UNIT-6

Application layer include following Topics.

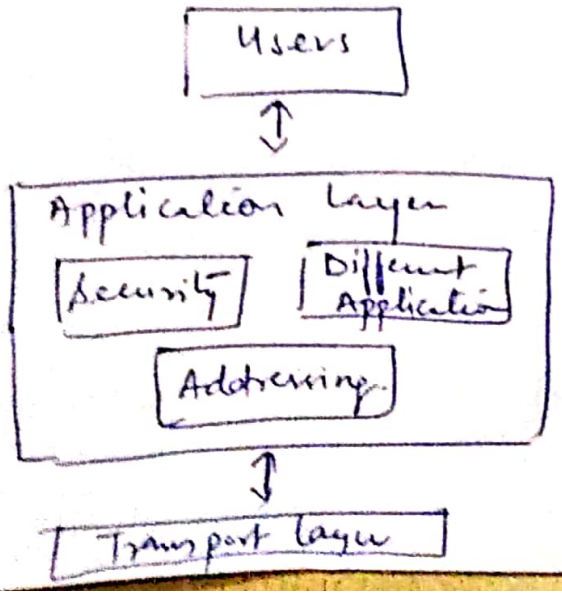
1) WWW
2) DNS
3) Multimedia
4) Electronic mail
5) FTP
6) HTTP + SMTP
7) Introduction to Network Security.

Objective - The application layer enables the user, whether human or H/W, to access the Network. It provides user Interface and support for services such as electronic mail, File access and transfer, access to system resources, surfing the world wide web (WWW) and Network Management.

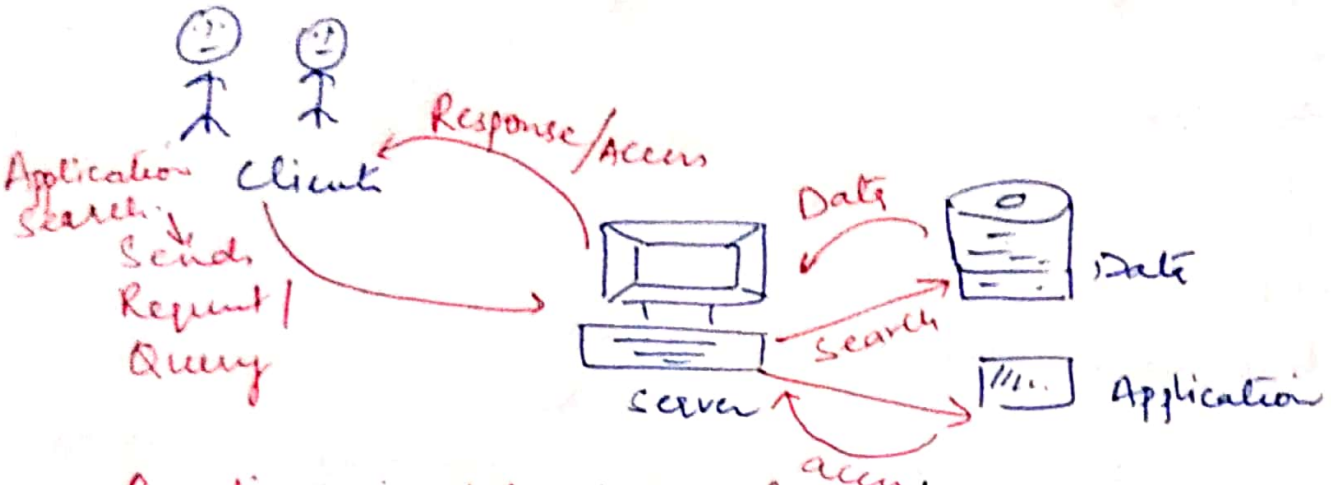
The Application layer is Responsible for Providing Services to the user.

Application layer - It is the topmost layer of the Internet model. Application layer allows people to use the Internet. It is intermediate layer between user and Transport layer.

Position of Application layer



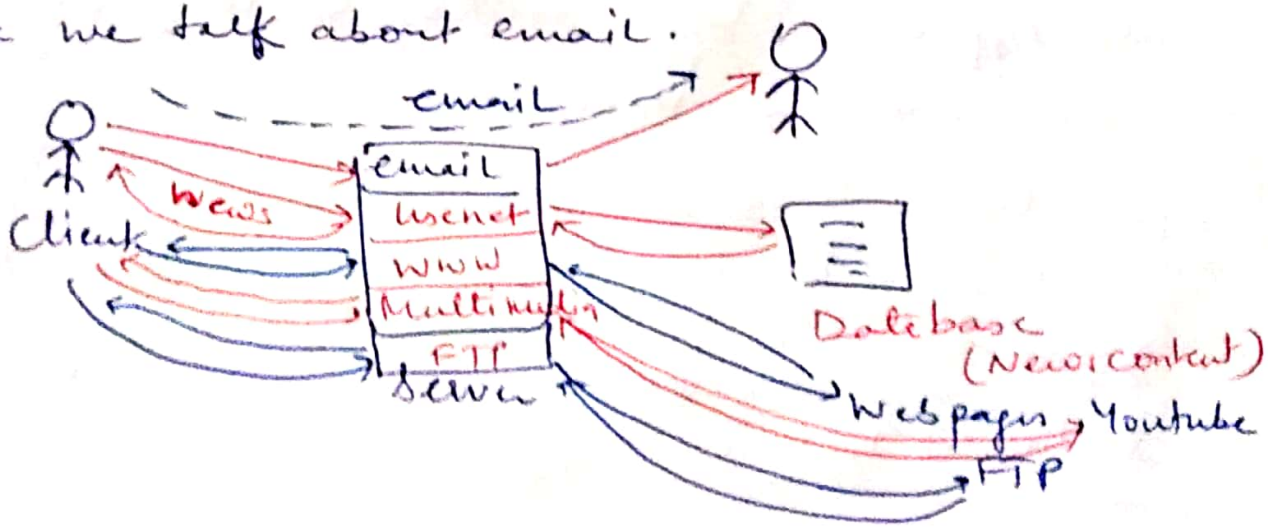
Application Layer Programs are based on the Client Server Model.



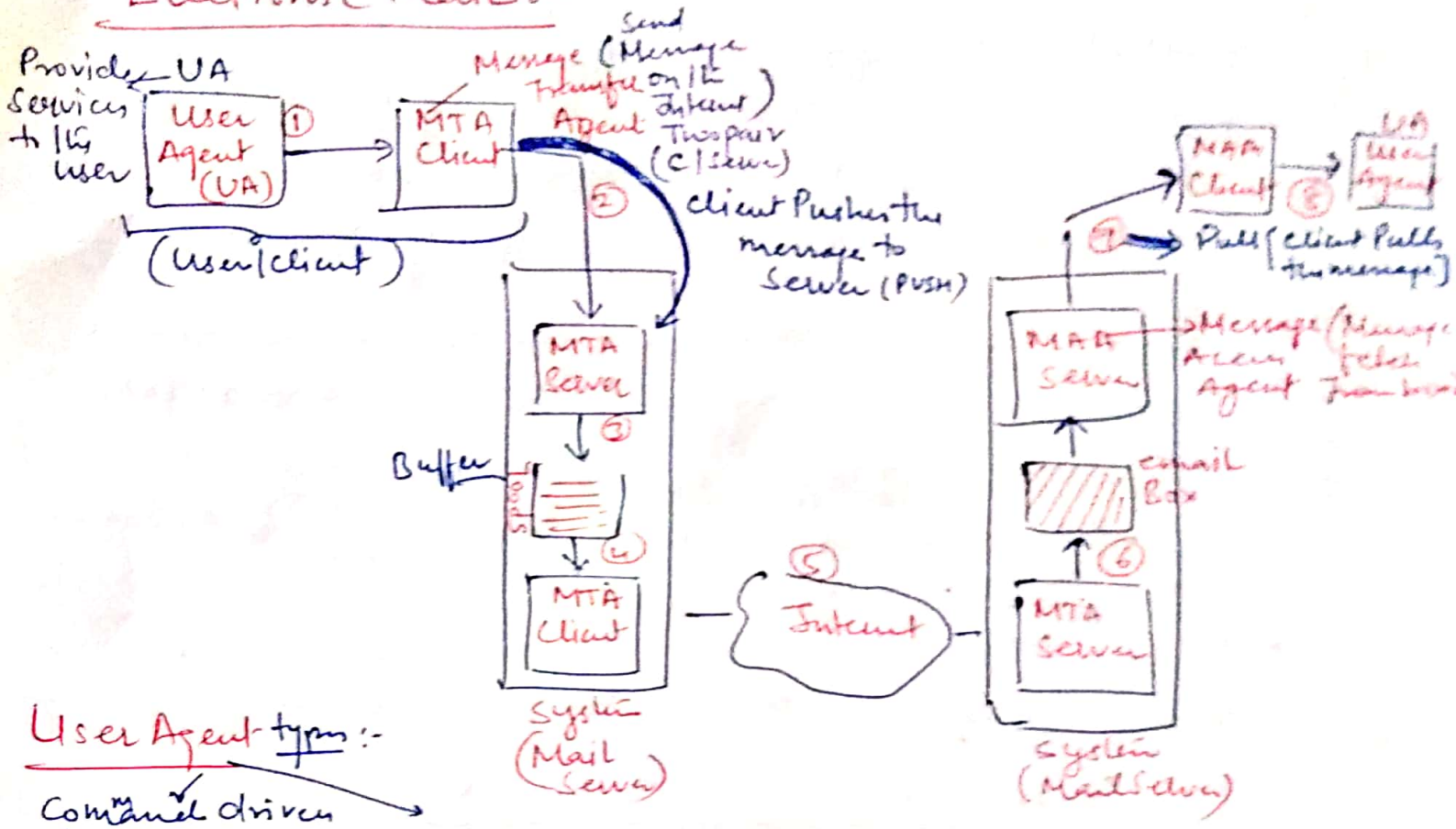
Application of Application layer / Uses

1) Electronic Mail (e-mail)
2) Net News (Usenet)
3) WWW (world wide web)
4) Multimedia
5) Remote file transfer and access.

