BCS502 COMPUTER NETWORKS

Module-5: Application Layer

Ch. 25. Introduction to Application Layer: Introduction, Client-Server Programming
Ch. 26. Standard Client-Server Protocols: World Wide Web and HTTP, FTP, Electronic Mail, Domain Name System (DNS), TELNET, Secure Shell (SSH).

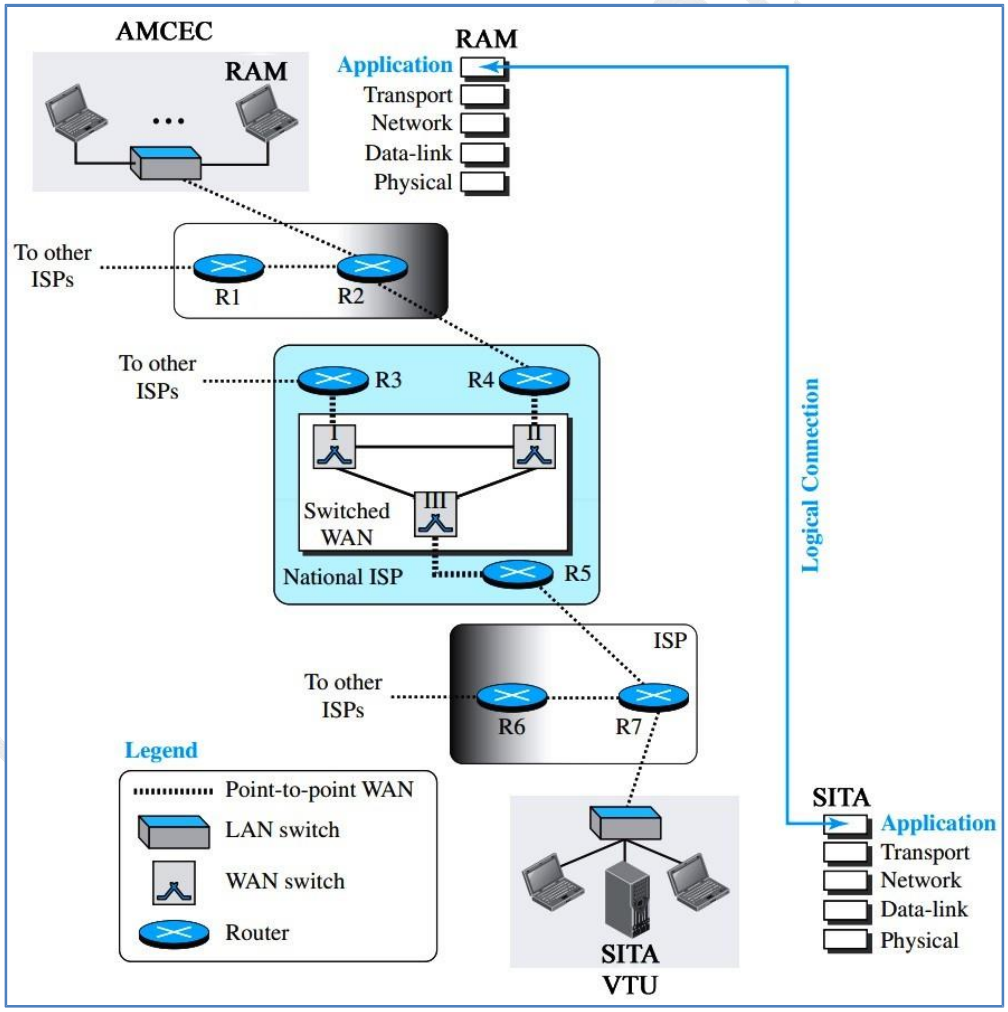
Textbook: Ch. 25.1-25.2, 26.1-26.6

Chapter 25: Introduction to Application Layer [P-854]

Introduction, Client-Server Programming

Introduction to Application Layer

- The whole Internet, hardware and software, was designed and developed to provide services at the application layer. The fifth layer of the TCP/IP protocol suite is where these services are provided for Internet users. The other four layers are there to make these services possible.
- Communication is provided using a logical connection, which means that the two application layers assume that there is an imaginary direct connection through which they can send and receive messages. Figure shows the idea behind this logical connection.



Providing Services

- ***The Internet was originally designed for the purpose: to provide service to users around the world***. The ***layered architecture of the TCP/IP protocol suite***, however, makes the ***Internet more flexible than other communication networks*** such as postal or telephone networks.

- Each layer in the suite was originally made up of one or more protocols, but new protocols can be added or some protocols can be removed or replaced by the Internet authorities. However, if a protocol is added to each layer, it should be designed in such a way that it uses the services provided by one of the protocols at the lower layer. If a protocol is removed from a layer, care should be taken to change the protocol at the next higher layer that supposedly uses the services of the removed protocol.
- The ***application layer*** is different from other layers in that it is the ***highest layer in the suite***. The protocols in this layer do not provide services to any other protocol in the suite; they ***only receive services from the protocols in the transport layer***. This means that protocols can be removed from this layer easily. New protocols can be also added to this layer as long as the new protocols can use the services provided by one of the transport-layer protocols.
- Since the application layer is the only layer that provides services to the Internet user, the flexibility of the application layer allows new application protocols to be easily added to the Internet, which has been occurring during the lifetime of the Internet. When the Internet was created, only a few application protocols were available to the users; today we cannot give a number for these protocols because new ones are being added constantly.

Standard Application-Layer Protocols

- There are several application-layer protocols that have been standardized and documented by the Internet authority, and we are using them in our daily interaction with the Internet. ***Each standard protocol is a pair of computer programs that interact with the user and the transport layer to provide a specific service to the user***.
- In the case of these application protocols, we should know what types of services they provide, how they work, and the options that we can use with these applications, and so on.
- The ***study of these protocols enables a network manager to easily solve the problems that may occur when using these protocols***. The deep understanding of how these protocols work will also give us some ideas about how to create new nonstandard protocols.

Nonstandard Application-Layer Protocols

- A programmer can create a nonstandard application-layer program if you can write two programs that provide service to the user by interacting with the transport layer.
- It is the creation of a nonstandard (proprietary) protocol, which does not even need the approval of the Internet authorities if privately used, that has made the Internet so popular worldwide.
- A private company can create a new customized application protocol to communicate with all of its offices around the world using the services provided by the first four layers of the TCP/IP protocol suite without using any of the standard application programs.
- What is needed is to write programs, in one of the computer languages that ***use the available services provided by the transport-layer protocols***.

Application-Layer Paradigms

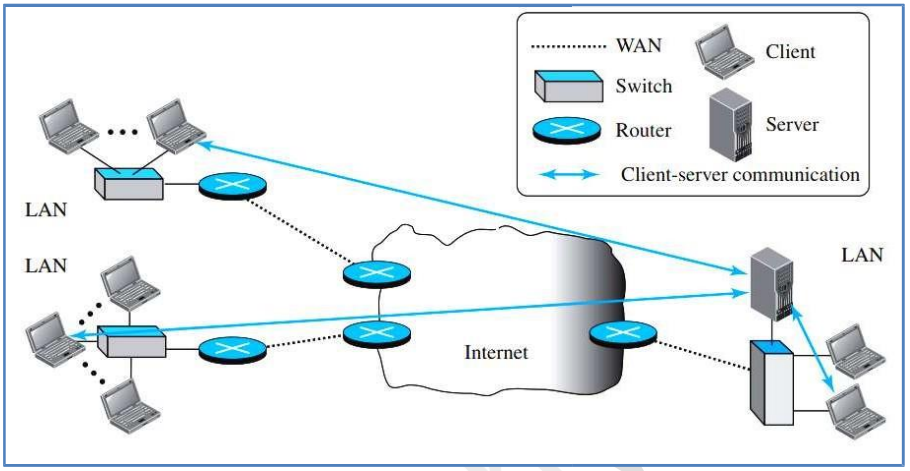
- To use the Internet we need ***two application programs*** to interact with each other
  - ***one running on a computer somewhere in the world***
  - ***the other running on another computer somewhere else in the world.***
- The ***two programs need to send messages to each other through the Internet infrastructure***.
- ***Two paradigms*** have been developed during the lifetime of the Internet:
  - ***the client-server paradigm***
  - ***the peer-to-peer paradigm.***

Traditional Paradigm: Client-Server

- The traditional paradigm is called the client-server paradigm. In this paradigm, ***the service provider*** is ***an application program***, ***called the server process***; it ***runs continuously***, ***waiting for another***

application program, called the ***client process***, to ***make a connection through the Internet and ask for service***.

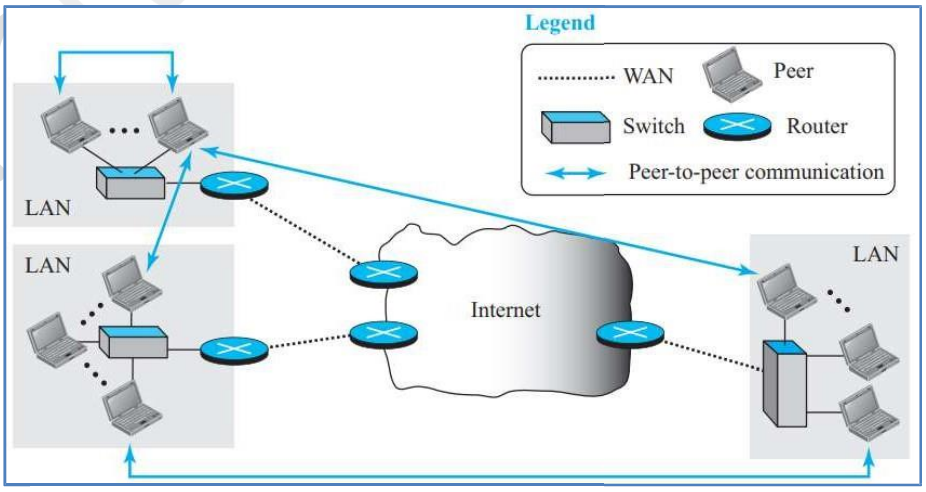
- There are normally some server processes that can provide a specific type of service, but there are many clients that request service from any of these server processes. The ***server process must be running all the time; the client process is started when the client needs to receive service***.
- Although the communication in the ***client-server paradigm*** is between ***two application programs***, the ***role of each program is totally different***. In other words, we cannot run a client program as a server program or vice versa.



- Figure shows an example of a client-server communication in which three clients communicate with one server to receive the services provided by this server.
- ***One problem*** with this paradigm is that the concentration of the ***communication load is on the server***, which means the server should be a powerful computer. Even a powerful computer may become overwhelmed if a large number of clients try to connect to the server at the same time.
- ***Another problem*** is that there should ***be a service provider willing to accept the cost and create a powerful server for a specific service***, which means the service must always return some type of income for the server in order to encourage such an arrangement. Several traditional services are still using this paradigm, including the ***World Wide Web (WWW)*** and its vehicle ***HyperText Transfer Protocol (HTTP), file transfer protocol (FTP), secure shell (SSH), e-mail***, and so on.

New Paradigm: Peer-to-Peer

- A new paradigm, called the ***peer-to-peer paradigm (P2P paradigm)*** has emerged to respond to the needs of some new applications. In this paradigm, there is no need for a server process to be running all the time and waiting for the client processes to connect. The ***responsibility is shared between peers***. A computer connected to the Internet can provide service at one time and receive service at another time. A computer can even provide and receive services at the same time. Figure shows an example of communication in this paradigm.



- One of the areas that really fits in this paradigm is the Internet telephony. Communication by phone is indeed a peer-to-peer activity; no party needs to be running forever waiting for the other party to call. Another area in which the peer-to-peer paradigm can be used is when some computers connected to the Internet have something to share with each other

- Although the peer-to-peer paradigm has been proved to be ***easily scalable and cost-effective in eliminating the need for expensive servers to be running and maintained all the time***, there are also some challenges.
- The main ***challenge has been security***; it is more difficult to create secure communication between distributed services than between those controlled by some dedicated servers.
- The ***other challenge is applicability***; it appears that not all applications can use this new paradigm. For example, not many Internet users are ready to become involved, if one day the Web can be implemented as a peer-to-peer service. There are ***some new applications***, such as ***BitTorrent, Skype, IPTV, and Internet telephony*** that use this paradigm.

Mixed Paradigm

- An application may choose to use a mixture of the two paradigms by combining the advantages of both.
- For example, a light-load client-server communication can be used to find the address of the peer that can offer a service. When the address of the peer is found, the actual service can be received from the peer by using the peer-topeer paradigm.

CLIENT-SERVER PROGRAMMING

Topics covered

- **Application Programming Interface**
- **Using Services of the Transport Layer**
- **Iterative Communication Using UDP**
- **Iterative Communication Using TCP**
- **Concurrent Communication**

- In a client-server paradigm, communication at the ***application layer is between two running application programs called processes: a client and a server***.
- A ***client*** is a running program that initializes the communication by sending a request; a server is another application program that waits for a request from a client.
- The ***server*** handles the request received from a client, prepares a result, and sends the result back to the client. This definition of a server implies that a server must be running when a request from a client arrives, but the client needs to be run only when it is needed.
- We need to be careful that the server program is started before we start running the client program. ***The lifetime of a server is infinite***: it should be started and run forever, waiting for the clients.
- The ***lifetime of a client is finite***: it normally sends a finite number of requests to the corresponding server, receives the responses, and stops.

