log-in.** 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 to a universal character set called *network virtual terminal (NVT) characters* and delivers them to the local TCP/IP protocol 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*

The mechanism to access a remote computer is complex. This is so because every computer and its operating system accept 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. For an illustration of this concept, see Figure 26.2.

**NVT Character Set** NVT uses two sets of characters, one for data and the other for control. Both are 8-bit bytes. For data, NVT is an 8-bit character set in which the 7 lowest-order bits are the same as 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.

Table 26.1 lists some of the control characters and their meanings.

**Figure 26.2** *Concept of NVT*

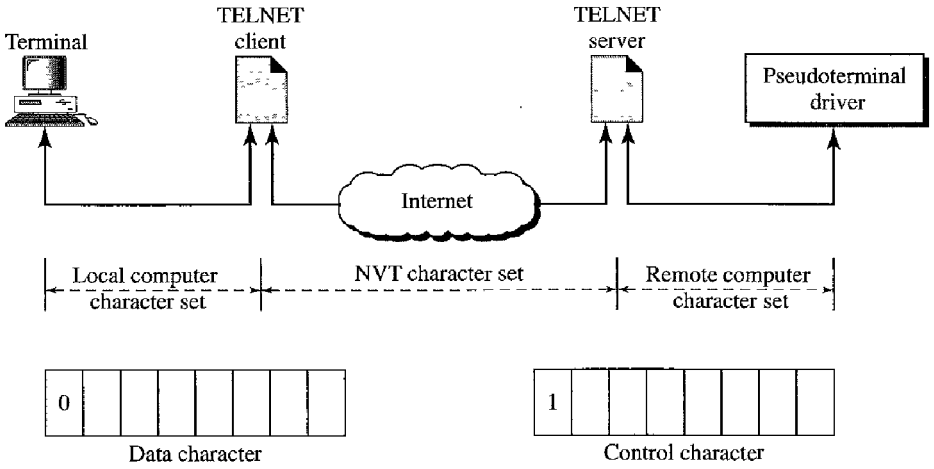



**Table 26.1** *Some NVT control characters*

| *Character* | *Decimal* | *Binary* | *Meaning* |
|---|---|---|---|
| EOF | 236 | 11101100 | End of file |
| EOR | 239 | 11101111 | End of record |
| SE | 240 | 11110000 | Suboption end |
| NOP | 241 | 11110001 | No operation |
| DM | 242 | 11110010 | Data mark |
| BRK | 243 | 11110011 | Break |
| IP | 244 | 11110100 | Interrupt process |
| AO | 245 | 11110101 | Abort output |
| AYT | 246 | 11110110 | Are you there? |
| EC | 247 | 11110111 | Erase character |
| EL | 248 | 11111000 | Erase line |
| GA | 249 | 11111001 | Go ahead |
| SB | 250 | 11111010 | Suboption begin |
| WILL | 251 | 11111011 | Agreement to enable option |
| WONT | 252 | 11111100 | Refusal to enable option |
| DO | 253 | 11111101 | Approval to option request |
| DONT | 254 | 11111110 | Denial of option request |
| IAC | 255 | 11111111 | Interpret (the next character) as control |

### *Embedding*

TELNET uses only one TCP connection. The server uses the well-known port 23, and the client uses an ephemeral port. The same connection is used for sending both data and

control characters. TELNET accomplishes this by embedding the control characters in the data stream. However, to distinguish data from control characters, each sequence of control characters is preceded by a special control character called *interpret as control* (IAC). For example, imagine a user wants a server to display a file (*file1*) on a remote server. She can type

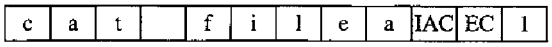
***cat file1***

However, suppose the name of the file has been mistyped (*filea* instead of *file1*). The user uses the backspace key to correct this situation.

***cat filea<backspace>1***

However, in the default implementation of TELNET, the user cannot edit locally; the editing is done at the remote server. The backspace character is translated into two remote characters (IAC EC), which are embedded in the data and sent to the remote server. What is sent to the server is shown in Figure 26.3.

**Figure 26.3** *An example of embedding*




Typed at the remote terminal

### *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. Some control characters discussed previously are used to define options. Table 26.2 shows some common options.

**Table 26.2** *Options*

| *Code* | *Option* | *Meaning* |
|---|---|---|
| 0 | Binary | Interpret as 8-bit binary transmission. |
| 1 | Echo | Echo the data received on one side to the other. |
| 3 | Suppress go ahead | Suppress go-ahead signals after data. |
| 5 | Status | Request the status of TELNET. |
| 6 | Timing mark | Define the timing marks. |
| 24 | Terminal type | Set the terminal type. |
| 32 | Terminal speed | Set the terminal speed. |
| 34 | Line mode | Change to line mode. |

**Option Negotiation** To use any of the options mentioned in the previous section first requires **option negotiation** between the client and the server. Four control characters are used for this purpose; these are shown in Table 26.3.

**Table 26.3** *NVT character set for option negotiation*

| *Character* | *Decimal* | *Binary* | *Meaning* |
|---|---|---|---|
| WILL | 251 | 11111011 | 1. Offering to enable<br>2. Accepting a request to enable |
| WONT | 252 | 11111100 | 1. Rejecting a request to enable<br>2. Offering to disable<br>3. Accepting a request to disable |
| DO | 253 | 11111101 | 1. Approving an offer to enable<br>2. Requesting to enable |
| DONT | 254 | 11111110 | 1. Disapproving an offer to enable<br>2. Approving an offer to disable<br>3. Requesting to disable |

A party can offer to enable or disable an option if it has the right to do so. The offering can be approved or disapproved by the other party. To offer enabling, the offering party sends the WILL command, which means "Will I enable the option?" The other party sends either the DO command, which means "Please do," or the DONT command, which means "Please don't." To offer disabling, the offering party sends the WONT command, which means "I won't use this option any more." The answer must be the DONT command, which means "Don't use it anymore."

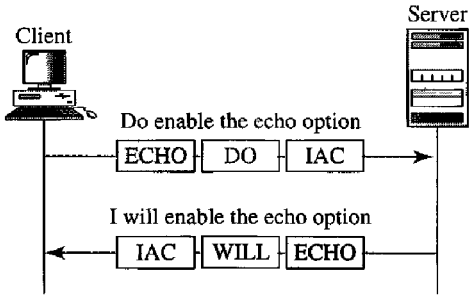
A party can request from the other party the enabling or the disabling of an option. To request enabling, the requesting party sends the DO command, which means "Please do enable the option." The other party sends either the WILL command, which means "I will," or the WONT command, which means "I won't." To request disabling, the requesting party sends the DONT command, which means "Please don't use this option anymore." The answer must be the WONT command, which means "I won't use it anymore."

### *Example 26.1*

Figure 26.4 shows an example of option negotiation. In this example, the client wants the server to echo each character sent to the server. In other words, when a character is typed at the user keyboard terminal, it goes to the server and is sent back to the screen of the user before being processed. The echo option is enabled by the server because it is the server that sends the characters back to the user terminal. Therefore, the client should *request* from the server the enabling of the option using DO. The request consists of three characters: IAC, DO, and ECHO. The server accepts the request and enables the option. It informs the client by sending the three-character approval: IAC, WILL, and ECHO.

**Suboption Negotiation** Some options require additional information. For example, to define the type or speed of a terminal, the negotiation includes a string or a number

**Figure 26.4** *Example 26.1: Echo option*




to define the type or speed. In either case, the two suboption characters indicated in Table 26.4 are needed for **suboption negotiation.**

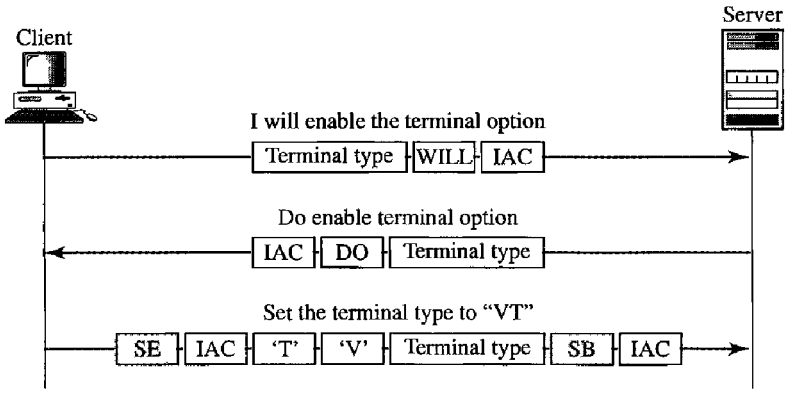
**Table 26.4** *NVT character set for suboption negotiation*

| *Character* | *Decimal* | *Binary* | *Meaning* |
|---|---|---|---|
| SE | 240 | 11110000 | Suboption end |
| SB | 250 | 11111010 | Suboption begin |

### *Example 26.2*

Figure 26.5 shows an example of suboption negotiation. In this example, the client wants to negotiate the type of the terminal.

**Figure 26.5** *Example of suboption negotiation*




### *Mode of Operation*

Most TELNET implementations operate in one of three modes: default mode, character mode, or line mode.

**Default Mode** The **default mode** is used if no other modes are invoked through option negotiation. In this mode, the echoing is done by the client. The user types a

character, and the client echoes the character on the screen (or printer) but does not send it until a whole line is completed.

**Character Mode** In the **character mode,** each character typed is sent by the client to the server. The server normally echoes the character back to be displayed on the client screen. In this mode the echoing of the character can be delayed if the transmission time is long (such as in a satellite connection). It also creates overhead (traffic) for the network because three TCP segments must be sent for each character of data.

**Line Mode** A new mode has been proposed to compensate for the deficiencies of the default mode and the character mode. In this mode, called the **line mode,** line editing (echoing, character erasing, line erasing, and so on) is done by the client. The client then sends the whole line to the server.

## 26.2 ELECTRONIC MAIL

One of the most popular Internet services is electronic mail (e-mail). The designers of the Internet probably never imagined the popularity of this application program. Its architecture consists of several components that we discuss in this chapter.

At the beginning of the Internet era, the messages sent by electronic mail were short and consisted of text only; they let people exchange quick memos. Today, electronic mail is much more complex. It allows a message to include text, audio, and video. It also allows one message to be sent to one or more recipients.

In this chapter, we first study the general architecture of an e-mail system including the three main components: user agent, message transfer agent, and message access agent. We then describe the protocols that implement these components.

### Architecture

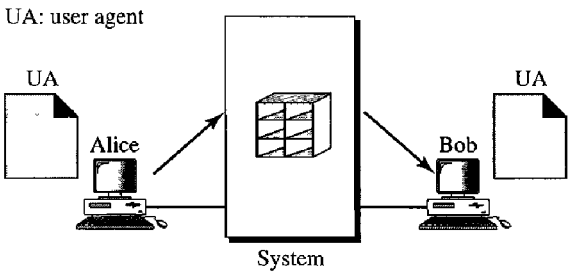
To explain the architecture of e-mail, we give four scenarios. We begin with the simplest situation and add complexity as we proceed. The fourth scenario is the most common in the exchange of email.

#### *First Scenario*

In the first scenario, the sender and the receiver of the e-mail are users (or application programs) on the same system; they are directly connected to a shared system. The administrator has created one mailbox for each user where the received messages are stored. A *mailbox* is part of a local hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it. When Alice, a user, needs to send a message to Bob, another user, Alice runs a *user agent (UA)* program to prepare the message and store it in Bob's mailbox. The message has the sender and recipient mailbox addresses (names of files). Bob can retrieve and read the contents of his mailbox at his convenience, using a user agent. Figure 26.6 shows the concept.

This is similar to the traditional memo exchange between employees in an office. There is a mailroom where each employee has a mailbox with his or her name on it.

**Figure 26.6** *First scenario in electronic mail*



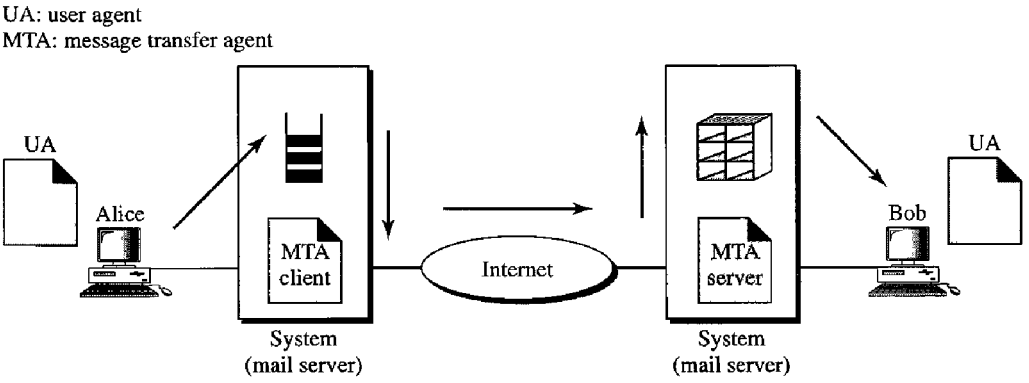

When Alice needs to send a memo to Bob, she writes the memo and inserts it into Bob's mailbox. When Bob checks his mailbox, he finds Alice's memo and reads it.

**When the sender and the receiver of an e-mail are on the same system, we need only two user agents.**

*Second Scenario*

In the second scenario, the sender and the receiver of the e-mail are users (or application programs) on two different systems.The message needs to be sent over the Internet. Here we need **user agents (UAs)** and **message transfer agents (MTAs),** as shown in Figure 26.7.

**Figure 26.7** *Second scenario in electronic mail*




Alice needs to use a user agent program to send her message to the system at her own site. The system (sometimes called the mail server) at her site uses a queue to store messages waiting to be sent. Bob also needs a user agent program to retrieve messages stored in the mailbox of the system at his site. The message, however, needs to be sent through the Internet from Alice's site to Bob's site. 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

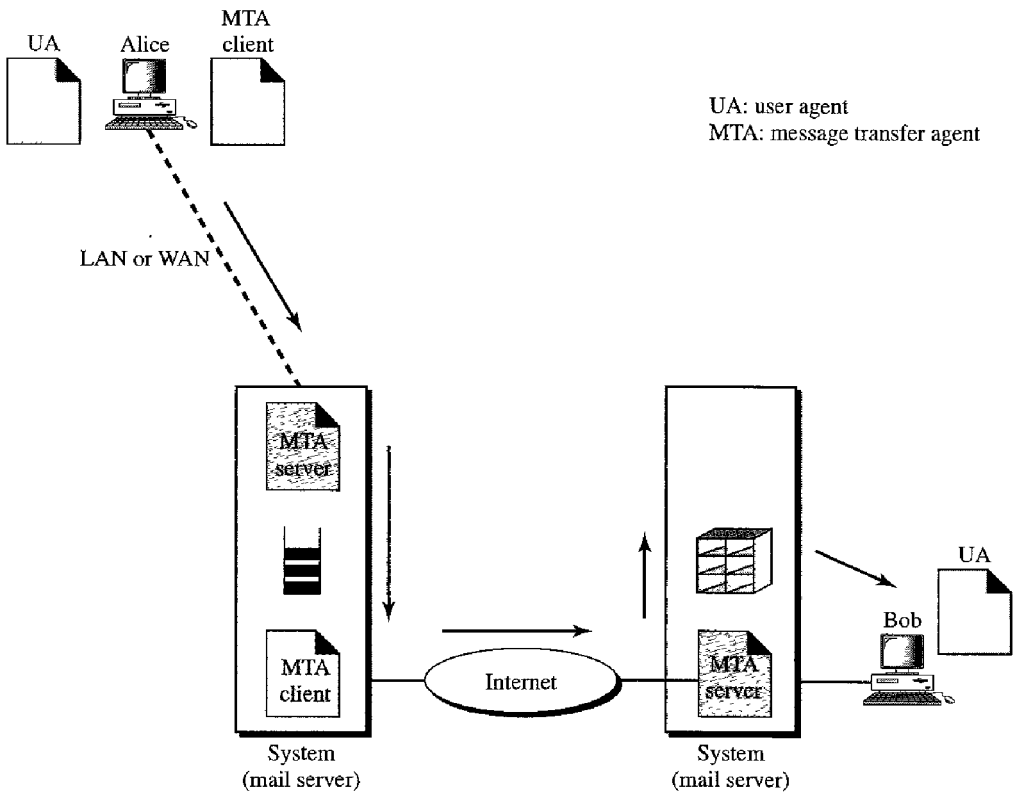
connection. The client, on the other hand, can be alerted by the system when there is a message in the queue to be sent.

**When the sender and the receiver of an e-mail are on different systems, we need two UAs and a pair of MTAs (client and server).**

### *Third Scenario*

In the third scenario, Bob, as in the second scenario, is directly connected to his system. Alice, however, is separated from her system. Either Alice is connected to the system via a point-to-point WAN, such as a dial-up modem, a DSL, or a cable modem; or she is connected to a LAN in an organization that uses one mail server for handling e-mails—all users need to send their messages to this mail server. Figure 26.8 shows the situation.

**Figure 26.8** *Third scenario in electronic mail*




Alice still needs a user agent to prepare her message. She then needs to send the message through the LAN or WAN. This can be done through a pair of message transfer agents (client and server). Whenever Alice has a message to send, she calls the user agent which, in turn, calls the MTA client. The MTA client establishes a connection with the MTA server on the system, which is running all the time. The system at Alice's site queues all messages received. It then uses an MTA client to send the messages to the system at Bob's site; the system receives the message and stores it in Bob's mailbox.

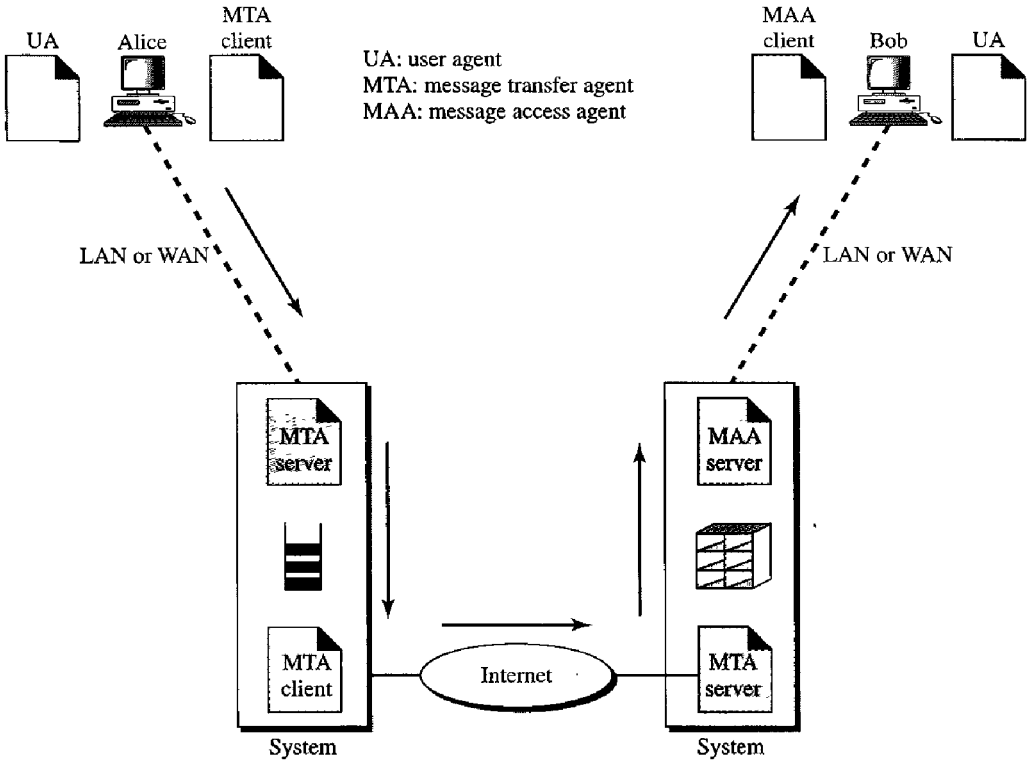
At his convenience, Bob uses his user agent to retrieve the message and reads it. Note that we need two pairs of MTA client/server programs.

**When the sender is connected to the mail server via a LAN or a WAN, we need two UAs and two pairs of MTAs (client and server).**

### *Fourth Scenario*

In the fourth and most common scenario, Bob is also connected to his mail server by a WAN or a LAN. After the message has arrived at Bob's mail server, Bob needs to retrieve it. Here, we need another set of client/server agents, which we call **message access agents (MAAs).** Bob uses an MAA client to retrieve his messages. The client sends a request to the MAA server, which is running all the time, and requests the transfer of the messages. The situation is shown in Figure 26.9.

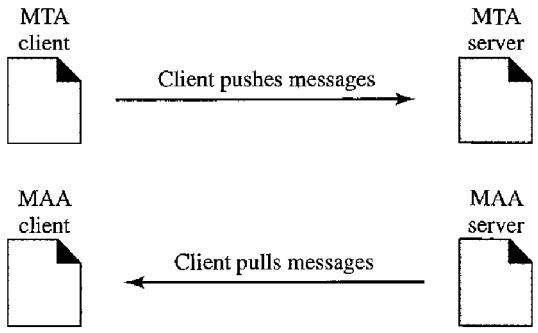
**Figure 26.9** *Fourth scenario in electronic mail*




There are two important points here. First, Bob cannot bypass the mail server and use the MTA server directly. To use 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 so 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. Figure 26.10 shows the difference.

**Figure 26.10** *Push versus pull in electronic email*




**When both sender and receiver are connected to the mail server via a LAN or a WAN, we need two UAs, two pairs of MTAs (client and server), and a pair of MAAs (client and server). This is the most common situation today.**

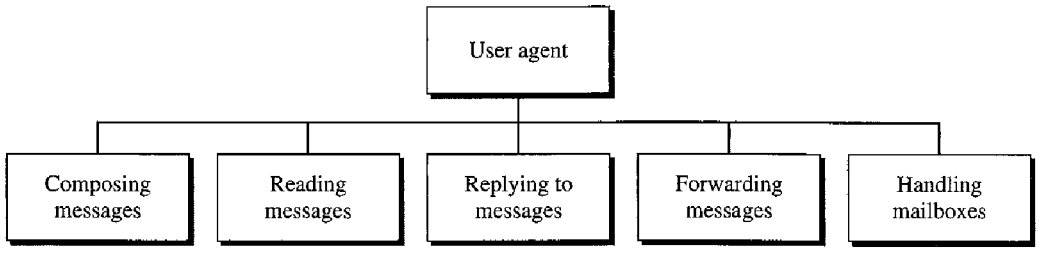
## User Agent

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.

### *Services Provided by a User Agent*

A user agent is a software package (program) that composes, reads, replies to, and forwards messages. It also handles mailboxes. Figure 26.11 shows the services of a typical user agent.

**Figure 26.11** *Services of user agent*




**Composing Messages** A user agent helps the user compose the e-mail message to be sent out. Most user agents provide a template on the screen to be filled in by the user. Some even have a built-in editor that can do spell checking, grammar checking, and

other tasks expected from a sophisticated word processor. A user, of course, could alternatively use his or her favorite text editor or word processor to create the message and import it, or cut and paste it, into the user agent template.

**Reading Messages** The second duty of the user agent is to read the incoming messages. When a user invokes a user agent, it first checks the mail in the incoming mailbox. Most user agents show a one-line summary of each received mail. Each e-mail contains the following fields.

1. A number field.
2. A flag field that shows the status of the mail such as new, already read but not replied to, or read and replied to.
3. The size of the message.
4. The sender.
5. The optional subject field.

**Replying to Messages** After reading a message, a user can use the user agent to reply to a message. A user agent usually allows the user to reply to the original sender or to reply to all recipients of the message. The reply message may contain the original message (for quick reference) and the new message.

**Forwarding Messages** *Replying* is defined as sending a message to the sender or recipients of the copy. *Forwarding* is defined as sending the message to a third party. A user agent allows the receiver to forward the message, with or without extra comments, to a third party.

### *Handling Mailboxes*

A user agent normally creates two mailboxes: an inbox and an outbox. Each box is a file with a special format that can be handled by the user agent. The inbox keeps all the received e-mails until they are deleted by the user. The outbox keeps all the sent e-mails until the user deletes them. Most user agents today are capable of creating customized mailboxes.

### *User Agent Types*

There are two types of user agents: command-driven and GUI-based.

**Command-Driven** Command-driven user agents belong to the early days of electronic mail. They are still present as the underlying user agents in servers. 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.*

---

**Some examples of command-driven user agents are *mail, pine,* and *elm.***

---

**GUI-Based** 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

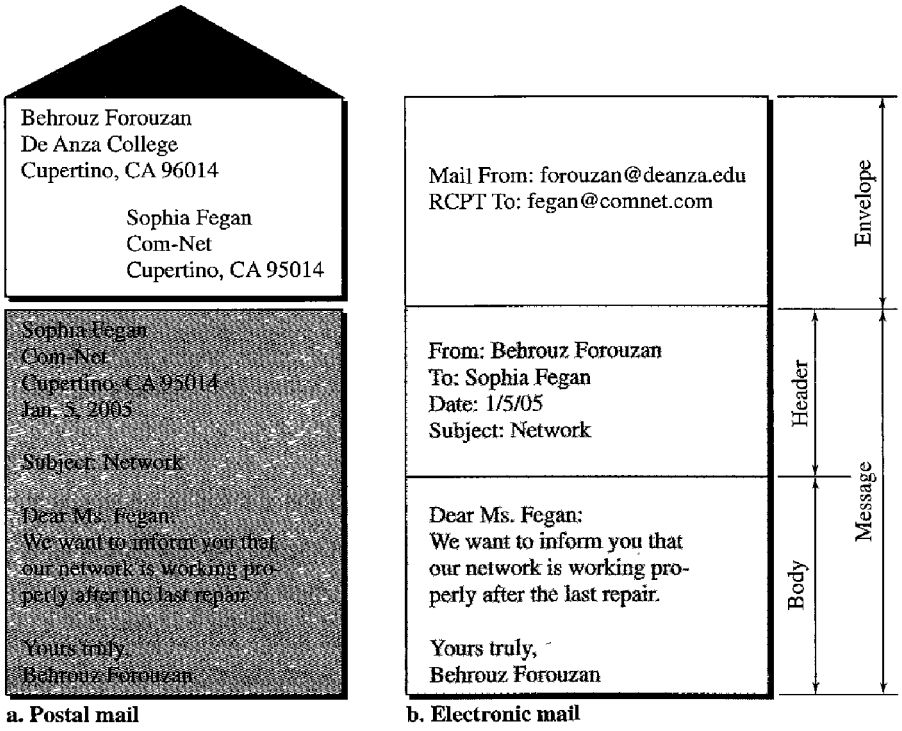
bars, and windows that make the services easy to access. Some examples of GUI-based user agents are Eudora, Microsoft's Outlook, and Netscape.

**Some examples of GUI-based user agents are *Eudora, Outlook,* and *Netscape.***

### *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 26.12).

**Figure 26.12** *Format of an e-mail*




**Envelope** The **envelope** usually contains the sender and the receiver addresses.

**Message** The message 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 (such as encoding type, as we see shortly). The body of the message contains the actual information to be read by the recipient.

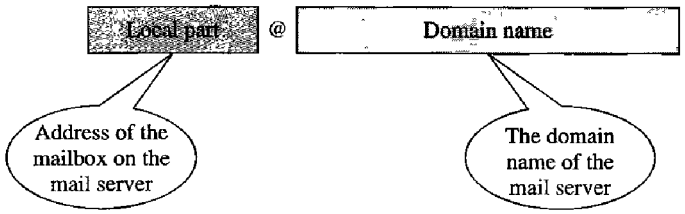
### *Receiving Mail*

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 26.13).

**Figure 26.13** *E-mail address*




**Local Part** 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.

**Domain Name** The second part of the address is the domain name. An organization usually selects one or more hosts to receive and send e-mail; the hosts 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*

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. If there is no mailing list for the alias, the name itself is the receiving address and a single message is delivered to the mail transfer entity.