Suppose we talk about email.



Electronic Mail:-



User Agent types :-

- Command driven (Command prompt users)
  - Mail, pine and users to push or pull.
- GUI Based
  - Ex Outlook, Netscape and web platform which users can access them.

(9) Architecture of email

It gives four Scenarios

(1) **First Scenario** :- the sender and Reciever of emails are users (or application programs) on the same system. that they are directly connected to a shared system. The administrator has created one mailbox for each user where received messages are stored.

A **Mailbox** is part of 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)

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

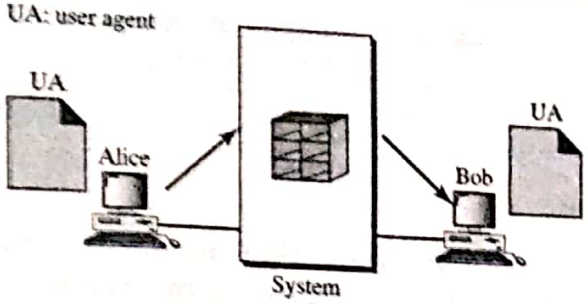
To explain the architecture of e-mail, we give four scenarios. We begin with the simplest situation and add complexity as we proceed. The fourth scenario is the most common in the exchange of email.

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*



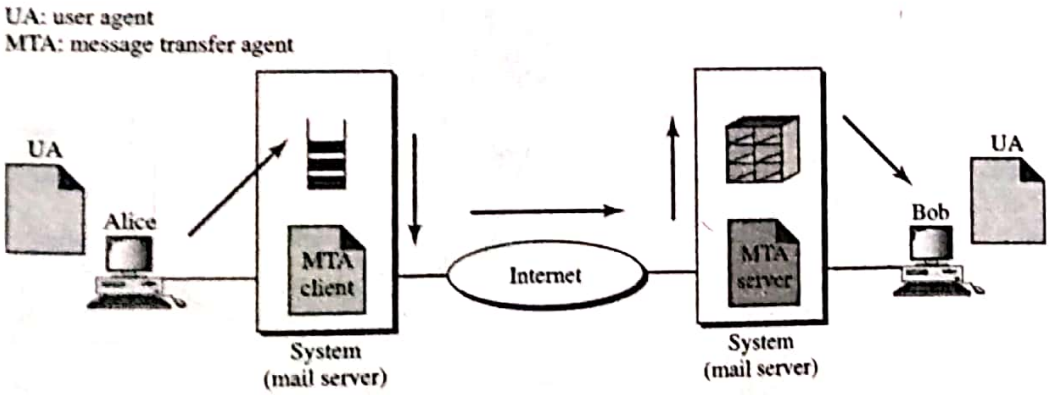
When Alice needs to send a memo to Bob, she writes the memo and inserts it into Bob's mailbox. When Bob checks his mailbox, he finds Alice's memo and reads it.

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

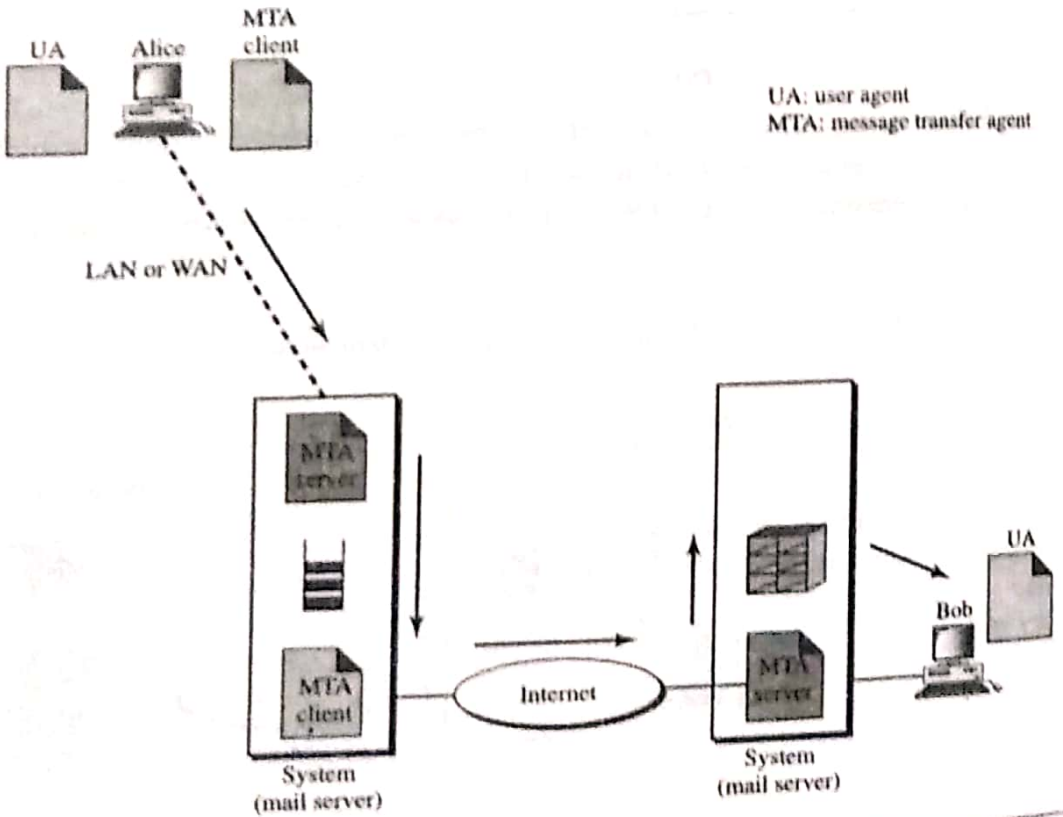
connection. The client, on the other hand, can be alerted by the system when there is a message in the queue to be sent.

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.

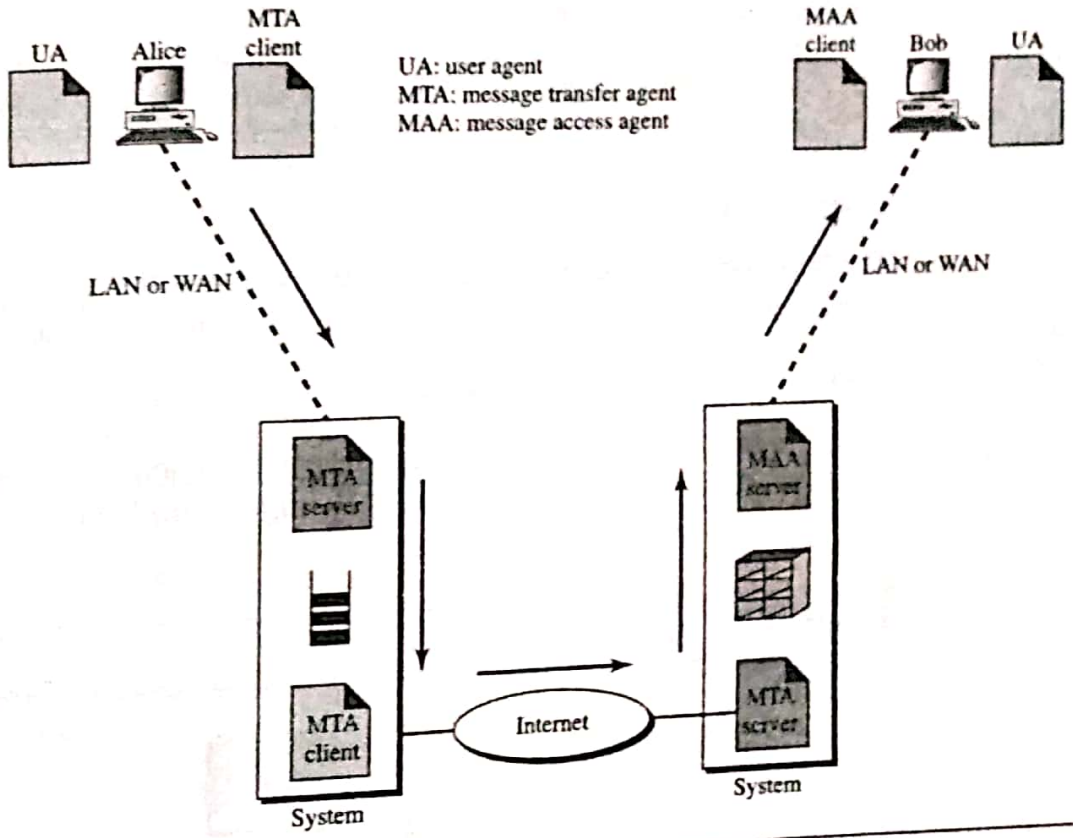
At his convenience, Bob uses his user agent to retrieve the message and reads it. Note that we need two pairs of MTA client/server programs.

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.

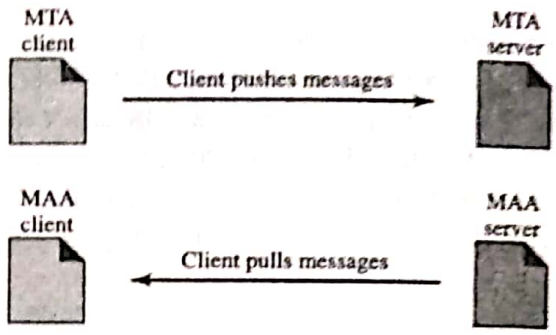
Figure 26.9 *Fourth scenario in electronic mail*



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.