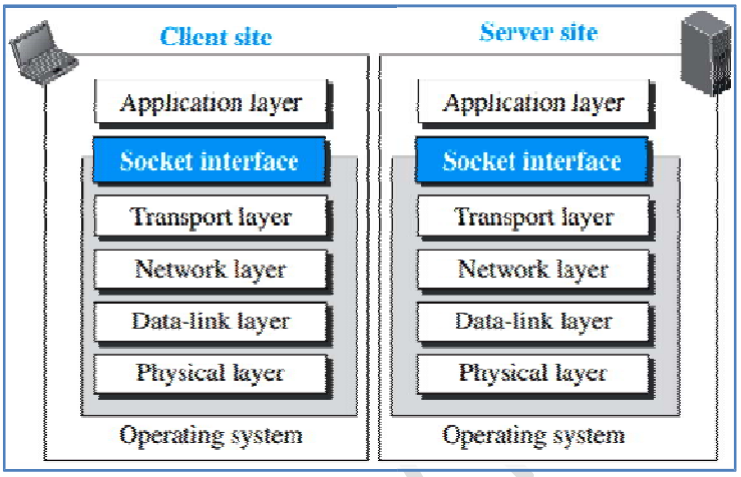
Application Programming Interface

(Sockets - Socket Addresses - Finding Socket Addresses - Server Site - Client Site)

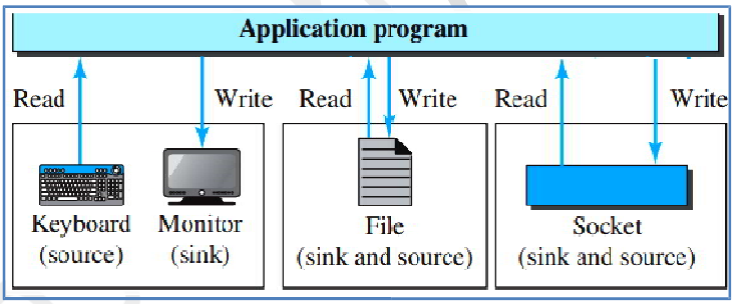
How can a client process communicate with a server process?

- If we need a process to be able to communicate with another process, we need a new set of instructions to tell the lowest four layers of the TCP/IP suite to open the connection, send and receive data from the other end, and close the connection. A set of instructions of this kind is normally referred to as an ***application programming interface (API).***
- An interface in programming is a set of instructions between two entities. In this case, ***one of the entities is the process at the application layer*** and the ***other is the operating system that encapsulates the first four layers of the TCP/IP protocol suite***.

- A ***computer manufacturer needs to build the first four layers of the suite in the operating system and include an API***. In this way, the processes running at the application layer are able to communicate with the operating system when sending and receiving messages through the Internet.



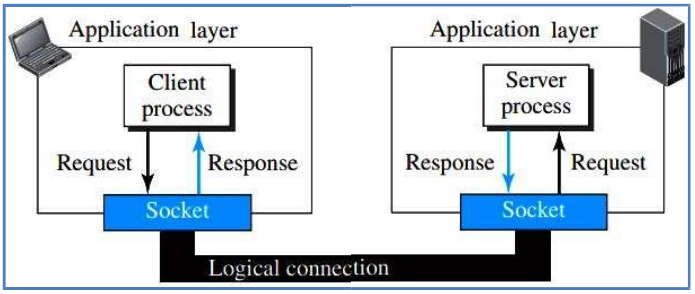
- ***Several APIs*** have been designed for communication. Three among them are common: ***socket interface, Transport Layer Interface (TLI), and STREAM***.
- Socket interface started in the early 1980s at UC Berkeley as part of a UNIX environment. The socket interface is a set of instructions that provide communication between the application layer and the operating system, ***as shown in Figure.***
- The idea of sockets allows us to use the set of all instructions already designed in a programming language for other sources and sinks. For example, in most computer languages, like C, C++, or Java, we have several instructions that can read and write data to other sources and sinks such as a keyboard (a source), a monitor (a sink), or a file (source and sink). We can use the same instructions to read from or write to sockets.



- We are ***adding only new sources and sinks to the programming language without changing the way we send data or receive data***. ***Figure shows the idea and compares the sockets with other sources and sinks.***

Sockets

- Socket is supposed to behave like a terminal or a file, it is not a physical entity like them; it is an abstraction. It is an object that is created and used by the application program.
- As far as the application layer is concerned, communication between a client process and a server process is communication between two sockets, created at two ends, ***as shown in Figure***.

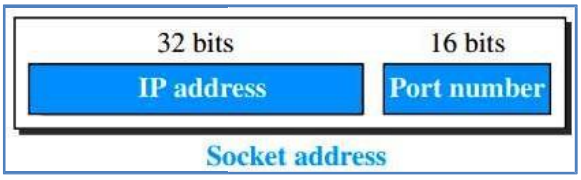


- The client thinks that the socket is the entity that receives the request and gives the response; the server thinks that the socket is the one that has a request and needs the response.
- If we create two sockets, one at each end, and define the source and destination addresses correctly, we can use the available instructions to send and receive data. The rest is the responsibility of the operating system and the embedded TCP/IP protocol.

Socket Addresses

- The ***interaction between a client and a server is two-way communication***. In a ***two-way*** communication, we ***need a pair of addresses***: ***local (sender)*** and ***remote (receiver).*** The local address in one direction is the remote address in the other direction and vice versa.
- Since communication in the client-server paradigm is between two sockets, ***we need a pair of socket addresses for communication: a local socket address and a remote socket address***. We need to define a socket address in terms of identifiers used in the TCP/IP protocol suite.

- A socket address should first define the computer on which a client or a server is running. A computer in the Internet is uniquely defined by its IP address, a 32-bit integer in the current Internet version.
- Several client or server processes may be running at the same time on a computer, which means that we need another identifier to define the specific client or server involved in the communication. ***An application program can be defined by a port number, a 16-bit integer***. This means that a ***socket address should be a combination of an IP address and a port number*** as shown in Figure.



- Since a socket defines the end-point of the communication, we can say that a socket is identified by a pair of socket addresses, a local and a remote.

Finding Socket Addresses - Server Site

- How can a client or a server find a pair of socket addresses for communication? The situation is different for each site.
- The server needs a local (server) and a remote (client) socket address for communication.
- ***Local Socket Address***: The local (server) socket address is ***provided by the operating system***. The ***operating system knows the IP address of the computer on which the server process is running***. The port number of a server process, however, needs to be assigned. If the server process is a standard one defined by the Internet authority, a port number is already assigned to it. - For example, the assigned port number for a Hypertext Transfer Protocol (HTTP) is the integer 80, which cannot be used by any other process. If the server process is not standard, the designer of the server process can choose a port number, in the range defined by the Internet authority, and assign it to the process. ***When a server starts running, it knows the local socket address***.
- ***Remote Socket Address***: The remote socket address ***for a server is the socket address of the client that makes the connection***. Since the server can serve many clients, it does not know beforehand the remote socket address for communication. The server can find this socket address when a client tries to connect to the server. The client socket address, which is contained in the request packet sent to the server, becomes the remote socket address that is used for responding to the client. In other words, although the local socket address for a server is fixed and used during its lifetime, the remote socket address is changed in each interaction with a different client.

Finding Socket Addresses - Client Site

- The ***client also needs a local (client) and a remote (server) socket address for communication***.
- ***Local Socket Address:*** The local (client) socket address is also provided by the operating system. The operating system knows the IP address of the computer on which the client is running. The port number, however, is a 16-bit temporary integer that is assigned to a client process each time the process needs to start the communication. The port number, however, needs to be assigned from a set of integers defined by the Internet authority and called the ephemeral (temporary) port numbers. The operating system, however, ***needs to guarantee that the new port number is not used by any other running client process***. The ***operating system needs to remember the port number to be able to redirect the response received from the server process to the client process that sent the request***.
- ***Remote Socket Address:*** Finding the remote (server) socket address for a client, however, needs more work. When a client process starts, it should know the socket address of the server it wants to connect to. We will have ***two situations*** in this case.
  1. Sometimes, the user who starts the client process knows both the server port number and IP address of the computer on which the server is running. This usually occurs in situations when we have written client and server applications and we want to test them. For example, write some simple client and server programs and we test them using this approach. In this

situation, the programmer can provide these two pieces of information when he runs the client program.

2. Although each standard application has a well-known port number, most of the time, we do not know the IP address. This happens in situations such as when we need to contact a web page, send an e-mail to a friend, copy a file from a remote site, and so on. In these situations, the server has a name, an identifier that uniquely defines the server process. Examples of these identifiers are URLs, such as www.xxx.yyy, or e-mail addresses, such as xxxx@yyyy.com. The client process should now change this identifier (name) to the corresponding server socket address. The client process normally knows the port number because it should be a well-known port number, but the IP address can be obtained using another client-server application called the ***Domain Name System (DNS).***

Using Services of the Transport Layer

- A pair of processes provide services to the users of the Internet, human or programs.
- A pair of processes need to use the services provided by the transport layer for communication because there is no physical communication at the application layer.
- There are three common transport-layer protocols in the TCP/IP suite: UDP, TCP, and SCTP. Most standard applications have been designed to use the services of one of these protocols.
- When we write a new application, we can decide which protocol we want to use. The choice of the transport layer protocol seriously affects the capability of the application processes.

UDP Protocol

- UDP provides connectionless, unreliable, datagram service. Connectionless service means that there is no logical connection between the two ends exchanging messages. Each message is an independent entity encapsulated in a datagram. UDP does not see any relation (connection) between consequent datagrams coming from the same source and going to the same destination.
- UDP is not a reliable protocol. Although it may check that the data is not corrupted during the transmission, it does not ask the sender to resend the corrupted or lost datagram.
- For some applications, UDP has an advantage: it is message-oriented. It gives boundaries to the messages exchanged. An application program may be designed to use UDP if it is sending small messages and the simplicity and speed is more important for the application than reliability. For example, some management and multimedia applications fit in this category.

TCP Protocol

- TCP provides connection-oriented, reliable, byte-stream service. TCP requires that two ends first create a logical connection between themselves by exchanging some connection-establishment packets. This phase, which is sometimes called ***handshaking***, establishes some parameters between the two ends, including the size of the data packets to be exchanged, the size of buffers to be used for holding the chunks of data until the whole message arrives, and so on.
- After the handshaking process, the two ends can send chunks of data in segments in each direction. By numbering the bytes exchanged, the continuity of the bytes can be checked.
- For example, if some bytes are lost or corrupted, the receiver can request the resending of those bytes, which makes TCP a reliable protocol. TCP also can ***provide flow control and congestion control***.
- ***One problem with the TCP protocol is that it is not message-oriented; it does not put boundaries on the messages exchanged***. Most of the standard applications that need to send long messages and require reliability may benefit from the service of the TCP.

SCTP Protocol

- SCTP provides a service which is a combination of the two other protocols. Like TCP, SCTP provides a connection-oriented, reliable service, but it is not byte-stream oriented.

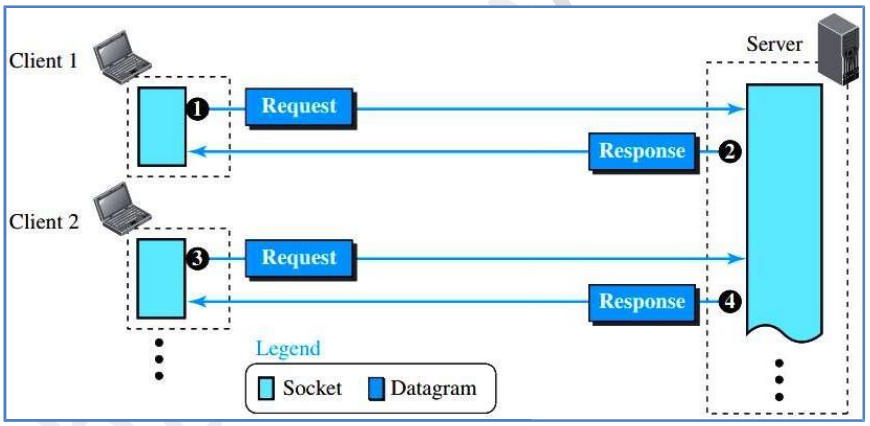
- It is a message-oriented protocol like UDP. In addition, SCTP can provide multi-stream service by providing multiple network-layer connections.
- SCTP is normally suitable for any application that needs reliability and at the same time needs to remain connected, even if a failure occurs in one network-layer connection.

Iterative Communication Using UDP

- Communication between a client program and a server program can occur iteratively or concurrently. Although several client programs can access the same server program at the same time, the server program can be designed to respond iteratively or concurrently.
- An iterative server can process one client request at a time; it receives a request, processes it, and sends the response to the requestor before handling another request. When the server is handling the request from a client, the requests from other clients, and even other requests from the same client, need to be queued at the server site and wait for the server to be freed.
- The ***received and queued requests are handled in the first-in, first-out fashion. In this section, we discuss iterative communication using UDP***.

Sockets Used for UDP

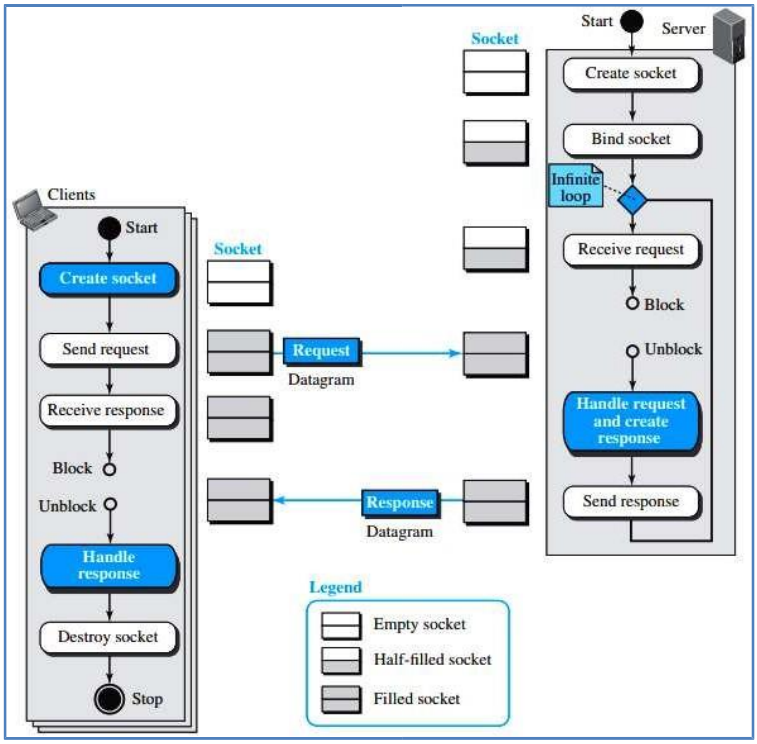
- In UDP communication, the client and server use only one socket each. The socket created at the server site lasts forever; the socket created at the client site is closed (destroyed) when the client process terminates. Figure shows the lifetime of the sockets in the server and client processes.