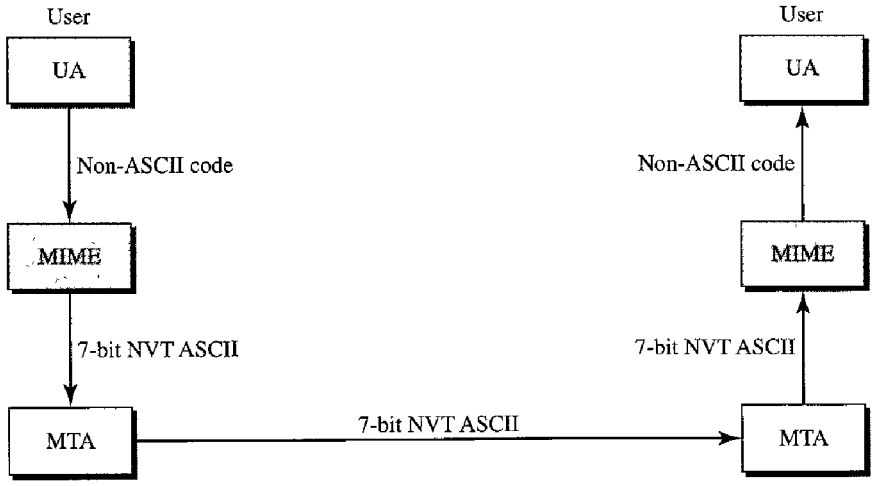
### *MIME*

Electronic mail has a simple structure. Its simplicity, however, comes at a price. It can send messages only in NVT 7-bit ASCII format. In other words, it has some limitations. For example, it cannot be used for languages that are not supported by 7-bit ASCII characters (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 them to the client MTA to be sent through the Internet. The message at the receiving side is transformed back to the original data.

We can think of MIME as a set of software functions that transforms non-ASCII data (stream of bits) to ASCII data and vice versa, as shown in Figure 26.14.

**Figure 26.14** *MIME*

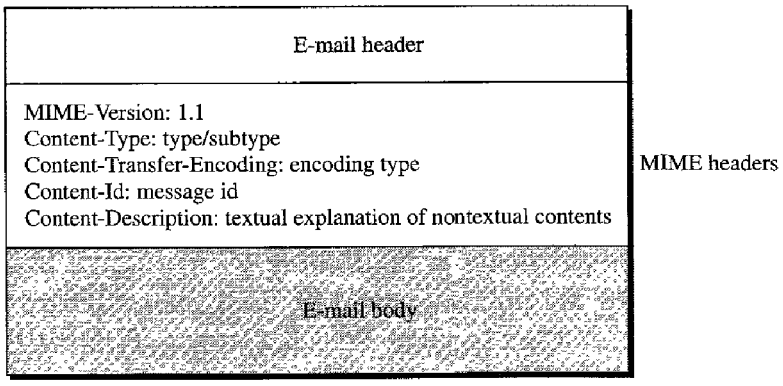



MIME defines five headers that can be added to the original e-mail header section to define the transformation parameters:

1. MIME-Version
2. Content-Type
3. Content-Transfer-Encoding
4. Content-Id
5. Content-Description

Figure 26.15 shows the MIME headers. We will describe each header in detail.

**Figure 26.15** *MIME header*




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

**MIME-Version:** 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.

**Content-Type:** <type / subtype; parameters>

MIME allows seven different types of data. These are listed in Table 26.5.

**Table 26.5** *Data types and subtypes in MIME*

| *Type* | *Subtype* | *Description* |
|---|---|---|
| Text | Plain | Unformatted |
| | HTML | HTML format (see Chapter 27) |
| Multipart | Mixed | Body contains ordered parts of different data types |
| | Parallel | Same as above, but no order |
| | Digest | Similar to mixed subtypes, 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 (8-bit bytes) |

**Content-Transfer-Encoding** This header defines the method used to encode the messages into 0s and 1s for transport:

**Content-Transfer-Encoding:** <type>

The five types of encoding methods are listed in Table 26.6.

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

**Content-Id:** id=<content-id>

**Table 26.6** *Content-transfer-encoding*

| *Type* | *Description* |
|---|---|
| 7-bit | NVT ASCII characters and short lines |
| 8-bit | Non-ASCII characters and short lines |
| Binary | Non-ASCII characters with unlimited-length lines |
| Base-64 | 6-bit blocks of data encoded into 8-bit ASCII characters |
| Quoted-printable | Non-ASCII characters encoded as an equals sign followed by an ASCII code |

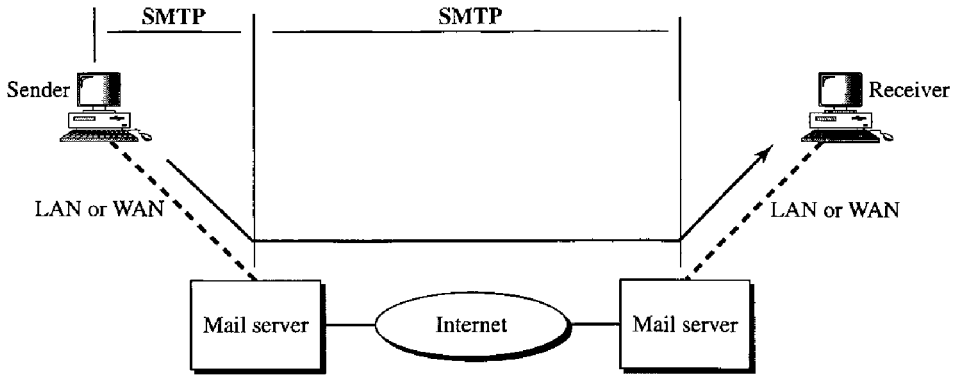
**Content-Description** This header defines whether the body is image, audio, or video.

**Content-Description:** <description>

## Message Transfer Agent: SMTP

The actual mail transfer is done through message transfer agents. To send mail, a system must have the client MTA, and to receive mail, a system must have a server MTA. The formal protocol that defines the MTA client and server in the Internet is called the **Simple Mail Transfer Protocol (SMTP).** As we said before, two pairs of MTA client/server programs are used in the most common situation (fourth scenario). Figure 26.16 shows the range of the SMTP protocol in this scenario.

**Figure 26.16** *SMTP range*



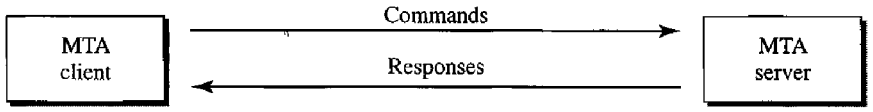

SMTP is used two times, between the sender and the sender's mail server and between the two mail servers. As we will see shortly, another protocol is needed between the mail server and the receiver.

SMTP simply defines how commands and responses must be sent back and forth. Each network is free to choose a software package for implementation. We discuss the mechanism of mail transfer by SMTP in the remainder of the section.

### *Commands and Responses*

SMTP uses commands and responses to transfer messages between an MTA client and an MTA server (see Figure 26.17).

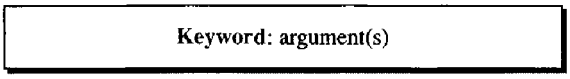
**Figure 26.17** *Commands and responses*




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 in Figure 26.18. It consists of a keyword followed by zero or more arguments. SMTP defines 14 commands. The first five are mandatory; every implementation must support these five commands. The next three are often used and highly recommended. The last six are seldom used.

**Figure 26.18** *Command format*




The commands are listed in Table 26.7.

**Table 26.7** *Commands*

| *Keyword* | *Argument(s)* |
|---|---|
| HELO | Sender's host name |
| MAIL FROM | Sender of the message |
| RCPT TO | Intended recipient of the message |
| DATA | Body of the mail |
| QUIT | |
| RSET | |
| VRFY | Name of recipient to be verified |
| NOOP | |
| TURN | |
| EXPN | Mailing list to be expanded |
| HELP | Command name |

**Table 26.7** *Commands (continued)*

| *Keyword* | *Argument(s)* |
|---|---|
| SEND FROM | Intended recipient of the message |
| SMOL FROM | Intended recipient of the message |
| SMAL FROM | Intended recipient of the message |

**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 26.8 lists some of the responses.

**Table 26.8** *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 |
| **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 |

As the table shows, responses are divided into four categories. The leftmost digit of the code (2, 3, 4, and 5) defines the category.

### *Mail Transfer Phases*

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

### *Example 26.3*

Let us see how we can directly use SMTP to send an e-mail and simulate the commands and responses we described in this section. We use TELNET to log into port 25 (the well-known port for SMTP). We then use the commands directly to send an e-mail. In this example, forouzanb@adelphia.net is sending an e-mail to himself. The first few lines show TELNET trying to connect to the Adelphia mail server.

After connection, we can type the SMTP commands and then receive the responses, as shown below. We have shown the commands in black and the responses in color. Note that we have added, for clarification, some comment lines, designated by the "=" signs. These lines are not part of the e-mail procedure.

```
$ telnet mail.adelphia.net 25
Trying 68.168.78.100 . . .
Connected to mail.adelphia.net (68.168.78.100).

================== Connection Establishment ==================
  220 mta13.adelphia.net SMTP server ready Fri, 6 Aug 2004 . . .
HELO mail.adelphia.net
  250 mta13.adelphia.net
==================      Mail Transfer      ==================
MAIL FROM: forouzanb@adelphia.net
  250 Sender <forouzanb@adelphia.net> Ok
RCPT TO: forouzanb@adelphia.net
  250 Recipient <forouzanb@adelphia.net> Ok
DATA
  354 Ok Send data ending with <CRLF>.<CRLF>
From: Forouzan
TO: Forouzan

This is a test message
to show SMTP in action.
.
==================  Connection Termination  ==================
  250 Message received: adelphia.net@mail.adelphia.net
QUIT
  221 mta13.adelphia.net SMTP server closing connection
Connection closed by foreign host.
```

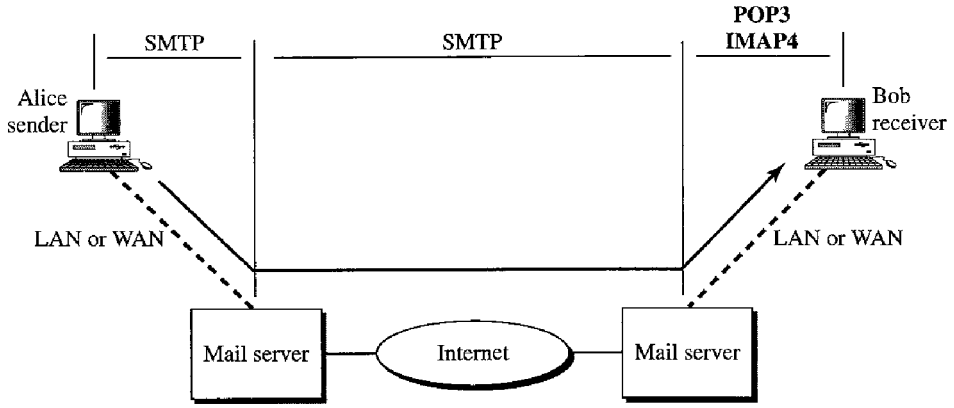
## Message Access Agent: POP and IMAP

The first and the 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 26.19 shows the position of these two protocols in the most common situation (fourth scenario).

**Figure 26.19** *POP3 and IMAP4*




### *POP3*

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

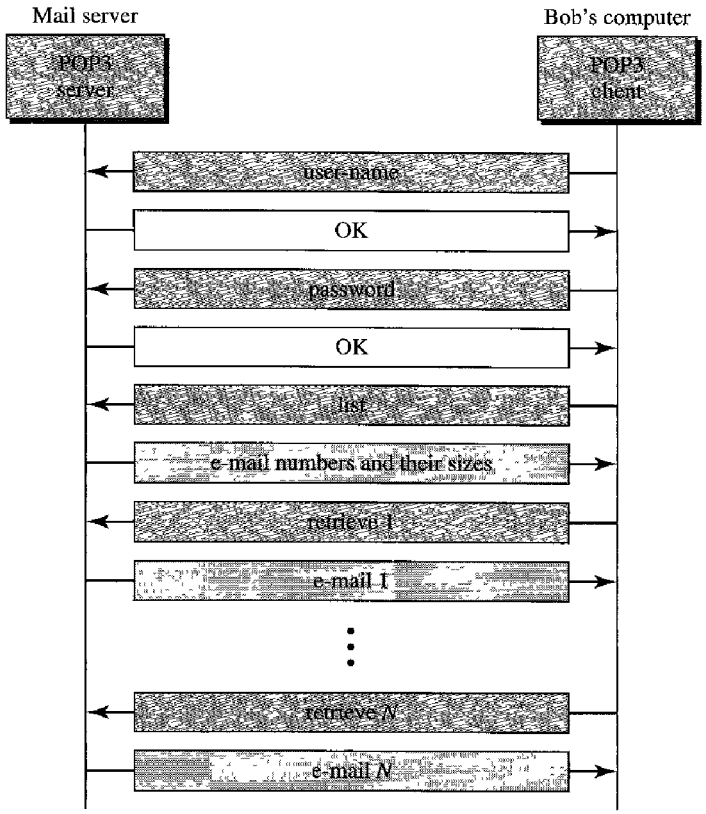
Mail access starts with the client when the user needs to download e-mail from the mailbox on the mail server. The client opens a connection to the server on TCP port 110. It then sends its user name and password to access the mailbox. The user can then list and retrieve the mail messages, one by one. Figure 26.20 shows an example of downloading using 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 (e.g., 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.

**Figure 26.20** *The exchange of commands and responses in POP3*




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. (Of course, the user can create folders on her own computer.) 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.

## Web-Based Mail

E-mail is such a common application that some websites today provide this service to anyone who accesses the site. Two common sites are Hotmail and Yahoo. The idea is very simple. Mail transfer from Alice's browser to her mail server is done through HTTP (see Chapter 27). The transfer of the message from the sending mail server to the receiving

mail server is still through SMTP. Finally, the message from the receiving server (the Web server) to Bob's browser is done through HTTP.

The last phase is very interesting. Instead of POP3 or IMAP4, HTTP is normally used. When Bob needs to retrieve his e-mails, he sends a 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 e-mail is transferred from the Web server to Bob's browser in HTML format.

## 26.3 FILE TRANSFER

Transferring files from one computer to another is one of the most common tasks expected from a networking or internetworking environment. As a matter of fact, the greatest volume of data exchange in the Internet today is due to file transfer. In this section, we discuss one popular protocol involved in transferring files: File Transfer Protocol (FTP).

### File Transfer Protocol (FTP)

**File Transfer Protocol (FTP)** is the standard mechanism 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 text and data. Two systems may have different directory structures. All these problems have been solved by FTP in a very simple and elegant approach.

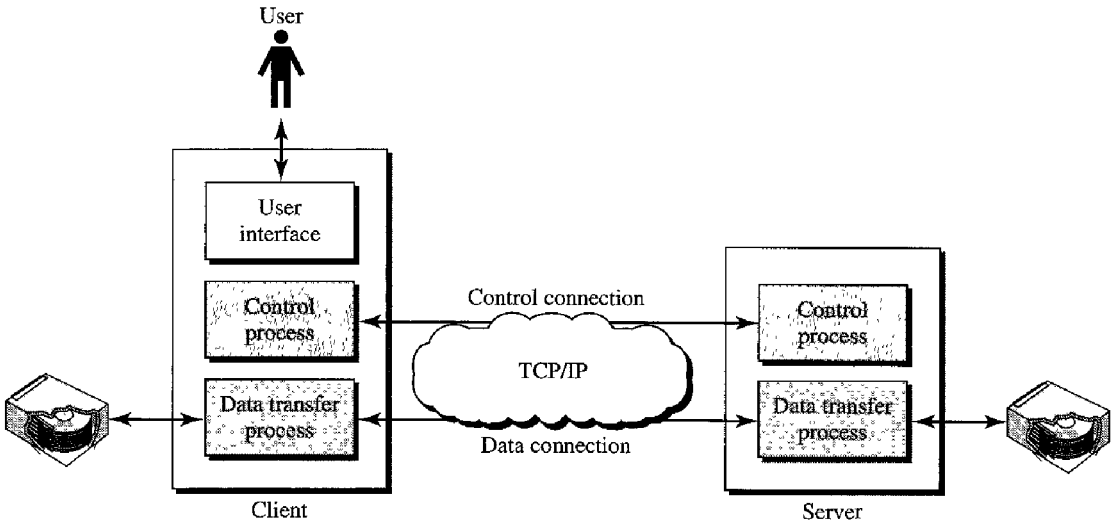
FTP differs from other client/server applications in that it establishes two connections between the hosts. One connection is used for data transfer, the other for control information (commands and responses). Separation of commands and data transfer makes FTP more efficient. The control connection uses 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. However, the difference in complexity is at the FTP level, not TCP. For TCP, both connections are treated the same.

FTP uses two well-known TCP ports: Port 21 is used for the control connection, and port 20 is used for the data connection.

**FTP uses the services of TCP. It needs two TCP connections. The well-known port 21 is used for the control connection and the well-known port 20 for the data connection.**

Figure 26.21 shows the basic model of FTP. The client has three components: user interface, 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.

**Figure 26.21** *FTP*

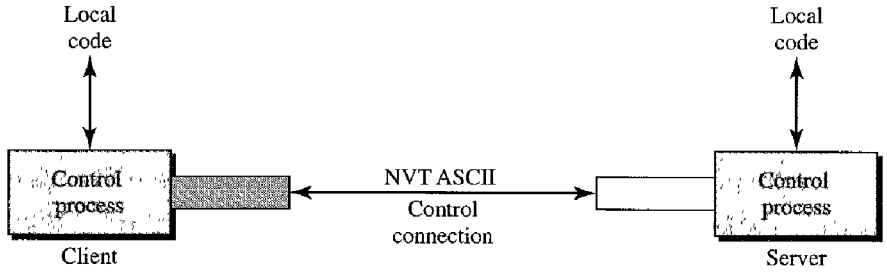



The **control connection** remains connected during the entire interactive FTP session. The **data connection** is opened and then closed for each file transferred. It opens each time commands that involve transferring files are used, and it closes when the file is transferred. In other words, when a 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.

### *Communication over Control Connection*

FTP uses the same approach as SMTP to communicate across the control connection. It uses the 7-bit ASCII character set (see Figure 26.22). 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 command or response is only one short line, so we need not worry about file format or file structure. Each line is terminated with a two-character (carriage return and line feed) end-of-line token.

**Figure 26.22** *Using the control connection*



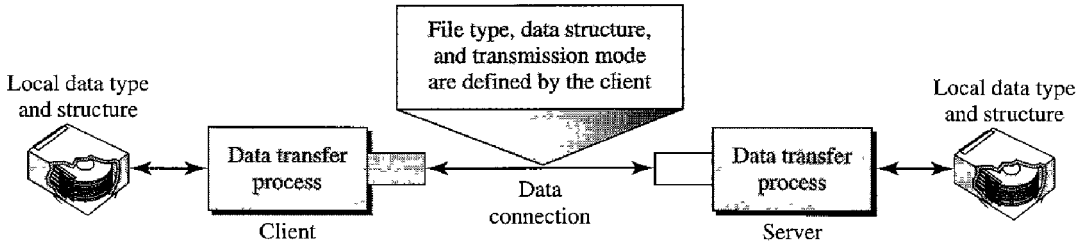
### *Communication over Data Connection*

The purpose of the data connection is different from that of the control connection. We want to transfer files through the data connection. 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:

- A file is to be copied from the server to the client. This is called *retrieving a file*. It is done under the supervision of the RETR command.
- A file is to be copied from the client to the server. This is called *storing a file*. It is done under the supervision of the STOR command.
- A list of directory or file names is to be sent from the server to the client. This is done under the supervision of the LIST command. Note that FTP treats a list of directory or file names as a file. It is sent over 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 (see Figure 26.23).

**Figure 26.23** *Using the data connection*




**File Type** FTP can transfer one of the following file types across the data connection: an ASCII file, EBCDIC file, or image file. The **ASCII file** is the default format for transferring text files. Each character is encoded using 7-bit ASCII. The sender transforms the file from its own representation into ASCII characters, and the receiver transforms the ASCII characters to its own representation. If one or both ends of the connection use EBCDIC encoding (the file format used by IBM), the file can be transferred using EBCDIC encoding. The **image file** is the default format for transferring binary files. The file is sent as continuous streams of bits without any interpretation or encoding. This is mostly used to transfer binary files such as compiled programs.

**Data Structure** FTP can transfer a file across the data connection by using one of the following interpretations about the structure of the data: file structure, record structure, and page structure. In the **file structure** format, the file 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 by using one of the following three transmission modes: stream mode, block mode, and compressed mode. The **stream mode** is the default mode. Data are delivered from FTP to TCP as a continuous stream of bytes. TCP is responsible for chopping data into segments of appropriate size. If the data are simply a stream of bytes (file structure), no end-of-file is needed. End-of-file in this case is the closing of the data connection by the sender. If the data are divided into records (record structure), each record will have a 1-byte end-of-record (EOR) character and the end of the file will have a 1-byte end-of-file (EOF) character. In **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 2 bytes define the size of the block in bytes. In the **compressed mode,** if the file is big, the data can be compressed. The compression method normally used is run-length encoding. In this method, consecutive appearances of a data unit are replaced by one occurrence and the number of repetitions. In a text file, this is usually spaces (blanks). In a binary file, null characters are usually compressed.

### *Example 26.4*

The following shows an actual FTP session for retrieving a list of items in a directory. 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 a 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.
ftp> ls reports
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.
```

1. After the control connection is created, the FTP server sends the 220 (service ready) response on the control connection.
2. The client sends its name.
3. The server responds with 331 (user name is OK, password is required).

4. The client sends the password (not shown).
5. The server responds with 230 (user log-in is OK).
6. The client sends the list command (ls reports) to find the list of files on the directory named report.
7. Now the server responds with 150 and opens the data connection.
8. The server then sends the list of the files or directories (as a file) on the data connection. When the whole list (file) is sent, the server responds with 226 (closing data connection) over the control connection.
9. The client now has two choices. It can use the QUIT command to request the closing of the control connection, or it can send another command to start another activity (and eventually open another data connection). In our example, the client sends a QUIT command.
10. After receiving the QUIT command, the server responds with 221 (service closing) and then closes the control connection.

## Anonymous FTP

To use FTP, a user needs an account (user name) and a password on the remote server. Some sites have a set of files available for public access, to enable **anonymous FTP.** To access these files, a user does not need to have an account or password. Instead, the user can use *anonymous* as the user name and *guest* as the password.

User access to the system is very limited. Some sites allow anonymous users only a subset of commands. For example, most sites allow the user to copy some files, but do not allow navigation through the directories.

### *Example 26.5*

We show an example of anonymous FTP. We assume that some public data are available at internic.net.