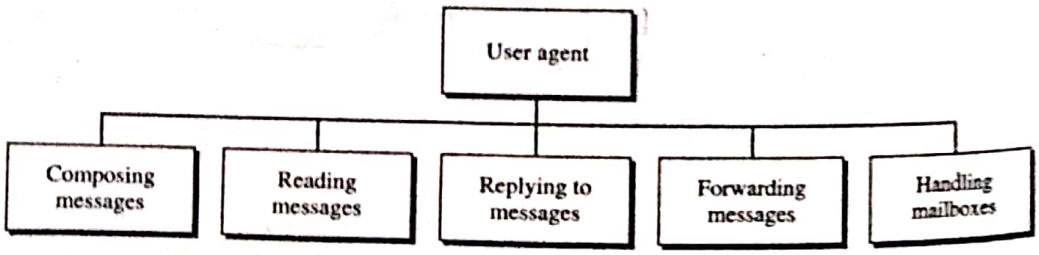
User Agent

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

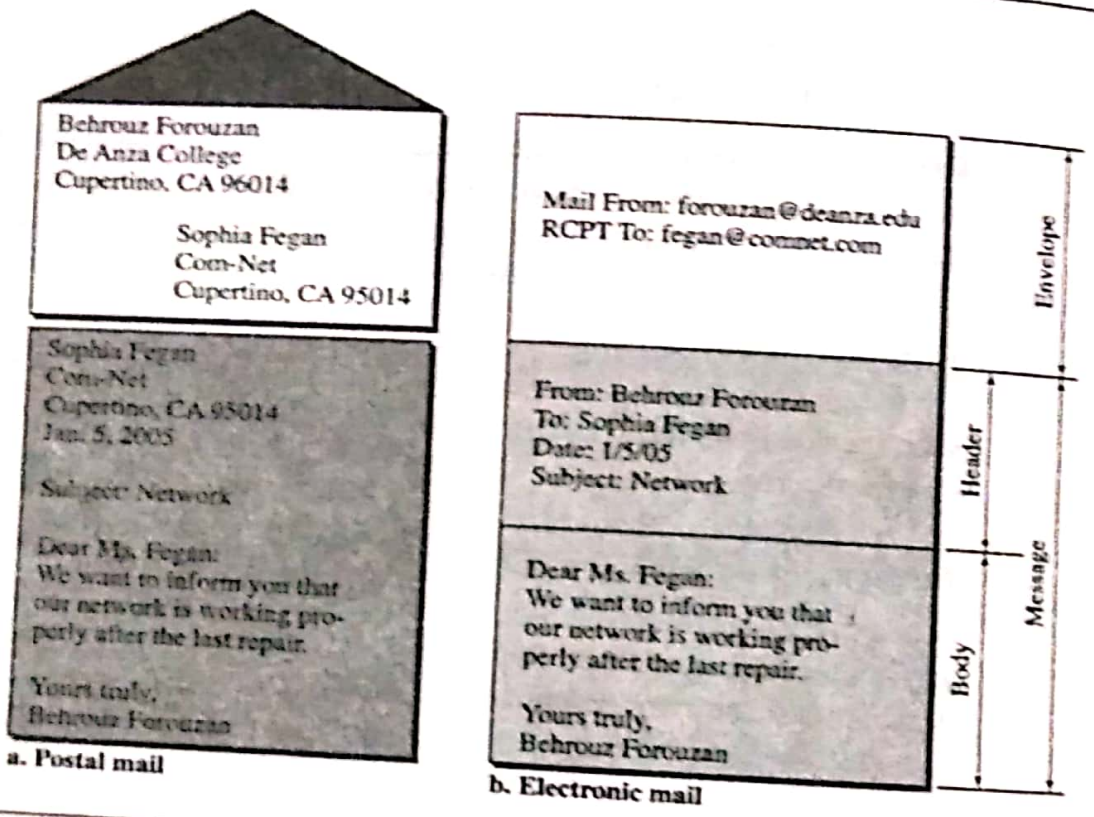
bars, and windows that make the services easy to access. Some examples of GUI-based user agents are Eudora, Microsoft's Outlook, and Netscape.

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.

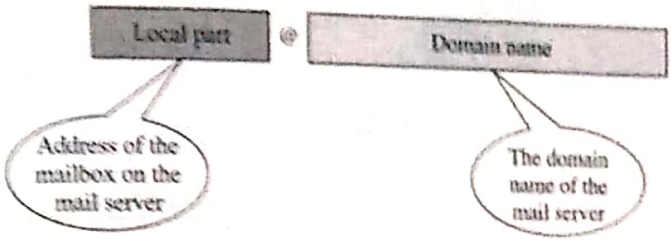
Receiving Mail

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 mail-box. 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.

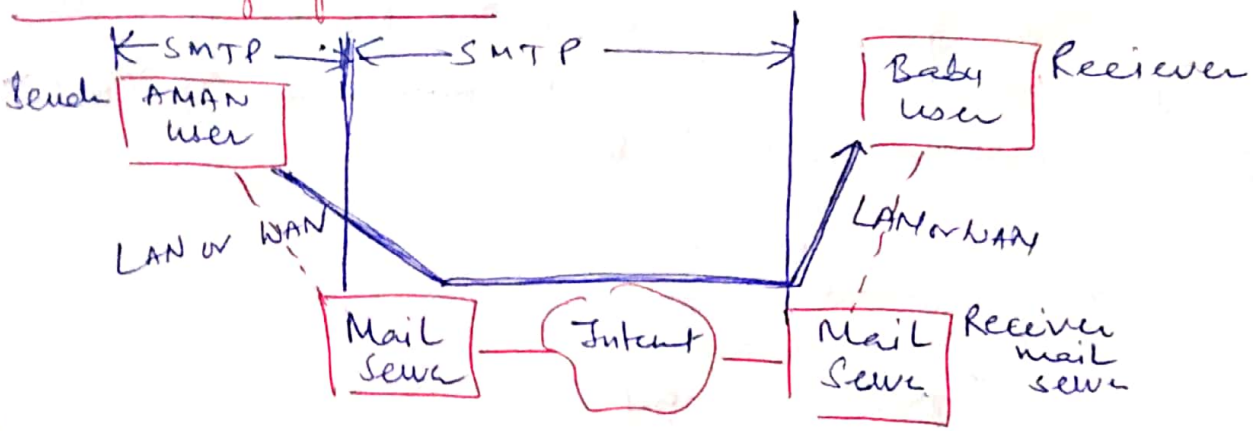
SIMPLE MAIL TRANSFER PROTOCOL (SMTP)

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 SMTP (Simple mail transfer Protocol.

This is the protocol that defines MTA client and server in the Internet.

Use/Range of SMTP -



SMTP Range

SMTP uses commands and Responses to transfer Message between an MTA client and MTA Server

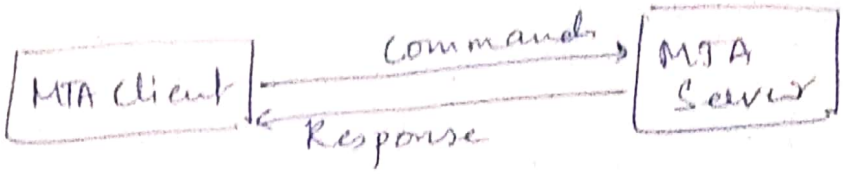
SMTP is used two times bet^n the Sender and Sender's mail Server and bet^n the two mail server. another protocol is needed bet^n the Mail Server and the Reciver

SMTP simply defines how command and Responses must be sent back and forth. Each N/W is free to choose a S/W package for Implementation

Commands + Response

Each command or reply is terminated by a two character (carriage return and line feed).

-End-of-line token



Command & Responses

a) **Commands**:- Commands are sent from client to the Server. The format of a command shown in diagram. It consist 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 recomended. The last six are seldom used.

Command format-

Keyword : argument(s)

Command List

1) HELO – used by client to identify itself - Ex – xyz.com (Client)
2) MAIL FROM – used by client to identify sender of message. (abc@xyz.com → Sender)
3) RCPT TO : Identify intended receipient of the message (RCPT TO: xyz@efg.com)
4) DATA – Send actual message (Body of mail) Ex DATA, This is to remind
5) QUIT:- QUIT – To terminate the message
6) RSET:- Reset – This Comd Current abort Current Mail transaction. Reset the Connection
7) VRFY:- Name of Receipient to be verified / Verify the Addl. of Recipt
8) HELP:- HELP: mail (xyz@efg.com)
9) NOOP
10) TURN

11) EXPN – Mail list to be expanded.
12) Send from – Intended Receipient of the Message
13) SMOL FROM – Intended Receipient of the Message
14) SMAL FROM – Intended Receipient of the Message

b) Responses – Responses are sent from the server to the client. A Response is three digit code that may be followed by additional textual Information.

Example of Responses

1) Positive Condition Reply
- 211 – System status
- 214 – Help
- 220 – Service Ready
- 221 – Service closing transmission channel
- 250 – Requested command completed
- 251 – User not local; the message will be forwarded.

2) Positive Intermediate Reply
- 354 – Start mail input

3) Transient Negative Completion Reply
- 421 – Service Not Available
- 450 – Mailbox not available
- 451 – Command Aborted : Local error
- 452 – Command aborted: insufficient storage

4) Permanent Negative Completion Reply :-
- 500 – Syntax error : unrecognized command.
- 501 – Syntax error in parameter or argument
- 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

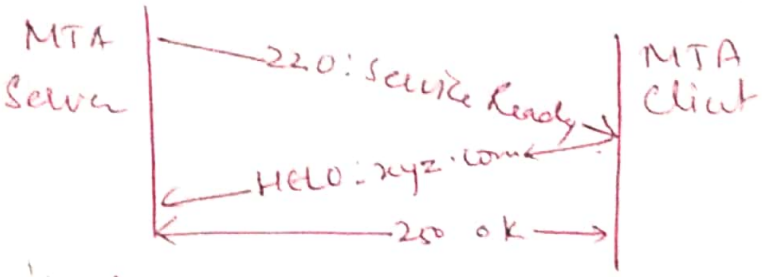
553 – Requested action not taken; mailbox name not allowed

554 – Transaction failed

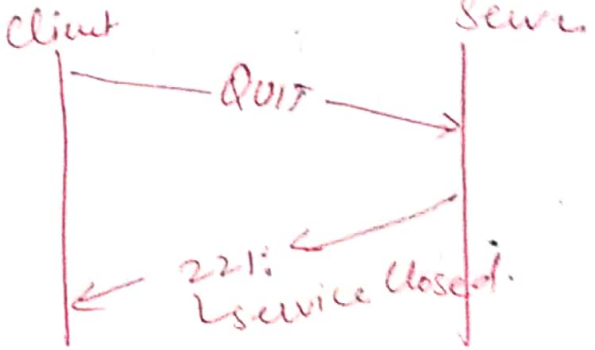
As the table shows Responses are divided into four categories the leftmost digit of each (2, 3, 4 and 5) defines the category.