- ***Different clients use different sockets, but the server creates only one socket and changes only the remote socket address each time a new client makes a connection***. ***This is logical***, because the server does know its own socket address, but does not know the socket addresses of the clients who need its services; it needs to wait for the client to connect before filling this part of the socket address.

Flow Diagram

- UDP provides a connectionless service, in which a client sends a request and the server sends back a response. Figure shows a simplified flow diagram for iterative communication.
- There are multiple clients, but only one server. Each client is served in each iteration of the loop in the server. Note that there is no connection establishment or connection termination.
- Each client sends a single datagram and receives a single datagram.



- In other words, if a client wants to send two datagrams, it is considered as two clients for the server. The second datagram needs to wait for its turn. The diagram also shows the status of the socket after each action.

Server Process

- The server makes a ***passive open***, in which it becomes ready for the communication, but ***it waits until a client process makes the connection***. It ***creates an empty socket***. It then binds the socket to the server and the well-know port, in which only ***part of the socket*** (the server socket address) is filled (binding can happen at the time of creation depending on the underlying language).
- The ***server then issues a receive request command, which blocks until it receives a request from a client***. The server then fills the rest of the socket (the client socket section) from the information obtained in the request. The request is the process and the response is sent back to the client.
- The ***server now starts another iteration waiting for another request to arrive*** (an infinite loop). Note that in each iteration, the socket becomes only half-filled again; the client socket address is erased. It is totally filled only when a request arrives.

Client Process

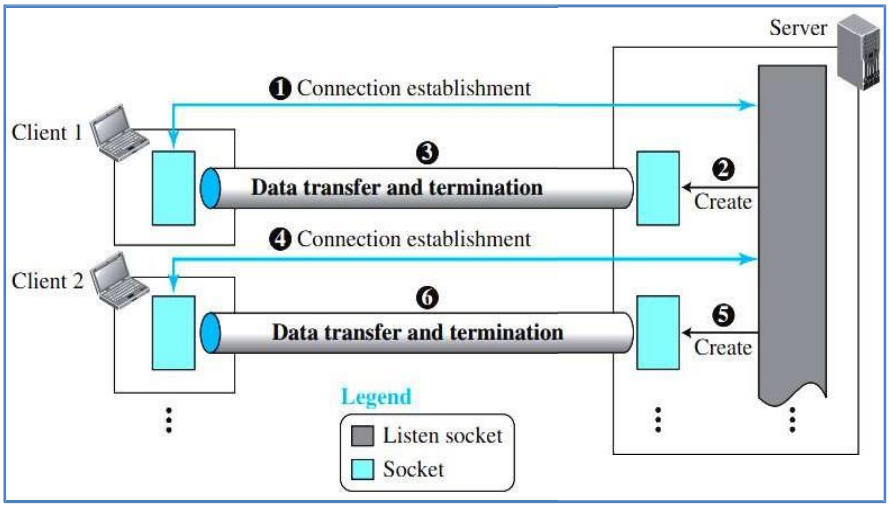
- The ***client process makes an active open***.
- It starts a connection. It creates an empty socket and then issues the send command, which fully fills the socket, and sends the request.
- The client then issues a receive command, which is blocked until a response arrives from the server. The response is then handled and the socket is destroyed.

Iterative Communication Using TCP

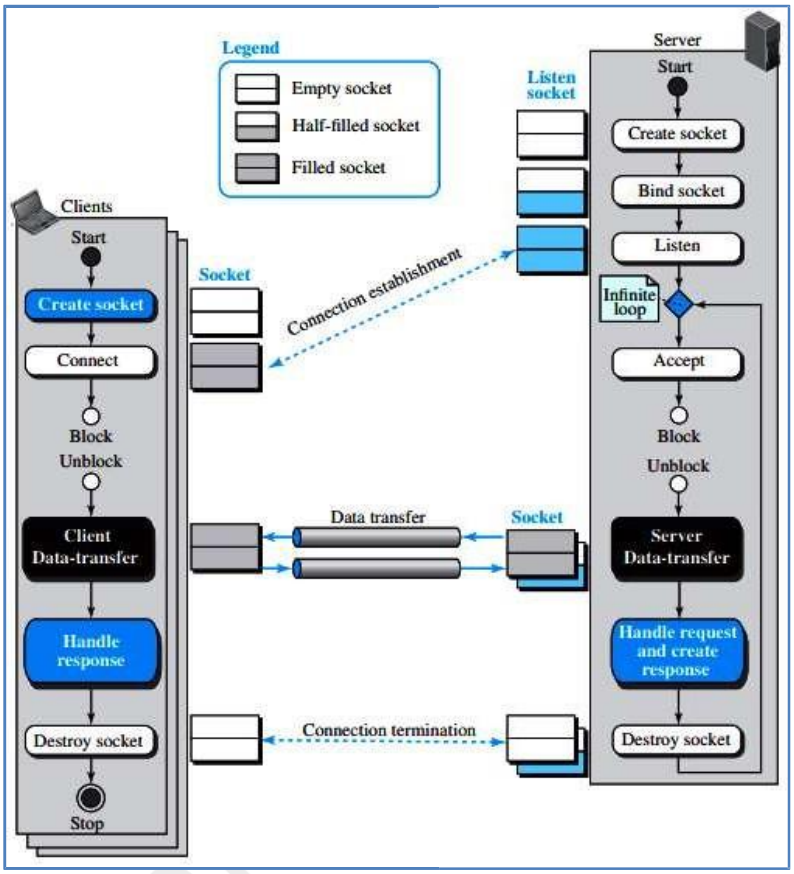
- TCP is a connection-oriented protocol.
- Before sending or receiving data, a connection needs to be established between the client and the server.
- After the connection is established, the two parties can send and receive chunks of data as long as they have data to do so.
- Although iterative communication using TCP is not very common, because it is simpler.

Sockets Used in TCP

- The TCP server uses ***two different sockets***, ***one for connection establishment (***listen socket***)*** and the ***other for data transfer (***socket***)***. The reason for having two types of sockets is to separate the connection phase from the data exchange phase.
- A server uses a listen socket to listen for a new client trying to establish connection.
- After the connection is established, the server creates a socket to exchange data with the client and finally to terminate the connection. The client uses only one socket for both connection establishment and data exchange (see Figure).



Flow Diagram



- Figure shows a simplified flow diagram for iterative communication using TCP. There are multiple clients, but only one server.
- Each client is served in each iteration of the loop. The flow diagram is almost similar to the one for UDP, but there are differences that we explain for each site.

Server Process

- In Figure, the TCP server process, like the UDP server process, creates a socket and binds it, but these two commands create the listen socket to be used only for the connection establishment phase.
- The server process then calls the listen procedure, to allow the operating system to start accepting the clients, completing the connection phase, and putting them in the waiting list to be served.
- The server process now starts a loop and serves the clients one by one. In each iteration, the server process issues the accept procedure that removes one client from the waiting list of the connected clients for serving.
- If the list is empty, the accept procedure blocks until there is a client to be served. When the accept procedure returns, it creates a new socket for data transfer. The server process now uses the client socket address obtained during the connection establishment to fill the remote socket address field in the newly created socket. At this time the client and server can exchange data.

Client Process

- The client flow diagram is almost similar to the UDP version except that the client data-transfer box needs to be defined for each specific case.

Concurrent Communication

- A concurrent server can process several client requests at the same time. This can be done using the available provisions in the underlying programming language.
- In C, a server can create several child processes, in which a child can handle a client. In Java, threading allows several clients to be handled by each thread.

Chapter 26: Standard Client-Server Protocols [P-854]

World Wide Web and HTTP, FTP, Electronic Mail, Domain Name System (DNS), TELNET, Secure Shell (SSH)

World Wide Web and HTTP

World Wide Web

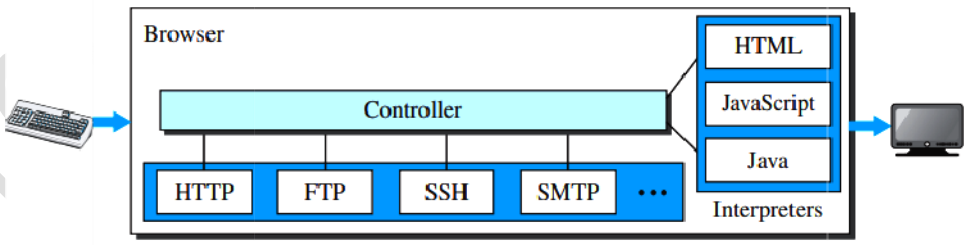
- The Web today is a repository of information in which the documents, called ***web pages***, are distributed all over the world and related documents are linked together.
- The popularity and growth of the Web can be related to two terms in the above statement: ***distributed*** and ***linked***.
-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

Architecture

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

Web Client (Browser)

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



- 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 such as ***HTTP or FTP***. The interpreter can be HTML, Java, or JavaScript, depending on the type of document. Some ***commercial browsers include Internet Explorer, Netscape Navigator, and Firefox***.

Web Server

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

Uniform Resource Locator (URL)

- A ***web page, as a file, needs to have a unique identifier to distinguish it from other web pages***.
- To define a web page, we need ***three identifiers***: ***host, port, and path***. However, before defining the web page, we need to tell the browser what client-server application we want to use, which is called the protocol. This means we need ***four identifiers to define the web page***.
  - The first is the type of vehicle to be used to fetch the web page;
  - the last three make up the combination that defines the destination object (web page).
- ***Protocol***: The first identifier is the abbreviation for the client-server program that we need in order to access the web page. Although most of the time the protocol is ***HTTP*** (HyperText Transfer Protocol), ***FTP*** (File Transfer Protocol).
- ***Host:*** The host identifier can be the IP address of the server or the unique name given to the server. IP addresses can be defined in dotted decimal notation (such as ***64.23.56.17***); the name is normally the domain name that uniquely defines the host, such as ***thagadoor.in***.
- ***Port:*** The port, a ***16-bit integer***, is normally ***predefined for the client-server application***. For example, if ***the HTTP protocol*** is ***used for accessing the web page***, the well-known ***port number is 80***. However, if a different port is used, the number can be explicitly given.
- ***Path:*** The path identifies the location and the name of the file in the underlying operating system. The format of this identifier normally depends on the operating system. In UNIX, a path is a set of directory names followed by the file name, all separated by a slash. For example, ***/vtu/bcs502/*** is a path that uniquely defines a file named ***bcs502***, stored in the directory last, which itself is part of the directory next, which itself is under the directory top.
- To combine these four pieces together, the uniform resource locator (URL) has been designed; it uses three different separators between the four pieces as shown below:

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

Web Documents

- The documents in the ***WWW*** can be grouped into ***3 broad categories***: ***static***, ***dynamic***, and ***active***.

Static Documents

- ***Static documents*** are ***fixed-content documents*** that are created and stored in a server. The client can get a copy of the document only. When a client accesses the document, a copy of the document is sent. The user can then use a browser to see the document. Static documents are ***prepared using one of several languages:***
  - ***HyperText Markup Language (HTML)***
  - ***Extensible Markup Language (XML)***
  - ***Extensible Style Language (XSL)***
  - ***Extensible Hypertext Markup Language (XHTML).***

Dynamic Documents

- A ***dynamic document is created by a web server whenever a browser requests the document***. When a request arrives, the web server runs an application program or a script that creates the dynamic document.

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

Active Documents

- For ***many applications, we need a program or a script to be run at the client site. These are called active documents***. For example, suppose we want to run a program that creates animated graphics on the screen or a program that interacts with the user.
- The program definitely needs to be run at the client site where the animation or interaction takes place. When a browser requests an active document, the server sends a copy of the document or a script. The ***document is then run at the client (browser) site***. ***One way to create*** an active document is to use ***Java applets***, a program written in ***Java on the server***. It is compiled and ready to be run. The document is in ***bytecode (binary) format***. Another way is to use ***JavaScripts*** but download and run the script at the client site.

