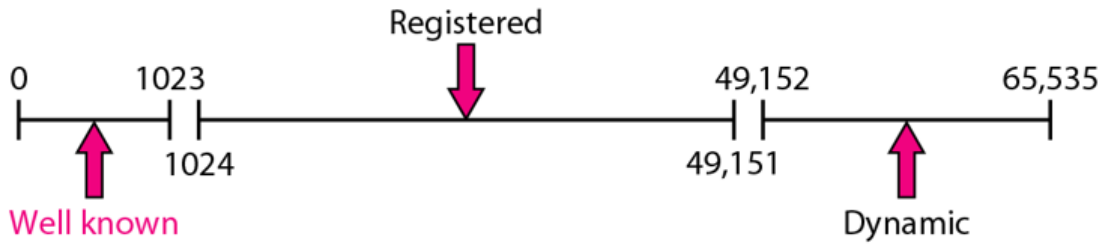
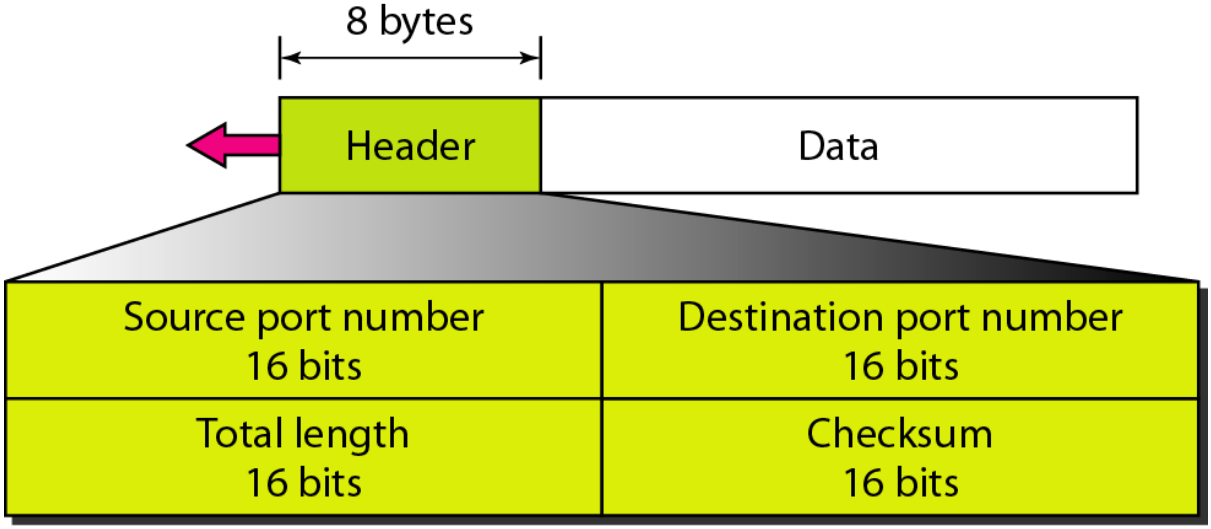
Module 4

1. Illustrate with diagram and explain the IANA ranges. **23-9**
   The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private), as shown in Figure
   - **Well-known ports.** The ports ranging from 0 to 1023 are assigned and controlled by lANA. These are the well-known ports.
   - **Registered ports.** The ports ranging from 1024 to 49,151 are not assigned or controlled by lANA. They can only be registered with lANA to prevent duplication.
   - **Dynamic ports**. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.