```
$ ftp internic.net
Connected to internic.net
220 Server ready
Name: anonymous
331 Guest login OK, send "guest" as password
Password: guest
ftp > pwd
257 '/' is current directory
ftp > ls
200 OK
150 Opening ASCII mode
bin
...
...
...
ftp > close
221 Goodbye
ftp > quit
```

## 26.4 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [. . .] refer to the reference list at the end of the text.

### Books

Remote logging is discussed in Chapter 18 of [For06] and Chapter 26 of [Ste94]. Electronic mail is discussed in Chapter 20 of [For06], Section 9.2 of [PD03], Chapter 32 of [Com04], Section 7.2 of [Tan03], and Chapter 28 of [Ste94]. FTP is discussed in Chapter 19 of [For06], Chapter 27 of [Ste94], and Chapter 34 of [Com04].

### Sites

The following sites are related to topics discussed in this chapter.

- www.ietf.org/rfc.html Information about RFCs

### RFCs

The following RFCs are related to TELNET:

137, 340, 393, 426, 435, 452, 466, 495, 513, 529, 562, 595, 596, 599, 669, 679, 701, 702, 703, 728, 764, 782, 818, 854, 855, 1184, 1205, 2355

The following RFCs are related to SMTP, POP, and IMAP:

196, 221, 224, 278, 524, 539, 753, 772, 780, 806, 821, 934, 974, 1047, 1081, 1082, 1225, 1460, 1496, 1426, 1427, 1652, 1653, 1711, 1725, 1734, 1740, 1741, 1767, 1869, 1870, 2045, 2046, 2047, 2048, 2177, 2180, 2192, 2193, 2221, 2342, 2359, 2449, 2683, 2503

The following RFCs are related to FTP:

114, 133, 141, 163, 171, 172, 238, 242, 250, 256, 264, 269, 281, 291, 354, 385, 412, 414, 418, 430, 438, 448, 463, 468, 478, 486, 505, 506, 542, 553, 624, 630, 640, 691, 765, 913, 959, 1635, 1785, 2228, 2577

DNS is discussed in [AL98], chapter 17 of [For06], section 9.1 of [PD03], and section 7.1 of [Tan03].

## 26.5 KEY TERMS

alias
anonymous FTP
ASCII file
block mode
body
character mode
compressed mode
control character
control connection
data connection

default mode
domain name
envelope
file structure
File Transfer Protocol (FTP)
header
image file
Internet Mail Access Protocol, version 4 (IMAP4)
line mode
local log-in
local part
message access agent (MAA)
message transfer agent (MTA)
Multipurpose Internet Mail Extensions (MIME)
network virtual terminal (NVT)
option negotiation
page structure
Post Office Protocol, version 3 (POP3)
record structure
remote log-in
Simple Mail Transfer Protocol (SMTP)
stream mode
suboption negotiation
terminal network (TELNET)
timesharing
user agent (UA)

## 26.6 SUMMARY

- TELNET is a client/server application that allows a user to log on to a remote machine, giving the user access to the remote system.
- TELNET uses the network virtual terminal (NVT) system to encode characters on the local system. On the server machine, NVT decodes the characters to a form acceptable to the rernote machine.
- NVT uses a set of characters for data and a set of characters for control.
- In TELNET, control characters are embedded in the data stream and preceded by the *interpret as control* (IAC) control character.
- Options are features that enhance the TELNET process.
- TELNET allows negotiation to set transfer conditions between the client and server before and during the use of the service.
- A TELNET implementation operates in the default, character, or line mode.
  1. In the default mode, the client sends one line at a time to the server.
  2. In the character mode, the client sends one character at a time to the server.
  3. In the line mode, the client sends one line at a time to the server.
- Several programs, including SMTP, POP3, and IMAP4, are used in the Internet to provide electronic mail services.
- In electronic mail, the UA prepares the message, creates the envelope, and puts the message in the envelope.
- In electronic mail, the mail address consists of two parts: a local part (user mailbox) and a domain name. The form is localpart@domainname.
- In electronic mail, Multipurpose Internet Mail Extension (MIME) allows the transfer of multimedia messages.
- In electronic mail, the MTA transfers the mail across the Internet, a LAN, or a WAN.

- SMTP uses commands and responses to transfer messages between an MTA client and an MTA server.
- The steps in transferring a mail message are
  1. Connection establishment
  2. Mail transfer
  3. Connection termination
- Post Office Protocol, version 3 (POP3) and Internet Mail Access Protocol, version 4 (IMAP4) are protocols used for pulling messages from a mail server.
- One of the programs used for file transfer in the Internet is File Transfer Protocol (FTP).
- FTP requires two connections for data transfer: a control connection and a data connection.
- FTP employs NVT ASCII for communication between dissimilar systems.
- Prior to the actual transfer of files, the file type, data structure, and transmission mode are defined by the client through the control connection.
- Responses are sent from the server to the client during connection establishment.
- There are three types of file transfer:
  1. A file is copied from the server to the client.
  2. A file is copied from the client to the server.
  3. A list of directories or file names is sent from the server to the client.
- Anonymous FTP provides a method for the general public to access files on remote sites.

## 26.7 PRACTICE SET

### Review Questions

1. What is the difference between local and remote log-in in TELNET?
2. How are control and data characters distinguished in NVT?
3. How are options negotiated in TELNET?
4. Describe the addressing system used by SMTP.
5. In electronic mail, what are the tasks of a user agent?
6. In electronic mail, what is MIME?
7. Why do we need POP3 or IMAP4 for electronic mail?
8. What is the purpose of FTP?
9. Describe the functions of the two FTP connections.
10. What kinds of file types can FTP transfer?
11. What are the three FTP transmission modes?
12. How does storing a file differ from retrieving a file?
13. What is anonymous FTP?

## Exercises

14. Show the sequence of bits sent from a client TELNET for the binary transmission of 11110011 00111100 11111111.
15. If TELNET is using the character mode, how many characters are sent back and forth between the client and server to copy a file named file1 to another file named file2 using the command *cp file1 file2*?
16. What is the minimum number of bits sent at the TCP level to accomplish the task in Exercise 15?
17. What is the minimum number of bits sent at the data link layer level (using Ethernet) to accomplish the task in Exercise 15?
18. What is the ratio of the useful bits to the total bits in Exercise 17?
19. Interpret the following sequences of characters (in hexadecimal) received by a TELNET client or server.
    a. FF FB 01
    b. FF FE 01
    c. FF F4
    d. FF F9
20. A sender sends unformatted text. Show the MIME header.
21. A sender sends a JPEG message. Show the MIME header.
22. Why is a connection establishment for mail transfer needed if TCP has already established a connection?
23. Why should there be limitations on anonymous FTP? What could an unscrupulous user do?
24. Explain why FTP does not have a message format.

## Research Activities

25. Show the sequence of characters exchanged between the TELNET client and the server to switch from the default mode to the character mode.
26. Show the sequence of characters exchanged between the TELNET client and the server to switch from the default mode to line mode.
27. In SMTP, show the connection establishment phase from aaa@xxx.com to bbb@yyy.com.
28. In SMTP, show the message transfer phase from aaa@xxx.com to bbb@yyy.com. The message is "Good morning my friend."
29. In SMTP, show the connection termination phase from aaa@xxx.com to bbb@yyy.com.
30. What do you think would happen if the control connection were accidentally severed during an FTP transfer?

31. Find the extended options proposed for TELNET.
32. Another log-in protocol is called Rlogin. Find some information about Rlogin and compare it with TELNET.
33. A more secure log-in protocol in UNIX is called Secure Shell (SSH). Find some information about this protocol.

# CHAPTER 27

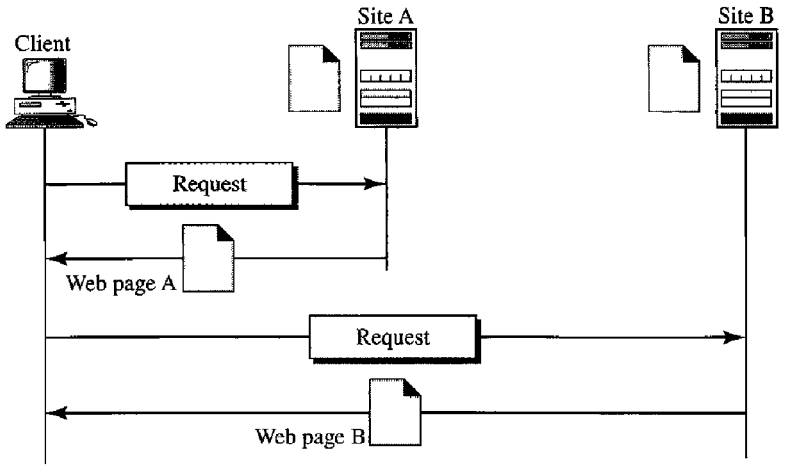
# *WWW and HTTP*

The **World Wide Web (WWW)** is a repository of information linked together from points all over the world. The WWW has a unique combination of flexibility, portability, and user-friendly features that distinguish it from other services provided by the Internet. The WWW project was initiated by CERN (European Laboratory for Particle Physics) to create a system to handle distributed resources necessary for scientific research. In this chapter we first discuss issues related to the Web. We then discuss a protocol, HTTP, that is used to retrieve information from the Web.

## 27.1 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*, as shown in Figure 27.1.

**Figure 27.1** *Architecture of WWW*

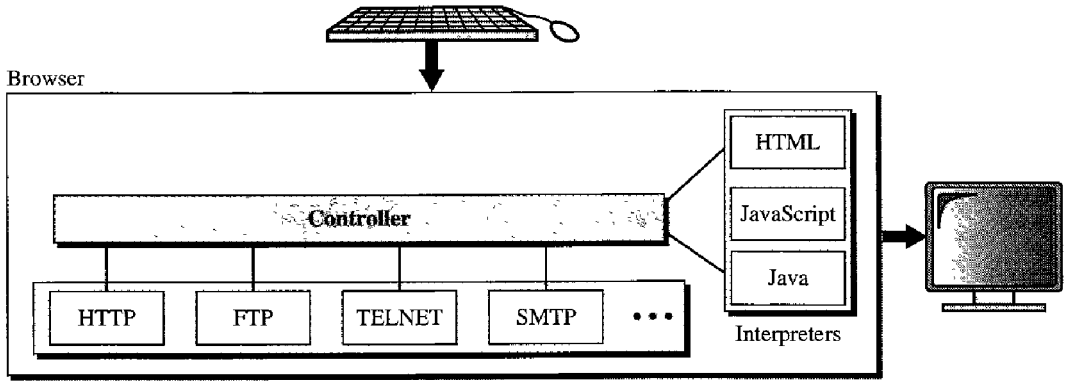


Each site holds one or more documents, referred to as *Web pages*. Each Web page can contain a link to other pages in the same site or at other sites. The pages can be retrieved and viewed by using browsers. Let us go through the scenario shown in Figure 27.1. The client needs to see some information that it knows belongs to site A. It sends a request through its browser, a program that is designed to fetch **Web** documents. The request, among other information, includes the address of the site and the Web page, called the URL, which we will discuss shortly. The server at site A finds the document and sends it to the client. When the user views the document, she finds some references to other documents, including a Web page at site B. The reference has the URL for the new site. The user is also interested in seeing this document. The client sends another request to the new site, and the new page is retrieved.

## Client (Browser)

A variety of vendors offer commercial browsers that interpret and display a Web document, and all use nearly the same architecture. Each **browser** usually consists of three parts: a controller, client protocol, and interpreters. 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 previously such as FTP or HTTP (described later in the chapter). The interpreter can be HTML, Java, or JavaScript, depending on the type of document. We discuss the use of these interpreters based on the document type later in the chapter (see Figure 27.2).

**Figure 27.2** *Browser*




## Server

The Web page is stored at the server. Each time a client 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 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.

## Uniform Resource Locator

A client that wants to access a Web page needs the address. To facilitate the access of documents distributed throughout the world, HTTP uses locators. The **uniform resource locator (URL)** is a standard for specifying any kind of information on the Internet. The URL defines four things: protocol, host computer, port, and path (see Figure 27.3).

**Figure 27.3** *URL*


The *protocol* is the client/server program used to retrieve the document. Many different protocols can retrieve a document; among them are FTP or HTTP. The most common today is HTTP.

The **host** is the computer on which the information is located, although the name of the computer can be an alias. Web pages are usually stored in computers, and computers are given alias names that usually begin with the characters "www". This is not mandatory, however, as the host can be any name given to the computer that hosts the Web page.

The URL can optionally contain the port number of the server. If the *port* is included, it is inserted between the host and the path, and it is separated from the host by a colon.

**Path** is the pathname of the file where the information is located. Note that the path can itself contain slashes that, in the UNIX operating system, separate the directories from the subdirectories and files.

## 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 design of WWW, retrieving publicly available documents, exactly fits this purpose. Today the Web has other functions; some are listed here.

1. Some websites need to allow access to registered clients only.
2. 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.
3. Some websites are used as portals: the user selects the Web pages he wants to see.
4. Some websites are just advertising.

For these purposes, the cookie mechanism was devised. We discussed the use of cookies at the transport layer in Chapter 23; we now discuss their use in Web pages.

### *Creation and Storage of Cookies*

The creation and storage 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 domain server 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:

1. 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.
2. An electronic store (e-commerce) can use a cookie for its client shoppers. When a client selects an item and inserts it into 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.
3. 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.
4. 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 banner address instead of the banner itself. When a user visits the main website and clicks on the icon of an advertised corporation, a request is sent to the advertising agency. The advertising agency sends the banner, a GIF file, for example, 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.

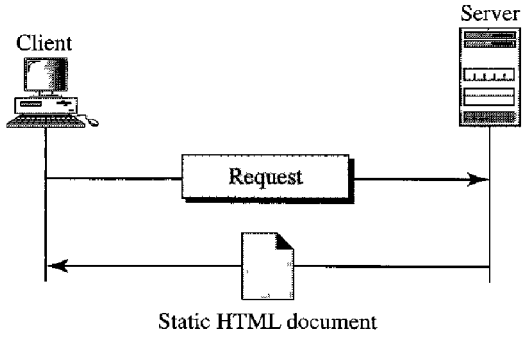
## 27.2 WEB DOCUMENTS

The documents in the WWW can be grouped into three broad categories: static, dynamic, and active. The category is based on the time at which the contents of the document are determined.

## Static Documents

**Static documents** are fixed-content documents that are created and stored in a server. The client can get only a copy of the document. 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 browsing program to display the document (see Figure 27.4).

**Figure 27.4** *Static document*

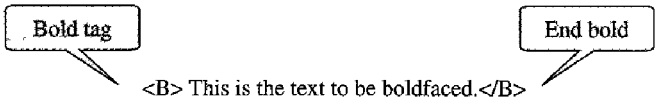



### *HTML*

**Hypertext Markup Language (HTML)** is a language for creating Web pages. The term *markup language* comes from the book publishing industry. Before a book is typeset and printed, a copy editor reads the manuscript and puts marks on it. These marks tell the compositor how to format the text. For example, if the copy editor wants part of a line to be printed in boldface, he or she draws a wavy line under that part. In the same way, data for a Web page are formatted for interpretation by a browser.

Let us clarify the idea with an example. To make part of a text displayed in boldface with HTML, we put beginning and ending boldface **tags** (marks) in the text, as shown in Figure 27.5.

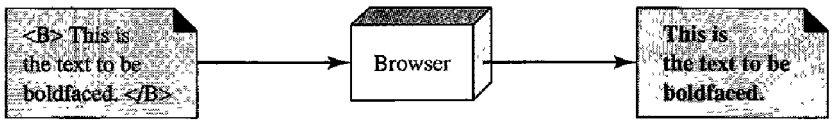
**Figure 27.5** *Boldface tags*




The two tags <B> and </B> are instructions for the browser. When the browser sees these two marks, it knows that the text must be boldfaced (see Figure 27.6).

A markup language such as HTML allows us to embed formatting instructions in the file itself. The instructions are included with the text. In this way, any browser can read the instructions and format the text according to the specific workstation. One might

**Figure 27.6** *Effect of boldface tags*




ask why we do not use the formatting capabilities of word processors to create and save formatted text. The answer is that different word processors use different techniques or procedures for formatting text. For example, imagine that a user creates formatted text on a Macintosh computer and stores it in a Web page. Another user who is on an IBM computer would not be able to receive the Web page because the two computers use different formatting procedures.

HTML lets us use only ASCII characters for both the main text and formatting instructions. In this way, every computer can receive the whole document as an ASCII document. The main text is the data, and the formatting instructions can be used by the browser to format the data.

A Web page is made up of two parts: the head and the body. The head is the first part of a Web page. The head contains the title of the page and other parameters that the browser will use. The actual contents of a page are in the body, which includes the text and the tags. Whereas the text is the actual information contained in a page, the tags define the appearance of the document. Every HTML tag is a name followed by an optional list of attributes, all enclosed between less-than and greater-than symbols (< and >).

An attribute, if present, is followed by an equals sign and the value of the attribute. Some tags can be used alone; others must be used in pairs. Those that are used in pairs are called *beginning* and *ending* tags. The beginning tag can have attributes and values and starts with the name of the tag. The ending tag cannot have attributes or values but must have a slash before the name of the tag. The browser makes a decision about the structure of the text based on the tags, which are embedded into the text. Figure 27.7 shows the format of a tag.

**Figure 27.7** *Beginning and ending tags*


**a. Beginning tag**


**b. Ending tag**

One commonly used tag category is the text formatting tags such as <B> and </B>, which make the text bold; <I> and </I>, which make the text italic; and <U> and </U>, which underline the text.

Another interesting tag category is the image tag. Nontextual information such as digitized photos or graphic images is not a physical part of an HTML document. But we can use an image tag to point to the file of a photo or image. The image tag defines the address (URL) of the image to be retrieved. It also specifies how the image can be inserted after retrieval. We can choose from several attributes. The most common are SRC (source), which defines the source (address), and ALIGN, which defines the alignment of the image. The SRC attribute is required. Most browsers accept images in the GIF or JPEG formats. For example, the following tag can retrieve an image stored as image1.gif in the directory /bin/images:

```
<IMG SRC= "/bin/images/image1.gif" ALIGN=MIDDLE >
```

A third interesting category is the hyperlink tag, which is needed to link documents together. Any item (word, phrase, paragraph, or image) can refer to another document through a mechanism called an *anchor*. The anchor is defined by <A . . . > and </A> tags, and the anchored item uses the URL to refer to another document. When the document is displayed, the anchored item is underlined, blinking, or boldfaced. The user can click on the anchored item to go to another document, which may or may not be stored on the same server as the original document. The reference phrase is embedded between the beginning and ending tags. The beginning tag can have several attributes, but the one required is HREF (hyperlink reference), which defines the address (URL) of the linked document. For example, the link to the author of a book can be

```
<A HREF= "http://www.deanza.edu/forouzan"> Author </A>
```

What appears in the text is the word *Author,* on which the user can click to go to the author's Web page.

## 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 output 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 can 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.

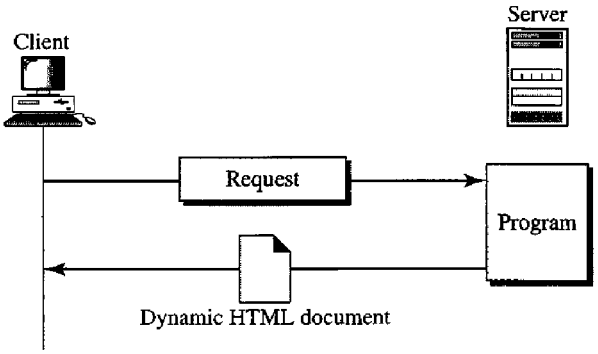
### *Common Gateway Interface (CGI)*

The **Common Gateway Interface (CGI)** is a technology that creates and handles dynamic documents. CGI is a set of standards that defines how a dynamic document is written, how data are input to the program, and how the output result is used.

CGI is not a new language; instead, it allows programmers to use any of several languages such as C, C++, Bourne Shell, Korn Shell, C Shell, Tcl, or Perl. The only thing that CGI defines is a set of rules and terms that the programmer must follow.

The term *common* in CGI indicates that the standard defines a set of rules that is common to any language or platform. The term *gateway* here means that a CGI program can be used to access other resources such as databases, graphical packages, and so on. The term *interface* here means that there is a set of predefined terms, variables, calls, and so on that can be used in any CGI program. A CGI program in its simplest form is code written in one of the languages supporting CGI. Any programmer who can encode a sequence of thoughts in a program and knows the syntax of one of the above-mentioned languages can write a simple CGI program. Figure 27.8 illustrates the steps in creating a dynamic program using CGI technology.

**Figure 27.8** *Dynamic document using CGI*




**Input** In traditional programming, when a program is executed, parameters can be passed to the program. Parameter passing allows the programmer to write a generic program that can be used in different situations. For example, a generic copy program can be written to copy any file to another. A user can use the program to copy a file named $x$ to another file named $y$ by passing $x$ and $y$ as parameters.

The input from a browser to a server is sent by using a *form*. If the information in a form is small (such as a word), it can be appended to the URL after a question mark. For example, the following URL is carrying form information (23, a value):

**http://www.deanza/cgi-bin/prog.pl?23**

When the server receives the URL, it uses the part of the URL before the question mark to access the program to be run, and it interprets the part after the question mark (23) as the input sent by the client. It stores this string in a variable. When the CGI program is executed, it can access this value.

If the input from a browser is too long to fit in the query string, the browser can ask the server to send a form. The browser can then fill the form with the input data and send it to the server. The information in the form can be used as the input to the CGI program.

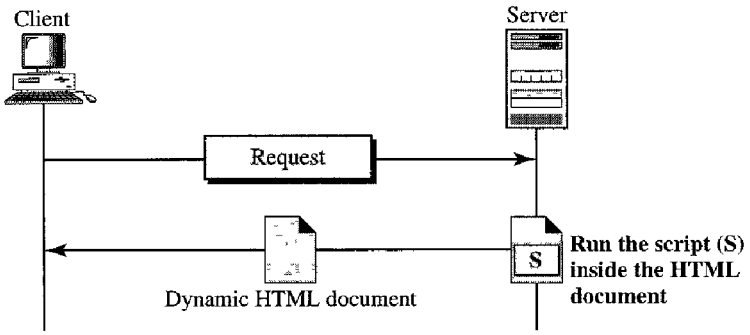
**Output** The whole idea of CGI is to execute a CGI program at the server site and send the output to the client (browser). The output is usually plain text or a text with HTML structures; however, the output can be a variety of other things. It can be graphics or binary data, a status code, instructions to the browser to cache the result, or instructions to the server to send an existing document instead of the actual output.

To let the client know about the type of document sent, a CGI program creates headers. As a matter of fact, the output of the CGI program always consists of two parts: a header and a body. The header is separated by a blank line from the body. This means any CGI program creates first the header, then a blank line, and then the body. Although the header and the blank line are not shown on the browser screen, the header is used by the browser to interpret the body.

### *Scripting Technologies for Dynamic Documents*

The problem with CGI technology is the inefficiency that results if part of the dynamic document that is to be created is fixed and not changing from request to request. For example, assume that we need to retrieve a list of spare parts, their availability, and prices for a specific car brand. Although the availability and prices vary from time to time, the name, description, and the picture of the parts are fixed. If we use CGI, the program must create an entire document each time a request is made. The solution is to create a file containing the fixed part of the document using HTML and embed a script, a source code, that can be run by the server to provide the varying availability and price section. Figure 27.9 shows the idea.

**Figure 27.9** *Dynamic document using server-site script*




A few technologies have been involved in creating dynamic documents using scripts. Among the most common are **Hypertext Preprocessor (PHP),** which uses the Perl language; **Java Server Pages (JSP),** which uses the Java language for scripting; **Active Server Pages (ASP),** a Microsoft product which uses Visual Basic language for scripting; and **ColdFusion,** which embeds SQL database queries in the HTML document.

**Dynamic documents are sometimes referred to as server-site dynamic documents.**

## Active Documents

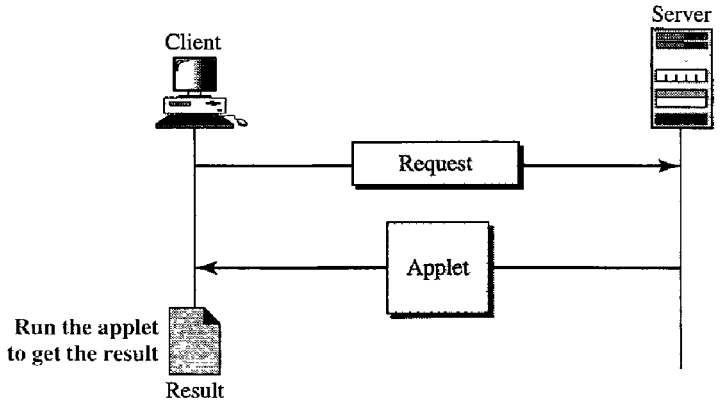
For many applications, we need a program or a script to be run at the client site. These are called **active documents.** For example, suppose we want to run a program that creates animated graphics on the screen or a program that interacts with the user. The program definitely needs to be run at the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

### *Java Applets*

One way to create an active document is to use Java **applets. Java** is a combination of a high-level programming language, a run-time environment, and a class library that allows a programmer to write an active document (an applet) and a browser to run it. It can also be a stand-alone program that doesn't use a browser.

An applet is a program written in Java on the server. It is compiled and ready to be run. The document is in byte-code (binary) format. The client process (browser) creates an instance of this applet and runs it. A Java applet can be run by the browser in two ways. In the first method, the browser can directly request the Java applet program in the URL and receive the applet in binary form. In the second method, the browser can retrieve and run an HTML file that has embedded the address of the applet as a tag. Figure 27.10 shows how Java applets are used in the first method; the second is similar but needs two transactions.

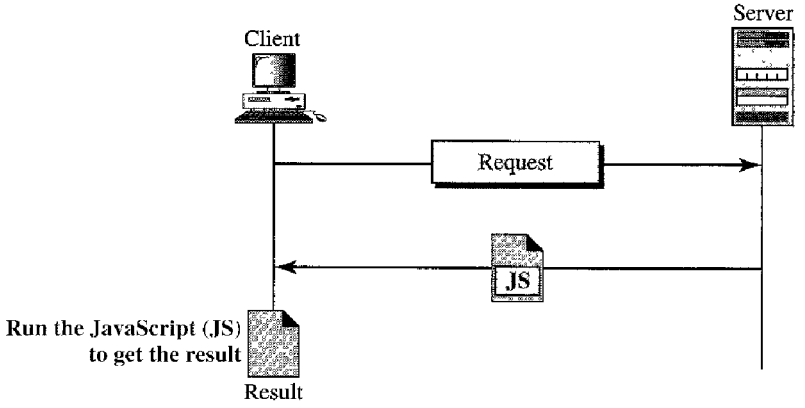
**Figure 27.10** *Active document using Java applet*




### *JavaScript*

The idea of scripts in dynamic documents can also be used for active documents. If the active part of the document is small, it can be written in a scripting language; then it can be interpreted and run by the client at the same time. The script is in source code (text) and not in binary form. The scripting technology used in this case is usually JavaScript. **JavaScript,** which bears a small resemblance to Java, is a very high level scripting language developed for this purpose. Figure 27.11 shows how JavaScript is used to create an active document.

**Figure 27.11** *Active document using client-site script*




**Active documents are sometimes referred to as client-site dynamic documents.**

## 27.3 HTTP

The **Hypertext Transfer Protocol (HTTP)** is a protocol used mainly to access data on the World Wide Web. HTTP functions as a combination of FTP and SMTP. It is similar to FTP because it transfers files and uses the services of TCP. However, it is much simpler than FTP because it uses only one TCP connection. There is no separate control connection; only data are transferred between the client and the server.

HTTP is like SMTP because the data transferred between the client and the server look like SMTP messages. In addition, the format of the messages is controlled by MIME-like headers. Unlike SMTP, the HTTP messages are not destined to be read by humans; they are read and interpreted by the HTTP server and HTTP client (browser). SMTP messages are stored and forwarded, but HTTP messages are delivered immediately. The commands from the client to the server are embedded in a request message. The contents of the requested file or other information are embedded in a response message. HTTP uses the services of TCP on well-known port 80.

**HTTP uses the services of TCP on well-known port 80.**

### HTTP Transaction

Figure 27.12 illustrates the HTTP transaction between the client and server. Although HTTP uses the services of TCP, HTTP itself is a stateless protocol. The client initializes the transaction by sending a request message. The server replies by sending a response.

#### *Messages*

The formats of the request and response messages are similar; both are shown in Figure 27.13. A request message consists of a request line, a header, and sometimes a body. A response message consists of a status line, a header, and sometimes a body.

**Figure 27.12** *HTTP transaction*

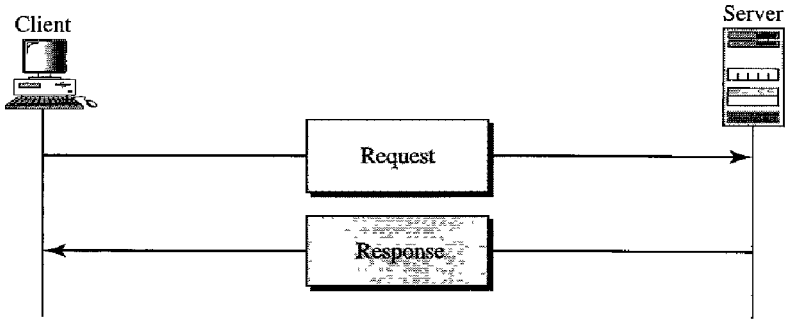
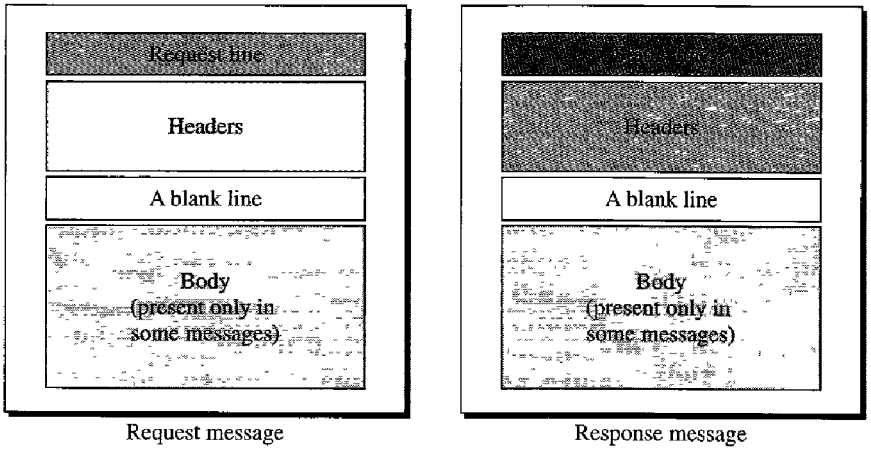
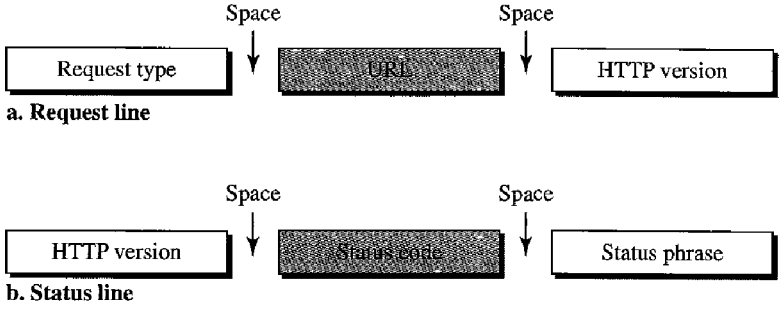



**Figure 27.13** *Request and response messages*




**Request and Status Lines** The first line in a request message is called a **request line;** the first line in the response message is called the **status line.** There is one common field, as shown in Figure 27.14.

**Figure 27.14** *Request and status lines*



- **Request type.** This field is used in the request message. In version 1.1 of HTTP, several request types are defined. The **request type** is categorized into *methods* as defined in Table 27.1.

**Table 27.1** *Methods*

| *Method* | *Action* |
|---|---|
| GET | Requests a document from the server |
| HEAD | Requests information about a document but not the document itself |
| POST | Sends some information from the client to the server |
| PUT | Sends a document from the server to the client |
| TRACE | Echoes the incoming request |
| CONNECT | Reserved |
| OPTION | Inquires about available options |

- **URL.** We discussed the URL earlier in the chapter.
- **Version.** The most current version of HTTP is 1.1.
- **Status code.** This field is used in the response message. The **status code** field is similar to those in the FTP and the SMTP protocols. 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. We list the most common codes in Table 27.2.
- **Status phrase.** This field is used in the response message. It explains the status code in text form. Table 27.2 also gives the status phrase.

**Table 27.2** *Status codes*

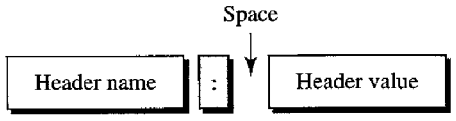
| *Code* | *Phrase* | *Description* |
|---|---|---|
| | **Informational** | |
| **100** | Continue | The initial part of the request has been received, and the client may continue with its request. |
| **101** | Switching | The server is complying with a client request to switch protocols defined in the upgrade header. |
| | **Success** | |
| **200** | OK | The request is successful. |
| **201** | Created | A new URL is created. |
| **202** | Accepted | The request is accepted, but it is not immediately acted upon. |
| **204** | No content | There is no content in the body. |

**Table 27.2** *Status codes (continued)*

| *Code* | *Phrase* | *Description* |
|---|---|---|
| | **Redirection** | |
| **301** | Moved permanently | The requested URL is no longer used by the server. |
| **302** | Moved temporarily | The requested URL has moved temporarily. |
| **304** | Not modified | The document has not been modified. |
| | **Client Error** | |
| **400** | Bad request | There is a syntax error in the request. |
| **401** | Unauthorized | The request lacks proper authorization. |
| **403** | Forbidden | Service is denied. |
| **404** | Not found | The document is not found. |
| **405** | Method not allowed | The method is not supported in this URL. |
| **406** | Not acceptable | The format requested is not acceptable. |
| | **Server Error** | |
| **500** | Internal server error | There is an error, such as a crash, at the server site. |
| **501** | Not implemented | The action requested cannot be performed. |
| **503** | Service unavailable | The service is temporarily unavailable, but may be requested in the future. |

**Header** The header exchanges additional information between the client and the server. For example, the client can request that the document be sent in a special format, or the server can send extra information about the document. The header can consist of one or more header lines. Each header line has a header name, a colon, a space, and a header value (see Figure 27.15). We will show some header lines in the examples at the end of this chapter. A header line belongs to one of four categories: **general header, request header, response header,** and **entity header.** A request message can contain only general, request, and entity headers. A response message, on the other hand, can contain only general, response, and entity headers.

**Figure 27.15** *Header format*




- **General header** The general header gives general information about the message and can be present in both a request and a response. Table 27.3 lists some general headers with their descriptions.

**Table 27.3** *General headers*

| *Header* | *Description* |
|---|---|
| Cache-control | Specifies information about caching |
| Connection | Shows whether the connection should be closed or not |
| Date | Shows the current date |
| MIME-version | Shows the MIME version used |
| Upgrade | Specifies the preferred communication protocol |

- **Request header** The request header can be present only in a request message. It specifies the client's configuration and the client's preferred document format. See Table 27.4 for a list of some request headers and their descriptions.

**Table 27.4** *Request headers*

| *Header* | *Description* |
|---|---|
| Accept | Shows the medium 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 |
| From | Shows the e-mail address of the user |
| Host | Shows the host and port number of the server |
| If-modified-since | Sends the document if newer than specified date |
| If-match | Sends the document only if it matches given tag |
| If-non-match | Sends the document only if it does not match given tag |
| If-range | Sends only the portion of the document that is missing |
| If-unmodified-since | Sends the document if not changed since specified date |
| Referrer | Specifies the URL of the linked document |
| User-agent | Identifies the client program |

- **Response header** The response header can be present only in a response message. It specifies the server's configuration and special information about the request. See Table 27.5 for a list of some response headers with their descriptions.

**Table 27.5** *Response headers*

| *Header* | *Description* |
|---|---|
| Accept-range | Shows if server accepts the range requested by client |
| Age | Shows the age of the document |
| Public | Shows the supported list of methods |
| Retry-after | Specifies the date after which the server is available |
| Server | Shows the server name and version number |

- **Entity header** The entity header gives information about the body of the document. Although it is mostly present in response messages, some request messages, such as POST or PUT methods, that contain a body also use this type of header. See Table 27.6 for a list of some entity headers and their descriptions.

**Table 27.6** *Entity headers*

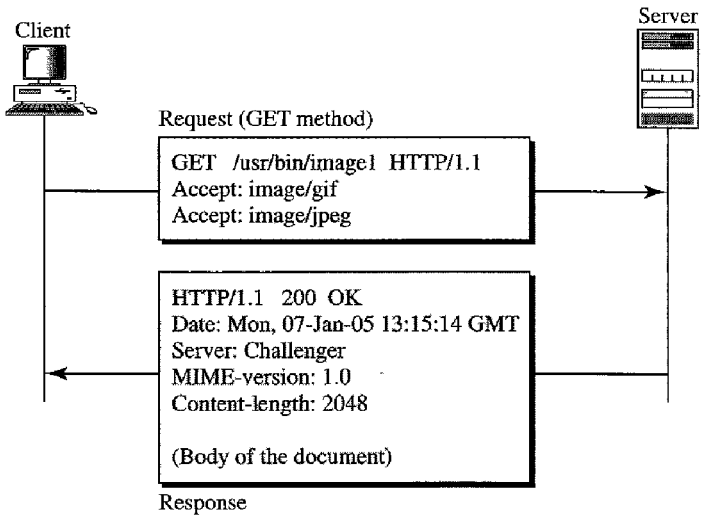
| *Header* | *Description* |
|---|---|
| Allow | Lists valid methods that can be used with a URL |
| Content-encoding | Specifies the encoding scheme |
| Content-language | Specifies the language |
| Content-length | Shows the length of the document |
| Content-range | Specifies the range of the document |
| Content-type | Specifies the medium type |
| Etag | Gives an entity tag |
| Expires | Gives the date and time when contents may change |
| Last-modified | Gives the date and time of the last change |
| Location | Specifies the location of the created or moved document |

**Body** The body can be present in a request or response message. Usually, it contains the document to be sent or received.

### *Example 27.1*

This example retrieves a document. 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, MIME version, and length of the document. The body of the document follows the header (see Figure 27.16).

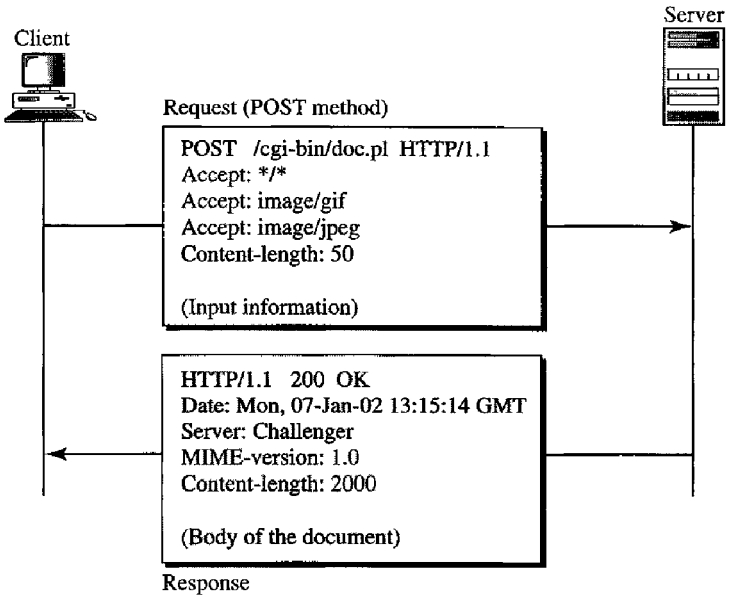
**Figure 27.16** *Example 27.1*



*Example 27.2*

In this example, the client wants to send data to the server. We use the POST method. The request line shows the method (POST), URL, and HTTP version (1.1). There are four lines of headers. The request body contains the input information. 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 27.17).

**Figure 27.17** *Example 27.2*




*Example 27.3*

HTTP uses ASCII characters. A client can directly connect to a server using TELNET, which logs into port 80. The next three lines show that the connection is successful.

We then type three lines. The first shows the request line (GET method), the second is the header (defining the host), the third is a blank, terminating the request.

The server response is seven lines starting with the status line. The blank line at the end terminates the server response. The file of 14,230 lines is received after the blank line (not shown here). The last line is the output by the client.