HyperText Transfer Protocol (HTTP)

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

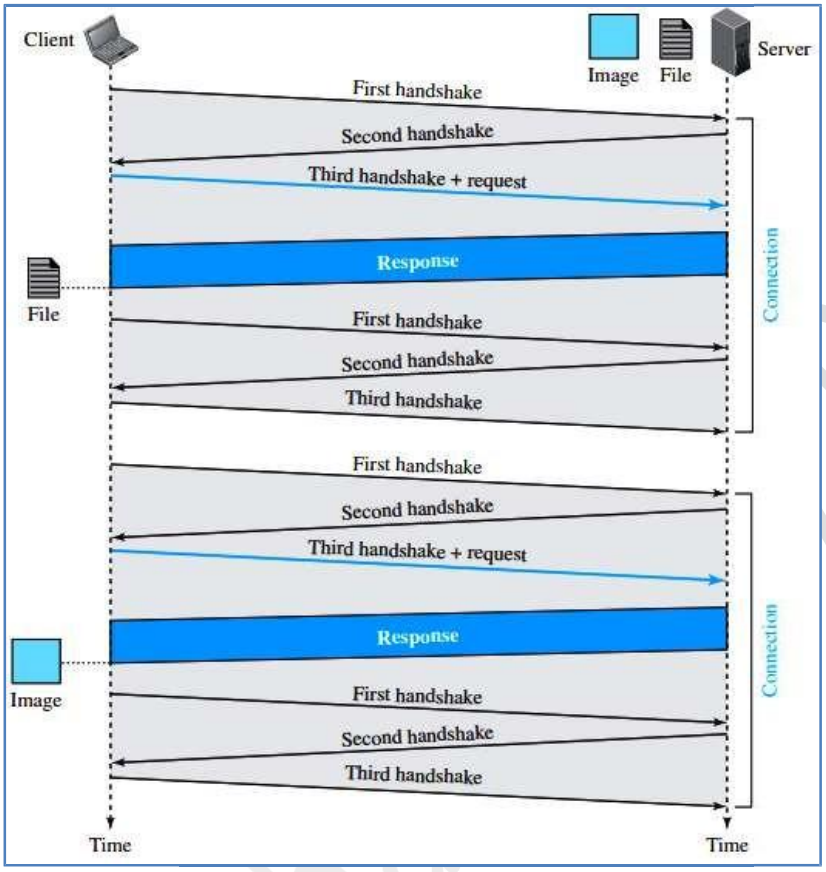
Non-persistent vs *Persistent Connections*

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

Non-persistent Connections

- In a non-persistent connection, one TCP connection is made for each ***request/response***. The following lists the steps in this strategy:
  1. The client opens a TCP connection and sends a request.
  2. The server sends the response and closes the connection.
  3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

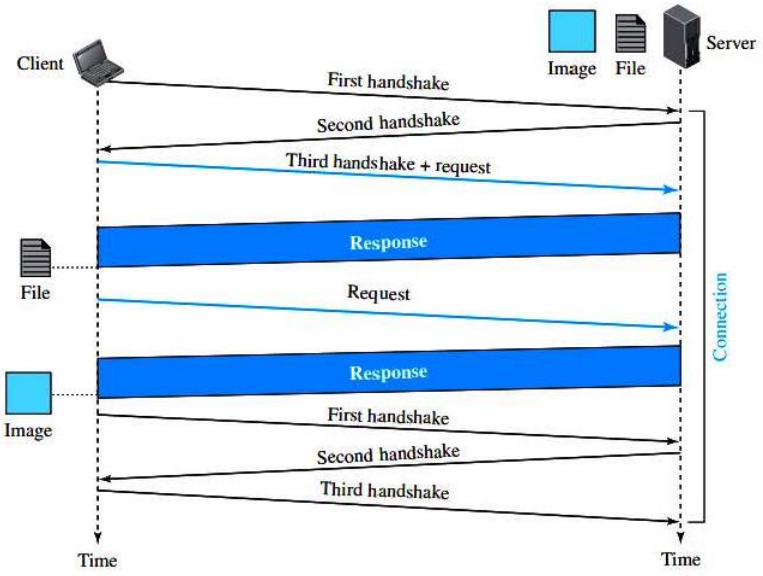
- In this strategy, if a file contains links to ***N different pictures*** in different files (all located on the same server), the connection must be opened and closed ***N + 1*** times. The non-persistent strategy imposes high overhead on the server because the server needs ***N + 1*** different buffers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



- The client needs to access a file that contains one link to an image. The text file and image are located on the same server. Here we need two connections. For each connection, TCP requires at least three handshake messages to establish the connection, but the request can be sent with the third one. After the connection is established, the object can be transferred. After receiving an object, another three handshake messages are needed to terminate the connection.

Persistent Connections

- HTTP version 1.1 specifies a persistent connection by default. In a persistent connection, the server leaves the connection open for more requests after sending a response.
- The server can close the connection at the request of a client or if a time-out has been reached. The sender usually sends the length of the data with each response.
- There are some occasions when the sender does not know the length of the data. This is the case when a document is created dynamically or actively. In these cases, the server informs the client that the length is not known and closes the connection after sending the data so the client knows that the end of the data has been reached.

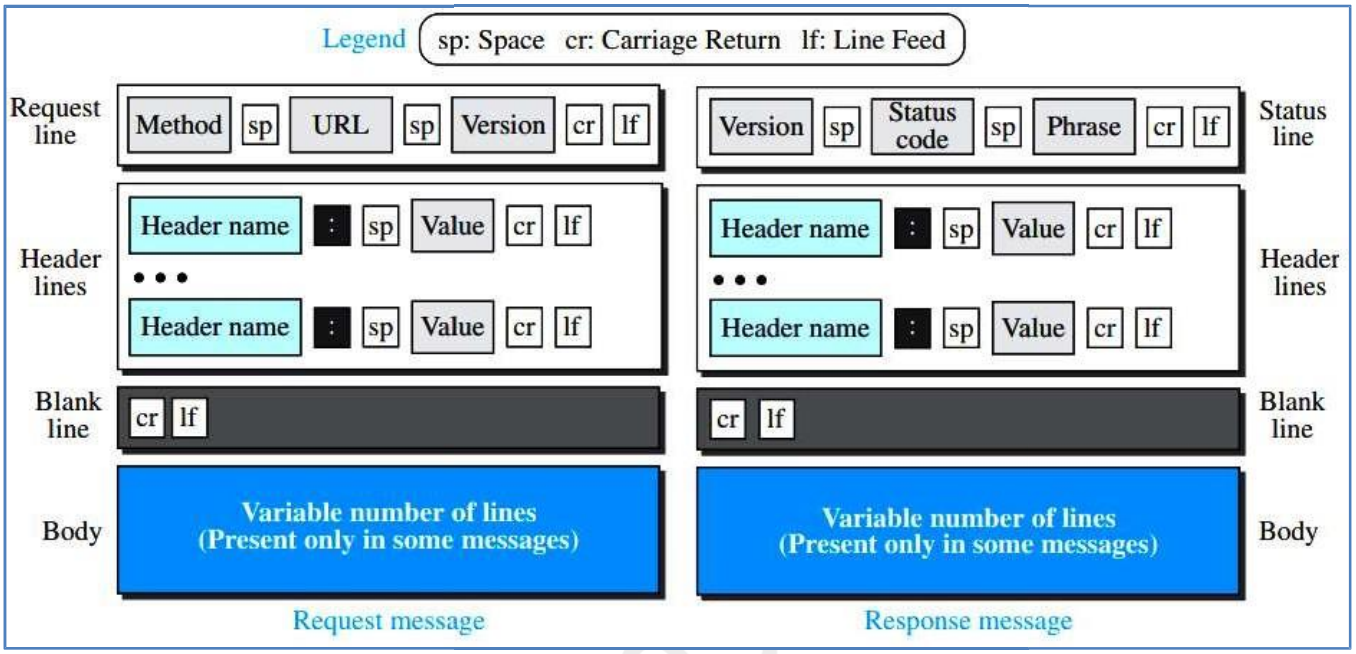


- ***Time and resources are saved using persistent connections***.

- Only one set of buffers and variables needs to be set for the connection at each site. The round trip time for connection establishment and connection termination is saved.

Message Formats

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



Request Message

- The first line in a request message is called a ***request line***. There are ***three fields*** in this line separated by one space and terminated by two characters (carriage return and line feed) as shown in Figure. The fields are called ***method***, ***URL***, and ***version***.
- The method field defines the request types. In version 1.1 of HTTP, several methods are defined, as shown in Table.

Table *Methods*

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

- Most of the time, the client uses the ***GET*** method to send a request. The ***PUT*** method is the inverse of the ***GET*** method; it allows the client to post a new web page on the server (if permitted). The POST method is similar to the ***PUT*** method, but it is used to send some information to the server to be added to the web page or to modify the web page.
- The ***TRACE*** method is used for debugging; the client asks the server to echo back the request to check whether the server is getting the requests. The ***DELETE*** method allows the client to delete a web page on the server if the client has permission to do so. The ***CONNECT*** method was originally made as a reserve method; it may be used by proxy servers, as discussed later. Finally, the ***OPTIONS*** method allows the client to ask about the properties of a web page.

- The ***second field***, URL. It ***defines the address and name of the corresponding web page***. The ***third field, version***, gives the version of the protocol; the most current version of HTTP is ***1.1***.
- After the request line, we can have zero or more request header lines. Each header line sends additional information from the client to the server.
- For example, the client can request that the document be sent in a special format. Each header line has a ***header name, a colon, a space***, and a ***header value***. The following ***Table shows some header names commonly used in a request***. The value field defines the values associated with each header name. The list of values can be found in the corresponding RFCs. The body can be present in a request message. Usually, it contains the comment to be sent or the file to be published on the website when the method is ***PUT*** or ***POST***.

Table *Request header names*

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

Response Message

- A ***response message consists of a status line, header lines, a blank line, and sometimes a body***. The ***first line*** in a response message is called the ***status line***.
- There are ***three fields*** in this line separated by spaces and terminated by a carriage return and line feed. The ***first field*** defines the ***version of HTTP*** protocol, currently ***1.1.*** The ***status code field*** defines the status of the request. It consists of ***three digits***. Whereas the codes in the 100 range are only informational, the codes in the 200 range indicate a successful request. The codes in the 300 range redirect the client to another URL, and the codes in the 400 range indicate an error at the client site. Finally, the codes in the 500 range indicate an error at the server site. The status phrase explains the status code in text form.

Table 26.3 *Response header names*

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

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

Conditional Request

- A ***client can add a condition in its request***. In this case, the ***server will send the requested web page if the condition is met or inform the client otherwise***. One of the most common conditions imposed by the client is the time and date the web page is modified. The client can send the header line If-Modified-Since with the request to tell the server that it needs the page only if it is modified after a certain point in time.

Cookies

- The World Wide Web was originally designed as a stateless entity. A client sends a request; a server responds. Their relationship is over. The original purpose of the Web, retrieving publicly available documents, exactly fits this design. Today the Web has other functions that need to remember some information about the clients; some are listed below:
  - Websites are being used as electronic stores that allow users to browse through the store, select wanted items, put them in an electronic cart, and pay at the end with a credit card.
  - Some websites need to allow access to registered clients only.
  - Some websites are used as portals: the user selects the web pages he wants to see.
  - Some websites are just advertising agencies. For these purposes, the cookie mechanism was devised.

Creating and Storing Cookies

- The creation and storing of cookies depend on the implementation.
  - 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.
  - The server includes the cookie in the response that it sends to the client.
  - When the client receives the response, the browser stores the cookie in the cookie directory, which is sorted by the server domain name.

Using Cookies

- When a client sends a request to a server, the browser looks in the cookie directory to see if it can find a cookie sent by that server. If found, the cookie is included in the request. When the server receives the request, it knows that this is an old client, not a new one.
- The contents of the cookie are never read by the browser or disclosed to the user. It is a cookie made by the server and eaten by the server. Now let us see how a cookie is used for the four previously mentioned purposes:
  - ***An electronic store (e-commerce) can use a cookie for its client shoppers***. When a client selects an item and inserts it in a cart, a cookie that contains information about the item, such as its number and unit price, is sent to the browser. If the client selects a second item, the cookie is updated with the new selection information, and so on. When the client finishes shopping and wants to check out, the last cookie is retrieved and the total charge is calculated.
  - The site that ***restricts access to registered clients only*** sends a cookie to the client when the client registers for the first time. For any repeated access, only those clients that send the appropriate cookie are allowed.
  - A ***web portal uses the cookie in a similar way***. When a user selects her favorite pages, a cookie is made and sent. If the site is accessed again, the cookie is sent to the server to show what the client is looking for.
  - ***A cookie is also used by advertising agencies***. An advertising agency can place banner ads on some main website that is often visited by users. The advertising agency supplies only a URL that gives the advertising agency's address instead of the banner itself. When a user visits the main website and clicks the icon of a corporation, a request is sent to the advertising agency.
  - The advertising agency sends the requested banner, also includes a cookie with the ID of the user.

Web Caching

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

Proxy Server Location

- The proxy servers are normally located at the client site. This means that we can have a hierarchy of proxy servers, as shown below:
  - A client computer can also be used as a proxy server, in a small capacity, that stores responses to requests often invoked by the client.
  - In a company, a proxy server may be installed on the computer LAN to reduce the load going out of and coming into the LAN.
  - An ISP with many customers can install a proxy server to reduce the load going out of and coming into the ISP network.

Cache Update

How long a response should remain in the proxy server before being deleted and replaced?

- ***One solution is to store the list of sites whose information remains the same for a while.***
- For example, a news agency may change its news page every morning. This means that a proxy server can get the news early in the morning and keep it until the next day.
- Another recommendation is to add some headers to show the last modification time of the information. The proxy server can then use the information in this header to guess how long the information would be valid.