Mail Transfer Phases

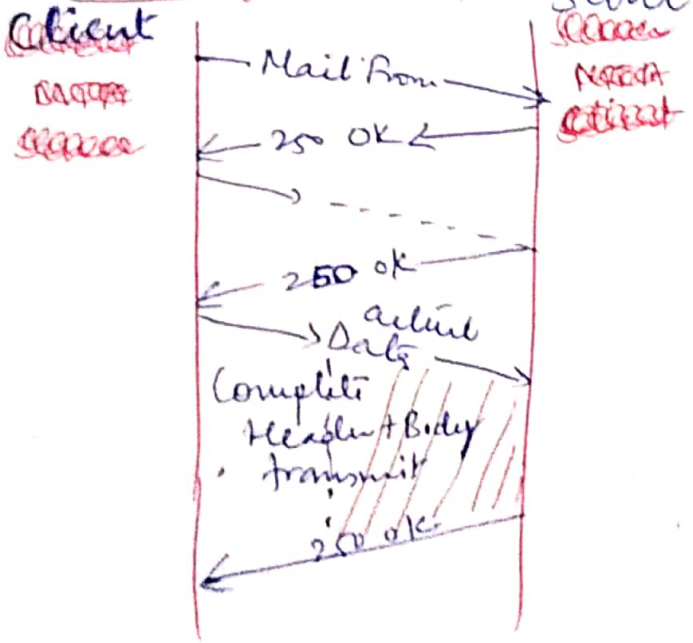
1) Connection Establishment



2) Connection Termination



ii) Message Transfer



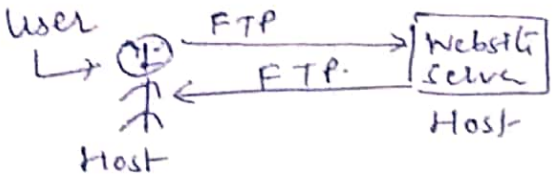
Mail transfer has three phases connection establishment, Message transfer and Connection Termination.

FTP - FILE TRANSFER PROTOCOL :-

Transferring file from one computer to another is one of the most common ~~tasks~~ tasks expected from a Networking and Internetworking environment. The greatest volume of data exchange on the Internet Today is due to File transfer

FTP is the standard Mechanism provided by TCP/IP for copying a file from one host to another.

FTP establishes two connections → Data transfer FTP (port 20)
→ Control Info" FTP (port 21)



For Example, two system may use different file name conventions, Two system may have different ways to Represent text and Data. Two system may have different directory structure. All these problems have solved by FTP in a very simple and Elegant approach.

FTP differs from other client/server Application in that- establishes two Connection between the hosts.

Ⓐ One connection is used for Data transfer

Ⓑ Other for Control information (Commands & Responses)

The control connection uses very simple rules of Communication. We need to transfer only a line of Commands or a line of response at a time.

The Data connection on the other hands, 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 connection are treated the same.

FTP uses two well-known TCP ports:

1) Port 21 is used for Control connection
2) Port 20 is used for Data Connection

IMP

FTP uses the service of TCP, It needs two TCP Connection. The well known port 21 is used for the Control Connection and well known port 20 for the Data Connection

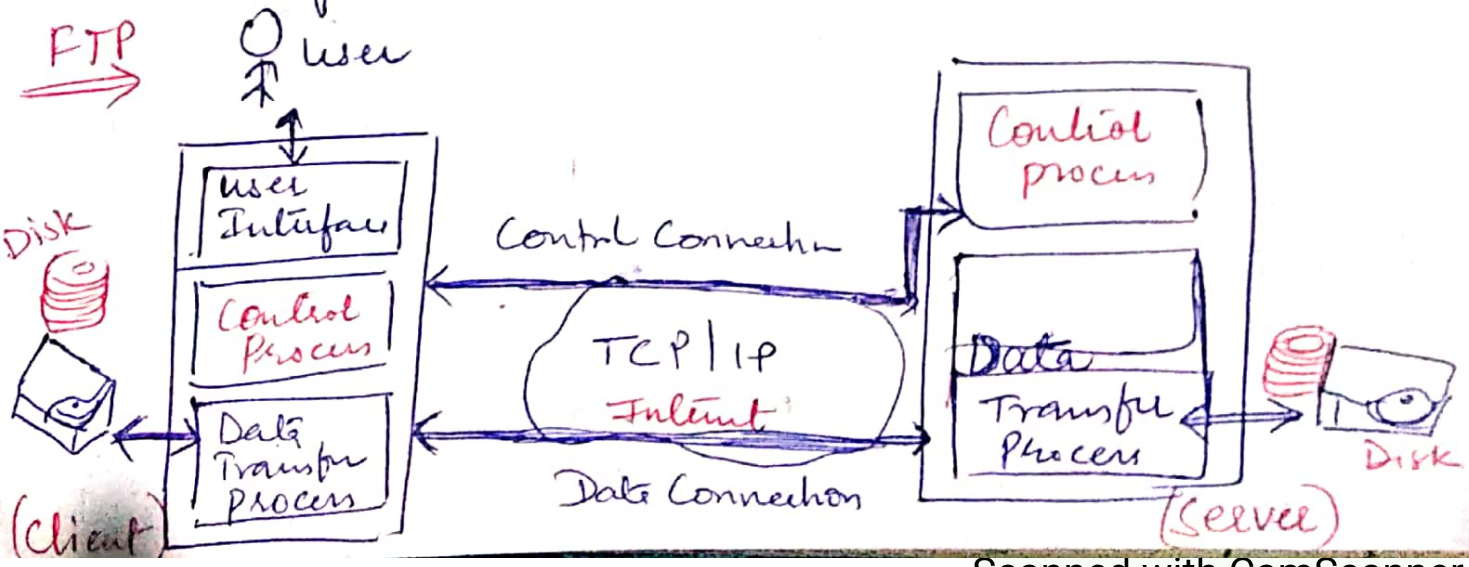
→ FTP Model - The client has three Component user Interface, Client Control process and the client data transfer process.

→ The Server has two Component - the Server Control process and Server Data transfer process.

→ The Control connection is made b/w the Control processes.

→ The Data Connection is made between the Data transfer Processes.

FTP



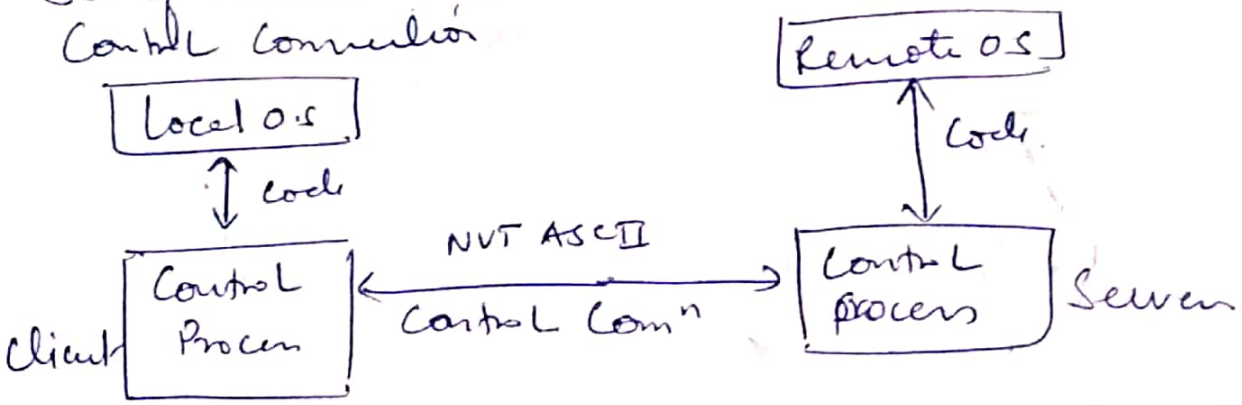
→ The Control Connection remain connected during the Entire Interactive FTP session. (Remain Connected through entire FTP Session)

→ 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. (User opens & closed for each file transfer)

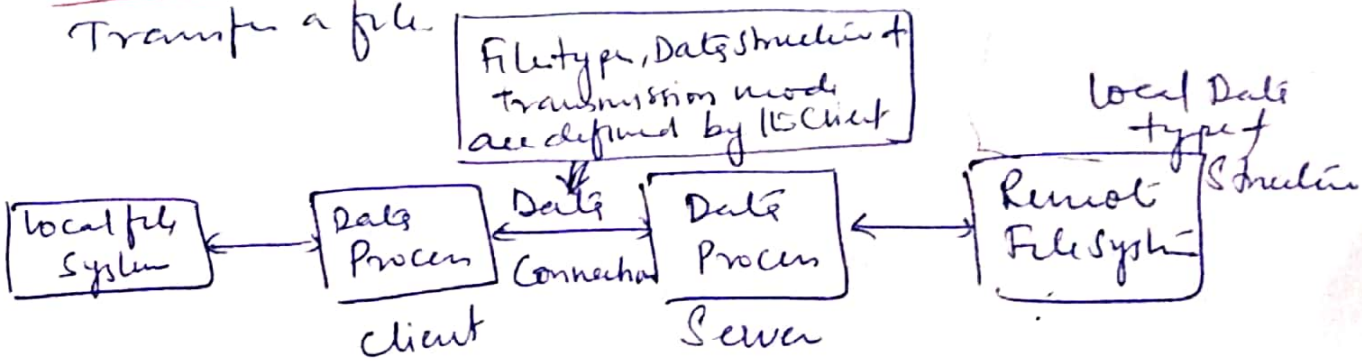
→ On the other hand. When a user starts an FTP session the Control Connection opens. While Control Connection is open, the Data Connection can be opened and closed Multiple times if several files are transferred.

Communication in FTP :-

1) Communication over Control Connection – FTP uses a Set of ASCII characters to communicate across the Control Connection



2) Communication over Data Connection :- Used to Transfer a file.



a) File Types :- FTP can transfer one of the following file types.