2. Explain the user datagram format with a neat diagram. **23-18**

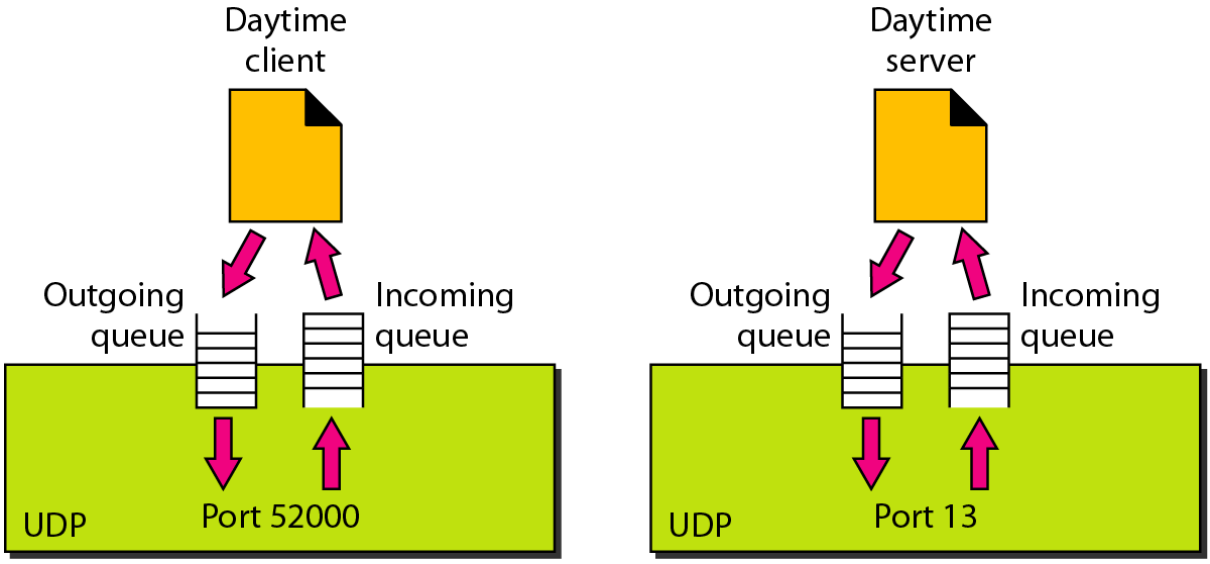


UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure 23.9 shows the format of a user datagram. The fields are as follows:

- **Source port number.** This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. Ifthe source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.

- Destination port number. This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.
- Length. This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes.
- The length field in a UDP user datagram is actually not necessary. A user datagram is encapsulated in an IP datagram. There is a field in the IP datagram that defines the total length. There is another field in the IP datagram that defines the length of the header. So if we subtract the value of the second field from the first, we can deduce the length of a UDP datagram that is encapsulated in an IP datagram.
- **UDP length = IP length - IP header's length**
- However, the designers of the UDP protocol felt that it was more efficient for the destination UDP to calculate the length of the data from the information provided in the UDP user datagram rather than ask the IP software to supply this information. We should remember that when the IP software delivers the UDP user datagram to the UDP layer, it has already dropped the IP header.
- Checksum. This field is used to detect errors over the entire user datagram (header plus data). The checksum is discussed next.
-

3. Illustrate the concept of queuing in UDP. **23-23**
   We have talked about ports without discussing the actual implementation of them. In UDP, queues are associated with ports



At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.

Note that even if a process wants to communicate with multiple processes, it obtains only one port number and eventually one outgoing and one incoming queue. The

queues opened by the client are, in most cases, identified by ephemeral port numbers. The queues function as long as the process is running. When the process terminates, the queues are destroyed.

The client process can send messages to the outgoing queue by using the source port number specified in the request. UDP removes the messages one by one and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system can ask the client process to wait before sending any more messages.

When a message arrives for a client, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the server. All the incoming messages for one particular client program, whether coming from the same or a different server, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the server.

At the server site, the mechanism of creating queues is different. In its simplest form, a server asks for incoming and outgoing queues, using its well-known port, when it starts running. The queues remain open as long as the server is running.

When a message arrives for a server, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field ofthe user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. Ifthere is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the client. All the incoming messages for one particular server, whether coming from the same or a different client, are sent to the same queue. An incoming queue can overflow. Ifthis happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the client.

When a server wants to respond to a client, it sends messages to the outgoing queue, using the source port number specified in the request. UDP removes the messages one by one and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system asks the server to wait before sending any more messages.