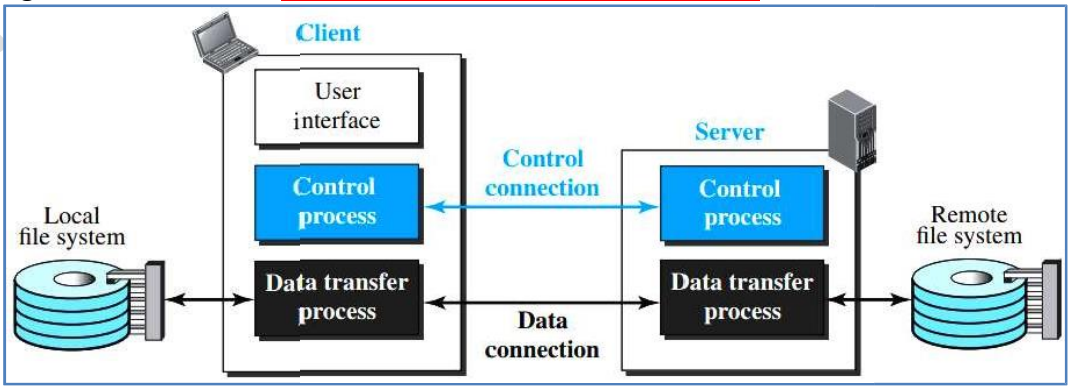
HTTP Security

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

FTP (File Transfer Protocol)

Topics covered: Two Connections - Control Connection - Data Connection - Security for FTP

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



- ***The client has three components***
  - The user interface
  - The client control process
  - The client data transfer process.
- ***The server has two components***
  - The server control process
  - The server data transfer process.
- The control connection is made between the control processes. The data connection is made between the data transfer processes.
- ***Separation of commands and data transfer makes FTP more efficient***. The control connection uses very simple rules of communication. We need to transfer only a line of command or a line of response at a time. The data connection, on the other hand, needs more complex rules due to the variety of data types transferred.

Two Connection

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

Control Connection

- ***For control communication, FTP uses*** the NVT ASCII character set as used by TELNET. Communication is achieved through commands and responses.
- Each line is terminated with a two-character (carriage return and line feed) end-of-line token. During this control connection, commands are sent from the client to the server and responses are sent from the server to the client.
- Commands, which are sent from the FTP client control process, are in the form of ASCII uppercase, which may or may not be followed by an argument. Some of the most common commands are shown in Table.

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

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

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

Data Connection

- The ***data connection uses the well-known port 20 at the server site***. However, the creation of a data connection is different from the control connection. The following shows the steps:
  - The client, not the server, issues a passive open using an ephemeral port. This must be done by the client because it is the client that issues the commands for transferring files.
  - Using the ***PORT command the client sends this port number to the server***.
  - The server receives the port number and issues an active open using the ***wellknown port 20*** and the ***received ephemeral port number***.

Communication over Data Connection

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

File Type

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

Data Structure

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

Transmission Mode

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

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

Security for FTP

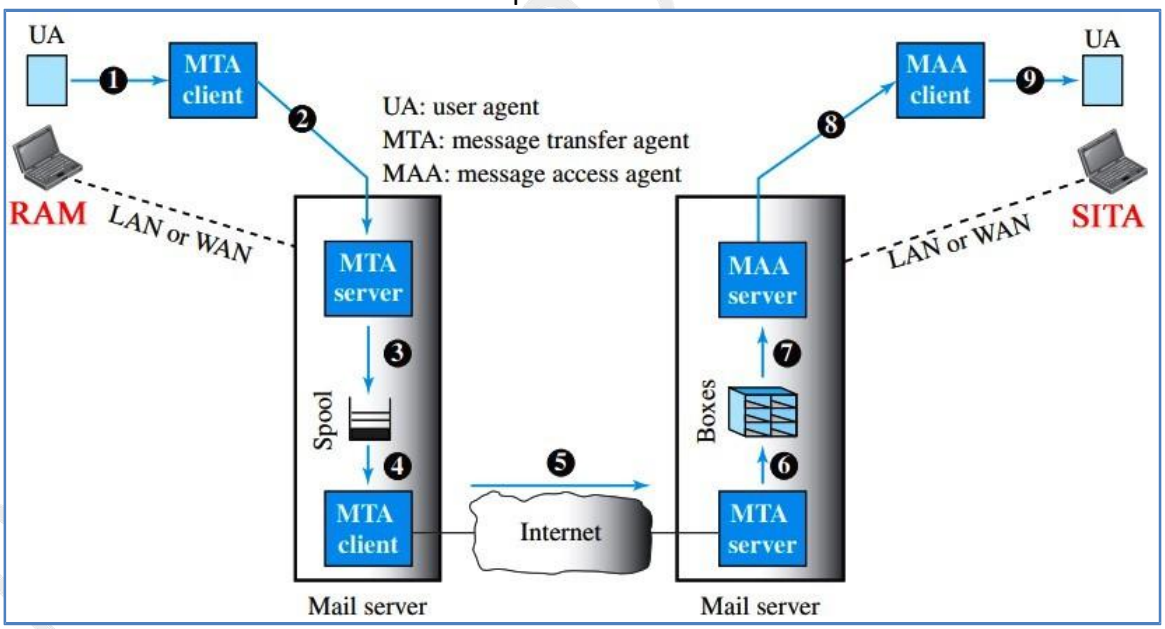
- The FTP protocol was designed when security was not a big issue.
- Although FTP requires a password, the password is sent in plaintext (unencrypted), which means it can be intercepted and used by an attacker.
- The data transfer connection also transfers data in plaintext, which is insecure.
- To be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

Electronic Mail

- ***Electronic mail (or e-mail) allows users to exchange messages.***
- The nature of this application, however, is different from other applications discussed so far. In an application such as HTTP or FTP, the server program is running all the time, waiting for a request from a client. When the request arrives, the server provides the service.
- There is a request and there is a response. In the case of electronic mail, the situation is different.
- First, e-mail is considered a one-way transaction.
- Second, it is neither feasible nor logical for a person1 to run a server program and wait until someone sends an e-mail to him. Person2 may turn off his computer when he is not using it.
- This means that the idea of client/server programming should be implemented in another way: using some ***intermediate computers (servers).*** The users run only client programs when they want and the ***intermediate servers apply the client/server paradigm.***

Architecture

- To explain the architecture of e-mail, we give a common scenario, as shown in Figure. Another possibility is the case in which ***RAM*** or ***SITA*** is directly connected to the corresponding mail server, in which LAN or WAN connection is not required.

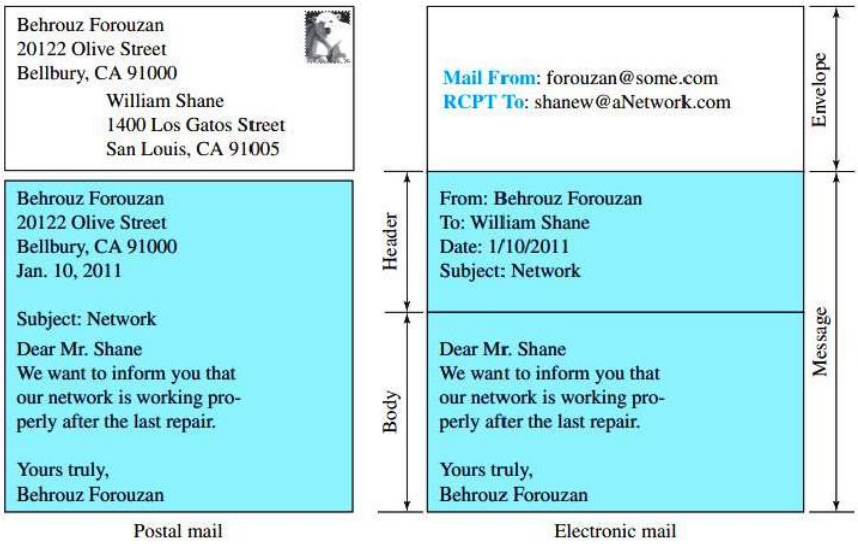


- In the common scenario, the sender and the receiver of the e-mail, RAM and SITA respectively, are connected via a LAN or a WAN to two mail servers. The administrator has created one mailbox for each user where the received messages are stored.
- A mailbox is part of a server hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it. The administrator has also created a queue (spool) to store messages waiting to be sent.
- A simple e-mail from RAM to SITA takes nine different steps, as shown in the figure. RAM and SITA use ***three different agents***: a ***user agent (UA),*** a ***message transfer agent (MTA),*** and a ***message access agent (MAA).***

- When RAM needs to send a message to SITA , she runs a UA program to prepare the message and send it to her mail server. The mail server at her site uses a queue (spool) to store messages waiting to be sent. The message, however, needs to be sent through the Internet from RAM's site to SITA 's site using an MTA. Here two message transfer agents are needed: one client and one server. Like most client-server programs on the Internet, the server needs to run all the time because it does not know when a client will ask for a connection. The client, on the other hand, can be triggered by the system when there is a message in the queue to be sent. The user agent at the SITA site allows SITA to read the received message. SITA later uses an MAA client to retrieve the message from an MAA server running on the second server.
- There are two important points we need to emphasize here.
  - ***First, SITA cannot bypass the mail server and use the MTA server directly.*** To use the MTA server directly, SITA would need to run the MTA server all the time because he does not know when a message will arrive. This implies that SITA 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, ***SITA needs another pair of client-server programs: message access programs.*** This is because an MTA client-server program is a push program: the client pushes the message to the server. SITA needs a pull program. The client needs to pull the message from the server. We discuss more about MAAs shortly.

User Agent

- The first component of an electronic mail system is the ***user agent (UA).*** It provides service to the user to make the process of sending and receiving a message easier. A ***user agent is a software package (program) that composes, reads, replies to, and forwards messages***. It ***also handles local mailboxes on the user computers***.
- There are ***two types of user agents***: ***command-driven*** and ***GUI-based***. Command-driven user agents belong to the early days of electronic mail. They are still present as the underlying user agents. A command-driven user agent normally accepts a onecharacter 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 commanddriven user agents are mail, pine, and elm.
- ***Modern user agents are GUI- based***. They contain graphical user interface (GUI) components that allow the user to interact with the software by using both the keyboard and the mouse. They have graphical components such as icons, menu bars, and windows that make the services easy to access. Some examples of GUI-based user agents are Eudora and Outlook.
- To ***send mail,*** the user, through the UA, creates mail that looks very similar to postal mail. It has an envelope and a message. The envelope usually contains the sender address, the receiver address, and other information. 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. The body of the message contains the actual information to be read by the recipient.



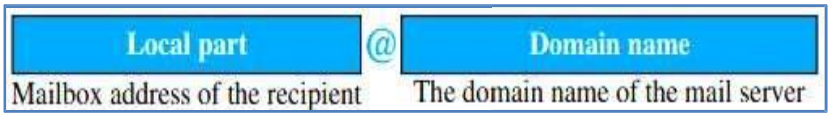
Receiving Mail

- The user agent is triggered by the user (or a timer). If a user has

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

Addresses

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

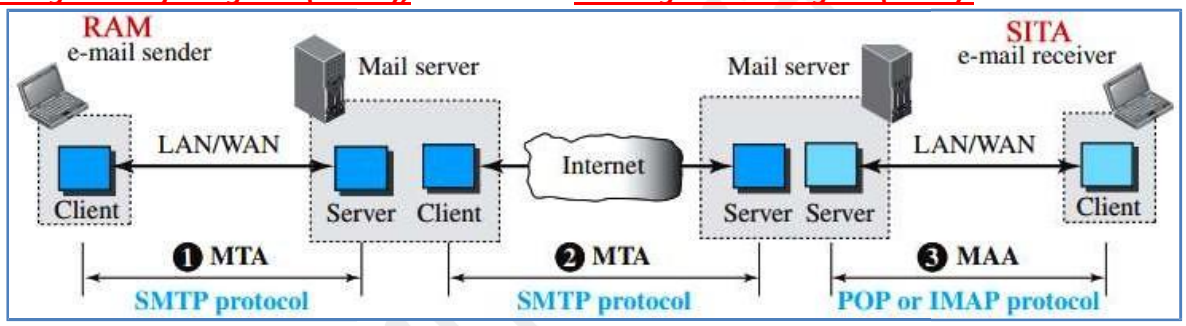


Mailing List or Group List

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

Message Transfer Agent: SMTP

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



- The formal protocol that defines the MTA client and server in the Internet is called ***Simple Mail Transfer Protocol (SMTP).*** SMTP is used two times, between the sender and the sender's mail server and between the two mail servers. As we will see shortly, another protocol is needed between the mail server and the receiver. SMTP simply defines how commands and responses must be sent back and forth.
- ***Commands and Responses:*** SMTP uses commands and responses to transfer messages between an MTA client and an MTA server. The command is from an MTA client to an MTA server; the response is from an MTA server to the MTA client. Each command or reply is terminated by a two character (carriage return and line feed) end-of-line token.
- Commands are sent from the client to the server. The format of a command is shown here-→

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

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

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

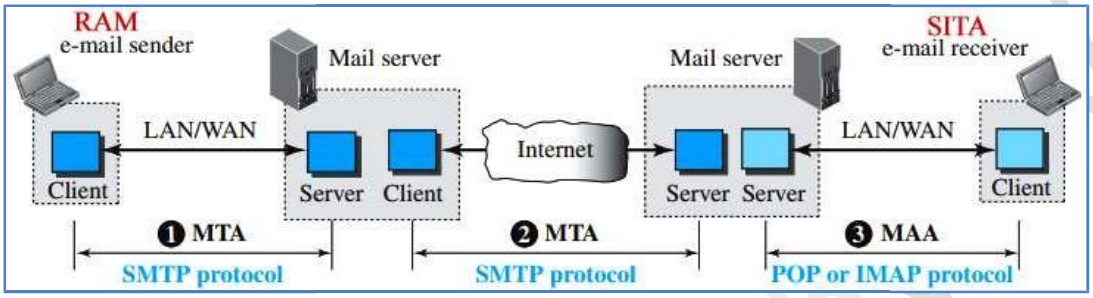
Code	*Description*
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command temporarily not implemented
550	Command is not executed; mailbox unavailable
551	User not local
552	Requested action aborted; exceeded storage location
553	Requested action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

Mail Transfer Phases

- The process of transferring a mail message occurs in ***three phases: connection establishment, mail transfer, and connection termination.***
- ***Connection Establishment***: After a client has made a TCP connection to the wellknown port 25, the SMTP server starts the connection phase. This phase involves the following three steps:
  - The server ***sends code 220 (service ready)*** to tell the client that it is ready to receive mail. If the server is not ready, it sends code 421 (service not available).
  - The ***client sends the HELO message to identify itself***, using its domain name address. This step is necessary to inform the server of the domain name of the client.
  - ***The server responds with code 250 (request command completed)*** or some other code depending on the situation.
- ***Message Transfer:*** After connection has been established between the SMTP client and server, a single message between a sender and one or more recipients can be exchanged. ***This phase involves eight steps.*** Steps 3 and 4 are repeated if there is more than one recipient.
  1. The ***client sends the MAIL FROM message to introduce the sender of the message***. It ***includes*** the ***mail address*** of the sender (mailbox and the domain name). This step is needed to give the server the return mail address for returning errors and reporting messages.
  2. The ***server responds with code 250*** or some other appropriate code.
  3. The ***client sends the RCPT TO (recipient) message***, which includes the mail address of the recipient.
  4. The ***server responds with code 250 or some other appropriate code***.
  5. The ***client sends the DATA message*** to initialize the message transfer.
  6. The ***server responds with code 354 (start mail input)*** or some other appropriate message.
  7. The ***client sends the contents of the message in consecutive lines***. Each line is terminated by a two-character end-of-line token (carriage return and line feed). The message is terminated by a line containing just one period.
  8. The ***server responds with code 250 (OK)*** or some other appropriate code.
- ***Connection Termination*** After the message is transferred successfully, the client terminates the connection. This ***phase involves two steps***.
  - The ***client sends the QUIT command***.
  - The ***server responds with code 221*** or some other appropriate code.