```
$ telnet www.mhhe.com 80
Trying 198.45.24.104 . . .
Connected to www.mhhe.com (198.45.24.104).
Escape character is '^]'.
GET /engcs/compsci/forouzan HTTP/1.1
From: forouzanbehrouz@fhda.edu

HTTP/1.1 200 OK
Date: Thu, 28 Oct 2004 16:27:46 GMT
Server: Apache/1.3.9 (Unix) ApacheJServ/1.1.2 PHP/4.1.2 PHP/3.0.18
MIME-version:1.0
Content-Type: text/html
```

**Last-modified: Friday, 15-Oct-04 02:11:31 GMT**
**Content-length: 14230**

Connection closed by foreign host.

## Persistent Versus Nonpersistent Connection

HTTP prior to version 1.1 specified a nonpersistent connection, while a persistent connection is the default in version 1.1.

### *Nonpersistent Connection*

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, for $N$ different pictures in different files, the connection must be opened and closed $N$ times. The nonpersistent strategy imposes high overhead on the server because the server needs $N$ different buffers and requires a slow start procedure each time a connection is opened.

### *Persistent Connection*

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.

---

**HTTP version 1.1 specifies a persistent connection by default.**

---

## Proxy Server

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.

## 27.4 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [. . .] refer to the reference list at the end of the text.

### Books

HTTP is discussed in Chapters 13 and 14 of [Ste96], Section 9.3 of [PD03], Chapter 35 of [Com04], and Section 7.3 of [Tan03].

### Sites

The following sites are related to topics discussed in this chapter.

- www.ietf.org/rfc.html Information about RFCs

### RFCs

The following RFCs are related to WWW:

1614, 1630, 1737, 1738

The following RFCs are related to HTTP:

2068, 2109

## 27.5 KEY TERMS

active document
Active Server Pages (ASP)
applet
browser
ColdFusion
Common Gateway Interface (CGI)
dynamic document
entity header
general header
host
Hypertext Markup Language (HTML)
Hypertext Preprocessor (PHP)
Hypertext Transfer Protocol (HTTP)
Java
JavaScript
Java Server Pages (JSP)
nonpersistent connection
path
persistent connection
proxy server
request header
request line
request type
response header
static document
status code
status line
tag
uniform resource locator (URL)
Web
World Wide Web (WWW)

## 27.6 SUMMARY

- The World Wide Web (WWW) is a repository of information linked together from points all over the world.
- Hypertexts are documents linked to one another through the concept of pointers.
- Browsers interpret and display a Web document.
- A browser consists of a controller, client programs, and interpreters.
- A Web document can be classified as static, dynamic, or active.
- A static document is one in which the contents are fixed and stored in a server. The client can make no changes in the server document.
- Hypertext Markup Language (HTML) is a language used to create static Web pages.
- Any browser can read formatting instructions (tags) embedded in an HTML document.
- Tags provide structure to a document, define titles and headers, format text, control the data flow, insert figures, link different documents together, and define executable code.
- A dynamic Web document is created by a server only at a browser request.
- The Common Gateway Interface (CGI) is a standard for creating and handling dynamic Web documents.
- A CGI program with its embedded CGI interface tags can be written in a language such as C, C++, Shell Script, or Perl.
- An active document is a copy of a program retrieved by the client and run at the client site.
- Java is a combination of a high-level programming language, a run-time environment, and a class library that allows a programmer to write an active document and a browser to run it.
- Java is used to create applets (small application programs).
- The Hypertext Transfer Protocol (HTTP) is the main protocol used to access data on the World Wide Web (WWW).
- HTTP uses a TCP connection to transfer files.
- An HTTP message is similar in form to an SMTP message.
- The HTTP request line consists of a request type, a URL, and the HTTP version number.
- The uniform resource locator (URL) consists of a method, host computer, optional port number, and path name to locate information on the WWW.
- The HTTP request type or method is the actual command or request issued by the client to the server.
- The status line consists of the HTTP version number, a status code, and a status phrase.
- The HTTP status code relays general information, information related to a successful request, redirection information, or error information.
- The HTTP header relays additional information between the client and server.

- An HTTP header consists of a header name and a header value.
- An HTTP general header gives general information about the request or response message.
- An HTTP request header specifies a client's configuration and preferred document format.
- An HTTP response header specifies a server's configuration and special information about the request.
- An HTTP entity header provides information about the body of a document.
- HTTP, version 1.1, specifies a persistent connection.
- A proxy server keeps copies of responses to recent requests.

## 27.7 PRACTICE SET

### Review Questions

1. How is HTTP related to WWW?
2. How is HTTP similar to SMTP?
3. How is HTTP similar to FTP?
4. What is a URL and what are its components?
5. What is a proxy server and how is it related to HTTP?
6. Name the common three components of a browser.
7. What are the three types of Web documents?
8. What does HTML stand for and what is its function?
9. What is the difference between an active document and a dynamic document?
10. What does CGI stand for and what is its function?
11. Describe the relationship between Java and an active document.

### Exercises

12. Where will each figure be shown on the screen?

```
Look at the following picture:
then tell me what you feel:
<IMG SRC="Pictures/Funny1.gif" ALIGN=middle>
<IMG SRC="Pictures/Funny2.gif" ALIGN=bottom>
<B>What is your feeling? </B>
```

13. Show the effect of the following HTML segment.

```
The publisher of this book is <A HREF="www.mhhe">
McGraw-Hill Publisher </A>
```

14. Show a request that retrieves the document /usr/users/doc/doc1. Use at least two general headers, two request headers, and one entity header.
15. Show the response to Exercise 14 for a successful request.
16. Show the response to Exercise 14 for a document that has permanently moved to /usr/deads/doc1.

17. Show the response to Exercise 14 if there is a syntax error in the request.
18. Show the response to Exercise 14 if the client is unauthorized to access the document.
19. Show a request that asks for information about a document at /bin/users/file. Use at least two general headers and one request header.
20. Show the response to Exercise 19 for a successful request.
21. Show the request to copy the file at location /bin/usr/bin/file1 to /bin/file1.
22. Show the response to Exercise 21.
23. Show the request to delete the file at location /bin/file1.
24. Show the response to Exercise 23.
25. Show a request to retrieve the file at location /bin/etc/file1. The client needs the document only if it was modified after January 23, 1999.
26. Show the response to Exercise 25.
27. Show a request to retrieve the file at location /bin/etc/file1. The client should identify itself.
28. Show the response to Exercise 27.
29. Show a request to store a file at location /bin/letter. The client identifies the types of documents it can accept.
30. Show the response to Exercise 29. The response shows the age of the document as well as the date and time when the contents may change.

# CHAPTER 28

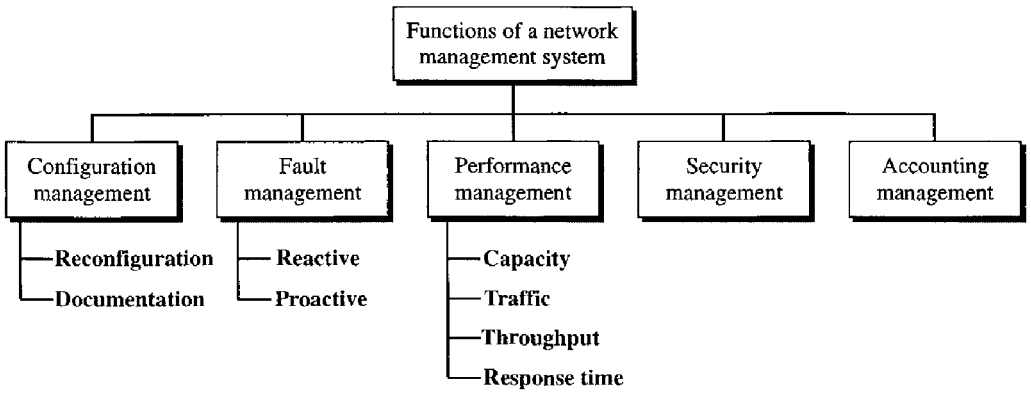
# *Network Management: SNMP*

We can define **network management** as monitoring, testing, configuring, and troubleshooting network components to meet a set of requirements defined by an organization. These requirements include the smooth, efficient operation of the network that provides the predefined quality of service for users. To accomplish this task, a network management system uses hardware, software, and humans. In this chapter, first we briefly discuss the functions of a network management system. Then we concentrate on the most common management system, the Simple Network Management Protocol (SNMP).

## 28.1 NETWORK MANAGEMENT SYSTEM

We can say that the functions performed by a network management system can be divided into five broad categories: configuration management, fault management, performance management, security management, and accounting management, as shown in Figure 28.1.

**Figure 28.1** *Functions of a network management system*



## Configuration Management

A large network is usually made up of hundreds of entities that are physically or logically connected to one another. These entities have an initial configuration when the network is set up, but can change with time. Desktop computers may be replaced by others; application software may be updated to a newer version; and users may move from one group to another. The **configuration management** system must know, at any time, the status of each entity and its relation to other entities. Configuration management can be divided into two subsystems: reconfiguration and documentation.

### *Reconfiguration*

Reconfiguration, which means adjusting the network components and features, can be a daily occurrence in a large network. There are three types of reconfiguration: hardware reconfiguration, software reconfiguration, and user-account reconfiguration.

Hardware reconfiguration covers all changes to the hardware. For example, a desktop computer may need to be replaced. A router may need to be moved to another part of the network. A subnetwork may be added or removed from the network. All these need the time and attention of network management. In a large network, there must be specialized personnel trained for quick and efficient hardware reconfiguration. Unfortunately, this type of reconfiguration cannot be automated and must be manually handled case by case.

Software reconfiguration covers all changes to the software. For example, new software may need to be installed on servers or clients. An operating system may need updating. Fortunately, most software reconfiguration can be automated. For example, updating an application on some or all clients can be electronically downloaded from the server.

User-account reconfiguration is not simply adding or deleting users on a system. You must also consider the user privileges, both as an individual and as a member of a group. For example, a user may have read and write permission with regard to some files, but only read permission with regard to other files. User-account reconfiguration can be, to some extent, automated. For example, in a college or university, at the beginning of each quarter or semester, new students are added to the system. The students are normally grouped according to the courses they take or the majors they pursue.

### *Documentation*

The original network configuration and each subsequent change must be recorded meticulously. This means that there must be documentation for hardware, software, and user accounts.

**Hardware documentation** normally involves two sets of documents: maps and specifications. Maps track each piece of hardware and its connection to the network. There can be one general map that shows the logical relationship between each subnetwork. There can also be a second general map that shows the physical location of each subnetwork. For each subnetwork, then, there is one or more maps that show all pieces of equipment. The maps use some kind of standardization to be easily read and understood by current and future personnel. Maps are not enough per se. Each piece of hardware also needs to be documented. There must be a set of specifications for each piece

of hardware connected to the network. These specifications must include information such as hardware type, serial number, vendor (address and phone number), time of purchase, and warranty information.

All software must also be documented. Software documentation includes information such as the software type, the version, the time installed, and the license agreement.

Most operating systems have a utility that allows the documentation of user accounts and their privileges. The management must make sure that the files with this information are updated and secured. Some operating systems record access privileges in two documents; one shows all files and access types for each user; the other shows the list of users that have a particular access to a file.

## Fault Management

Complex networks today are made up of hundreds and sometimes thousands of components. Proper operation of the network depends on the proper operation of each component individually and in relation to each other. **Fault management** is the area of network management that handles this issue.

An effective fault management system has two subsystems: reactive fault management and proactive fault management.

### *Reactive Fault Management*

A reactive fault management system is responsible for detecting, isolating, correcting, and recording faults. It handles short-term solutions to faults.

The first step taken by a reactive fault management system is to detect the exact location of the fault. A fault is defined as an abnormal condition in the system. When a fault occurs, either the system stops working properly or the system creates excessive errors. A good example of a fault is a damaged communication medium. This fault may interrupt communication or produce excessive errors.

The next step taken by a reactive fault management system is to isolate the fault. A fault, if isolated, usually affects only a few users. After isolation, the affected users are immediately notified and given an estimated time of correction.

The third step is to correct the fault. This may involve replacing or repairing the faulty component(s).

After the fault is corrected, it must be documented. The record should show the exact location of the fault, the possible cause, the action or actions taken to correct the fault, the cost, and time it took for each step. Documentation is extremely important for several reasons:

- The problem may recur. Documentation can help the present or future administrator or technician solve a similar problem.
- The frequency of the same kind of failure is an indication of a major problem in the system. If a fault happens frequently in one component, it should be replaced with a similar one, or the whole system should be changed to avoid the use of that type of component.
- The statistic is helpful to another part of network management, performance management.

### *Proactive Fault Management*

Proactive fault management tries to prevent faults from occurring. Although this is not always possible, some types of failures can be predicted and prevented. For example, if a manufacturer specifies a lifetime for a component or a part of a component, it is a good strategy to replace it before that time. As another example, if a fault happens frequently at one particular point of a network, it is wise to carefully reconfigure the network to prevent the fault from happening again.

## Performance Management

**Performance management,** which is closely related to fault management, tries to monitor and control the network to ensure that it is running as efficiently as possible. Performance management tries to quantify performance by using some measurable quantity such as capacity, traffic, throughput, or response time.

### *Capacity*

One factor that must be monitored by a performance management system is the capacity of the network. Every network has a limited capacity, and the performance management system must ensure that it is not used above this capacity. For example, if a LAN is designed for 100 stations at an average data rate of 2 Mbps, it will not operate properly if 200 stations are connected to the network. The data rate will decrease and blocking may occur.

### *Traffic*

Traffic can be measured in two ways: internally and externally. Internal traffic is measured by the number of packets (or bytes) traveling inside the network. External traffic is measured by the exchange of packets (or bytes) outside the network. During peak hours, when the system is heavily used, blocking may occur if there is excessive traffic.

### *Throughput*

We can measure the throughput of an individual device (such as a router) or a part of the network. Performance management monitors the throughput to make sure that it is not reduced to unacceptable levels.

### *Response Time*

Response time is normally measured from the time a user requests a service to the time the service is granted. Other factors such as capacity and traffic can affect the response time. Performance management monitors the average response time and the peak-hour response time. Any increase in response time is a very serious condition as it is an indication that the network is working above its capacity.

## Security Management

**Security management** is responsible for controlling access to the network based on the predefined policy. We discuss security and in particular network security in Chapters 31 and 32.

### Accounting Management

**Accounting management** is the control of users' access to network resources through charges. Under accounting management, individual users, departments, divisions, or even projects are charged for the services they receive from the network. Charging does not necessarily mean cash transfer; it may mean debiting the departments or divisions for budgeting purposes. Today, organizations use an accounting management system for the following reasons:

- It prevents users from monopolizing limited network resources.
- It prevents users from using the system inefficiently.
- Network managers can do short- and long-term planning based on the demand for network use.

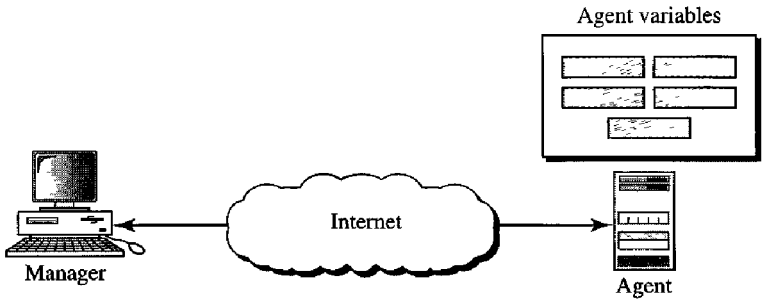
## 28.2 SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)

The **Simple Network Management Protocol (SNMP)** is a framework for managing devices in an internet using the TCP/IP protocol suite. It provides a set of fundamental operations for monitoring and maintaining an internet.

### Concept

SNMP uses the concept of manager and agent. That is, a manager, usually a host, controls and monitors a set of agents, usually routers (see Figure 28.2).

**Figure 28.2** *SNMP concept*




SNMP is an application-level protocol in which a few manager stations control a set of agents. The protocol is designed at the application level so that it can monitor devices made by different manufacturers and installed on different physical networks. In other words, SNMP frees management tasks from both the physical characteristics of the managed devices and the underlying networking technology. It can be used in a heterogeneous internet made of different LANs and WANs connected by routers made by different manufacturers.

### *Managers and Agents*

A management station, called a **manager,** is a host that runs the SNMP client program. A managed station, called an **agent,** is a router (or a host) that runs the SNMP server program. Management is achieved through simple interaction between a manager and an agent.

The agent keeps performance information in a database. The manager has access to the values in the database. For example, a router can store in appropriate variables the number of packets received and forwarded. The manager can fetch and compare the values of these two variables to see if the router is congested or not.

The manager can also make the router perform certain actions. For example, a router periodically checks the value of a reboot counter to see when it should reboot itself. It reboots itself, for example, if the value of the counter is 0. The manager can use this feature to reboot the agent remotely at any time. It simply sends a packet to force a 0 value in the counter.

Agents can also contribute to the management process. The server program running on the agent can check the environment, and if it notices something unusual, it can send a warning message, called a **trap,** to the manager.

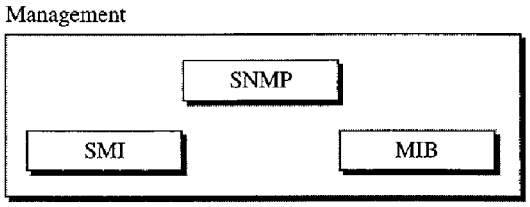
In other words, management with SNMP is based on three basic ideas:

1. A manager checks an agent by requesting information that reflects the behavior of the agent.
2. A manager forces an agent to perform a task by resetting values in the agent database.
3. An agent contributes to the management process by warning the manager of an unusual situation.

## Management Components

To do management tasks, SNMP uses two other protocols: **Structure of Management Information (SMI)** and **Management Information Base (MIB).** In other words, management on the Internet is done through the cooperation of the three protocols SNMP, SMI, and MIB, as shown in Figure 28.3.

**Figure 28.3** *Components of network management on the Internet*




Let us elaborate on the interactions between these protocols.

### *Role of SNMP*

SNMP has some very specific roles in network management. It defines the format of the packet to be sent from a manager to an agent and vice versa. It also interprets the

result and creates statistics (often with the help of other management software). The packets exchanged contain the object (variable) names and their status (values). SNMP is responsible for reading and changing these values.

**SNMP defines the format of packets exchanged between a manager and an agent. It reads and changes the status (values) of objects (variables) in SNMP packets.**

### *Role of SMI*

To use SNMP, we need rules. We need rules for naming objects. This is particularly important because the objects in SNMP form a hierarchical structure (an object may have a parent object and some children objects). Part of a name can be inherited from the parent. We also need rules to define the type of the objects. What types of objects are handled by SNMP? Can SNMP handle simple types or structured types? How many simple types are available? What are the sizes of these types? What is the range of these types? In addition, how are each of these types encoded?

We need these universal rules because we do not know the architecture of the computers that send, receive, or store these values. The sender may be a powerful computer in which an integer is stored as 8-byte data; the receiver may be a small computer that stores an integer as 4-byte data.

SMI is a protocol that defines these rules. However, we must understand that SMI only defines the rules; it does not define how many objects are managed in an entity or which object uses which type. SMI is a collection of general rules to name objects and to list their types. The association of an object with the type is not done by SMI.

**SMI defines the general rules for naming objects, defining object types (including range and length), and showing how to encode objects and values.**

**SMI does not define the number of objects an entity should manage or name the objects to be managed or define the association between the objects and their values.**

### *Role of MIB*

We hope it is clear that we need another protocol. For each entity to be managed, this protocol must define the number of objects, name them according to the rules defined by SMI, and associate a type to each named object. This protocol is MIB. MIB creates a set of objects defined for each entity similar to a database (mostly metadata in a database, names and types without values).

**MIB creates a collection of named objects, their types, and their relationships to each other in an entity to be managed.**

### *An Analogy*

Before discussing each of these protocols in greater detail, we give an analogy. The three network management components are similar to what we need when we write a program in a computer language to solve a problem.

Before we write a program, the syntax of the language (such as C or Java) must be predefined. The language also defines the structure of variables (simple, structured, pointer, and so on) and how the variables must be named. For example, a variable name must be 1 to $N$ characters in length and start with a letter followed by alphanumeric characters. The language also defines the type of data to be used (integer, float, char, etc.). In programming the rules are defined by the language. In network management the rules are defined by SMI.

Most computer languages require that variables be declared in each specific program. The declaration names each variable and defines the predefined type. For example, if a program has two variables (an integer named *counter* and an array named *grades* of type char), they must be declared at the beginning of the program:

```
int counter;
char grades [40];
```

Note that the declarations name the variables (counter and grades) and define the type of each variable. Because the types are predefined in the language, the program knows the range and size of each variable.

MIB does this task in network management. MIB names each object and defines the type of the objects. Because the type is defined by SMI, SNMP knows the range and size.

After declaration in programming, the program needs to write statements to store values in the variables and change them if needed. SNMP does this task in network management. SNMP stores, changes, and interprets the values of objects already declared by MIB according to the rules defined by SMI.

---

**We can compare the task of network management to the task of writing a program.**

- ❑ Both tasks need rules. In network management this is handled by SMI.
- ❑ Both tasks need variable declarations. In network management this is handled by MIB.
- ❑ Both tasks have actions performed by statements. In network management this is handled by SNMP.

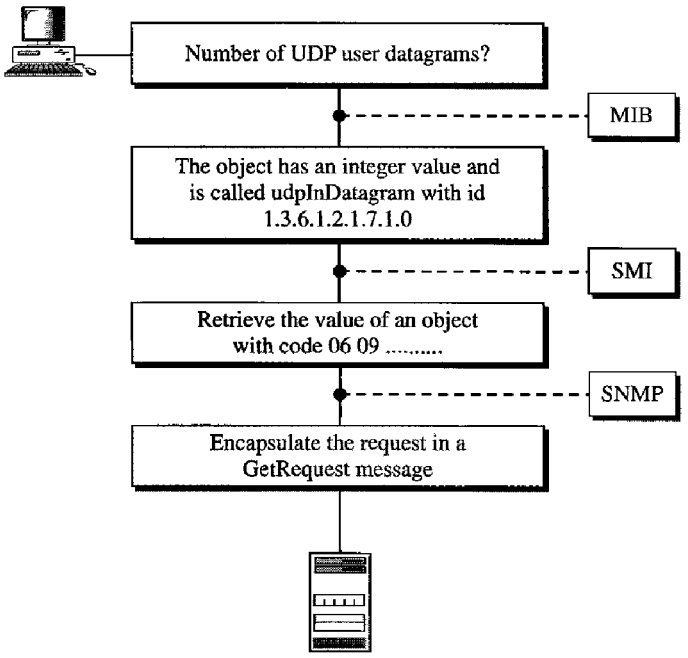
---

### *An Overview*

Before discussing each component in detail, we show how each is involved in a simple scenario. This is an overview that will be developed later at the end of the chapter. A manager station (SNMP client) wants to send a message to an agent station (SNMP server) to find the number of UDP user datagrams received by the agent. Figure 28.4 shows an overview of steps involved.

MIB is responsible for finding the object that holds the number of the UDP user datagrams received. SMI, with the help of another embedded protocol, is responsible for encoding the name of the object. SNMP is responsible for creating a message, called a GetRequest message, and encapsulating the encoded message. Of course, things are more complicated than this simple overview, but we first need more details of each protocol.

**Figure 28.4** *Management overview*



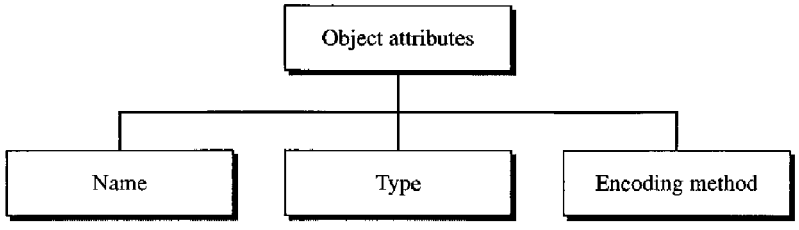

## Structure of Management Information

The Structure of Management Information, version 2 (SMIv2) is a component for network management. Its functions are

1. To name objects
2. To define the type of data that can be stored in an object
3. To show how to encode data for transmission over the network

SMI is a guideline for SNMP. It emphasizes three attributes to handle an object: name, data type, and encoding method (see Figure 28.5).

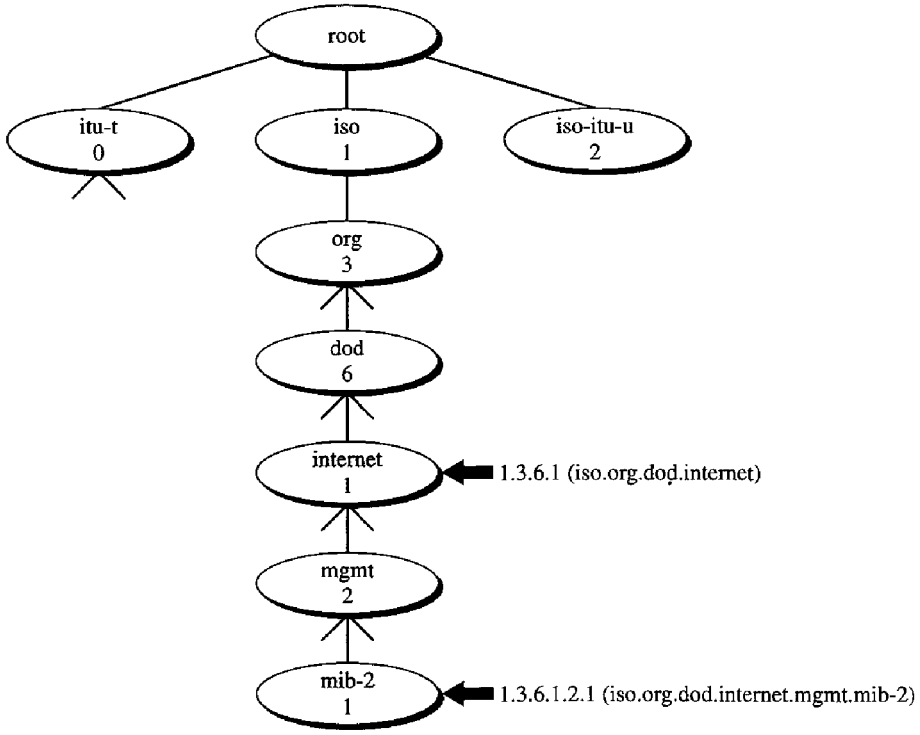
**Figure 28.5** *Object attributes*




### *Name*

SMI requires that each managed object (such as a router, a variable in a router, a value) have a unique name. To name objects globally, SMI uses an **object identifier,** which is a hierarchical identifier based on a tree structure (see Figure 28.6).

**Figure 28.6** *Object identifier*




The tree structure starts with an unnamed root. Each object can be defined by using a sequence of integers separated by dots. The tree structure can also define an object by using a sequence of textual names separated by dots. The integer-dot representation is used in SNMP. The name-dot notation is used by people. For example, the following shows the same object in two different notations:

iso.org.dod.internet.mgmt.mib-2 ➡ 1.3.6.1.2.1

The objects that are used in SNMP are located under the *mib-2* object, so their identifiers always start with 1.3.6.1.2.1.

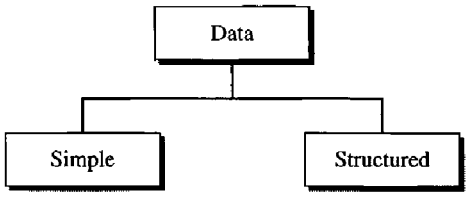
**All objects managed by SNMP are given an object identifier. The object identifier always starts with 1.3.6.1.2.1.**

### *Type*

The second attribute of an object is the type of data stored in it. To define the data type, SMI uses fundamental **Abstract Syntax Notation 1 (ASN.1)** definitions and adds some new definitions. In other words, SMI is both a subset and a superset of ASN.1.

SMI has two broad categories of data type: *simple* and *structured*. We first define the simple types and then show how the structured types can be constructed from the simple ones (see Figure 28.7).

**Figure 28.7** *Data type*




**Simple Type** The **simple data types** are atomic data types. Some of them are taken directly from ASN.1; others are added by SMI. The most important ones are given in Table 28.1. The first five are from ASN.1; the next seven are defined by SMI.

**Table 28.1** *Data types*

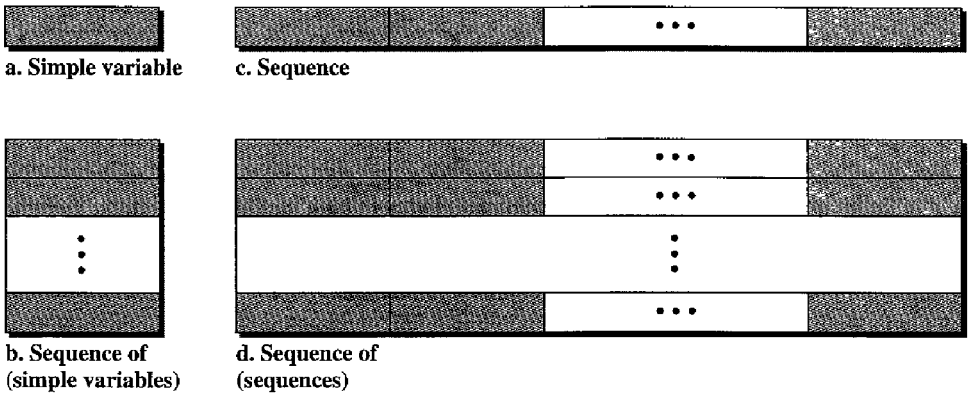
| *Type* | *Size* | *Description* |
|---|---|---|
| INTEGER | 4 bytes | An integer with a value between $-2^{31}$ and $2^{31} - 1$ |
| Integer32 | 4 bytes | Same as INTEGER |
| Unsigned32 | 4 bytes | Unsigned with a value between 0 and $2^{32} - 1$ |
| OCTET STRING | Variable | Byte string up to 65,535 bytes long |
| OBJECT IDENTIFIER | Variable | An object identifier |
| IPAddress | 4 bytes | An IP address made of four integers |
| Counter32 | 4 bytes | An integer whose value can be incremented from 0 to $2^{32}$; when it reaches its maximum value, it wraps back to 0. |
| Counter64 | 8 bytes | 64-bit counter |
| Gauge32 | 4 bytes | Same as Counter32, but when it reaches its maximum value, it does not wrap; it remains there until it is reset |
| TimeTicks | 4 bytes | A counting value that records time in $\frac{1}{100}$ s |
| BITS | | A string of bits |
| Opaque | Variable | Uninterpreted string |

**Structured Type** By combining simple and structured data types, we can make new structured data types. SMI defines two **structured data types:** *sequence* and *sequence of.*

- **Sequence.** A *sequence* data type is a combination of simple data types, not necessarily of the same type. It is analogous to the concept of a *struct* or a *record* used in programming languages such as C.
- **Sequence of.** A *sequence of* data type is a combination of simple data types all of the same type or a combination of sequence data types all of the same type. It is analogous to the concept of an *array* used in programming languages such as C.

Figure 28.8 shows a conceptual view of data types.

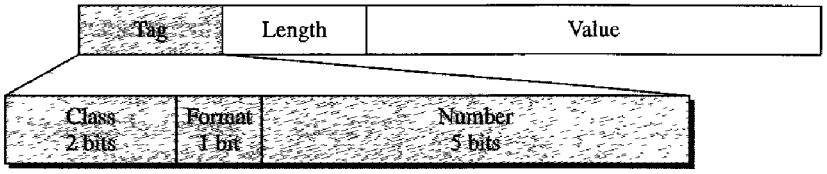
**Figure 28.8** *Conceptual data types*




### *Encoding Method*

SMI uses another standard, **Basic Encoding Rules (BER),** to encode data to be transmitted over the network. BER specifies that each piece of data be encoded in triplet format: tag, length, and value, as illustrated in Figure 28.9.

**Figure 28.9** *Encoding format*




- **Tag.** The tag is a 1-byte field that defines the type of data. It is composed of three subfields: *class* (2 bits), *format* (1 bit), and *number* (5 bits). The class subfield defines the scope of the data. Four classes are defined: universal (00), applicationwide (01), context-specific (10), and private (11). The universal data types are those taken from ASN.1 (INTEGER, OCTET STRING, and ObjectIdentifier). The applicationwide data types are those added by SMI (IPAddress, Counter, Gauge, and TimeTicks). The five context-specific data types have meanings that may change from one protocol to another. The private data types are vendor-specific.

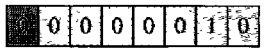
  The format subfield indicates whether the data are simple (0) or structured (1). The number subfield further divides simple or structured data into subgroups. For example, in the universal class, with simple format, INTEGER has a value of 2, OCTET STRING has a value of 4, and so on. Table 28.2 shows the data types we use in this chapter and their tags in binary and hexadecimal numbers.

**Table 28.2** *Codes for data types*

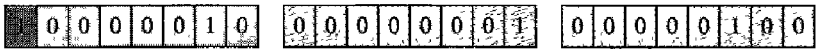
| *Data Type* | *Class* | *Format* | *Number* | *Tag (Binary)* | *Tag (Hex)* |
|---|---|---|---|---|---|
| INTEGER | 00 | 0 | 00010 | **00000010** | **02** |
| OCTET STRING | 00 | 0 | 00100 | **00000100** | **04** |
| OBJECT IDENTIFIER | 00 | 0 | 00110 | **00000110** | **06** |
| NULL | 00 | 0 | 00101 | **00000101** | **05** |
| Sequence, sequence of | 00 | 1 | 10000 | **00110000** | **30** |
| IPAddress | 01 | 0 | 00000 | **01000000** | **40** |
| Counter | 01 | 0 | 00001 | **01000001** | **41** |
| Gauge | 01 | 0 | 00010 | **01000010** | **42** |
| TimeTicks | 01 | 0 | 00011 | **01000011** | **43** |
| Opaque | 01 | 0 | 00100 | **01000100** | **44** |

- **Length.** The length field is 1 or more bytes. If it is 1 byte, the most significant bit must be 0. The other 7 bits define the length of the data. If it is more than 1 byte, the most significant bit of the first byte must be 1. The other 7 bits of the first byte define the number of bytes needed to define the length. See Figure 28.10 for a depiction of the length field.

**Figure 28.10** *Length format*




**a. The colored part defines the length (2).**




**b. The shaded part defines the length of the length (2 bytes); the colored bytes define the length (260 bytes).**

- **Value.** The value field codes the value of the data according to the rules defined in BER.

To show how these three fields—tag, length, and value—can define objects, we give some examples.

### *Example 28.1*

Figure 28.11 shows how to define INTEGER 14.

### *Example 28.2*

Figure 28.12 shows how to define the OCTET STRING "HI."

**Figure 28.11** *Example 28.1, INTEGER 14*

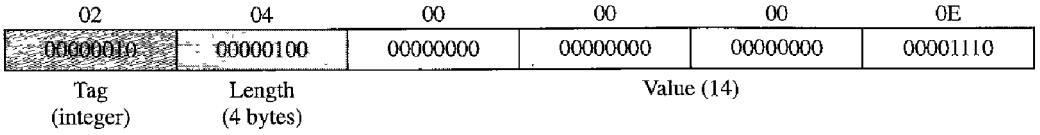
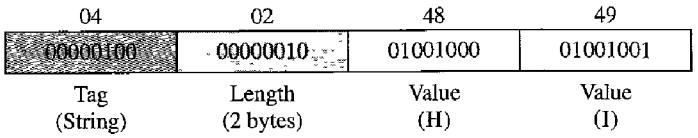


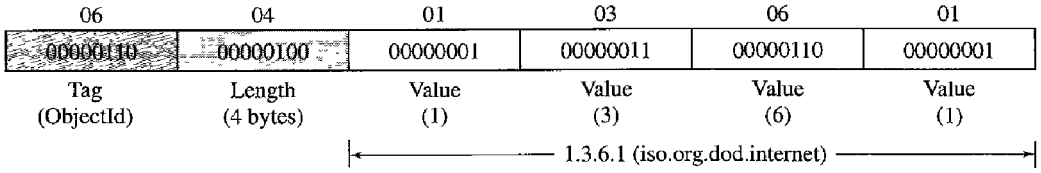

**Figure 28.12** *Example 28.2, OCTET STRING "HI"*




### *Example 28.3*

Figure 28.13 shows how to define ObjectIdentifier 1.3.6.1 (iso.org.dod.internet).

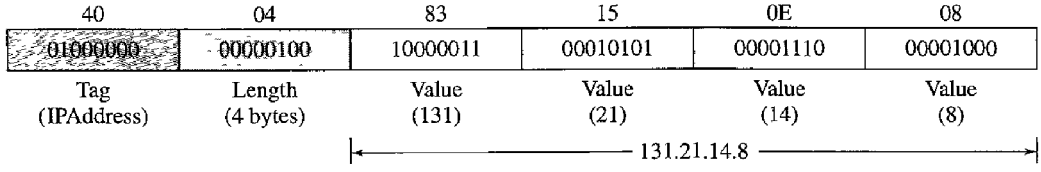
**Figure 28.13** *Example 28.3, ObjectIdentifier 1.3.6.1*




### *Example 28.4*

Figure 28.14 shows how to define IPAddress 131.21.14.8.

**Figure 28.14** *Example 28.4, IPAddress 131.21.14.8*

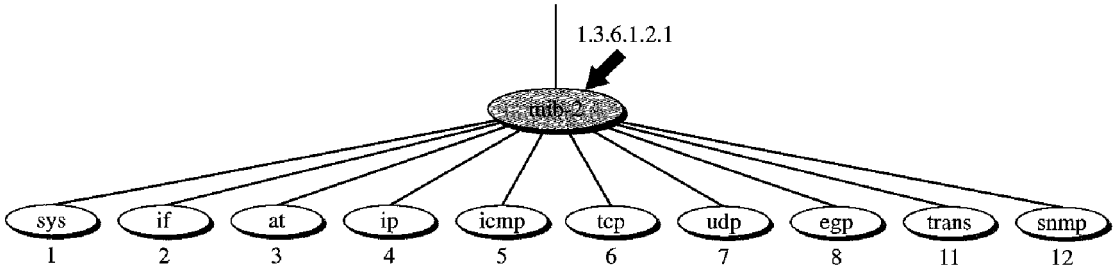



## Management Information Base (MIB)

The Management Information Base, version 2 (MIB2) is the second component used in network management. Each agent has its own MIB2, which is a collection of all the objects that the manager can manage. The objects in MIB2 are categorized under 10

different groups: system, interface, address translation, ip, icmp, tcp, udp, egp, transmission, and snmp. These groups are under the mib-2 object in the object identifier tree (see Figure 28.15). Each group has defined variables and/or tables.

**Figure 28.15** *mib-2*




The following is a brief description of some of the objects:

- **sys** This object (*system*) defines general information about the node (system), such as the name, location, and lifetime.
- **if** This object (*interface*) defines information about all the interfaces of the node including interface number, physical address, and IP address.
- **at** This object (*address translation*) defines the information about the ARP table.
- **ip** This object defines information related to IP, such as the routing table and the IP address.
- **icmp** This object defines information related to ICMP, such as the number of packets sent and received and total errors created.
- **tcp** This object defines general information related to TCP, such as the connection table, time-out value, number of ports, and number of packets sent and received.
- **udp** This object defines general information related to UDP, such as the number of ports and number of packets sent and received.
- **snmp** This object defines general information related to SNMP itself.

### *Accessing MIB Variables*

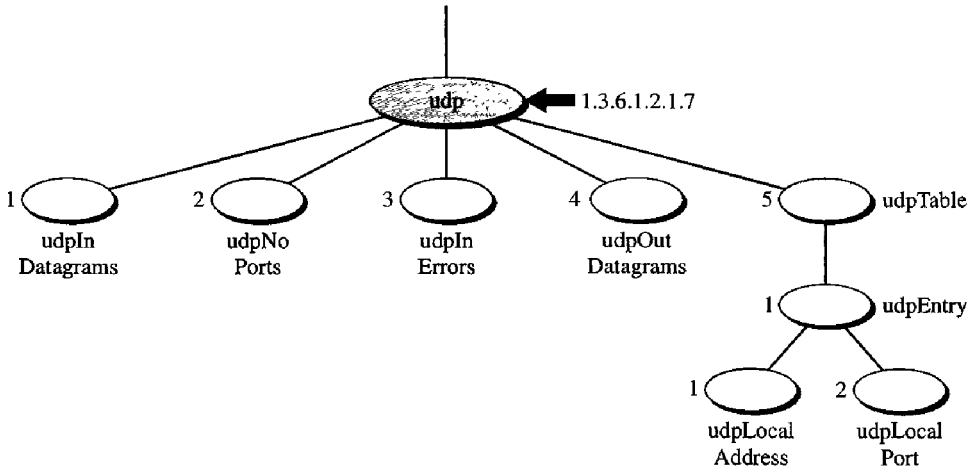
To show how to access different variables, we use the udp group as an example. There are four simple variables in the udp group and one sequence of (table of) records. Figure 28.16 shows the variables and the table.

We will show how to access each entity.

**Simple Variables** To access any of the simple variables, we use the id of the group (1.3.6.1.2.1.7) followed by the id of the variable. The following shows how to access each variable.

udpInDatagrams ➡ 1.3.6.1.2.1.7.1
udpNoPorts ➡ 1.3.6.1.2.1.7.2
udpInErrors ➡ 1.3.6.1.2.1.7.3
udpOutDatagrams ➡ 1.3.6.1.2.1.7.4

**Figure 28.16** *udp group*




However, these object identifiers define the variable, not the instance (contents). To show the instance or the contents of each variable, we must add an instance suffix. The instance suffix for a simple variable is simply a 0. In other words, to show an instance of the above variables, we use the following:

udpInDatagrams.0 ➡ 1.3.6.1.2.1.7.1.**0**
udpNoPorts.0 ➡ 1.3.6.1.2.1.7.2.**0**
udpInErrors.0 ➡ 1.3.6.1.2.1.7.3.**0**
udpOutDatagrams.0 ➡ 1.3.6.1.2.1.7.4.**0**

**Tables** To identify a table, we first use the table id. The udp group has only one table (with id 5) as illustrated in Figure 28.17.

So to access the table, we use the following:

udpTable ➡ 1.3.6.1.2.1.7.5

However, the table is not at the leaf level in the tree structure. We cannot access the table; we define the entry (sequence) in the table (with id of 1), as follows:

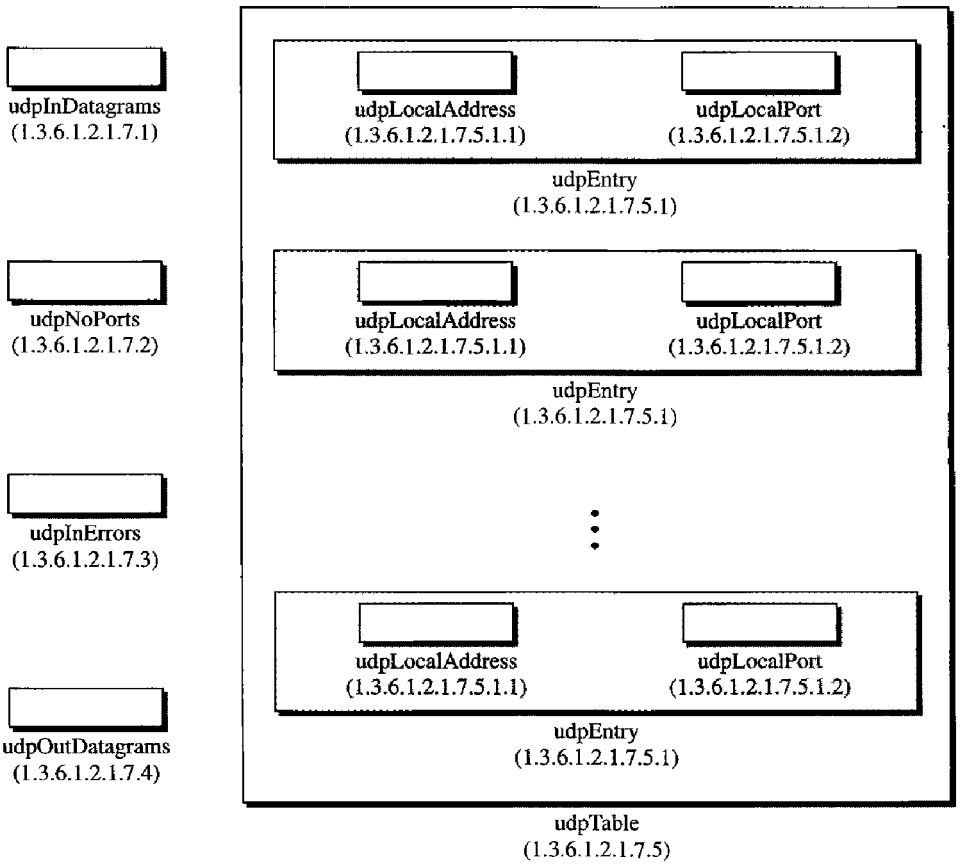
udpEntry ➡ 1.3.6.1.2.1.7.5.**1**

This entry is also not a leaf and we cannot access it. We need to define each entity (field) in the entry.

udpLocalAddress ➡ 1.3.6.1.2.1.7.5.1.**1**
udpLocalPort ➡ 1.3.6.1.2.1.7.5.1.**2**

These two variables are at the leaf of the tree. Although we can access their instances, we need to define *which* instance. At any moment, the table can have several values for

**Figure 28.17** *udp variables and tables*




each local address/local port pair. To access a specific instance (row) of the table, we add the index to the above ids. In MIB, the indexes of arrays are not integers (like most programming languages). The indexes are based on the value of one or more fields in the entries. In our example, the udpTable is indexed based on both the local address and the local port number. For example, Figure 28.18 shows a table with four rows and values for each field. The index of each row is a combination of two values.

To access the instance of the local address for the first row, we use the identifier augmented with the instance index:

udpLocalAddress.181.23.45.14.23 ➡ 1.3.6.1.2.7.5.1.1.181.23.45.14.23

Note that not all tables are indexed in the same way. Some tables are indexed by using the value of one field, others by using the value of two fields, and so on.

## Lexicographic Ordering

One interesting point about the MIB variables is that the object identifiers (including the instance identifiers) follow in lexicographic order. Tables are ordered according to column-row rules, which means one should go column by column. In each column, one should go from the top to the bottom, as shown in Figure 28.19.

**Figure 28.18** *Indexes for udpTable*

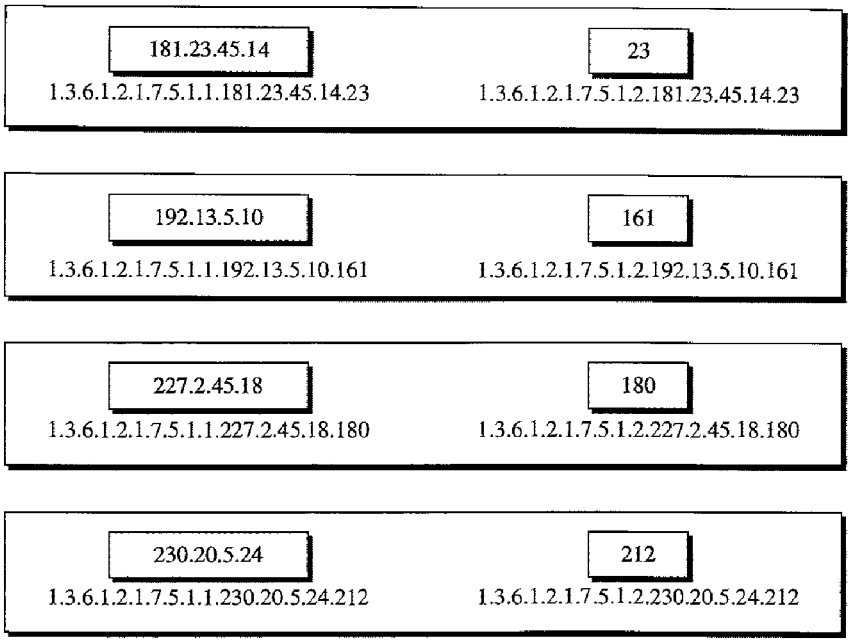
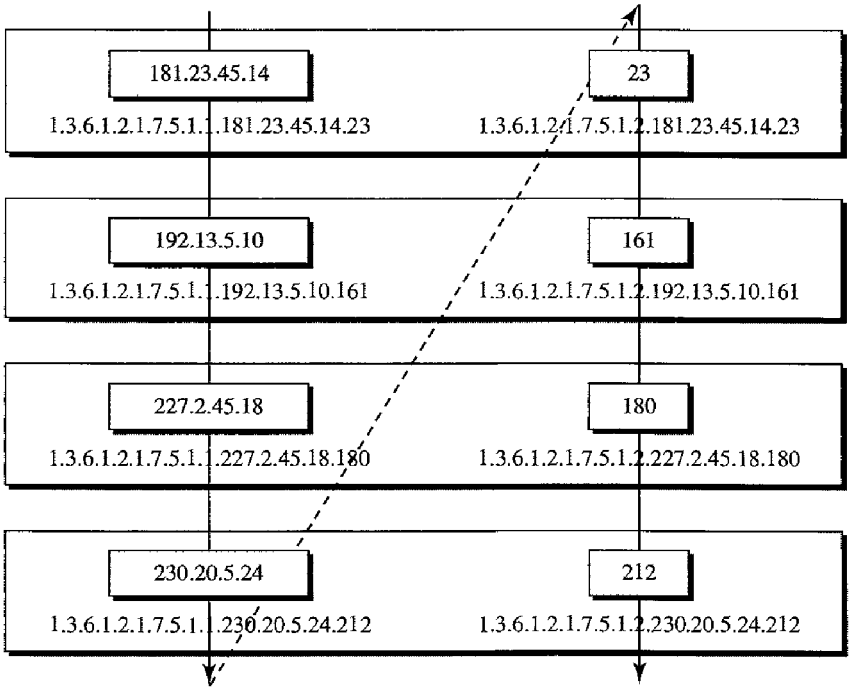



**Figure 28.19** *Lexicographic ordering*



The **lexicographic ordering** enables a manager to access a set of variables one after another by defining the first variable, as we will see in the GetNextRequest command in the next section.

## SNMP

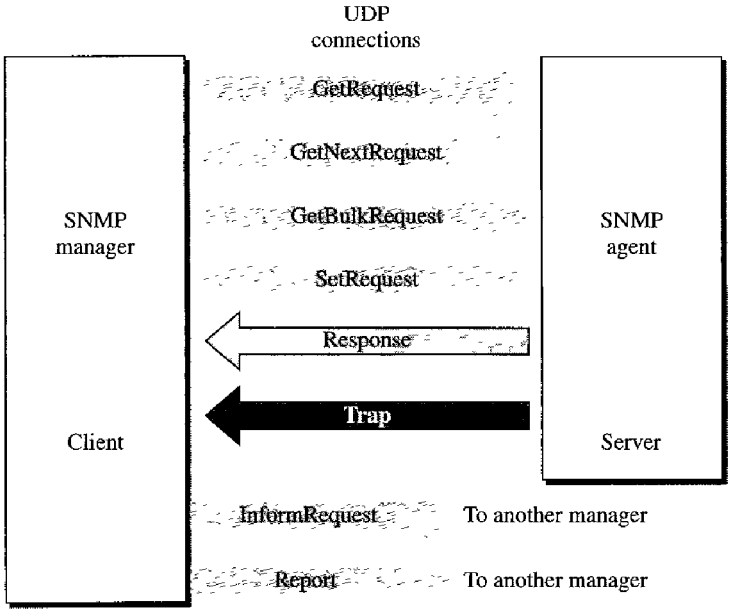
SNMP uses both SMI and MIB in Internet network management. It is an application program that allows

1. A manager to retrieve the value of an object defined in an agent
2. A manager to store a value in an object defined in an agent
3. An agent to send an alarm message about an abnormal situation to the manager

### *PDUs*

SNMPv3 defines eight types of packets (or PDUs): GetRequest, GetNextRequest, GetBulkRequest, SetRequest, Response, Trap, InformRequest, and Report (see Figure 28.20).

**Figure 28.20** *SNMP PDUs*




**GetRequest** The GetRequest PDU is sent from the manager (client) to the agent (server) to retrieve the value of a variable or a set of variables.

**GetNextRequest** The GetNextRequest PDU is sent from the manager to the agent to retrieve the value of a variable. The retrieved value is the value of the object following the defined ObjectId in the PDU. It is mostly used to retrieve the values of the entries in a table. If the manager does not know the indexes of the entries, it cannot retrieve the values. However, it can use GetNextRequest and define the ObjectId of the table. Because the first entry has the ObjectId immediately after the ObjectId of the table, the value of the first entry is returned. The manager can use this ObjectId to get the value of the next one, and so on.

**GetBulkRequest** The GetBulkRequest PDU is sent from the manager to the agent to retrieve a large amount of data. It can be used instead of multiple GetRequest and GetNextRequest PDUs.

**SetRequest** The SetRequest PDU is sent from the manager to the agent to set (store) a value in a variable.

**Response** The Response PDU is sent from an agent to a manager in response to GetRequest or GetNextRequest. It contains the value(s) of the variable(s) requested by the manager.

**Trap** The **Trap** (also called SNMPv2 Trap to distinguish it from SNMPv1 Trap) PDU is sent from the agent to the manager to report an event. For example, if the agent is rebooted, it informs the manager and reports the time of rebooting.

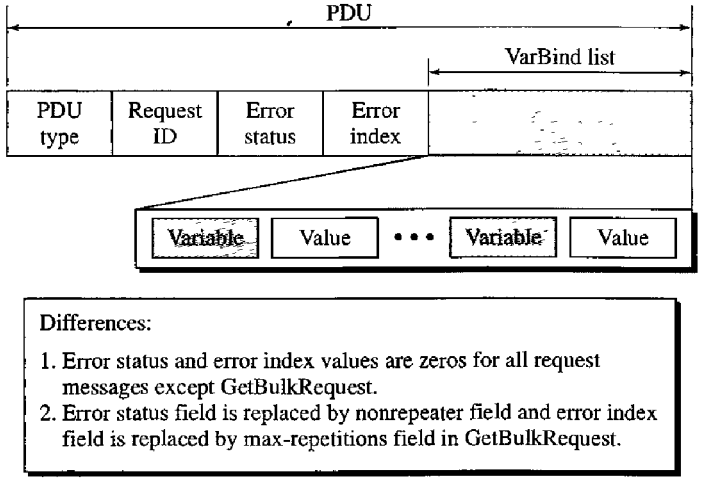
**InformRequest** The InformRequest PDU is sent from one manager to another remote manager to get the value of some variables from agents under the control of the remote manager. The remote manager responds with a Response PDU.

**Report** The Report PDU is designed to report some types of errors between managers. It is not yet in use.

*Format*

The format for the eight SNMP PDUs is shown in Figure 28.21. The GetBulkRequest PDU differs from the others in two areas, as shown in the figure.

**Figure 28.21** *SNMP PDU format*




The fields are listed below:

- **PDU type.** This field defines the type of the PDU (see Table 28.4).
- **Request ID.** This field is a sequence number used by the manager in a Request PDU and repeated by the agent in a response. It is used to match a request to a response.

- **Error status.** This is an integer that is used only in Response PDUs to show the types of errors reported by the agent. Its value is 0 in Request PDUs. Table 28.3 lists the types of errors that can occur.

**Table 28.3** *Types of errors*

| *Status* | *Name* | *Meaning* |
|---|---|---|
| 0 | noError | No error |
| 1 | tooBig | Response too big to fit in one message |
| 2 | noSuchName | Variable does not exist |
| 3 | badValue | The value to be stored is invalid |
| 4 | readOnly | The value cannot be modified |
| 5 | genErr | Other errors |

- **Nonrepeaters.** This field is used only in GetBulkRequest and replaces the error status field, which is empty in Request PDUs.
- **Error index.** The error index is an offset that tells the manager which variable caused the error.
- **Max-repetition.** This field is also used only in GetBulkRequest and replaces the error index field, which is empty in Request PDUs.
- **VarBind list.** This is a set of variables with the corresponding values the manager wants to retrieve or set. The values are null in GetRequest and GetNextRequest. In a Trap PDU, it shows the variables and values related to a specific PDU.

## Messages

SNMP does not send only a PDU, it embeds the PDU in a message. A message in SNMPv3 is made of four elements: version, header, security parameters, and data (which include the encoded PDU), as shown in Figure 28.22.