↳ 1) ASCII file – ASCII file is default format for transferring Text file.
2) EBCDIC File – EBCDIC Encoding.
3) Image file – Binary file.

b) Data structure :-

1) File structure (default) – No structure / Continuous stream of bytes
2) Record structure – File divided into Records.
3) Page structure – Text files. The file divided into Pages, with each page having a Page Number and page header.

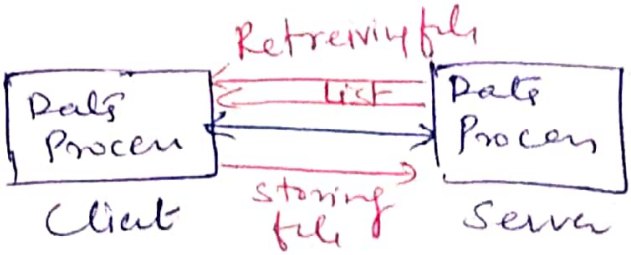
c) Transmission mode :-

1) Stream mode – Continuous stream of byte data transfer take place.

2) Block Mode – Data transfer in the form of Blocks. Stream of byte are combine to make Blocks.

3) Compressed Mode – Used for Big file and these big file are Compressed by encoding scheme

d) File transfer



1) Retreiving file :- Copying a file from server to Client [DOWNLOAD]

2) Storing file :- client to Server [UPLOAD] file.

3) Retreiving list :-

List of file or Directory from Server to client.

e) FTP Commands (3 diffent type of Command)

a) Transfer files

- GET - (Server → Client)
- PUT (Client → Server)

b) Connect to Remote Host :-

- Open = Connection open
- User = Username Pass
- Pass = Password Provide
- Site = Remote host Information Provide to site.

c) Terminate Session.

- QUIT
  Disconnect from Remote HOST and Terminate FTP
- CLOSE
  Just disconnect doesn't terminate.

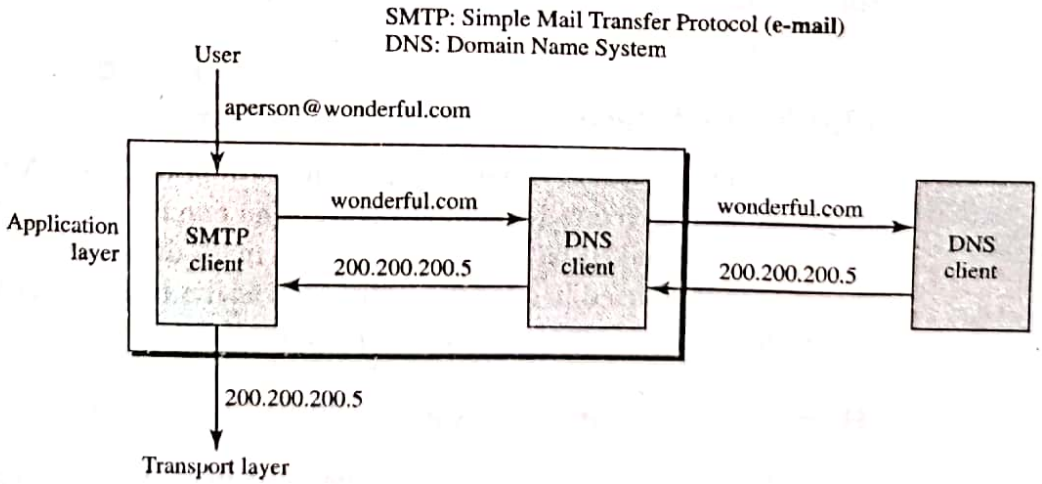
CHAPTER 25

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

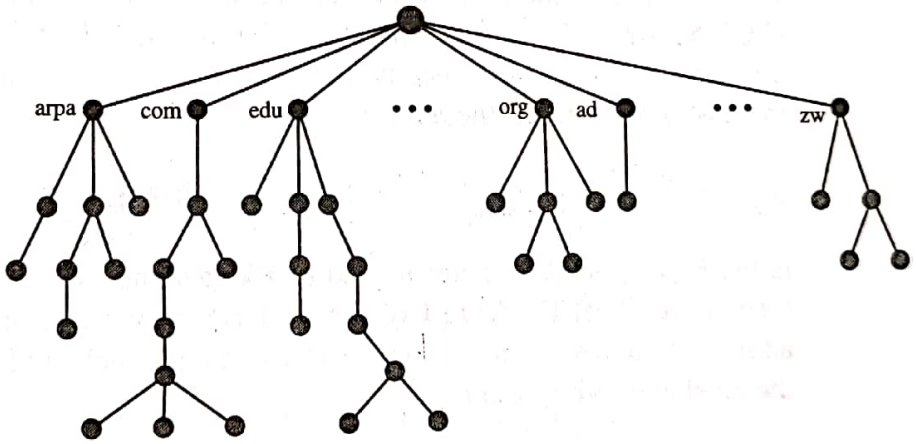
In a **hierarchical name space,** each name is made of several parts. The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on. In this case, the authority to assign and control 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*