4. Explain the three phases of a TCP connection. **23-40.44.45**

Figure 23.18 *Connection establishment using three-way handshaking*

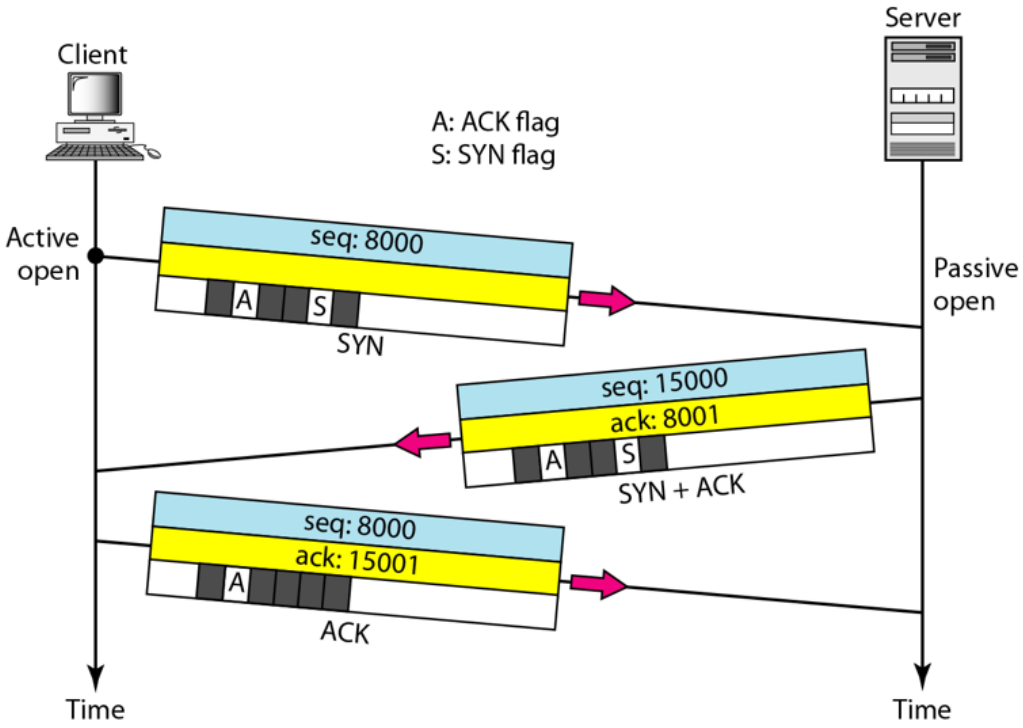


Figure 23.19 *Data transfer*

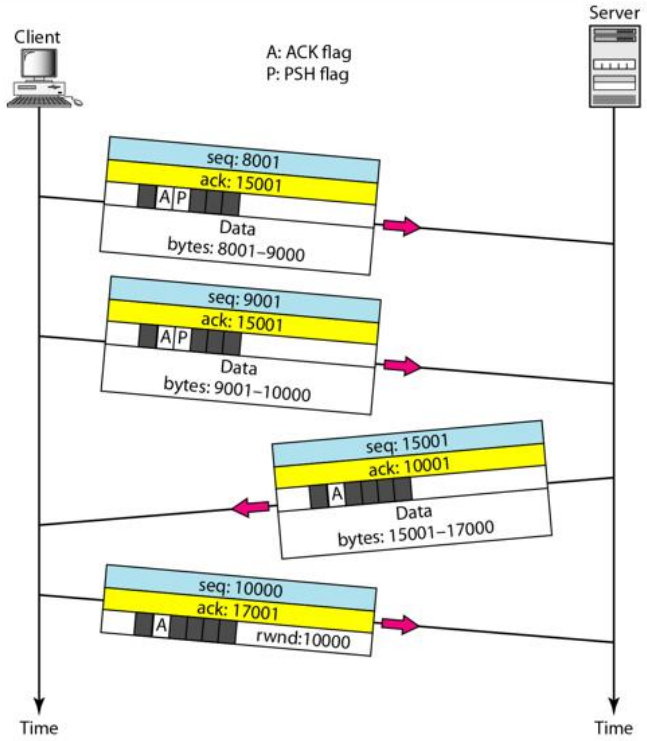
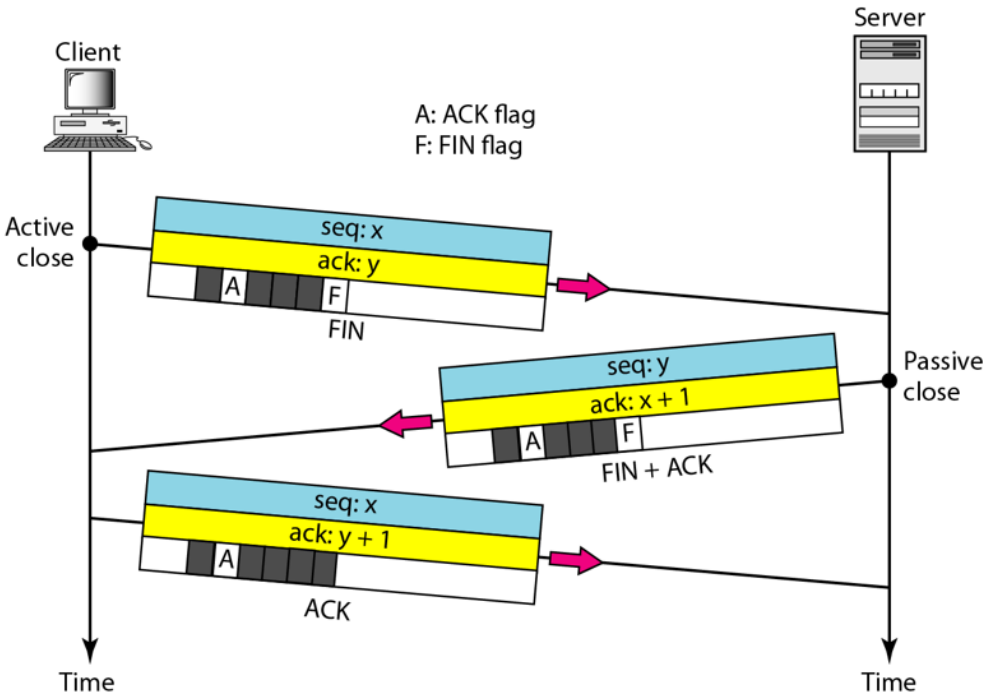