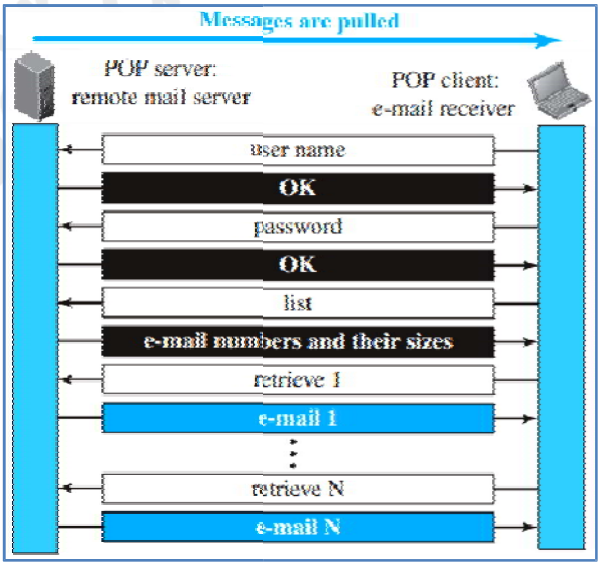
Message Access Agent: POP and IMAP

- The ***first and second stages of mail delivery use SMTP - is a push protocol***; it pushes the message from the client to the server.
- The ***direction of the bulk data (messages) is from the client to the server***. The ***third stage needs a pull protocol***; the client must pull messages from the server. The direction of the bulk data is from the server to the client. The ***third stage uses a message access agent***.
- Currently ***two message access protocols are available***: ***Post Office Protocol, version 3 (POP3)*** and ***Internet Mail Access Protocol, version 4 (IMAP4).*** Figure shows the position of these two protocols.



POP3

- ***Post Office Protocol, version 3 (POP3) is simple but limited in functionality***. The ***client POP3 software is installed on the recipient computer; the server POP3 software is installed on the mail server***.
- Mail access starts with the client when the user needs to download its e-mail from the mailbox on the mail server. ***The client opens a connection to the server on TCP port 110***.
- It then ***sends its user name and password to access the mailbox***. The user can then list and retrieve the mail messages, one by one. ***The following figure shows an example of downloading using POP3***. We have put the client on the right hand side because the e-mail receiver (Sita) is running the client process to pull messages from the remote mail server.



- ***POP3 has two modes***: the ***delete mode*** and the ***keep mode***.
- ***In the delete mode***
  - the mail is deleted from the mailbox after each retrieval.
  - used when the user is working at her permanent computer and can save and organize the received mail after reading or replying.
- ***In the keep mode***
  - the mail remains in the mailbox after retrieval.
  - used when the user accesses her mail away from her primary computer (for example, from a laptop). The mail is read but kept in the system for later retrieval and organizing.

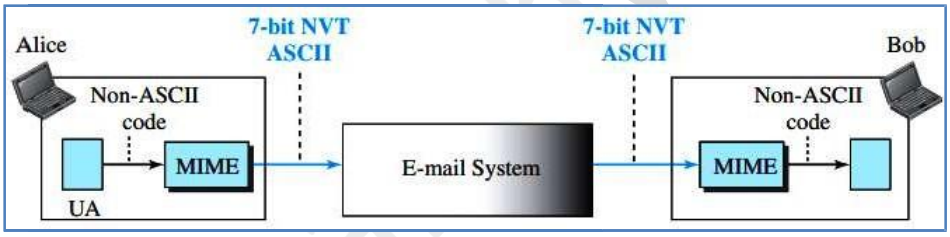
IMAP4

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

- POP3 is deficient in several ways. It does not allow the user to organize her mail on the server; the user cannot have different folders on the server. In addition, POP3 does not allow the user to partially check the contents of the mail before downloading.
- ***IMAP4 provides the following extra functions***:
  - A user can check the e-mail header prior to downloading.
  - A user can search the contents of the e-mail for a specific string of characters prior to downloading.
  - A user can partially download e-mail. This is especially useful if bandwidth is limited and the e-mail contains multimedia with high bandwidth requirements.
  - A user can create, delete, or rename mailboxes on the mail server.
  - A user can create a hierarchy of mailboxes in a folder for e-mail storage.

MIME

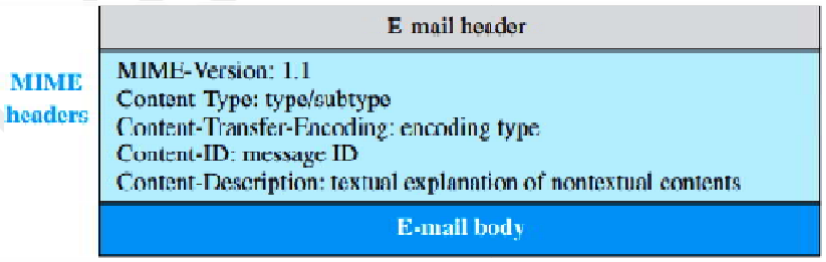
- Electronic mail has a simple structure. Its simplicity, however, comes with a price. It can send messages only in NVT 7-bit ASCII format.
- ***Multipurpose Internet Mail Extensions (MIME) is a supplementary protocol that allows non-ASCII data to be sent through e-mail***. MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the Internet. The message at the receiving site is transformed back to the original data.



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

MIME Headers

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



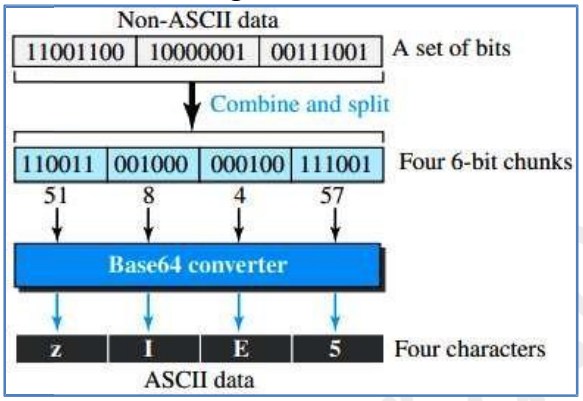
- ***MIME-Version:*** This header defines the version of MIME used. The current version is 1.1.
- ***Content-Type:*** This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters. MIME allows seven different types of data, listed in Table.

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

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

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

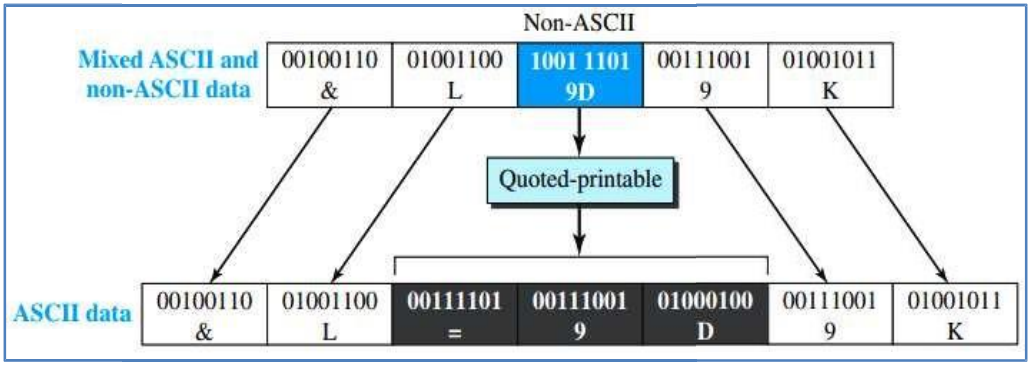
- The last two encoding methods are interesting. In the Base64 encoding, data, as a string of bits, is first divided into 6-bit chunks as shown in Figure.



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

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

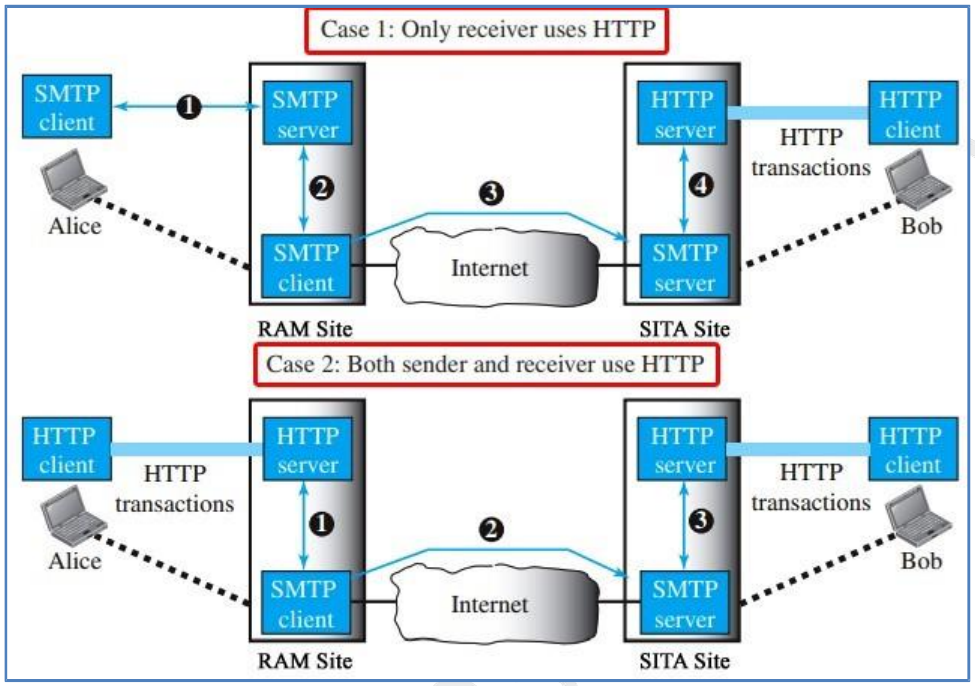
- Base64 is a redundant encoding scheme; that is, every six bits become one ***ASCII character and are sent as eight bits***. We have an overhead of 25%. If the data consist mostly of ASCII characters with a small non-ASCII portion, we can use quoted-printable encoding. In quoted-printable, if a character is ASCII, it is sent as is.
- If a ***character is not ASCII, it is sent as three characters***. The first character - equal sign (=). The next two characters are the hexadecimal representations of the byte. In Figure - The third character is a non-ASCII because it starts with bit 1. It is interpreted as two hexadecimal digits (9D16), which is replaced by three ASCII characters (=, 9, and D).



- ***Content-ID***: This header uniquely identifies the whole message in a multiple message environment.
- ***Content-Description***: This header defines whether the body is image, audio, or video.

Web-Based Mail

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



Case I

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

Case II

- ***In the second case, both RAM and SITA use web servers, but not necessarily the same server.***
- RAM sends the message to the web server using HTTP transactions. RAM sends an HTTP request message to her web server using the name and address of SITA's mailbox as the URL.
- The server at the RAM site passes the message to the SMTP client and sends it to the server at the SITA site using SMTP protocol. SITA receives the message using HTTP transactions.
- The message from the server at the RAM site to the server at the SITA site still takes place using SMTP protocol.