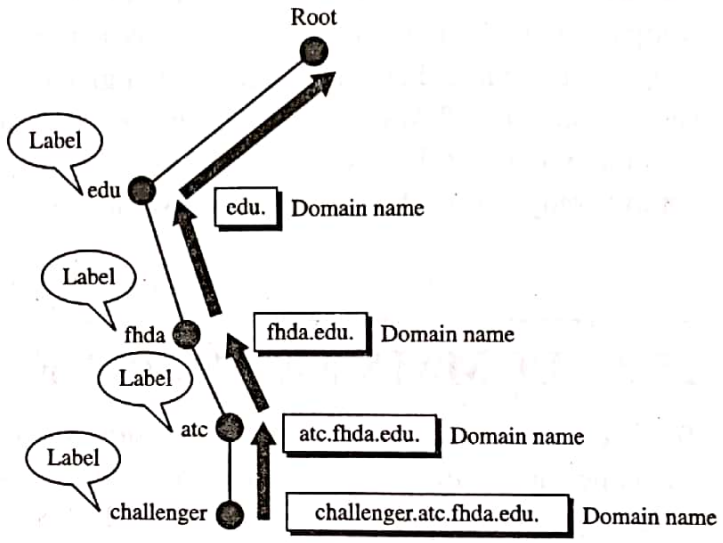
Label

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
```

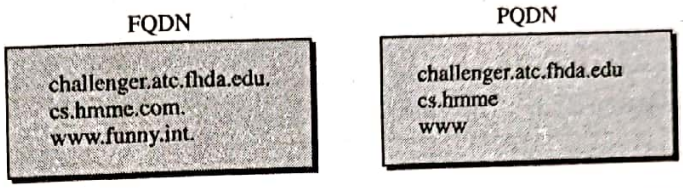
The DNS client adds the suffix *atc.fhda.edu.* before passing the address to the DNS server.

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.

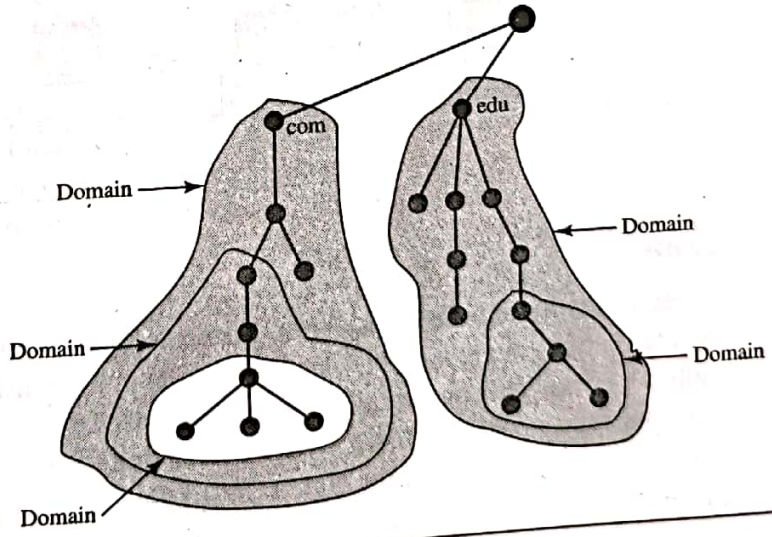
Figure 25.4 *FQDN and PQDN*



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*



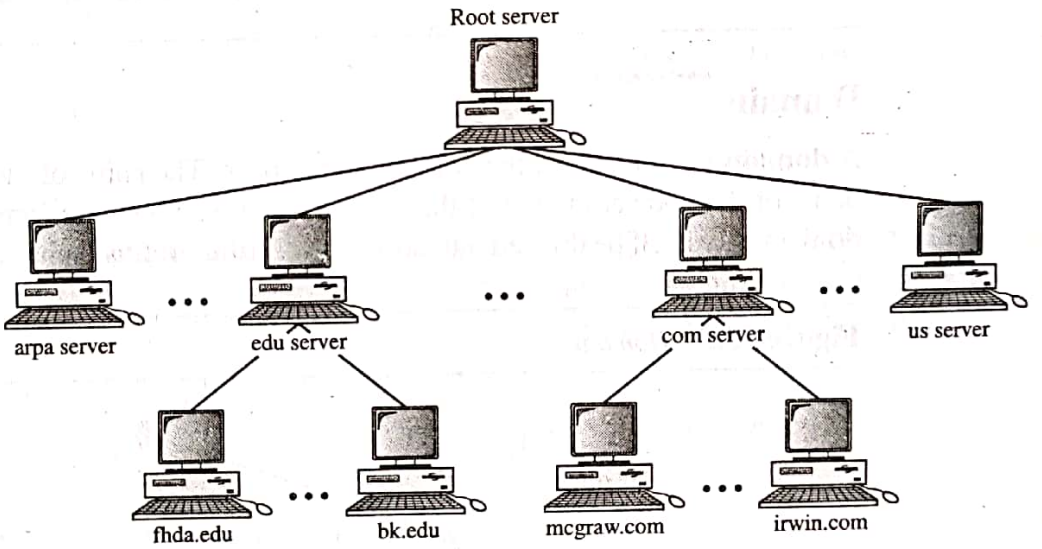
25.3 DISTRIBUTION OF NAME SPACE

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*

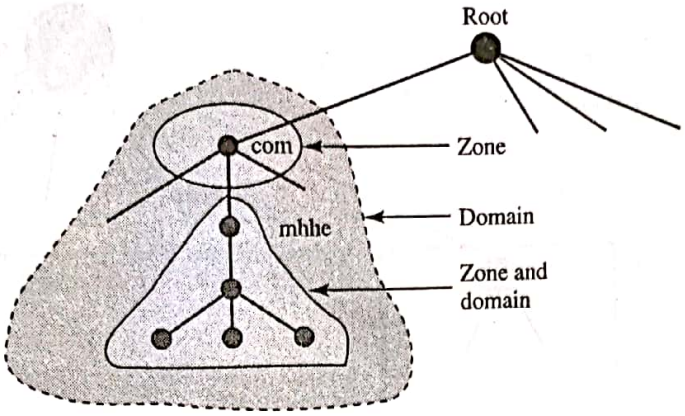


Zone

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.

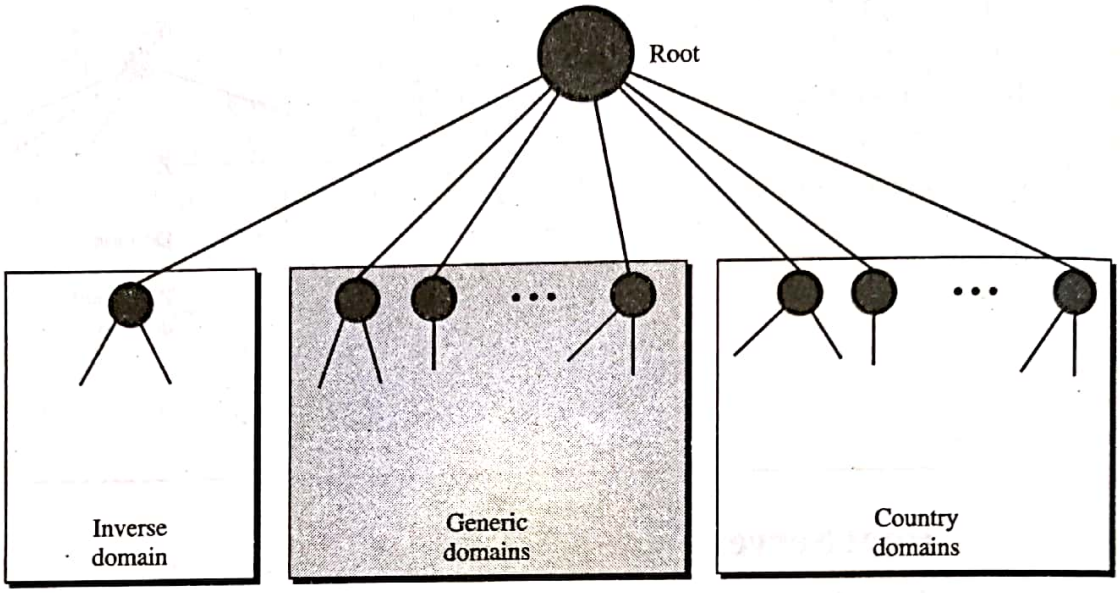
The primary and secondary servers are both authoritative for the zones they serve. The idea is not to put the secondary server at a lower level of authority but to create redundancy for the data so that if one server fails, the other can continue serving clients. Note also that a server can be a primary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

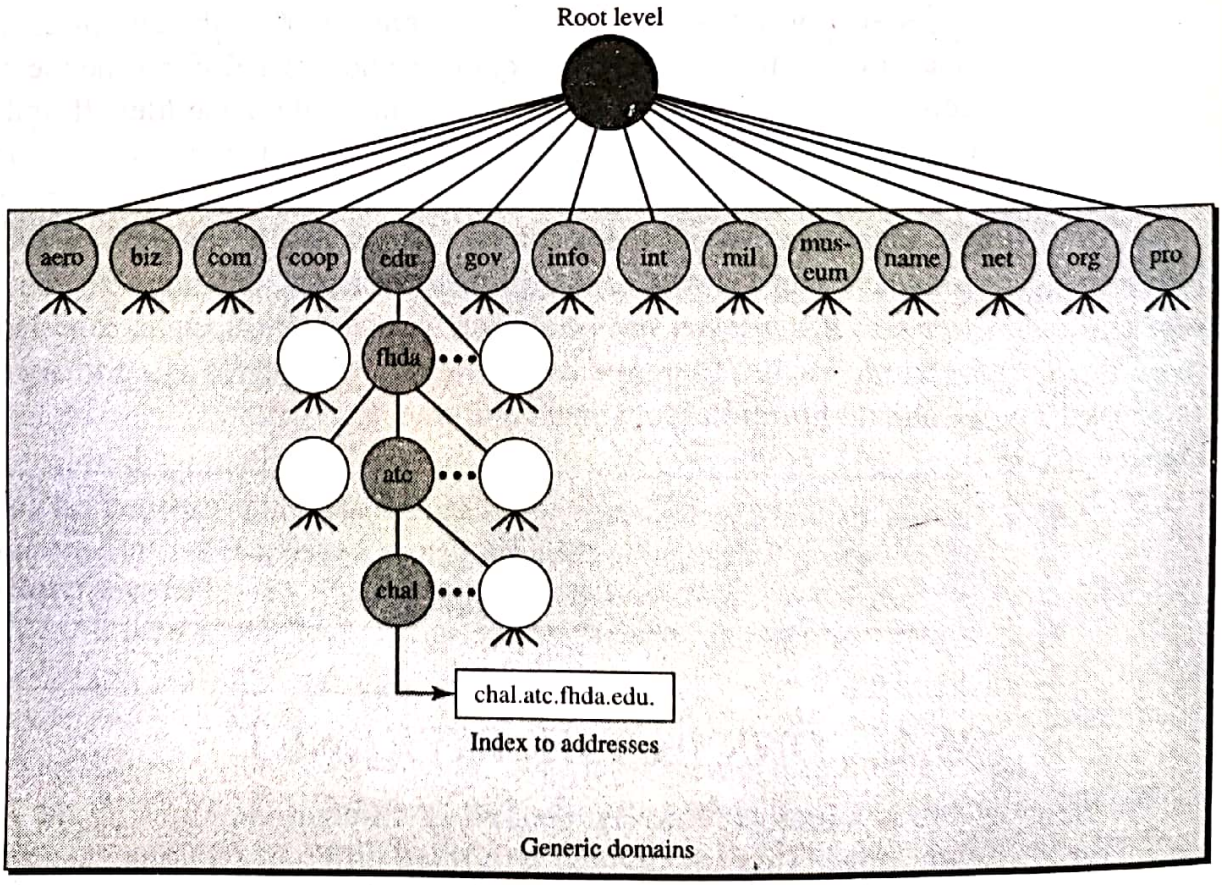
Figure 25.8 *DNS used in the Internet*



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.

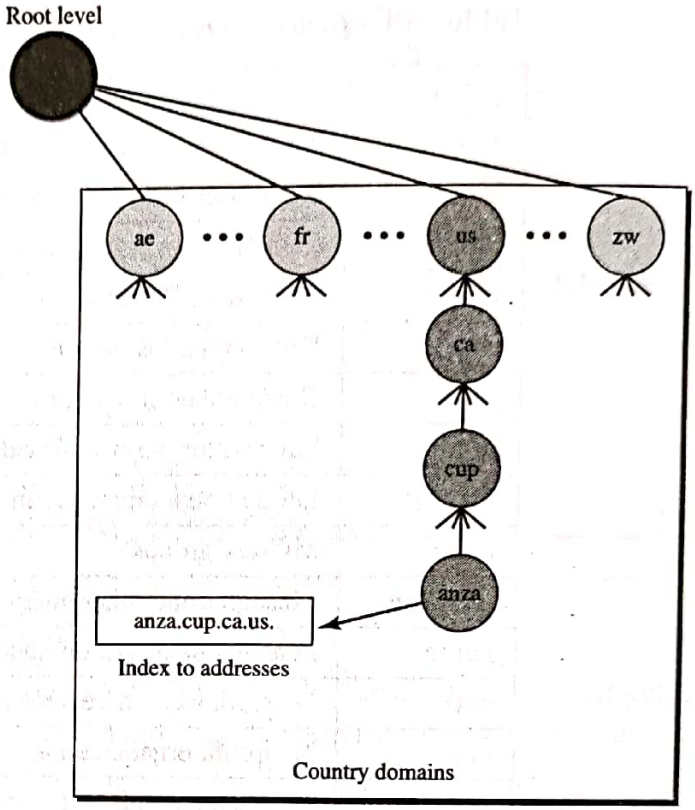
Inverse Domain

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

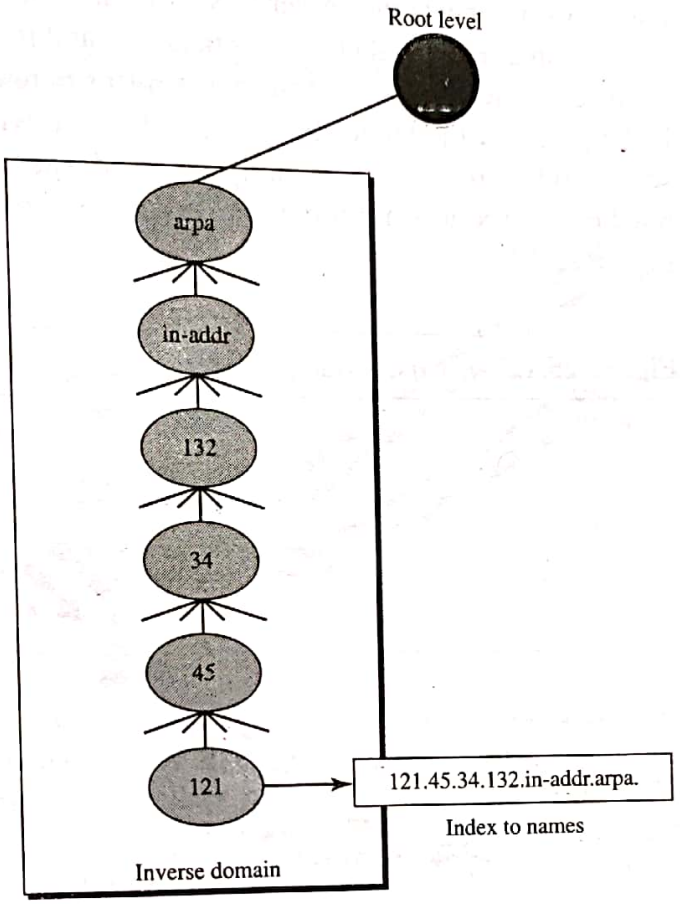
Mapping a name to an address or an address to a name is called *name-address resolution.*

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.

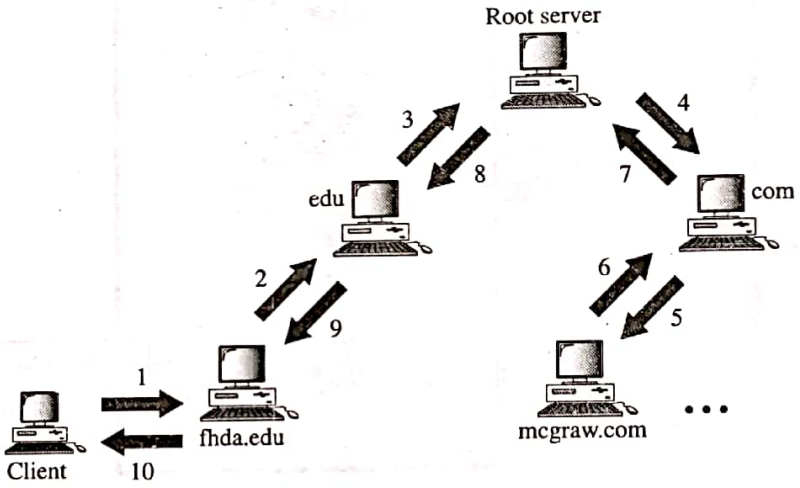
Mapping Addresses to Names

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*



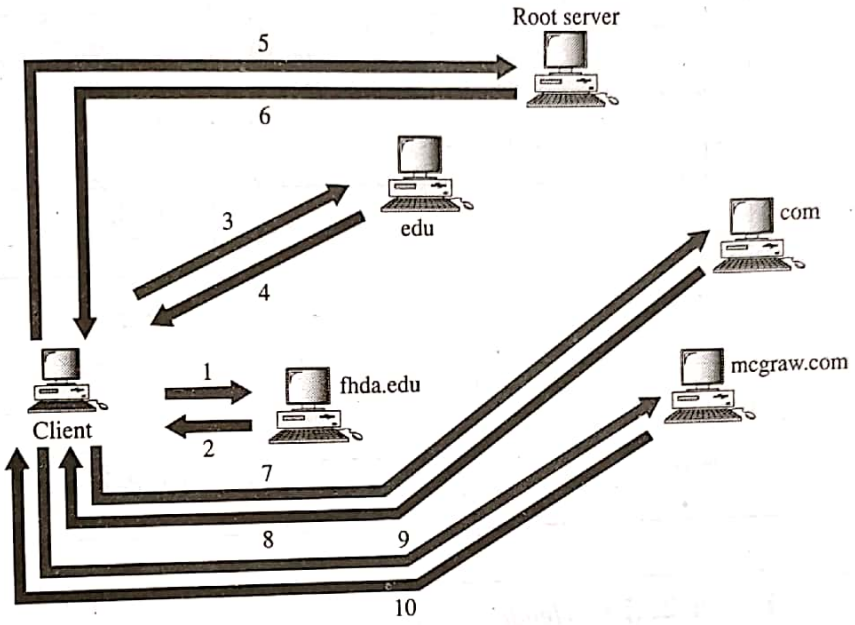