Figure 23.20 *Connection termination using three-way handshaking*



5. With neat diagram, explain the TCP segment. Also, explain the control filed of TCP in detail. **23-37,38**

Figure 23.16 *TCP segment format*

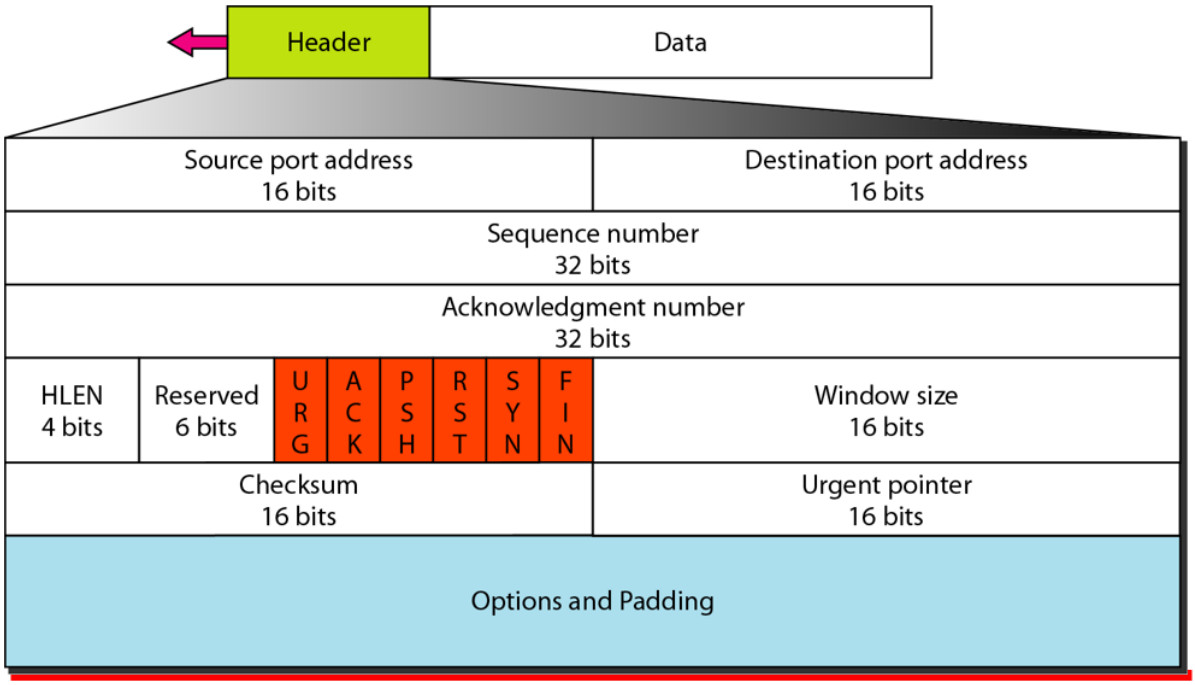
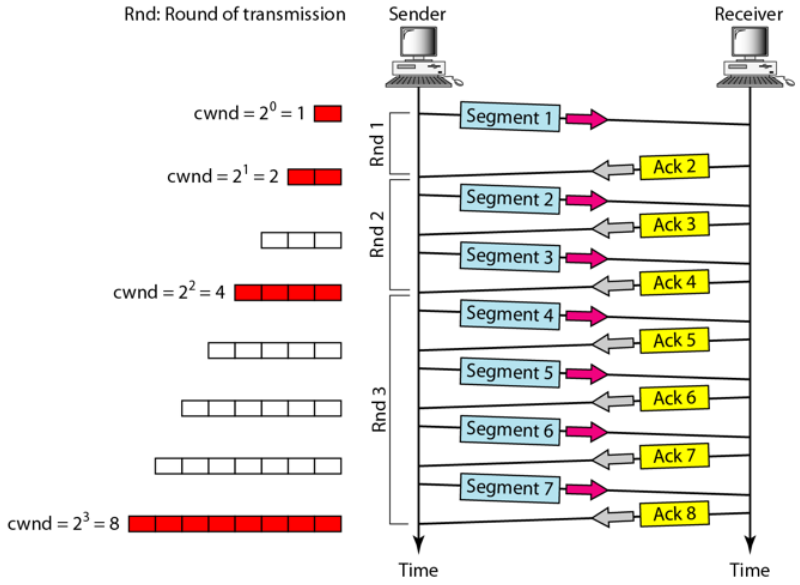