Because the length of these elements is different from message to message, SNMP uses BER to encode each element. Remember that BER uses the tag and the length to define a value. The *version* defines the current version (3). The *header* contains values for message identification, maximum message size (the maximum size of the reply), message flag (one octet of data type OCTET STRING where each bit defines security type, such as privacy or authentication, or other information), and a message security model (defining the security protocol). The message *security parameter* is used to create a message digest (see Chapter 31). The data contain the PDU. If the data are encrypted, there is information about the encrypting engine (the manager program that did the encryption) and the encrypting context (the type of encryption) followed by the encrypted PDU. If the data are not encrypted, the data consist of just the PDU.

To define the type of PDU, SNMP uses a tag. The class is context-sensitive (10), the format is structured (1), and the numbers are 0, 1, 2, 3, 5, 6, 7, and 8 (see Table 28.4). Note that SNMPv1 defined A4 for Trap, which is obsolete today.

**Figure 28.22** *SNMP message*

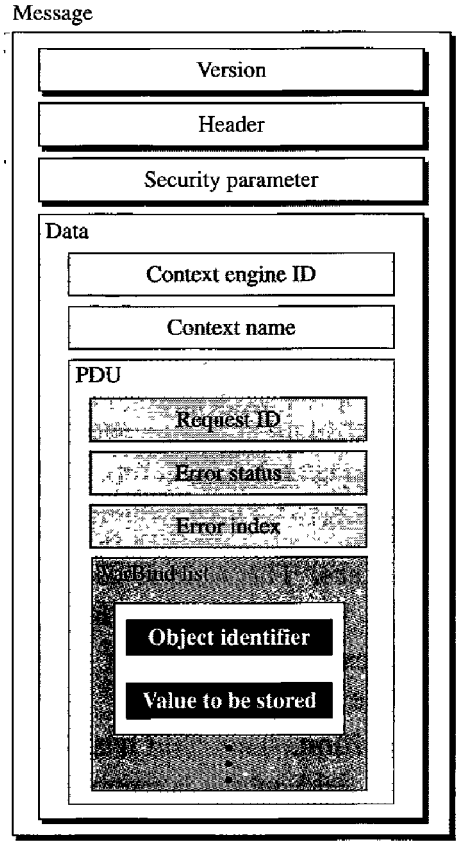



**Table 28.4** *Codes for SNMP messages*

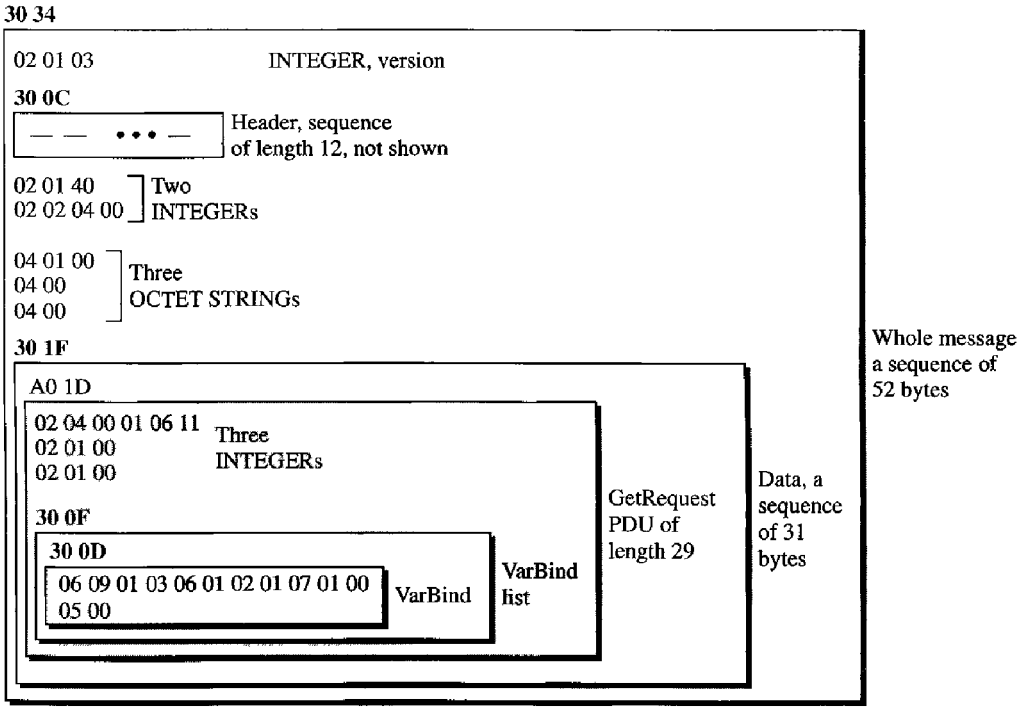
| *Data* | *Class* | *Format* | *Number* | *Whole Tag (Binary)* | *Whole Tag (Hex)* |
|---|---|---|---|---|---|
| GetRequest | 10 | 1 | 00000 | **10100000** | **A0** |
| GetNextRequest | 10 | 1 | 00001 | **10100001** | **A1** |
| Response | 10 | 1 | 00010 | **10100010** | **A2** |
| SetRequest | 10 | 1 | 00011 | **10100011** | **A3** |
| GetBulkRequest | 10 | 1 | 00101 | **10100101** | **A5** |
| InformRequest | 10 | 1 | 00110 | **10100110** | **A6** |
| Trap (SNMPv2) | 10 | 1 | 00111 | **10100111** | **A7** |
| Report | 10 | 1 | 01000 | **10101000** | **A8** |

### *Example 28.5*

In this example, a manager station (SNMP client) uses the GetRequest message to retrieve the number of UDP datagrams that a router has received.

There is only one VarBind entity. The corresponding MIB variable related to this information is udpInDatagrams with the object identifier 1.3.6.1.2.1.7.1.0. The manager wants to retrieve a value (not to store a value), so the value defines a null entity. Figure 28.23 shows the conceptual

**Figure 28.23** *Example 28.5*




view of the packet and the hierarchical nature of sequences. We have used white and colored boxes for the sequences and a gray one for the PDU.

The VarBind list has only one VarBind. The variable is of type 06 and length 09. The value is of type 05 and length 00. The whole VarBind is a sequence of length 0D (13). The VarBind list is also a sequence of length 0F (15). The GetRequest PDU is of length ID (29).

Now we have three OCTET STRINGs related to the security parameter, security model, and flags. Then we have two integers defining maximum size (1024) and message ID (64). The header is a sequence of length 12, which we left blank for simplicity. There is one integer, version (version 3). The whole message is a sequence of 52 bytes.

Figure 28.24 shows the actual message sent by the manager station (client) to the agent (server).

## UDP Ports

SNMP uses the services of UDP on two well-known ports, 161 and 162. The well-known port 161 is used by the server (agent), and the well-known port 162 is used by the client (manager).

The agent (server) issues a passive open on port 161. It then waits for a connection from a manager (client). A manager (client) issues an active open, using an ephemeral port. The request messages are sent from the client to the server, using the ephemeral port as the source port and the well-known port 161 as the destination port. The response messages are sent from the server to the client, using the well-known port 161 as the source port and the ephemeral port as the destination port.

The manager (client) issues a passive open on port 162. It then waits for a connection from an agent (server). Whenever it has a Trap message to send, an agent (server) issues an active open, using an ephemeral port. This connection is only one-way, from the server to the client (see Figure 28.25).

**Figure 28.24** *GetRequest message*

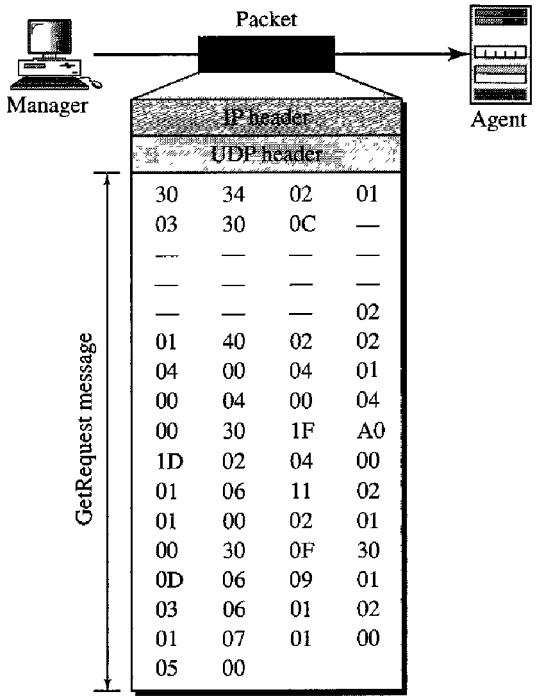
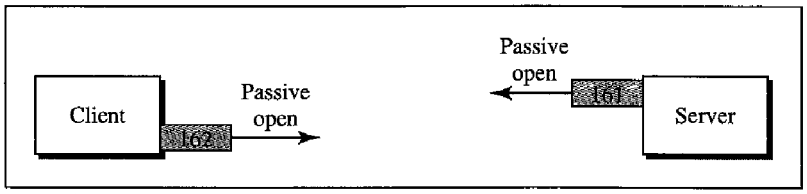
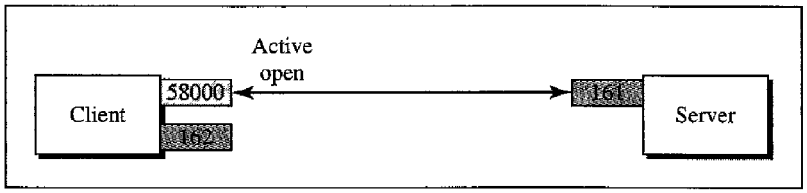


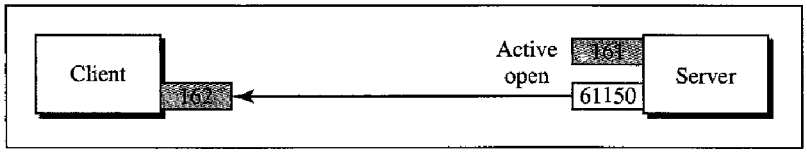

**Figure 28.25** *Port numbers for SNMP*




**a. Passive open by both client and server**




**b. Exchange of request and response messages**




**c. Server sends trap message**

The client/server mechanism in SNMP is different from other protocols. Here both the client and the server use well-known ports. In addition, both the client and the server are running infinitely. The reason is that request messages are initiated by a manager (client), but Trap messages are initiated by an agent (server).

### Security

The main difference between SNMPv3 and SNMPv2 is the enhanced security. SNMPv3 provides two types of security: general and specific. SNMPv3 provides message authentication, privacy, and manager authorization. We discuss these three aspects in Chapter 31. In addition, SNMPv3 allows a manager to remotely change the security configuration, which means that the manager does not have to be physically present at the manager station.

## 28.3 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [. . .] refer to the reference list at the end of the text.

### Books

SNMP is discussed in [MS01], Chapter 25 of [Ste94], Section 22.3 of [Sta04], and Chapter 39 of [Com04]. Network management is discussed in [Sub01].

### Sites

The following sites are related to topics discussed in this chapter.

- www.ietf.org/rfc.html Information about RFCs

### RFCs

The following RFCs are related to SNMP, MIB, and SMI:

1065, 1067, 1098, 1155, 1157, 1212, 1213, 1229, 1231, 1243, 1284, 1351, 1352, 1354, 1389, 1398, 1414, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1461, 1472, 1474, 1537, 1623, 1643, 1650, 1657, 1665, 1666, 1696, 1697, 1724, 1742, 1743, 1748, 1749

## 28.4 KEY TERMS

Abstract Syntax Notation 1 (ASN.1)
accounting management
agent
Basic Encoding Rules (BER)
configuration management
fault management
hardware documentation
lexicographic ordering
Management Information Base (MIB)
manager
network management
object identifier
performance management
security management

simple data type
Simple Network Management Protocol (SNMP)
structured data type
Structure of Management Information (SMI)
trap

## 28.5 SUMMARY

- The five areas comprising network management are configuration management, fault management, performance management, accounting management, and security management.
- Configuration management is concerned with the physical or logical changes of network entities. It includes the reconfiguration and documentation of hardware, software, and user accounts.
- Fault management is concerned with the proper operation of each network component. It can be reactive or proactive.
- Performance management is concerned with the monitoring and control of the network to ensure the network runs as efficiently as possible. It is quantified by measuring the capacity, traffic, throughput, and response time.
- Security management is concerned with controlling access to the network.
- Accounting management is concerned with the control of user access to network resources through charges.
- Simple Network Management Protocol (SNMP) is a framework for managing devices in an internet using the TCP/IP protocol suite.
- A manager, usually a host, controls and monitors a set of agents, usually routers.
- The manager is a host that runs the SNMP client program.
- The agent is a router or host that runs the SNMP server program.
- SNMP frees management tasks from both the physical characteristics of the managed devices and the underlying networking technology.
- SNMP uses the services of two other protocols: Structure of Management Information (SMI) and Management Information Base (MIB).
- SMI names objects, defines the type of data that can be stored in an object, and encodes the data.
- SMI objects are named according to a hierarchical tree structure.
- SMI data types are defined according to Abstract Syntax Notation 1 (ASN.1).
- SMI uses Basic Encoding Rules (BER) to encode data.
- MIB is a collection of groups of objects that can be managed by SNMP.
- MIB uses lexicographic ordering to manage its variables.
- SNMP functions in three ways:
  1. A manager can retrieve the value of an object defined in an agent.
  2. A manager can store a value in an object defined in an agent.
  3. An agent can send an alarm message to the manager.

- SNMP defines eight types of packets: GetRequest, GetNextRequest, SetRequest, GetBulkRequest, Trap, InformRequest, Response, and Report.
- SNMP uses the services of UDP on two well-known ports, 161 and 162.
- SNMPv3 has enhanced security features over previous versions.

## 28.6 PRACTICE SET

### Review Questions

1. Define network management.
2. List five functions of network management.
3. Define configuration management and its purpose.
4. List two subfunctions of configuration management.
5. Define fault management and its purpose.
6. List two subfunctions of fault management.
7. Define performance management and its purpose.
8. List four measurable quantities of performance management.
9. Define security management and its purpose.
10. Define account management and its purpose.

### Exercises

11. Show the encoding for INTEGER 1456.
12. Show the encoding for the OCTET STRING "Hello World."
13. Show the encoding for an arbitrary OCTET STRING of length 1000.
14. Show how the following record (sequence) is encoded.

| INTEGER | OCTET STRING | IP Address |
|---|---|---|
| 2345 | "COMPUTER" | 185.32.1.5 |

15. Show how the following record (sequence) is encoded.

| Time Tick | INTEGER | Object Id |
|---|---|---|
| 12000 | 14564 | 1.3.6.1.2.1.7 |

16. Show how the following array (sequence of) is encoded. Each element is an integer.

| |
|---|
| 2345 |
| 1236 |
| 122 |
| 1236 |

17. Show how the following array of records (sequence of sequence) is encoded.

| INTEGER | OCTET STRING | Counter |
|---|---|---|
| 2345 | "COMPUTER" | 345 |
| 1123 | "DISK" | 1430 |
| 3456 | "MONITOR" | 2313 |

18. Decode the following.
    a. 02 04 01 02 14 32
    b. 30 06 02 01 11 02 01 14
    c. 30 09 04 03 41 43 42 02 02 14 14
    d. 30 0A 40 04 23 51 62 71 02 02 14 12

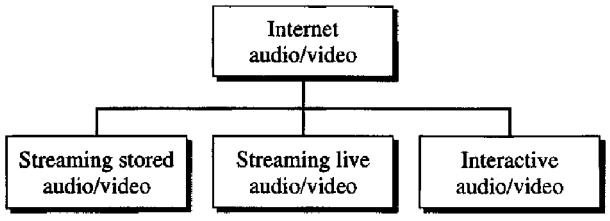
# CHAPTER 29

# *Multimedia*

Recent advances in technology have changed our use of audio and video. In the past, we listened to an audio broadcast through a radio and watched a video program broadcast through a TV. We used the telephone network to interactively communicate with another party. But times have changed. People want to use the Internet not only for text and image communications, but also for audio and video services. In this chapter, we concentrate on applications that use the Internet for audio and video services.

We can divide audio and video services into three broad categories: **streaming stored audio/video, streaming live audio/video,** and **interactive audio/video,** as shown in Figure 29.1. Streaming means a user can listen to (or watch) the file after the downloading has started.

**Figure 29.1** *Internet audio/video*




In the first category, streaming stored audio/video, the files are compressed and stored on a server. A client downloads the files through the Internet. This is sometimes referred to as **on-demand audio/video.** Examples of stored audio files are songs, symphonies, books on tape, and famous lectures. Examples of stored video files are movies, TV shows, and music video clips.

**Streaming stored audio/video refers to on-demand requests for compressed audio/video files.**

In the second category, streaming live audio/video, a user listens to broadcast audio and video through the Internet. A good example of this type of application is the Internet radio. Some radio stations broadcast their programs only on the Internet; many broadcast them both on the Internet and on the air. Internet TV is not popular yet, but many people believe that TV stations will broadcast their programs on the Internet in the future.

**Streaming live audio/video refers to the broadcasting of radio and TV programs through the Internet.**

In the third category, interactive audio/video, people use the Internet to interactively communicate with one another. A good example of this application is Internet telephony and Internet teleconferencing.

**Interactive audio/video refers to the use of the Internet for interactive audio/video applications.**

We will discuss these three applications in this chapter, but first we need to discuss some other issues related to audio/video: digitizing audio and video and compressing audio and video.

## 29.1 DIGITIZING AUDIO AND VIDEO

Before audio or video signals can be sent on the Internet, they need to be digitized. We discuss audio and video separately.

### Digitizing Audio

When sound is fed into a microphone, an electronic analog signal is generated which represents the sound amplitude as a function of time. The signal is called an *analog audio signal*. An analog signal, such as audio, can be digitized to produce a digital signal. According to the Nyquist theorem, if the highest frequency of the signal is $f$, we need to sample the signal $2f$ times per second. There are other methods for digitizing an audio signal, but the principle is the same.

Voice is sampled at 8000 samples per second with 8 bits per sample. This results in a digital signal of 64 kbps. Music is sampled at 44,100 samples per second with 16 bits per sample. This results in a digital signal of 705.6 kbps for monaural and 1.411 Mbps for stereo.

### Digitizing Video

A video consists of a sequence of frames. If the frames are displayed on the screen fast enough, we get an impression of motion. The reason is that our eyes cannot distinguish the rapidly flashing frames as individual ones. There is no standard number of frames per second; in North America 25 frames per second is common. However, to avoid a

condition known as flickering, a frame needs to be refreshed. The TV industry repaints each frame twice. This means 50 frames need to be sent, or if there is memory at the sender site, 25 frames with each frame repainted from the memory.

Each frame is divided into small grids, called picture elements or **pixels.** For black-and-white TV, each 8-bit pixel represents one of 256 different gray levels. For a color TV, each pixel is 24 bits, with 8 bits for each primary color (red, green, and blue).

We can calculate the number of bits in 1 s for a specific resolution. In the lowest resolution a color frame is made of $1024 \times 768$ pixels. This means that we need

$$2 \times 25 \times 1024 \times 768 \times 24 = 944 \text{ Mbps}$$

This data rate needs a very high data rate technology such as SONET. To send video using lower-rate technologies, we need to compress the video.

***Compression* is needed to send video over the Internet.**

## 29.2 AUDIO AND VIDEO COMPRESSION

To send audio or video over the Internet requires **compression.** In this section, we discuss audio compression first and then video compression.

### Audio Compression

Audio compression can be used for speech or music. For speech, we need to compress a 64-kHz digitized signal; for music, we need to compress a 1.411-MHz signal. Two categories of techniques are used for audio compression: predictive encoding and perceptual encoding.

#### *Predictive Encoding*

In **predictive encoding,** the differences between the samples are encoded instead of encoding all the sampled values. This type of compression is normally used for speech. Several standards have been defined such as GSM (13 kbps), G.729 (8 kbps), and G.723.3 (6.4 or 5.3 kbps). Detailed discussions of these techniques are beyond the scope of this book.

#### *Perceptual Encoding: MP3*

The most common compression technique that is used to create CD-quality audio is based on the **perceptual encoding** technique. As we mentioned before, this type of audio needs at least 1.411 Mbps; this cannot be sent over the Internet without compression. **MP3** (MPEG audio layer 3), a part of the MPEG standard (discussed in the video compression section), uses this technique.

Perceptual encoding is based on the science of psychoacoustics, which is the study of how people perceive sound. The idea is based on flaws in our auditory system: Some sounds can mask other sounds. Masking can happen in frequency and time. In

**frequency masking,** a loud sound in a frequency range can partially or totally mask a softer sound in another frequency range. For example, we cannot hear what our dance partner says in a room where a loud heavy metal band is performing. In **temporal masking,** a loud sound can numb our ears for a short time even after the sound has stopped.

MP3 uses these two phenomena, frequency and temporal masking, to compress audio signals. The technique analyzes and divides the spectrum into several groups. Zero bits are allocated to the frequency ranges that are totally masked. A small number of bits are allocated to the frequency ranges that are partially masked. A larger number of bits are allocated to the frequency ranges that are not masked.

MP3 produces three data rates: 96 kbps, 128 kbps, and 160 kbps. The rate is based on the range of the frequencies in the original analog audio.

## Video Compression

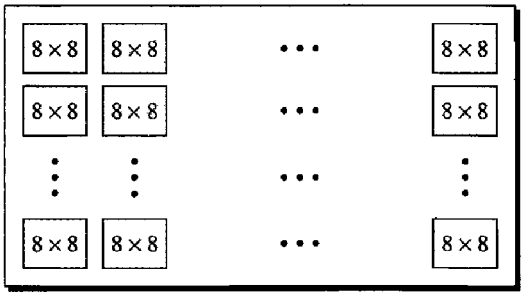
As we mentioned before, video is composed of multiple frames. Each frame is one image. We can compress video by first compressing images. Two standards are prevalent in the market. **Joint Photographic Experts Group (JPEG)** is used to compress images. **Moving Picture Experts Group (MPEG)** is used to compress video. We briefly discuss JPEG and then MPEG.

### *Image Compression: JPEG*

As we discussed previously, if the picture is not in color (gray scale), each pixel can be represented by an 8-bit integer (256 levels). If the picture is in color, each pixel can be represented by 24 bits ($3 \times 8$ bits), with each 8 bits representing red, blue, or green (RBG). To simplify the discussion, we concentrate on a gray scale picture.

In JPEG, a gray scale picture is divided into blocks of $8 \times 8$ pixels (see Figure 29.2).

**Figure 29.2** *JPEG gray scale*

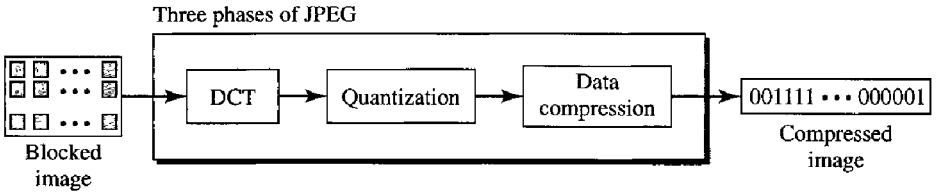



The purpose of dividing the picture into blocks is to decrease the number of calculations because, as you will see shortly, the number of mathematical operations for each picture is the square of the number of units.

The whole idea of JPEG is to change the picture into a linear (vector) set of numbers that reveals the redundancies. The redundancies (lack of changes) can then be removed

by using one of the text compression methods. A simplified scheme of the process is shown in Figure 29.3.

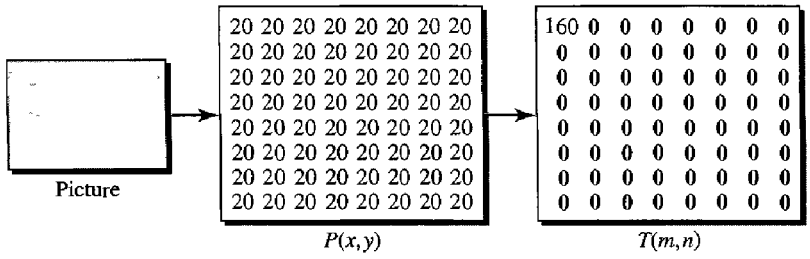