Iterative Resolution

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*

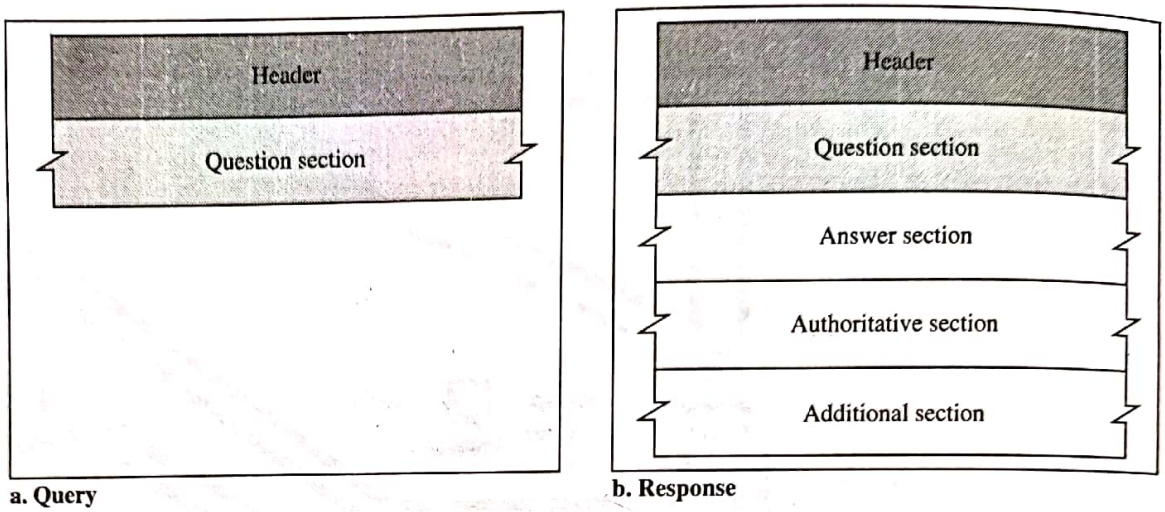


Figure 25.15 *Header format*

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.

CHAPTER 27

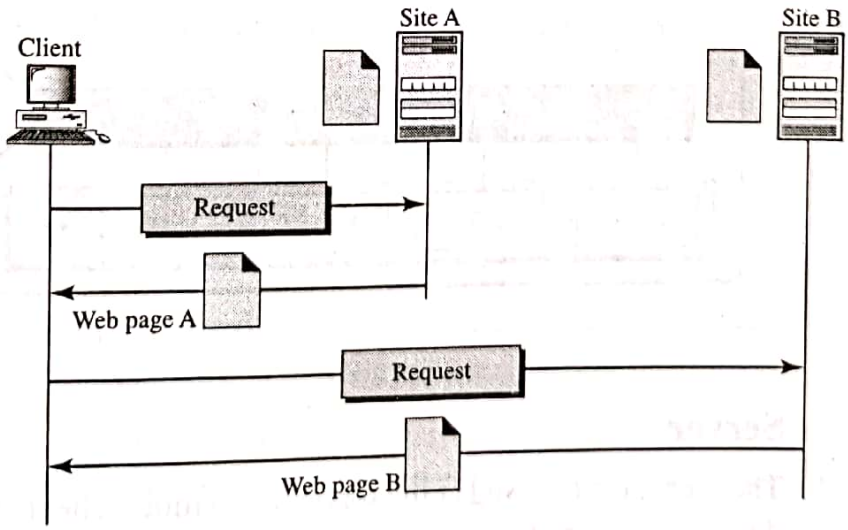
WWW and HTTP

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*

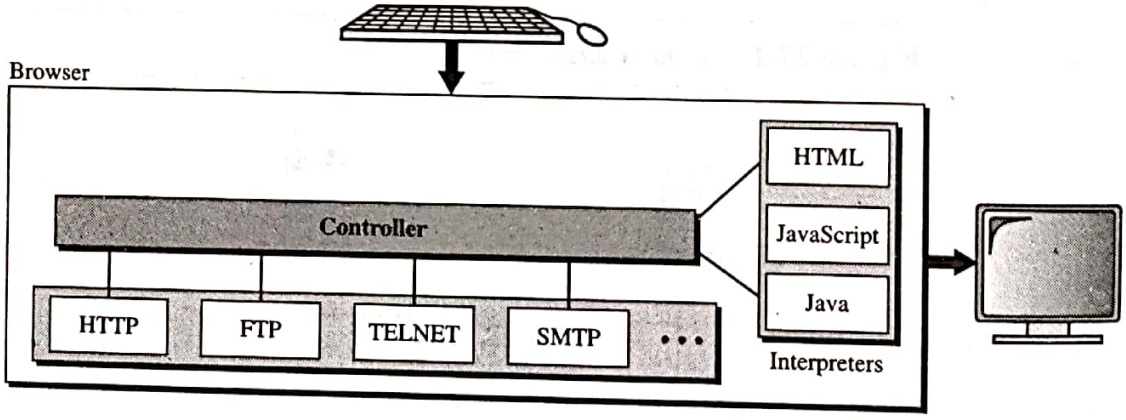


Each site holds one or more documents, referred to as *Web pages*. Each Web page can contain a link to other pages in the same site or at other sites. The pages can be retrieved and viewed by using browsers. Let us go through the scenario shown in Figure 27.1. The client needs to see some information 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*



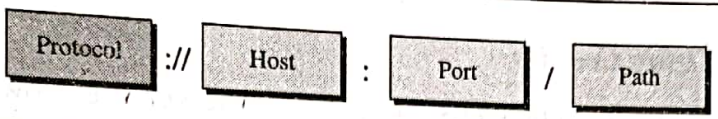
Server

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.

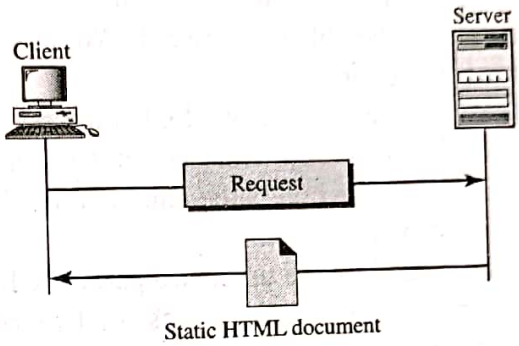
27.2 WEB DOCUMENTS

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*

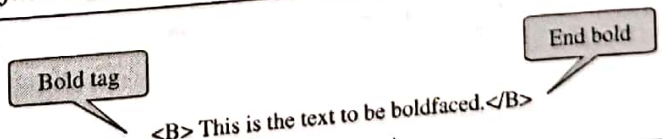


HTML

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.

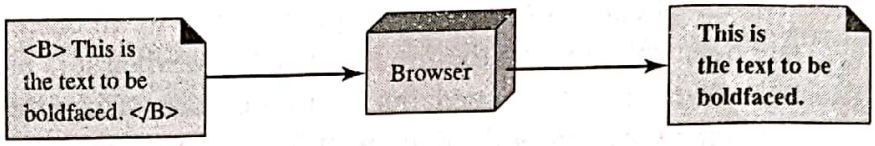
Figure 27.5 *Boldface tags*



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.

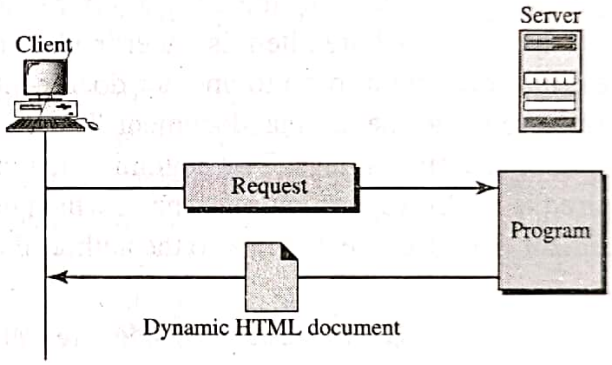
Common Gateway Interface (CGI)

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.

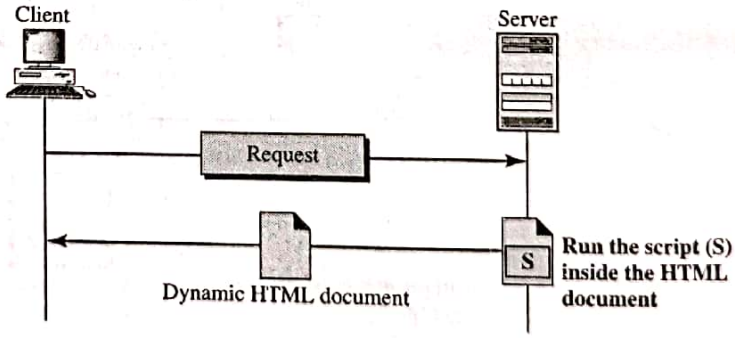
Output The whole idea of CGI is to execute a CGI program at the server site and send the output to the client (browser). The output is usually plain text or a text with HTML structures; however, the output can be a variety of other things. It can be graphics or binary data, a status code, instructions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