E-Mail Security

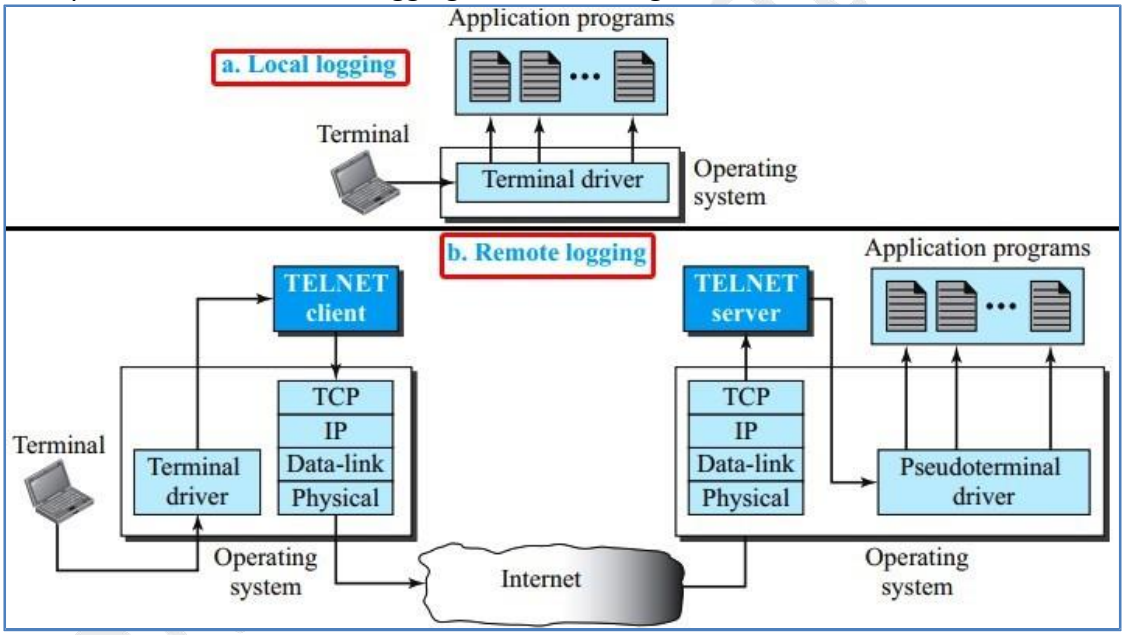
- E-mail exchanges can be secured using two application-layer securities designed in particular for e-mail systems. ***Two of these protocols, Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME).***

TELNET

- A server program can provide a specific service to its corresponding client program. For example, the FTP server is designed to let the FTP client store or retrieve files on the server site. However, it is impossible to have a client/server pair for each type of service we need; the number of servers soon becomes intractable.
- We refer to these generic client/server pairs as remote logging applications. One of the ***original remote logging protocols is TELNET, which is an abbreviation for TErminaL NETwork***. Although TELNET requires a logging name and password, it is vulnerable to hacking because it sends all data including the password in plaintext (not encrypted).
- A hacker can eavesdrop and obtain the logging name and password. Because of this security issue, the use of TELNET has diminished in favor of another protocol, ***Secure Shell (SSH)***. Although TELNET is almost replaced by SSH.
- ***TELNET here for two reasons***
  - The simple plaintext architecture of TELNET allows us to explain the issues and challenges related to the concept of remote logging, which is also used in SSH when it serves as a remote logging protocol.
  - Network administrators often use TELNET for diagnostic and debugging purposes.

Local versus Remote Logging

- The concept of local and remote logging as shown in Figure.

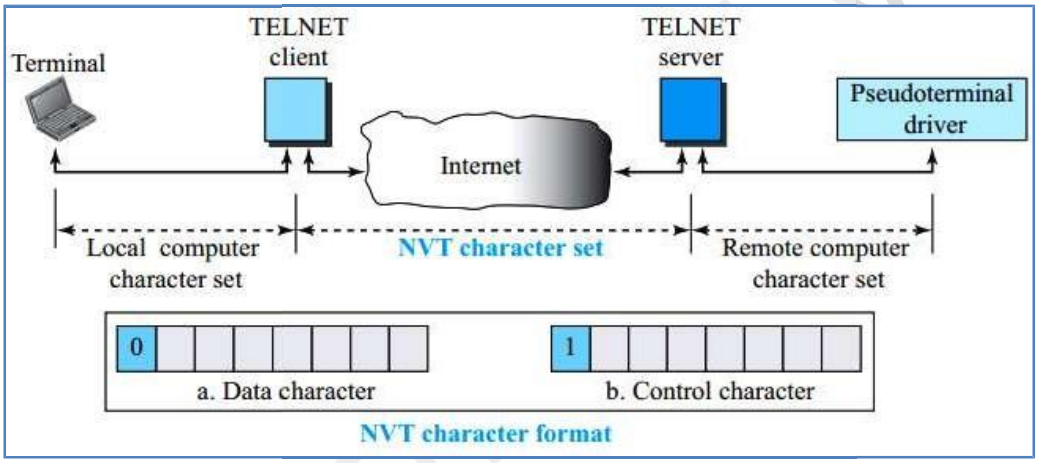


- When a user logs into a local system, it is called local logging. As a user types at a terminal or at a workstation running a terminal emulator, the keystrokes are accepted by the terminal driver. The terminal driver passes the characters to the operating system. The operating system, in turn, interprets the combination of characters and invokes the desired application program or utility.
- However, when a user wants to access an application program or utility located on a remote machine, she performs remote logging. Here the TELNET client and server programs come into use. The user sends the keystrokes to the terminal driver where the local operating system accepts the characters but does not interpret them. The characters are sent to the TELNET client, which transforms the characters into a universal character set called Network Virtual Terminal (NVT) characters (discussed below) and delivers them to the local TCP/IP stack.
- The commands or text, in NVT form, travel through the Internet and arrive at the TCP/IP stack at the remote machine. Here the characters are delivered to the operating system and passed to the TELNET server, which changes the characters to the corresponding characters understandable by the remote computer. However, the characters cannot be passed directly to the operating system

because the remote operating system is not designed to receive characters from a TELNET server; it is designed to receive characters from a terminal driver. The solution is to add a piece of software called a pseudoterminal driver, which pretends that the characters are coming from a terminal. The operating system then passes the characters to the appropriate application program.

Network Virtual Terminal (NVT)

- The mechanism to access a remote computer is complex. This is because every computer and its operating system accepts a special combination of characters as tokens. For example, the end-of-file token in a computer running the DOS operating system is Ctrl+z, while the UNIX operating system recognizes Ctrl+d. We are dealing with heterogeneous systems. If we want to access any remote computer in the world, we must first know what type of computer we will be connected to, and we must also install the specific terminal emulator used by that computer. TELNET solves this problem by defining a universal interface called the Network Virtual Terminal (NVT) character set. Via this interface, the client TELNET translates characters (data or commands) that come from the local terminal into NVT form and delivers them to the network. The server TELNET, on the other hand, translates data and commands from NVT form into the form acceptable by the remote computer. Figure shows the concept.



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

Options

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

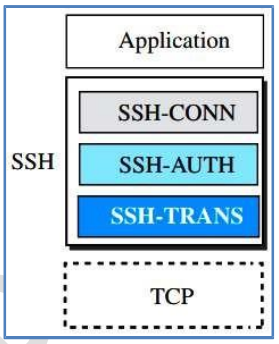
User Interface

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

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

Secure Shell (SSH) – Components - Applications

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



Components

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

SSH Transport-Layer Protocol (SSH-TRANS)

- Since TCP is not a secured transport-layer protocol, SSH first uses a protocol that creates a secured channel on top of the TCP. This new layer is an independent protocol referred to as ***SSH-TRANS***.
- When the procedure implementing this protocol is called, the client and server ***first use the TCP protocol to establish an insecure connection***. Then they exchange several security parameters to establish a secure channel on top of the TCP.
- ***The services provided by this protocol***
  - Privacy or confidentiality of the message exchanged.
  - Data integrity, which means that it is guaranteed that the messages exchanged between the client and server are not changed by an intruder.
  - Server authentication, which means that the client is now sure that the server is the one that it claims to be.
  - Compression of the messages, which improves the efficiency of the system and makes attack more difficult.

SSH Authentication Protocol (SSH-AUTH)

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

SSH Connection Protocol (SSH-CONN)

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

Applications

- Although SSH is often thought of as a replacement for TELNET, SSH is, in fact, a general-purpose protocol that provides a secure connection between a client and server.
  - SSH for Remote Logging
  - SSH for File Transfer
  - Port Forwarding
  - Format of the SSH Packets

SSH for Remote Logging

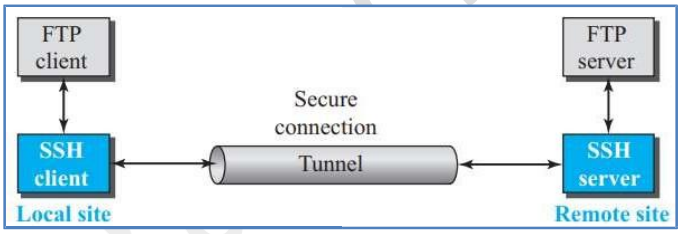
- Several free and commercial applications use SSH for remote logging. Among them, we can mention ***PuTTy***, by Simon Tatham, which is a client SSH program that can be used for remote logging. Another ***application program*** is ***Tectia***, which can be used on several platforms.

SSH for File Transfer

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

Port Forwarding

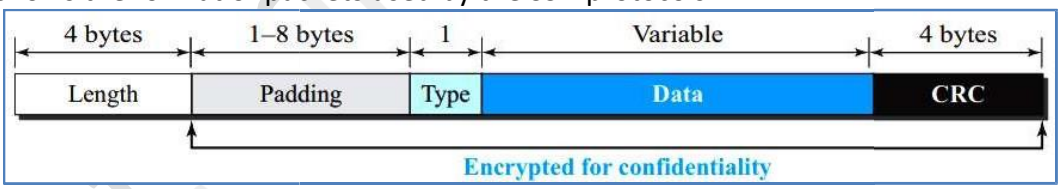
- One of the interesting services provided by the ***SSH protocol is port forwarding***. We can use the secured channels available in SSH to access an application program that does not provide security services.
- ***Applications such as TELNET and Simple Mail Transfer Protocol (SMTP)***, can use the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



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

Format of the SSH Packets

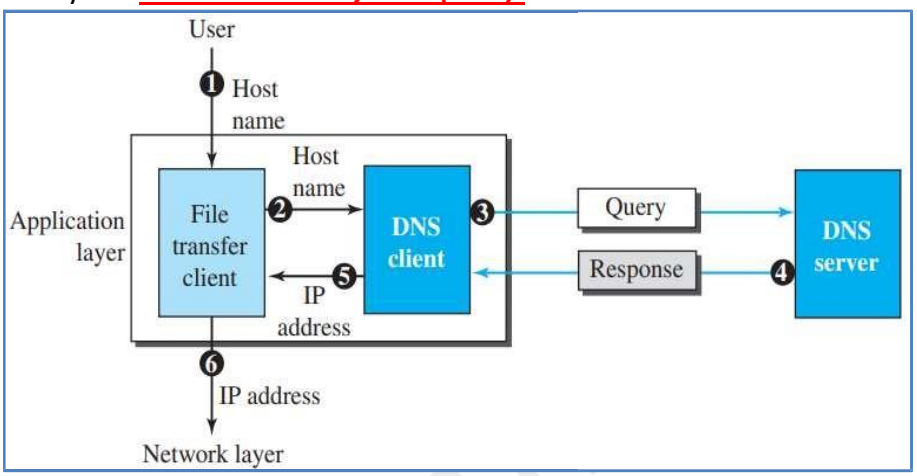
- Figure shows the format of packets used by the SSH protocols.



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

Domain Name System (DNS)

- Since the Internet is so huge today, a central directory system cannot hold all the mapping. In addition, if the central computer fails, the whole communication network will collapse.
- A better solution is to distribute the information among many computers in the world. In this method, the ***host that needs mapping can contact the closest computer holding the needed information***. This method is used by the ***Domain Name System (DNS).***
- Figure shows how TCP/IP uses a DNS client and a DNS server to map a name to an address. A user wants to use a file transfer client to access the corresponding file transfer server running on a remote host.



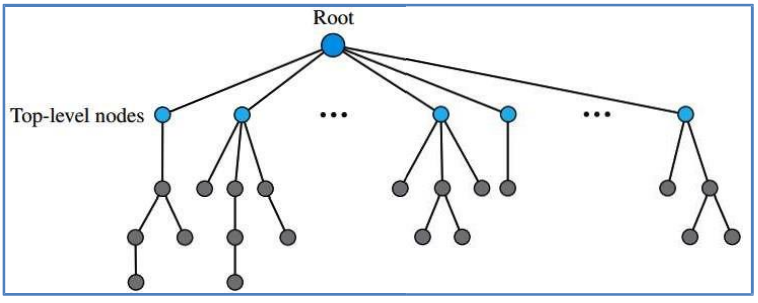
- The user knows only the file transfer server name, such as ***incerd.in***. However, the TCP/IP suite needs the IP address of the file transfer server to make the connection. The ***following six steps map the host name to an IP address***:
  - The user passes the host name to the file transfer client.
  - The file transfer client passes the host name to the DNS client.
  - Each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server name using the known IP address of the DNS server.
  - The DNS server responds with the IP address of the desired file transfer server.
  - The DNS server passes the IP address to the file transfer client.
  - The file transfer client now uses the received IP address to access the file transfer server.

Name Space

- The names must be unique because the addresses are unique. A name space that maps each address to a unique name can be organized in two ways: ***flat*** or ***hierarchical***.
- In a ***flat name space***, a name is assigned to an address. A name in this space is a sequence of characters without structure. The names may or may not have a common section; if they do, it has no meaning. The main disadvantage of a flat name space is that it cannot be used in a large system such as the Internet because it must be centrally controlled to avoid ambiguity and duplication.
- In a ***hierarchical name space***, ***each name is made of several parts***. The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on. In this case, the authority to assign and control the name spaces can be decentralized.

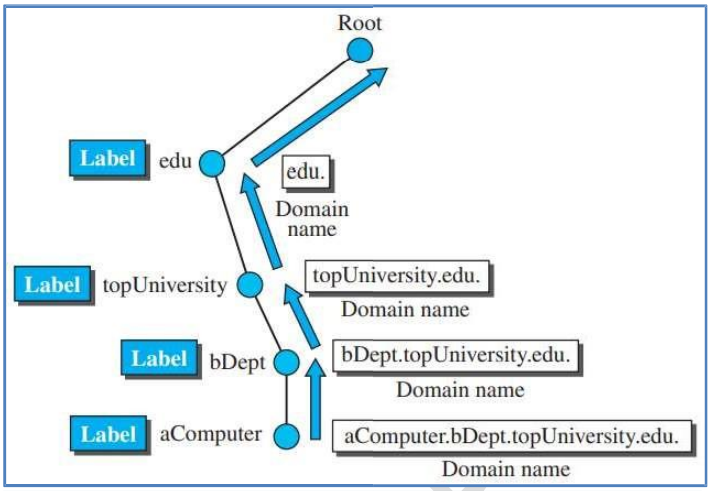
Domain Name Space

- 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).