**Figure 29.3** *JPEG process*




**Discrete Cosine Transform (DCT)** In this step, each block of 64 pixels goes through a transformation called the **discrete cosine transform (DCT).** The transformation changes the 64 values so that the relative relationships between pixels are kept but the redundancies are revealed. We do not give the formula here, but we do show the results of the transformation for three cases.

**Case 1** In this case, we have a block of uniform gray, and the value of each pixel is 20. When we do the transformations, we get a nonzero value for the first element (upper left corner); the rest of the pixels have a value of 0. The value of $T(0,0)$ is the average (multiplied by a constant) of the $P(x,y)$ values and is called the *dc value* (direct current, borrowed from electrical engineering). The rest of the values, called *ac values*, in $T(m,n)$ represent changes in the pixel values. But because there are no changes, the rest of the values are 0s (see Figure 29.4).

**Figure 29.4** *Case 1: uniform gray scale*




**Case 2** In the second case, we have a block with two different uniform gray scale sections. There is a sharp change in the values of the pixels (from 20 to 50). When we do the transformations, we get a dc value as well as nonzero ac values. However, there are only a few nonzero values clustered around the dc value. Most of the values are 0 (see Figure 29.5).

**Case 3** In the third case, we have a block that changes gradually. That is, there is no sharp change between the values of neighboring pixels. When we do the transformations, we get a dc value, with many nonzero ac values also (Figure 29.6).

**Figure 29.5** *Case 2: two sections*

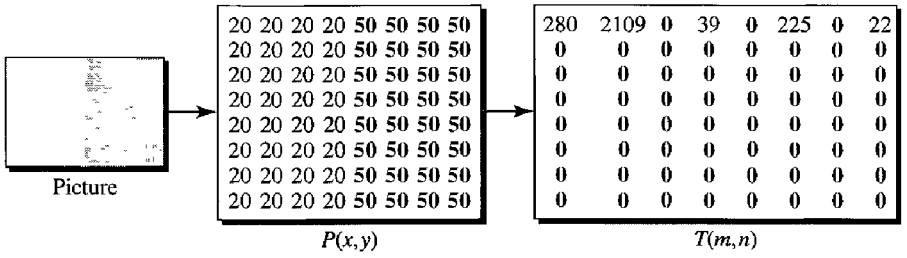
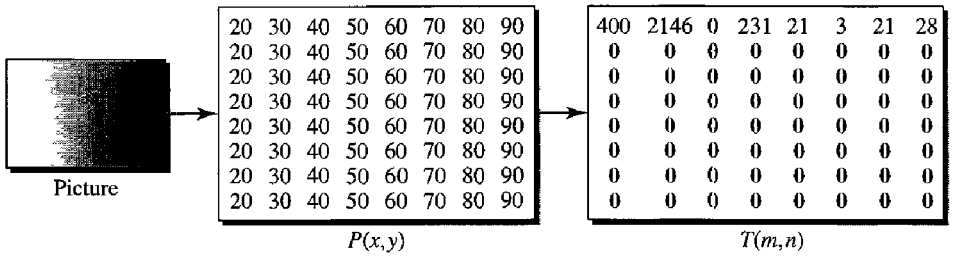



**Figure 29.6** *Case 3: gradient gray scale*



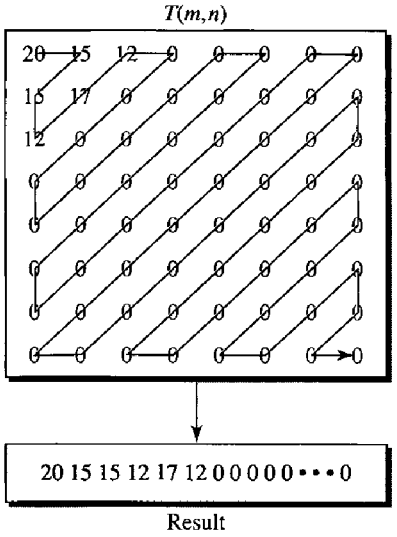

From Figures 29.4, 29.5, and 29.6, we can state the following:

- The transformation creates table $T$ from table $P$.
- The dc value is the average value (multiplied by a constant) of the pixels.
- The ac values are the changes.
- Lack of changes in neighboring pixels creates 0s.

**Quantization** After the $T$ table is created, the values are quantized to reduce the number of bits needed for encoding. Previously in **quantization,** we dropped the fraction from each value and kept the integer part. Here, we divide the number by a constant and then drop the fraction. This reduces the required number of bits even more. In most implementations, a quantizing table ($8 \times 8$) defines how to quantize each value. The divisor depends on the position of the value in the $T$ table. This is done to optimize the number of bits and the number of 0s for each particular application. Note that the only phase in the process that is not reversible is the quantizing phase. We lose some information here that is not recoverable. As a matter of fact, the only reason that JPEG is called *lossy compression* is because of this quantization phase.

**Compression** After quantization, the values are read from the table, and redundant 0s are removed. However, to cluster the 0s together, the table is read diagonally in a zigzag fashion rather than row by row or column by column. The reason is that if the picture changes smoothly, the bottom right corner of the $T$ table is all 0s. Figure 29.7 shows the process.

**Figure 29.7** *Reading the table*




*Video Compression: MPEG*

The Moving Picture Experts Group method is used to compress video. In principle, a motion picture is a rapid flow of a set of frames, where each frame is an image. In other words, a frame is a spatial combination of pixels, and a video is a temporal combination of frames that are sent one after another. Compressing video, then, means spatially compressing each frame and temporally compressing a set of frames.

**Spatial Compression** The **spatial compression** of each frame is done with JPEG (or a modification of it). Each frame is a picture that can be independently compressed.

**Temporal Compression** In **temporal compression,** redundant frames are removed. When we watch television, we receive 50 frames per second. However, most of the consecutive frames are almost the same. For example, when someone is talking, most of the frame is the same as the previous one except for the segment of the frame around the lips, which changes from one frame to another.

To temporally compress data, the MPEG method first divides frames into three categories: I-frames, P-frames, and B-frames.

- **I-frames.** An **intracoded frame (I-frame)** is an independent frame that is not related to any other frame (not to the frame sent before or to the frame sent after). They are present at regular intervals (e.g., every ninth frame is an I-frame). An I-frame must appear periodically to handle some sudden change in the frame that the previous and following frames cannot show. Also, when a video is broadcast, a viewer may tune in at any time. If there is only one I-frame at the beginning of the broadcast, the viewer who tunes in late will not receive a complete picture. I-frames are independent of other frames and cannot be constructed from other frames.
- **P-frames.** A **predicted frame (P-frame)** is related to the preceding I-frame or P-frame. In other words, each P-frame contains only the changes from the preceding

frame. The changes, however, cannot cover a big segment. For example, for a fast-moving object, the new changes may not be recorded in a P-frame. P-frames can be constructed only from previous I- or P-frames. P-frames carry much less information than other frame types and carry even fewer bits after compression.

- **B-frames.** A **bidirectional frame (B-frame)** is related to the preceding and following I-frame or P-frame. In other words, each B-frame is relative to the past and the future. Note that a B-frame is never related to another B-frame.

Figure 29.8 shows a sample sequence of frames.

**Figure 29.8** *MPEG frames*

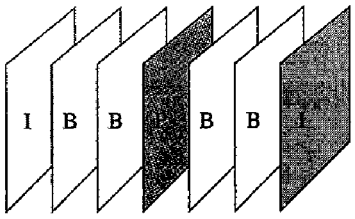
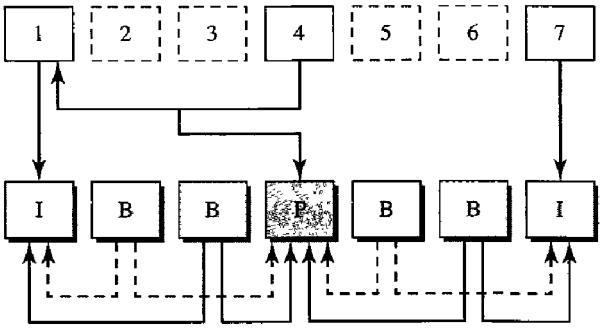



Figure 29.9 shows how I-, P-, and B-frames are constructed from a series of seven frames.

**Figure 29.9** *MPEG frame construction*




MPEG has gone through two versions. MPEG1 was designed for a CD-ROM with a data rate of 1.5 Mbps. MPEG2 was designed for high-quality DVD with a data rate of 3 to 6 Mbps.

## 29.3 STREAMING STORED AUDIO/VIDEO

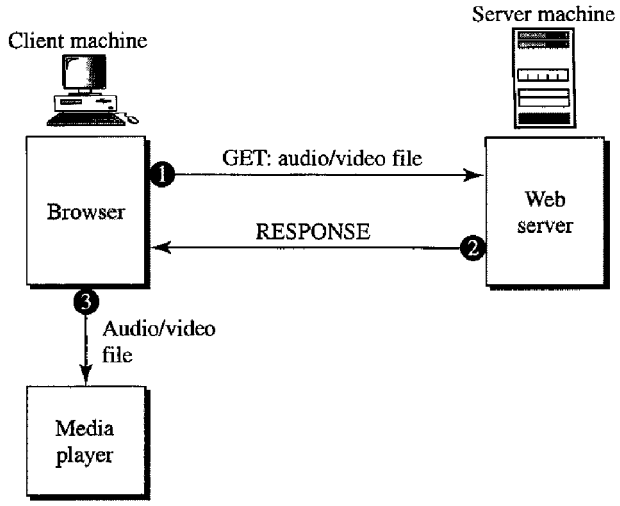
Now that we have discussed digitizing and compressing audio/video, we turn our attention to specific applications. The first is streaming stored audio and video. Downloading these types of files from a Web server can be different from downloading other

types of files. To understand the concept, let us discuss four approaches, each with a different complexity.

## First Approach: Using a Web Server

A compressed audio/video file can be downloaded as a text file. The client (browser) can use the services of HTTP and send a GET message to download the file. The Web server can send the compressed file to the browser. The browser can then use a help application, normally called a **media player,** to play the file. Figure 29.10 shows this approach.

**Figure 29.10** *Using a Web server*



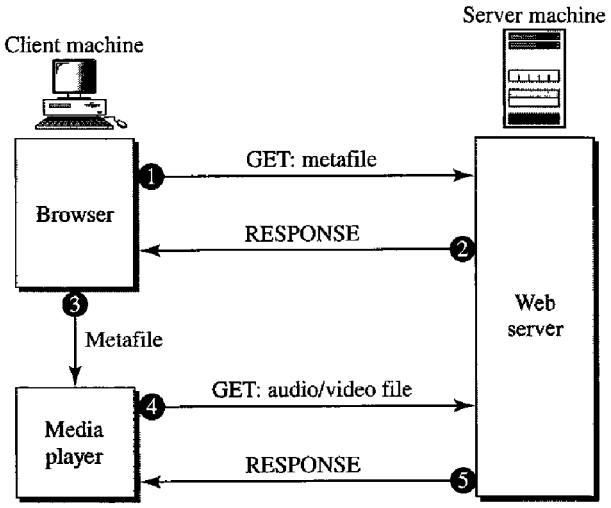

This approach is very simple and does not involve *streaming*. However, it has a drawback. An audio/video file is usually large even after compression. An audio file may contain tens of megabits, and a video file may contain hundreds of megabits. In this approach, the file needs to download completely before it can be played. Using contemporary data rates, the user needs some seconds or tens of seconds before the file can be played.

## Second Approach: Using a Web Server with Metafile

In another approach, the media player is directly connected to the Web server for downloading the audio/video file. The Web server stores two files: the actual audio/video file and a **metafile** that holds information about the audio/video file. Figure 29.11 shows the steps in this approach.

1. The HTTP client accesses the Web server by using the GET message.
2. The information about the metafile comes in the response.
3. The metafile is passed to the media player.
4. The media player uses the URL in the metafile to access the audio/video file.
5. The Web server responds.

**Figure 29.11** *Using a Web server with a metafile*

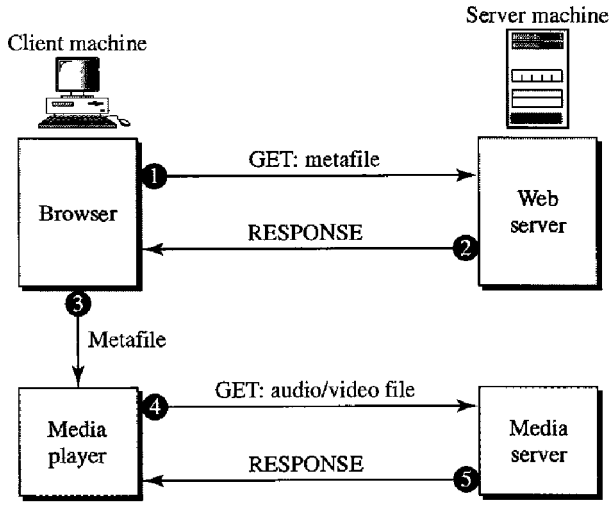



## Third Approach: Using a Media Server

The problem with the second approach is that the browser and the media player both use the services of HTTP. HTTP is designed to run over TCP. This is appropriate for retrieving the metafile, but not for retrieving the audio/video file. The reason is that TCP retransmits a lost or damaged segment, which is counter to the philosophy of streaming. We need to dismiss TCP and its error control; we need to use UDP. However, HTTP, which accesses the Web server, and the Web server itself are designed for TCP; we need another server, a **media server.** Figure 29.12 shows the concept.

1. The HTTP client accesses the Web server by using a GET message.
2. The information about the metafile comes in the response.

**Figure 29.12** *Using a media server*

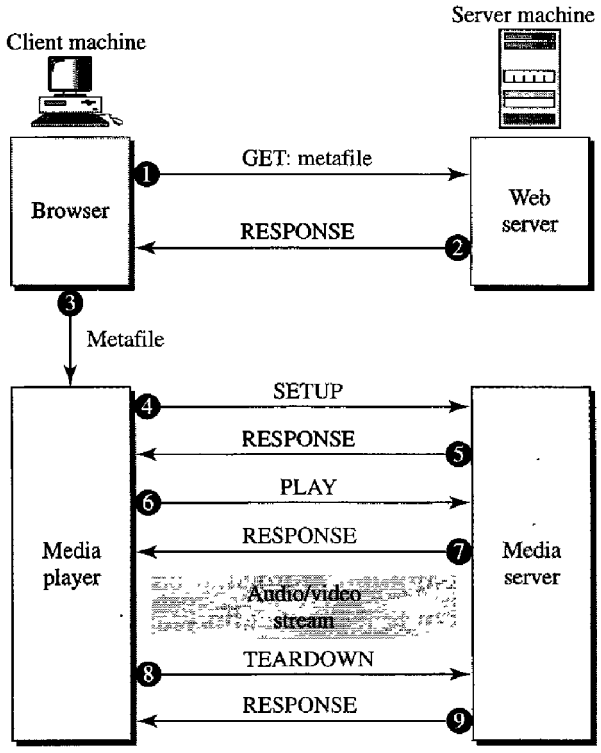


3. The metafile is passed to the media player.
4. The media player uses the URL in the metafile to access the media server to download the file. Downloading can take place by any protocol that uses UDP.
5. The media server responds.

## Fourth Approach: Using a Media Server and RTSP

The **Real-Time Streaming Protocol (RTSP)** is a control protocol designed to add more functionalities to the streaming process. Using RTSP, we can control the playing of audio/video. RTSP is an out-of-band control protocol that is similar to the second connection in FTP. Figure 29.13 shows a media server and RTSP.

**Figure 29.13** *Using a media server and RTSP*




1. The HTTP client accesses the Web server by using a GET message.
2. The information about the metafile comes in the response.
3. The metafile is passed to the media player.
4. The media player sends a SETUP message to create a connection with the media server.
5. The media server responds.
6. The media player sends a PLAY message to start playing (downloading).
7. The audio/video file is downloaded by using another protocol that runs over UDP.

8. The connection is broken by using the TEARDOWN message.
9. The media server responds.

The media player can send other types of messages. For example, a PAUSE message temporarily stops the downloading; downloading can be resumed with a PLAY message.

## 29.4 STREAMING LIVE AUDIO/VIDEO

Streaming live audio/video is similar to the broadcasting of audio and video by radio and TV stations. Instead of broadcasting to the air, the stations broadcast through the Internet. There are several similarities between streaming stored audio/video and streaming live audio/video. They are both sensitive to delay; neither can accept retransmission. However, there is a difference. In the first application, the communication is unicast and on-demand. In the second, the communication is multicast and live. Live streaming is better suited to the multicast services of IP and the use of protocols such as UDP and RTP (discussed later). However, presently, live streaming is still using TCP and multiple unicasting instead of multicasting. There is still much progress to be made in this area.

## 29.5 REAL-TIME INTERACTIVE AUDIO/VIDEO

In real-time interactive audio/video, people communicate with one another in real time. The Internet phone or voice over IP is an example of this type of application. Video conferencing is another example that allows people to communicate visually and orally.

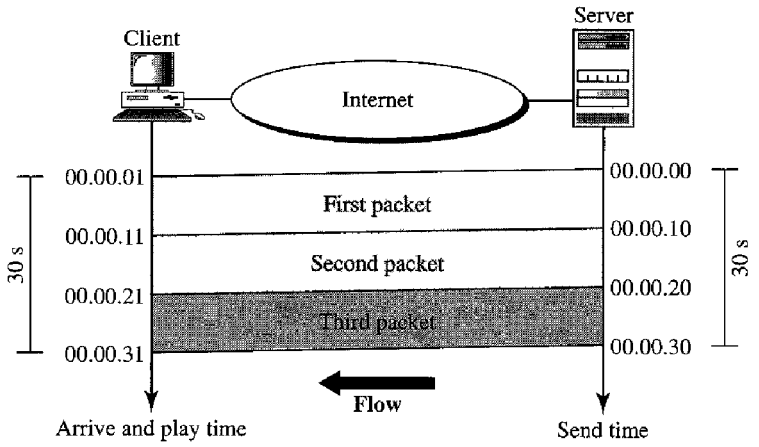
### Characteristics

Before addressing the protocols used in this class of applications, we discuss some characteristics of real-time audio/video communication.

#### *Time Relationship*

Real-time data on a packet-switched network require the preservation of the time relationship between packets of a session. For example, let us assume that a real-time video server creates live video images and sends them online. The video is digitized and packetized. There are only three packets, and each packet holds 10 s of video information. The first packet starts at 00:00:00, the second packet starts at 00:00:10, and the third packet starts at 00:00:20. Also imagine that it takes 1 s (an exaggeration for simplicity) for each packet to reach the destination (equal delay). The receiver can play back the first packet at 00:00:01, the second packet at 00:00:11, and the third packet at 00:00:21. Although there is a 1-s time difference between what the server sends and what the client sees on the computer screen, the action is happening in real time. The time relationship between the packets is preserved. The 1-s delay is not important. Figure 29.14 shows the idea.

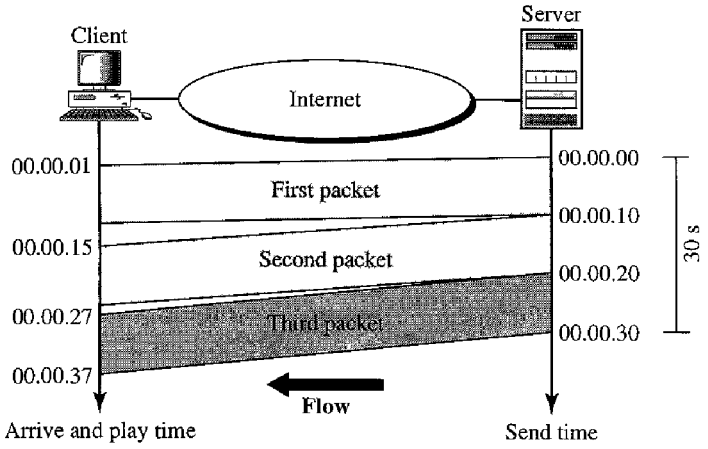
**Figure 29.14** *Time relationship*




But what happens if the packets arrive with different delays? For example, say the first packet arrives at 00:00:01 (1-s delay), the second arrives at 00:00:15 (5-s delay), and the third arrives at 00:00:27 (7-s delay). If the receiver starts playing the first packet at 00:00:01, it will finish at 00:00:11. However, the next packet has not yet arrived; it arrives 4 s later. There is a gap between the first and second packets and between the second and the third as the video is viewed at the remote site. This phenomenon is called **jitter.** Figure 29.15 shows the situation.

**Jitter is introduced in real-time data by the delay between packets.**

**Figure 29.15** *Jitter*

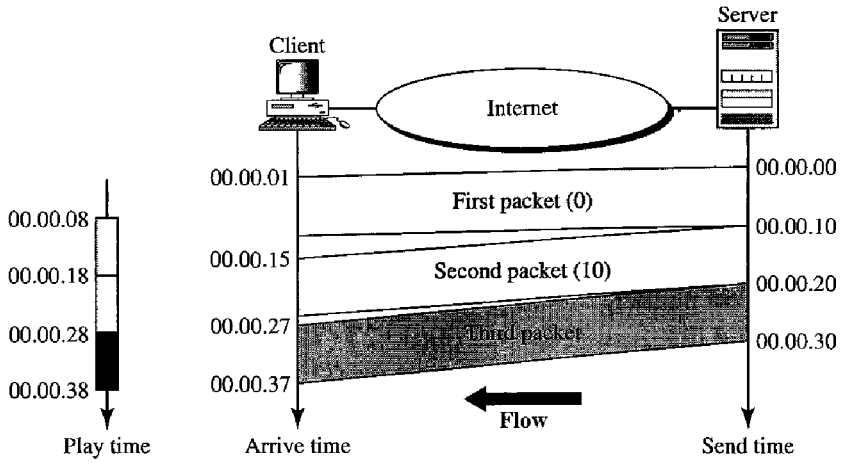



### *Timestamp*

One solution to jitter is the use of a **timestamp.** If each packet has a timestamp that shows the time it was produced relative to the first (or previous) packet, then the receiver

can add this time to the time at which it starts the playback. In other words, the receiver knows when each packet is to be played. Imagine the first packet in the previous example has a timestamp of 0, the second has a timestamp of 10, and the third has a timestamp of 20. If the receiver starts playing back the first packet at 00:00:08, the second will be played at 00:00:18 and the third at 00:00:28. There are no gaps between the packets. Figure 29.16 shows the situation.

**Figure 29.16** *Timestamp*




**To prevent jitter, we can time-stamp the packets and separate the arrival time from the playback time.**

### *Playback Buffer*

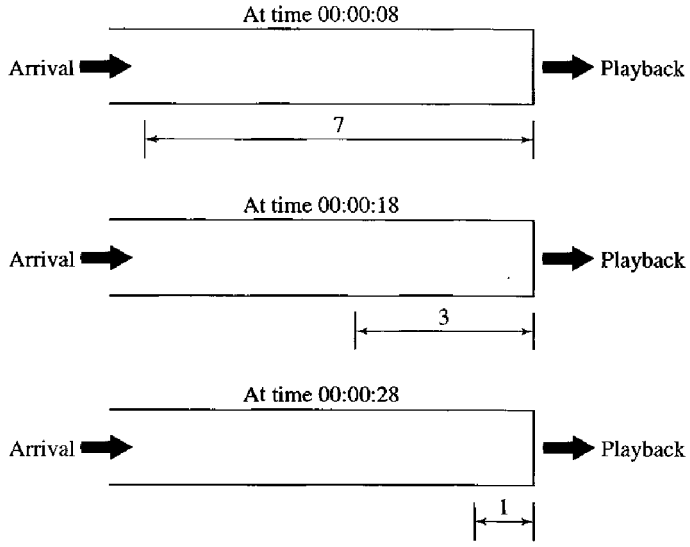
To be able to separate the arrival time from the playback time, we need a buffer to store the data until they are played back. The buffer is referred to as a **playback buffer.** When a session begins (the first bit of the first packet arrives), the receiver delays playing the data until a threshold is reached. In the previous example, the first bit of the first packet arrives at 00:00:01; the threshold is 7 s, and the playback time is 00:00:08. The threshold is measured in time units of data. The replay does not start until the time units of data are equal to the threshold value.

Data are stored in the buffer at a possibly variable rate, but they are extracted and played back at a fixed rate. Note that the amount of data in the buffer shrinks or expands, but as long as the delay is less than the time to play back the threshold amount of data, there is no jitter. Figure 29.17 shows the buffer at different times for our example.

### *Ordering*

In addition to time relationship information and timestamps for **real-time traffic,** one more feature is needed. We need a *sequence number* for each packet. The timestamp alone cannot inform the receiver if a packet is lost. For example, suppose the timestamps

**Figure 29.17** *Playback buffer*




**A playback buffer is required for real-time traffic.**

are 0, 10, and 20. If the second packet is lost, the receiver receives just two packets with timestamps 0 and 20. The receiver assumes that the packet with timestamp 20 is the second packet, produced 20 s after the first. The receiver has no way of knowing that the second packet has actually been lost. A sequence number to order the packets is needed to handle this situation.

**A sequence number on each packet is required for real-time traffic.**

### *Multicasting*

Multimedia play a primary role in audio and video conferencing. The traffic can be heavy, and the data are distributed by using **multicasting** methods. Conferencing requires two-way communication between receivers and senders.

**Real-time traffic needs the support of multicasting.**

### *Translation*

Sometimes real-time traffic needs **translation.** A translator is a computer that can change the format of a high-bandwidth video signal to a lower-quality narrow-bandwidth signal. This is needed, for example, for a source creating a high-quality video signal at 5 Mbps and sending to a recipient having a bandwidth of less than 1 Mbps. To receive the signal, a translator is needed to decode the signal and encode it again at a lower quality that needs less bandwidth.

**Translation means changing the encoding of a payload to a lower quality to match the bandwidth of the receiving network.**

### *Mixing*

If there is more than one source that can send data at the same time (as in a video or audio conference), the traffic is made of multiple streams. To converge the traffic to one stream, data from different sources can be mixed. A **mixer** mathematically adds signals coming from different sources to create one single signal.

**Mixing means combining several streams of traffic into one stream.**

### *Support from Transport Layer Protocol*

The procedures mentioned in the previous sections can be implemented in the application layer. However, they are so common in real-time applications that implementation in the transport layer protocol is preferable. Let's see which of the existing transport layers is suitable for this type of traffic.

TCP is not suitable for interactive traffic. It has no provision for time-stamping, and it does not support multicasting. However, it does provide ordering (sequence numbers). One feature of TCP that makes it particularly unsuitable for interactive traffic is its error control mechanism. In interactive traffic, we cannot allow the retransmission of a lost or corrupted packet. If a packet is lost or corrupted in interactive traffic, it must be ignored. Retransmission upsets the whole idea of time-stamping and playback. Today there is so much redundancy in audio and video signals (even with compression) that we can simply ignore a lost packet. The listener or viewer at the remote site may not even notice it.

**TCP, with all its sophistication, is not suitable for interactive multimedia traffic because we cannot allow retransmission of packets.**

UDP is more suitable for interactive multimedia traffic. UDP supports multicasting and has no retransmission strategy. However, UDP has no provision for time-stamping, sequencing, or mixing. A new transport protocol, Real-time Transport Protocol (RTP), provides these missing features.