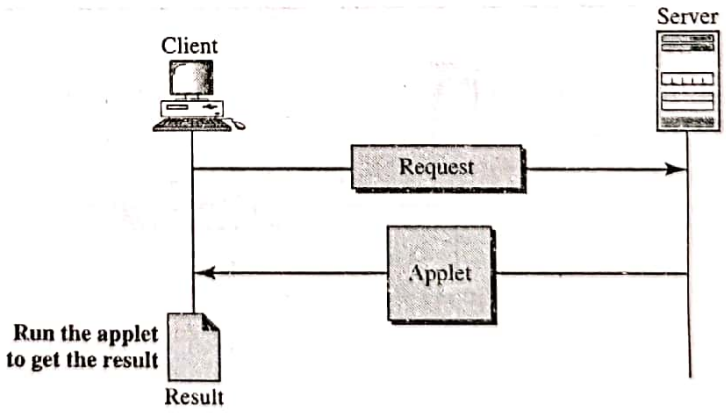
For many applications, we need a program or a script to be run at the client site. These are called **active documents.** For example, suppose we want to run a program that creates animated graphics on the screen or a program that interacts with the user. The program definitely needs to be run at the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

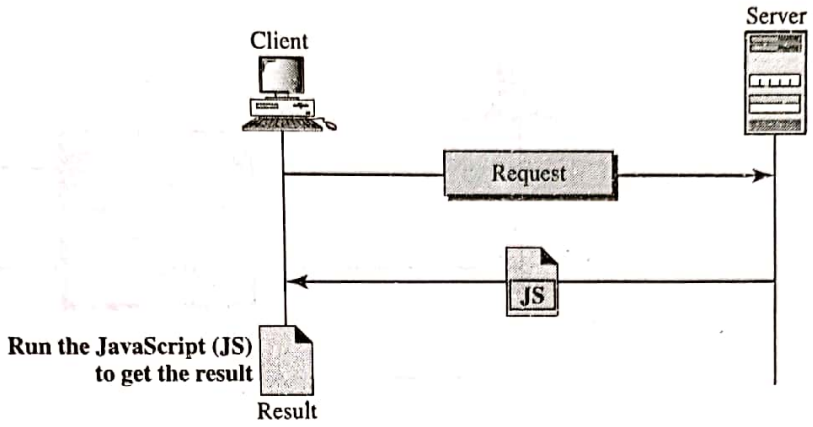
Figure 27.10 *Active document using Java applet*



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*

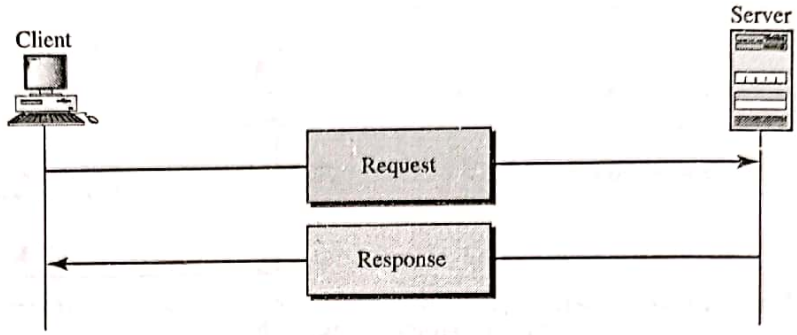
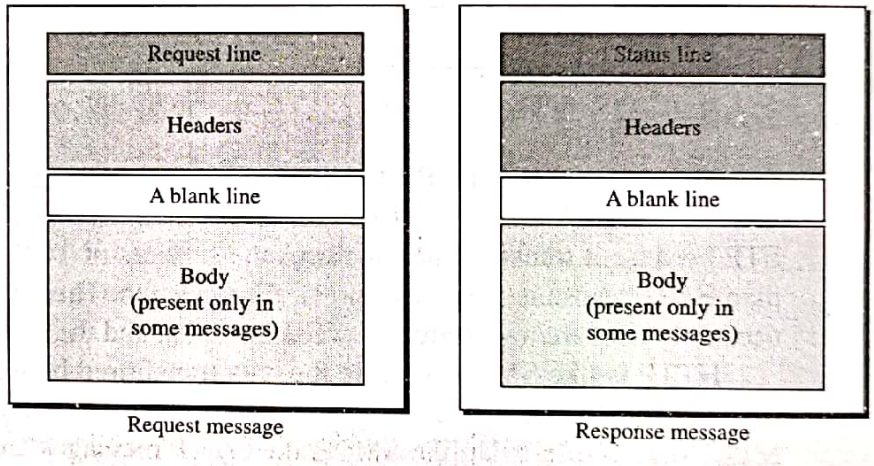
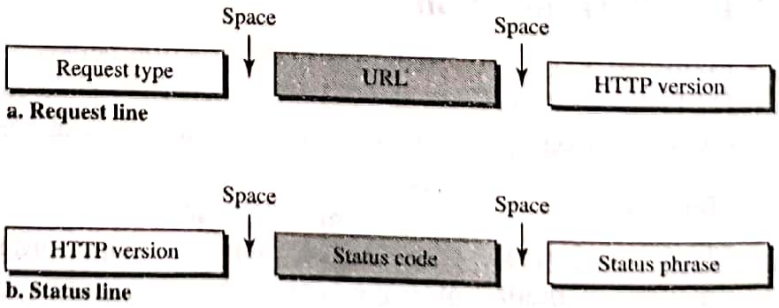


Figure 27.13 *Request and response messages*



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*

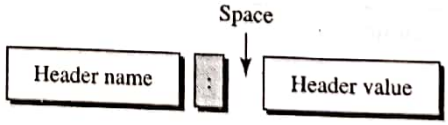
Code	*Phrase*	*Description*
	Informational	
100	Continue	The initial part of the request has been received, and the client may continue with its request.
101	Switching	The server is complying with a client request to switch protocols defined in the upgrade header.
	Success	
200	OK	The request is successful.
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*

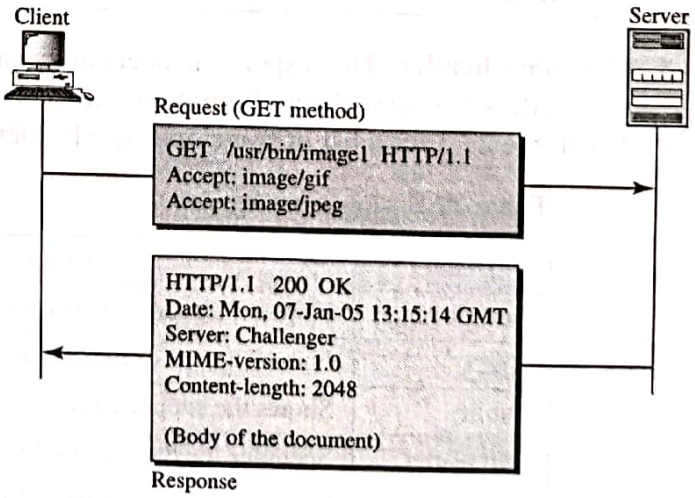
Header	*Description*
Allow	Lists valid methods that can be used with a URL
Content-encoding	Specifies the encoding scheme
Content-language	Specifies the language
Content-length	Shows the length of the document
Content-range	Specifies the range of the document
Content-type	Specifies the medium type
Etag	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*



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.