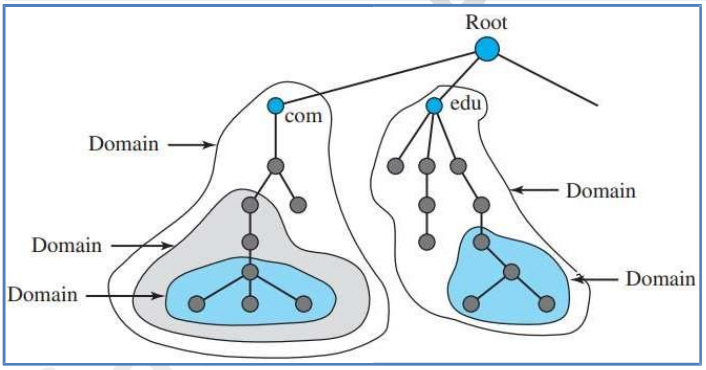


- ***Label***: Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that

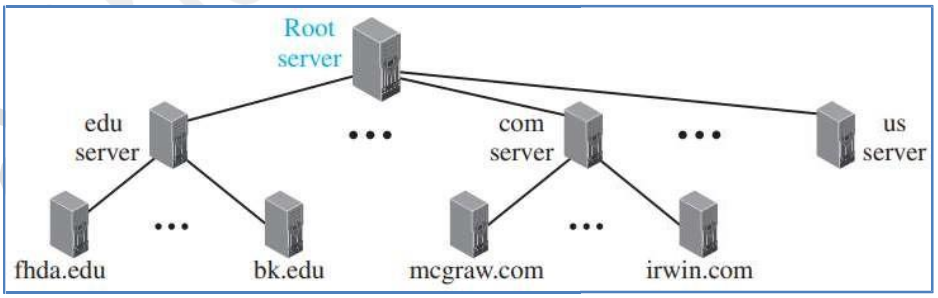
children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.



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

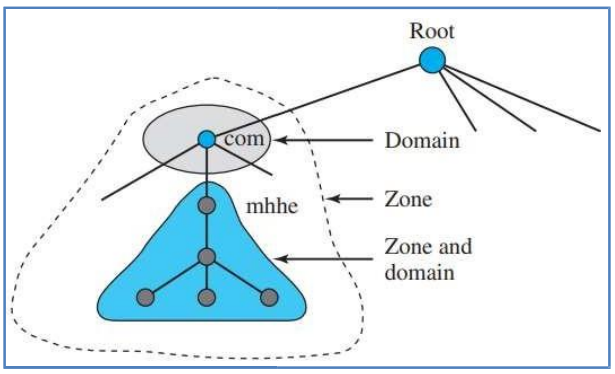


- ***Distribution of Name Space*** The information contained in the domain name space must be stored. However, it is very inefficient and also not reliable to have just one computer store such a huge amount of information. It is inefficient because responding to requests from all over the world places a heavy load on the system. It is not reliable because any failure makes the data inaccessible.
- ***Hierarchy of Name Servers*** The solution to these problems is to distribute the information among many computers called DNS servers. One way to do this is to divide the whole space into many domains based on the first level. In other words, we let the root stand alone and create as many domains (subtrees) as there are first-level nodes. Because a domain created this way could be very large, DNS allows domains to be divided further into smaller domains (sub domains). Each server can be responsible (authoritative) for either a large or small domain. In other words, we have a hierarchy of servers in the same way that we have a hierarchy of names (see Figure).



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***.

Root Server

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

Primary and Secondary Servers

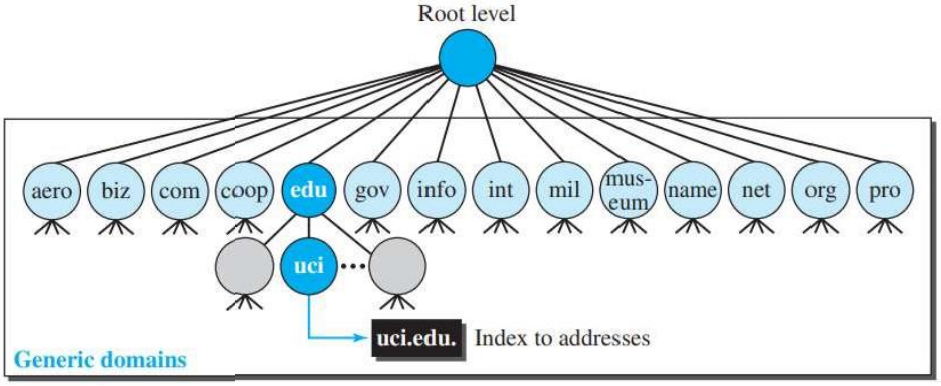
- DNS defines ***two types of servers: primary and secondary***. The primary and secondary servers are both authoritative for the zones they serve.
- A ***primary server is a server that stores a file about the zone for which it is an authority***. It is responsible for creating, maintaining, and updating 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.

DNS in the Internet

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

Generic Domains

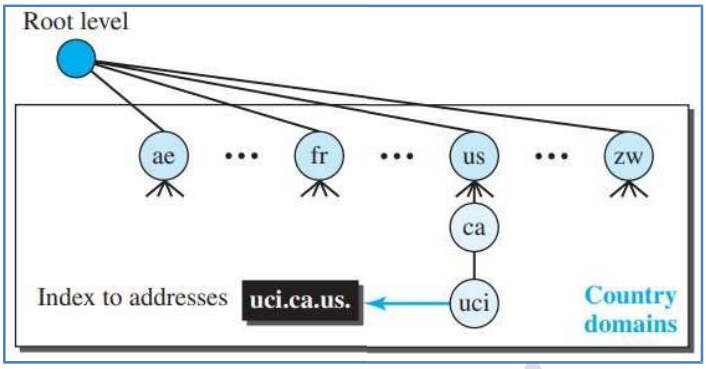
- The generic domains define registered hosts according to their generic behavior. Each node in the tree defines a domain, which is an index to the domain name space database (see Figure).



- 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.

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

Country Domains



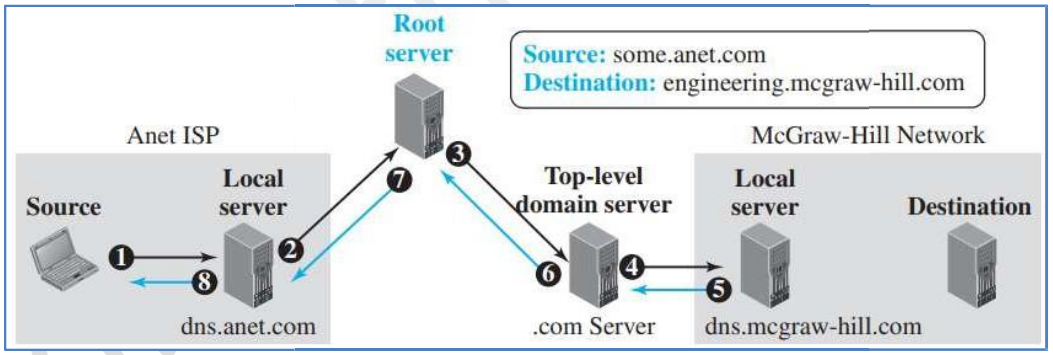
- The country domains section uses two-character country abbreviations (e.g., us for United States). Second labels can be organizational, or they can be more specific national designations. The United States, for example, uses state abbreviations as a subdivision of us (e.g., ca.us.). Figure shows the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

Resolution

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

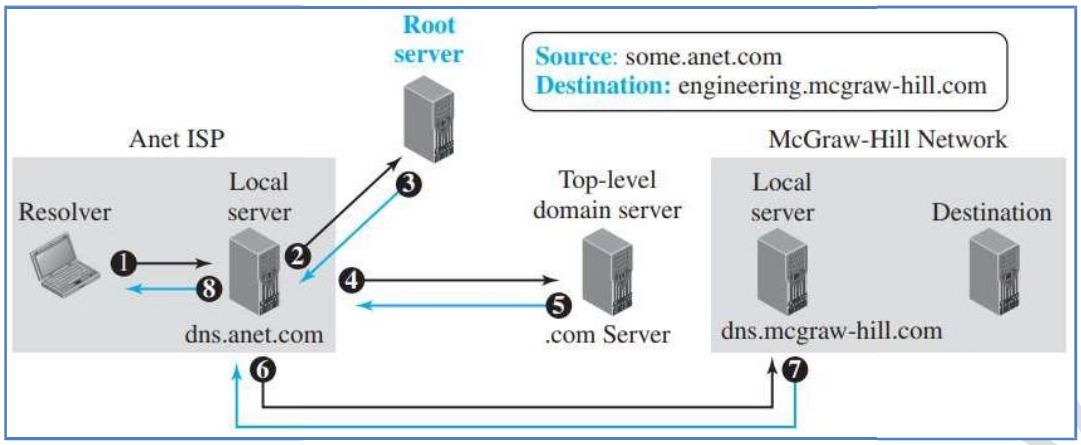
Recursive Resolution

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



Iterative Resolution

- In iterative resolution, each server that does not know the mapping sends the IP address of the next server back to the one that requested it.
- Figure shows the flow of information in an iterative resolution in the same scenario as the one depicted in the above figure. Normally the iterative resolution takes place between two local servers; the original resolver gets the final answer from the local server.
- Note that the messages shown by events 2, 4, and 6 contain the same query. However, the message shown by event 3 contains the IP address of the top-level domain server, the message shown by event 5 contains the IP address of the McGraw-Hill local DNS server, and the message shown by event 7 contains the IP address of the destination.
- When the Anet local DNS server receives the IP address of the destination, it sends it to the resolver (event 8).



Caching

- ***Each time a server receives a query for a name that is not in its domain, it needs to search its database for a server IP address. Reduction of this search time would increase efficiency.***
- ***DNS handles this with a mechanism called caching***. When a server asks for a mapping from another server and receives the response, it stores this information in its cache memory before sending it to the client. If the same or another client asks for the same mapping, it can check its cache memory and resolve the problem.
- To inform the client that the response is coming from the cache memory and not from an authoritative source, the server marks the response as unauthoritative. Caching speeds up resolution, but it can also be problematic.
- If a ***server caches a mapping for a long time***, it may send an outdated mapping to the client.
- To counter this, ***two techniques are used***.
    - ***First***, the authoritative server always adds information to the mapping called time to live (TTL). It defines the time in seconds that the receiving server can cache the information. After that time, the mapping is invalid and any query must be sent again to the authoritative server.
    - ***Second***, DNS requires that each server keep a TTL counter for each mapping it caches. The cache memory must be searched periodically and those mappings with an expired TTL must be purged.

Resource Records

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

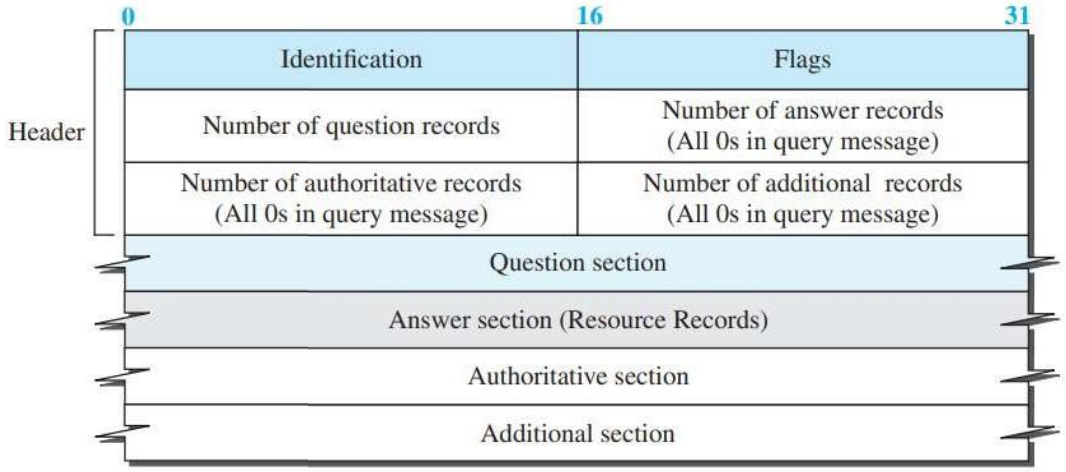
(Domain Name, Type, Class, TTL, Value)

- The domain name field is what identifies the resource record. The value defines the information kept about the domain name. The TTL defines the number of seconds for which the information is valid. The class defines the type of network; we are only interested in the class IN (Internet). The type defines how the value should be interpreted. Table lists the common types and how the value is interpreted for each type.

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

DNS Messages

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



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.

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***

DDNS

- When the DNS was designed, no one predicted that there would be so many address changes. In DNS, when there is a change, such as adding a new host, removing a host, or changing an IP address, the change must be made to the DNS master file. These types of changes involve a lot of manual updating. The size of today's Internet does not allow for this kind of manual operation.
- The DNS master file must be updated dynamically. The Dynamic Domain Name System (DDNS) therefore was devised to respond to this need. In DDNS, when a binding between a name and an address is determined, the information is sent, usually by DHCP (discussed in Chapter 18) to a primary DNS server. The primary server updates the zone. The secondary servers are notified either actively or passively. In active notification, the primary server sends a message to the secondary servers about the change in the zone, whereas in passive notification, the secondary servers periodically check for any changes. In either case, after being notified about the change, the secondary server requests information about the entire zone (called the zone transfer). To provide security and prevent unauthorized changes in the DNS records, DDNS can use an authentication mechanism.