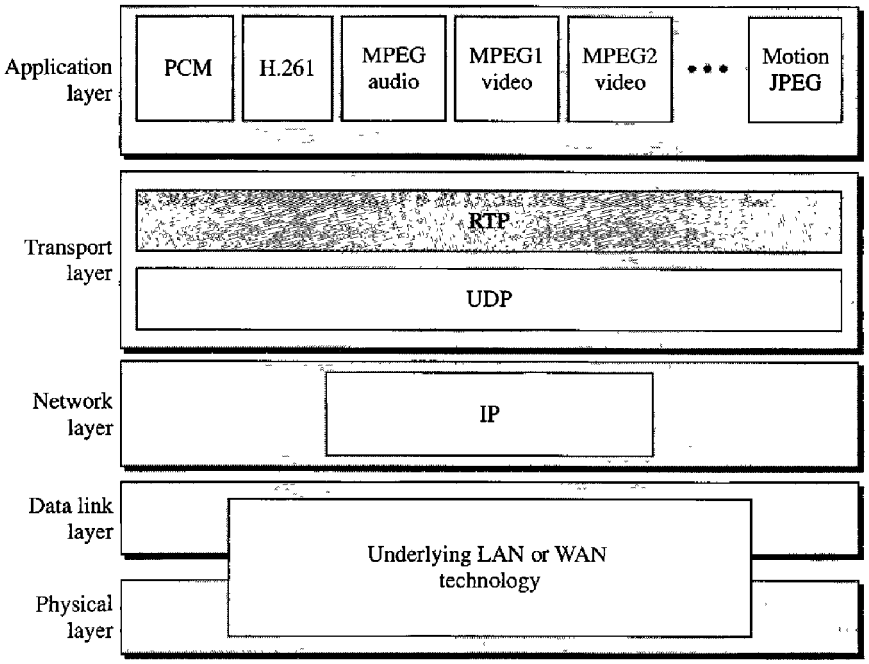
**UDP is more suitable than TCP for interactive traffic. However, we need the services of RTP, another transport layer protocol, to make up for the deficiencies of UDP.**

## 29.6 RTP

**Real-time Transport Protocol (RTP)** is the protocol designed to handle real-time traffic on the Internet. RTP does not have a delivery mechanism (multicasting, port numbers, and so on); it must be used with UDP. RTP stands between UDP and the application

program. The main contributions of RTP are time-stamping, sequencing, and mixing facilities. Figure 29.18 shows the position of RTP in the protocol suite.

**Figure 29.18** *RTP*

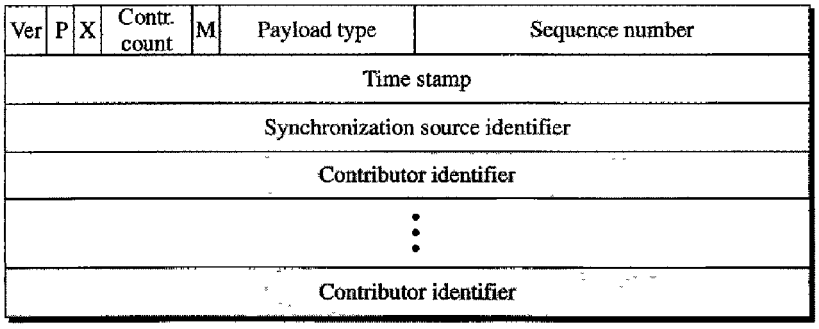



## RTP Packet Format

Figure 29.19 shows the format of the RTP packet header. The format is very simple and general enough to cover all real-time applications. An application that needs more information adds it to the beginning of its payload. A description of each field follows.

- **Ver.** This 2-bit field defines the version number. The current version is 2.

**Figure 29.19** *RTP packet header format*



- **P.** This 1-bit field, if set to 1, indicates the presence of padding at the end of the packet. In this case, the value of the last byte in the padding defines the length of the padding. Padding is the norm if a packet is encrypted. There is no padding if the value of the P field is 0.
- **X.** This 1-bit field, if set to 1, indicates an extra extension header between the basic header and the data. There is no extra extension header if the value of this field is 0.
- **Contributor count.** This 4-bit field indicates the number of contributors. Note that we can have a maximum of 15 contributors because a 4-bit field only allows a number between 0 and 15.
- **M.** This 1-bit field is a marker used by the application to indicate, for example, the end of its data.
- **Payload type.** This 7-bit field indicates the type of the payload. Several payload types have been defined so far. We list some common applications in Table 29.1. A discussion of the types is beyond the scope of this book.

**Table 29.1** *Payload types*

| *Type* | *Application* | *Type* | *Application* | *Type* | *Application* |
|---|---|---|---|---|---|
| 0 | PCMμ Audio | 7 | LPC audio | 15 | G728 audio |
| 1 | 1016 | 8 | PCMA audio | 26 | Motion JPEG |
| 2 | G721 audio | 9 | G722 audio | 31 | H.261 |
| 3 | GSM audio | 10–11 | L16 audio | 32 | MPEG1 video |
| 5–6 | DV14 audio | 14 | MPEG audio | 33 | MPEG2 video |

- **Sequence number.** This field is 16 bits in length. It is used to number the RTP packets. The sequence number of the first packet is chosen randomly; it is incremented by 1 for each subsequent packet. The sequence number is used by the receiver to detect lost or out-of-order packets.
- **Timestamp.** This is a 32-bit field that indicates the time relationship between packets. The timestamp for the first packet is a random number. For each succeeding packet, the value is the sum of the preceding timestamp plus the time the first byte is produced (sampled). The value of the clock tick depends on the application. For example, audio applications normally generate chunks of 160 bytes; the clock tick for this application is 160. The timestamp for this application increases 160 for each RTP packet.
- **Synchronization source identifier.** If there is only one source, this 32-bit field defines the source. However, if there are several sources, the mixer is the synchronization source and the other sources are contributors. The value of the source identifier is a random number chosen by the source. The protocol provides a strategy in case of conflict (two sources start with the same sequence number).
- **Contributor identifier.** Each of these 32-bit identifiers (a maximum of 15) defines a source. When there is more than one source in a session, the mixer is the synchronization source and the remaining sources are the contributors.

### UDP Port

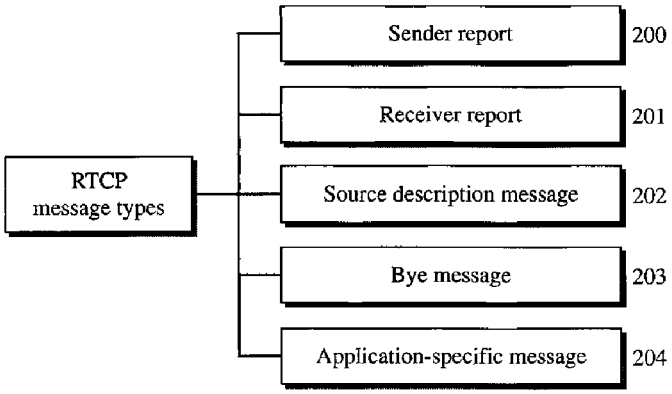
Although RTP is itself a transport layer protocol, the RTP packet is not encapsulated directly in an IP datagram. Instead, RTP is treated as an application program and is encapsulated in a UDP user datagram. However, unlike other application programs, no well-known port is assigned to RTP. The port can be selected on demand with only one restriction: The port number must be an even number. The next number (an odd number) is used by the companion of RTP, Real-time Transport Control Protocol (RTCP).

**RTP uses a temporary even-numbered UDP port.**

## 29.7 RTCP

RTP allows only one type of message, one that carries data from the source to the destination. In many cases, there is a need for other messages in a session. These messages control the flow and quality of data and allow the recipient to send feedback to the source or sources. **Real-time Transport Control Protocol (RTCP)** is a protocol designed for this purpose. RTCP has five types of messages, as shown in Figure 29.20. The number next to each box defines the type of the message.

**Figure 29.20** *RTCP message types*




### Sender Report

The sender report is sent periodically by the active senders in a conference to report transmission and reception statistics for all RTP packets sent during the interval. The sender report includes an absolute timestamp, which is the number of seconds elapsed since midnight on January 1, 1970. The absolute timestamp allows the receiver to synchronize different RTP messages. It is particularly important when both audio and video are transmitted (audio and video transmissions use separate relative timestamps).

### Receiver Report

The receiver report is for passive participants, those that do not send RTP packets. The report informs the sender and other receivers about the quality of service.

### Source Description Message

The source periodically sends a source description message to give additional information about itself. This information can be the name, e-mail address, telephone number, and address of the owner or controller of the source.

### Bye Message

A source sends a bye message to shut down a stream. It allows the source to announce that it is leaving the conference. Although other sources can detect the absence of a source, this message is a direct announcement. It is also very useful to a mixer.

### Application-Specific Message

The application-specific message is a packet for an application that wants to use new applications (not defined in the standard). It allows the definition of a new message type.

### UDP Port

RTCP, like RTP, does not use a well-known UDP port. It uses a temporary port. The UDP port chosen must be the number immediately following the UDP port selected for RTP. It must be an odd-numbered port.

---

**RTCP uses an odd-numbered UDP port number that follows the port number selected for RTP.**

---

## 29.8 VOICE OVER IP

Let us concentrate on one real-time interactive audio/video application: **voice over IP,** or Internet telephony. The idea is to use the Internet as a telephone network with some additional capabilities. Instead of communicating over a circuit-switched network, this application allows communication between two parties over the packet-switched Internet. Two protocols have been designed to handle this type of communication: SIP and H.323. We briefly discuss both.

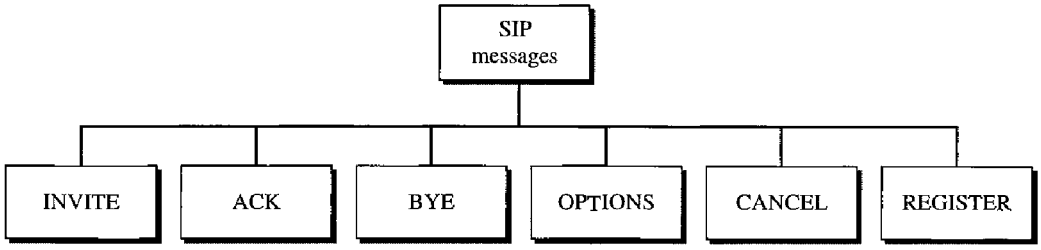
### SIP

The **Session Initiation Protocol (SIP)** was designed by IETF. It is an application layer protocol that establishes, manages, and terminates a multimedia session (call). It can be used to create two-party, multiparty, or multicast sessions. SIP is designed to be independent of the underlying transport layer; it can run on UDP, TCP, or SCTP.

### *Messages*

SIP is a text-based protocol, as is HTTP. SIP, like HTTP, uses messages. Six messages are defined, as shown in Figure 29.21.

**Figure 29.21** *SIP messages*



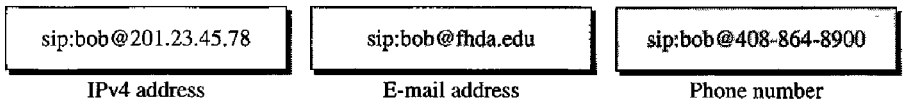

Each message has a header and a body. The header consists of several lines that describe the structure of the message, caller's capability, media type, and so on. We give a brief description of each message. Then we show their applications in a simple session.

The caller initializes a session with the INVITE message. After the callee answers the call, the caller sends an ACK message for confirmation. The BYE message terminates a session. The OPTIONS message queries a machine about its capabilities. The CANCEL message cancels an already started initialization process. The REGISTER message makes a connection when the callee is not available.

### *Addresses*

In a regular telephone communication, a telephone number identifies the sender, and another telephone number identifies the receiver. SIP is very flexible. In SIP, an e-mail address, an IP address, a telephone number, and other types of addresses can be used to identify the sender and receiver. However, the address needs to be in SIP format (also called *scheme*). Figure 29.22 shows some common formats.

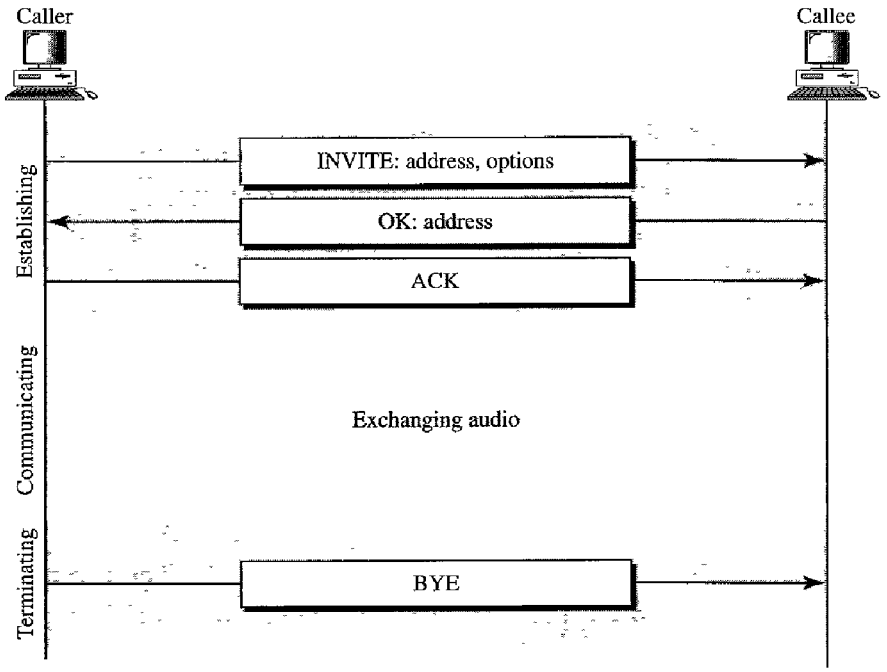
**Figure 29.22** *SIP formats*




### *Simple Session*

A simple session using SIP consists of three modules: establishing, communicating, and terminating. Figure 29.23 shows a simple session using SIP.

**Figure 29.23** *SIP simple session*




**Establishing a Session** Establishing a session in SIP requires a three-way handshake. The caller sends an INVITE message, using UDP, TCP, or SCTP to begin the communication. If the callee is willing to start the session, she sends a reply message. To confirm that a reply code has been received, the caller sends an ACK message.

**Communicating** After the session has been established, the caller and the callee can communicate by using two temporary ports.

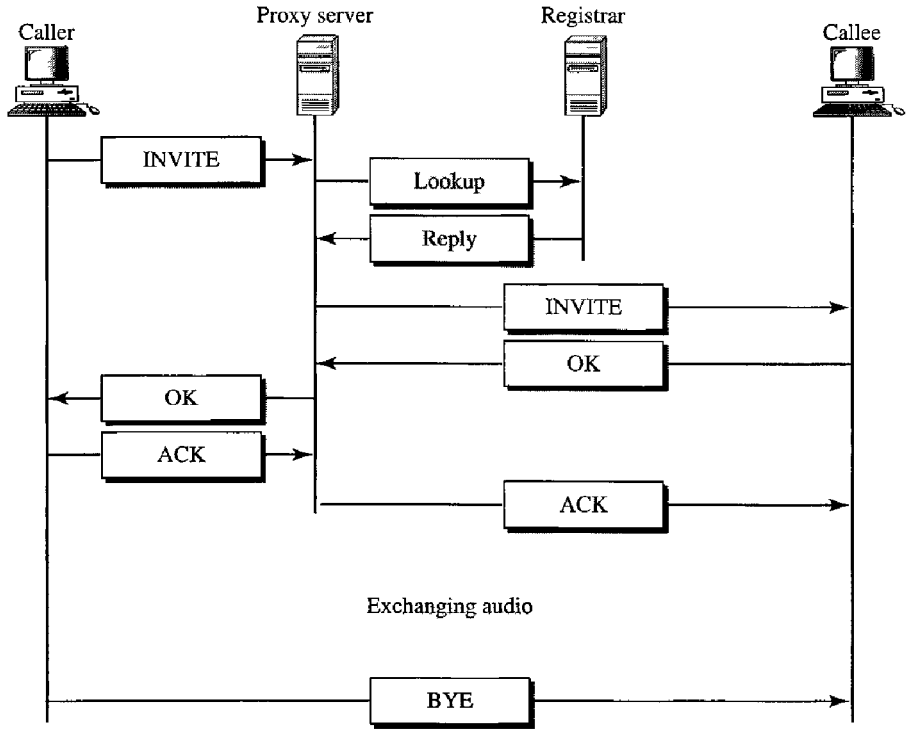
**Terminating the Session** The session can be terminated with a BYE message sent by either party.

### *Tracking the Callee*

What happens if the callee is not sitting at her terminal? She may be away from her system or at another terminal. She may not even have a fixed IP address if DHCP is being used. SIP has a mechanism (similar to one in DNS) that finds the IP address of the terminal at which the callee is sitting. To perform this tracking, SIP uses the concept of registration. SIP defines some servers as registrars. At any moment a user is registered with at least one **registrar server;** this server knows the IP address of the callee.

When a caller needs to communicate with the callee, the caller can use the e-mail address instead of the IP address in the INVITE message. The message goes to a proxy server. The proxy server sends a lookup message (not part of SIP) to some registrar server that has registered the callee. When the proxy server receives a reply message from the registrar server, the proxy server takes the caller's INVITE message and inserts the newly discovered IP address of the callee. This message is then sent to the callee. Figure 29.24 shows the process.

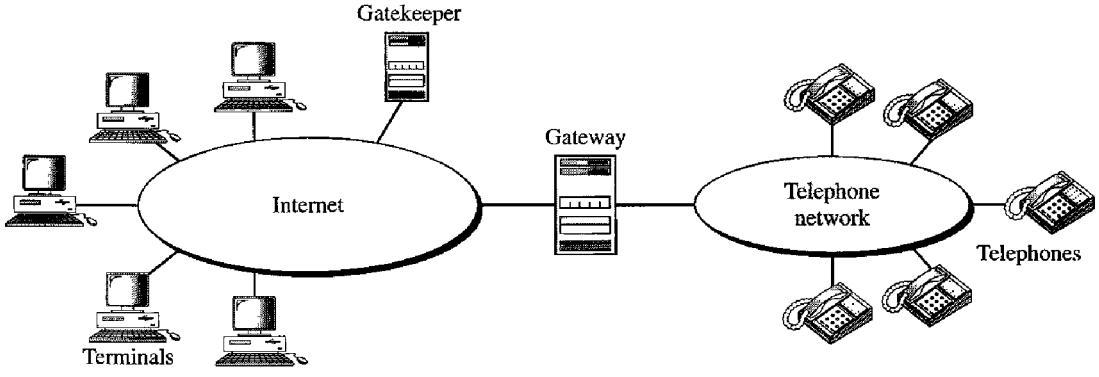
**Figure 29.24** *Tracking the callee*




## H.323

**H.323** is a standard designed by ITU to allow telephones on the public telephone network to talk to computers (called *terminals* in H.323) connected to the Internet. Figure 29.25 shows the general architecture of H.323.

**Figure 29.25** *H.323 architecture*



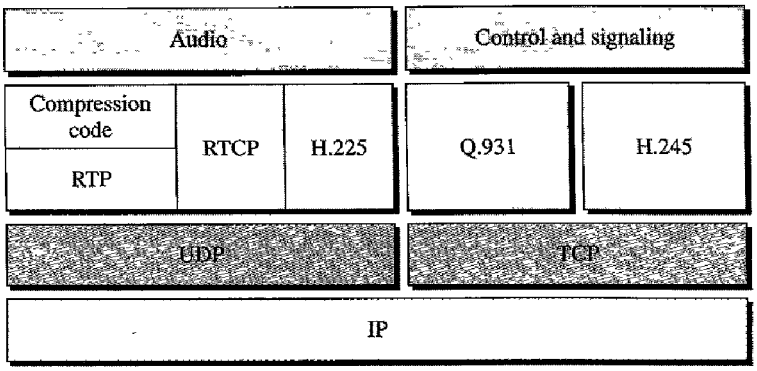

A **gateway** connects the Internet to the telephone network. In general, a gateway is a five-layer device that can translate a message from one protocol stack to another. The

gateway here does exactly the same thing. It transforms a telephone network message to an Internet message. The **gatekeeper** server on the local area network plays the role of the registrar server, as we discussed in the SIP.

### *Protocols*

H.323 uses a number of protocols to establish and maintain voice (or video) communication. Figure 29.26 shows these protocols.

**Figure 29.26** *H.323 protocols*



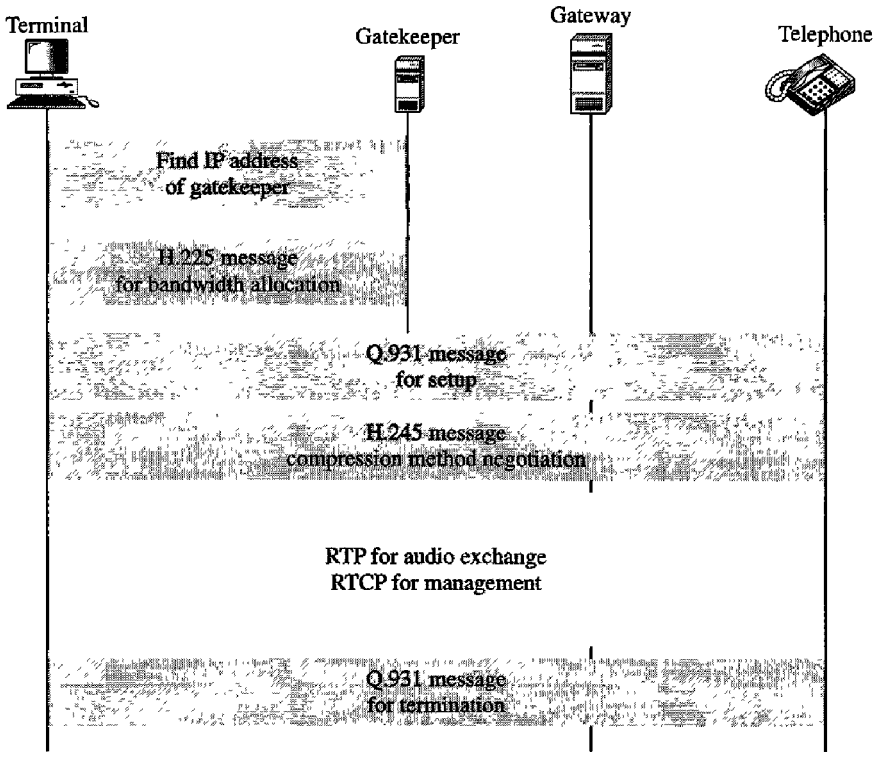

H.323 uses G.71 or G.723.1 for compression. It uses a protocol named H.245 which allows the parties to negotiate the compression method. Protocol Q.931 is used for establishing and terminating connections. Another protocol called H.225, or RAS (Registration/Administration/Status), is used for registration with the gatekeeper.

### *Operation*

Let us show the operation of a telephone communication using H.323 with a simple example. Figure 29.27 shows the steps used by a terminal to communicate with a telephone.

1. The terminal sends a broadcast message to the gatekeeper. The gatekeeper responds with its IP address.
2. The terminal and gatekeeper communicate, using H.225 to negotiate bandwidth.
3. The terminal, gatekeeper, gateway, and telephone communicate by using Q.931 to set up a connection.
4. The terminal, gatekeeper, gateway, and telephone communicate by using H.245 to negotiate the compression method.
5. The terminal, gateway, and telephone exchange audio by using RTP under the management of RTCP.
6. The terminal, gatekeeper, gateway, and telephone communicate by using Q.931 to terminate the communication.

**Figure 29.27** *H.323 example*




## 29.9 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [. . .] refer to the reference list at the end of the text.

### Books

Multimedia communication is discussed in [Hal01] and in Section 7.4 of [Tan03]. Image and video compression are fully discussed in [Dro02].

### Sites

- www.ietf.org/rfc.html Information about RFCs

## 29.10 KEY TERMS

bidirectional frame (B-frame)
compression
discrete cosine transform (DCT)
frequency masking
gatekeeper
gateway
H.323
interactive audio/video
intracoded frame (I-frame)
jitter

Joint Photographic Experts Group (JPEG)
media player
media server
metafile
mixer
Moving Picture Experts Group (MPEG)
MP3
multicasting
on-demand audio/video
perceptual encoding
pixel
playback buffer
predicted frame (P-frame)
predictive encoding
quantization
Real-Time Streaming Protocol (RTSP)
real-time traffic
Real-time Transport Control Protocol (RTCP)
Real-time Transport Protocol (RTP)
registrar server
Session Initiation Protocol (SIP)
spatial compression
streaming live audio/video
streaming stored audio/video
temporal compression
temporal masking
timestamp
translation
voice over IP

## 29.11 SUMMARY

- Audio/video files can be downloaded for future use (streaming stored audio/video) or broadcast to clients over the Internet (streaming live audio/video). The Internet can also be used for live audio/video interaction.
- Audio and video need to be digitized before being sent over the Internet.
- Audio files are compressed through predictive encoding or perceptual encoding.
- Joint Photographic Experts Group (JPEG) is a method to compress pictures and graphics.
- The JPEG process involves blocking, the discrete cosine transform, quantization, and lossless compression.
- Moving Pictures Experts Group (MPEG) is a method to compress video.
- MPEG involves both spatial compression and temporal compression. The former is similar to JPEG, and the latter removes redundant frames.
- We can use a Web server, or a Web server with a metafile, or a media server, or a media server and RTSP to download a streaming audio/video file.
- Real-time data on a packet-switched network require the preservation of the time relationship between packets of a session.
- Gaps between consecutive packets at the receiver cause a phenomenon called jitter.
- Jitter can be controlled through the use of timestamps and a judicious choice of the playback time.
- A playback buffer holds data until they can be played back.
- A receiver delays playing back real-time data held in the playback buffer until a threshold level is reached.

- Sequence numbers on real-time data packets provide a form of error control.
- Real-time data are multicast to receivers.
- Real-time traffic sometimes requires a translator to change a high-bandwidth signal to a lower-quality narrow-bandwidth signal.
- A mixer combines signals from different sources into one signal.
- Real-time multimedia traffic requires both UDP and Real-time Transport Protocol (RTP).
- RTP handles time-stamping, sequencing, and mixing.
- Real-time Transport Control Protocol (RTCP) provides flow control, quality of data control, and feedback to the sources.
- Voice over IP is a real-time interactive audio/video application.
- The Session Initiation Protocol (SIP) is an application layer protocol that establishes, manages, and terminates multimedia sessions.
- H.323 is an ITU standard that allows a telephone connected to a public telephone network to talk to a computer connected to the Internet.

## 29.12 PRACTICE SET

### Review Questions

1. How does streaming live audio/video differ from streaming stored audio/video?
2. How does frequency masking differ from temporal masking?
3. What is the function of a metafile in streaming stored audio/video?
4. What is the purpose of RTSP in streaming stored audio/video?
5. How does jitter affect real-time audio/video?
6. Discuss how SIP is used in the transmission of multimedia.
7. When would you use JPEG? When would you use MPEG?
8. In JPEG, what is the function of blocking?
9. Why is the DCT needed in JPEG?
10. What is spatial compression compared to temporal compression?

### Exercises

11. In Figure 29.17 what is the amount of data in the playback buffer at each of the following times?
    a. 00:00:17
    b. 00:00:20
    c. 00:00:25
    d. 00:00:30
12. Compare and contrast TCP with RTP. Are both doing the same thing?
13. Can we say UDP plus RTP is the same as TCP?
14. Why does RTP need the service of another protocol, RTCP, but TCP does not?

15. In Figure 29.12, can the Web server and media server run on different machines?
16. We discuss the use of SIP in this chapter for audio. Is there any drawback to prevent using it for video?
17. Do you think H.323 is actually the same as SIP? What are the differences? Make a comparison between the two.
18. What are the problems for full implementation of voice over IP? Do you think we will stop using the telephone network very soon?
19. Can H.323 also be used for video?

## Research Activities

20. Find the format of an RTCP sender report. Pay particular attention to the packet length and the parts repeated for each source. Describe each field.
21. Find the format of an RTCP receiver report. Pay particular attention to the packet length and the parts repeated for each source. Describe each field.
22. Find the format of an RTCP source description. Pay particular attention to the packet length and the parts repeated for each source. Describe each field.
23. Find the meaning of the source description items used in the RTCP source description packet. Specifically, find the meaning of CNAME, NAME, EMAIL, PHONE, LOC, TOOL, NOTE, and PRIV.
24. Find the format of an RTCP bye message. Pay particular attention to the packet length and the parts repeated for each source. Describe each field.