Figure 23.17 *Control field*

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

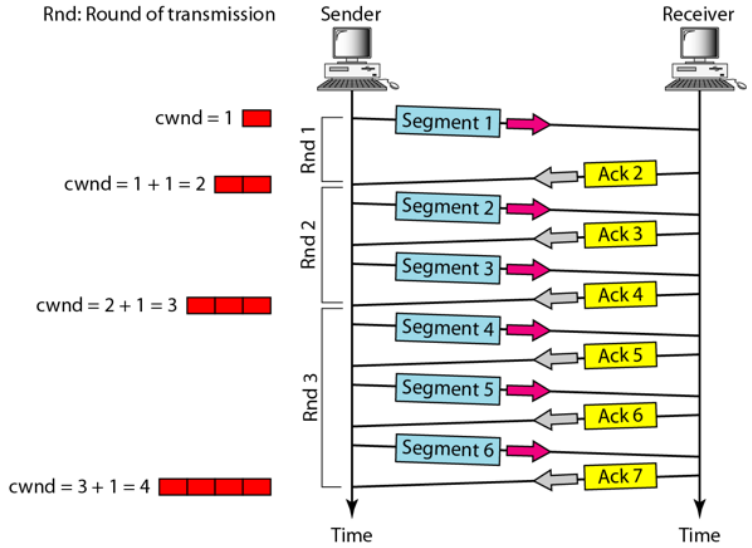
6. Explain the three phases of congestion control mechanism used in TCP with necessary diagrams.**24-13** **PAGE NO 769 IN TB**

Figure 24.8 *Slow start, exponential increase*



In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Figure 24.9 *Congestion avoidance, additive increase*



In the congestion avoidance algorithm, the size of the congestion window increases additively until congestion is detected.

An implementation reacts to congestion detection in one of the following ways:

- **If detection is by time-out, a new slow start phase starts.**
- **If detection is by three ACKs, a new congestion avoidance phase starts.**

7. Discuss the different techniques used to improve the Quality of Service. **24-2**

Bandwidth

Different applications need different bandwidths. In video conferencing we need to send millions of bits per second to refresh a color screen while the total number of bits in an e-mail may not reach even a million.

Flow Classes

Based on the flow characteristics, we can classify flows into groups, with each group having similar levels of characteristics. This categorization is not formal or universal; some protocols such as ATM have defined classes, as we will see later.

24.6 lECHNIQUES TO IMPROVE QoS

In Section 24.5 we tried to define QoS in terms of its characteristics. In this section, we discuss some techniques that can be used to improve the quality of service. We briefly discuss four common methods: scheduling, traffic shaping, admission control, and resource reservation.

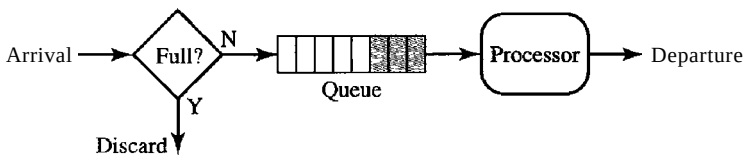
Scheduling

Packets from different flows arrive at a switch or router for processing. A good scheduling technique treats the different flows in a fair and appropriate manner. Several scheduling techniques are designed to improve the quality of service. We discuss three of them here: FIFO queuing, priority queuing, and weighted fair queuing.

FIFO Queuing

In first-in, first-out (FIFO) queuing, packets wait in a buffer (queue) until the node (router or switch) is ready to process them. If the average arrival rate is higher than the average processing rate, the queue will fill up and new packets will be discarded. A FIFO queue is familiar to those who have had to wait for a bus at a bus stop. Figure 24.16 shows a conceptual view of a FIFO queue.

Figure 24.16 *FIFO queue*

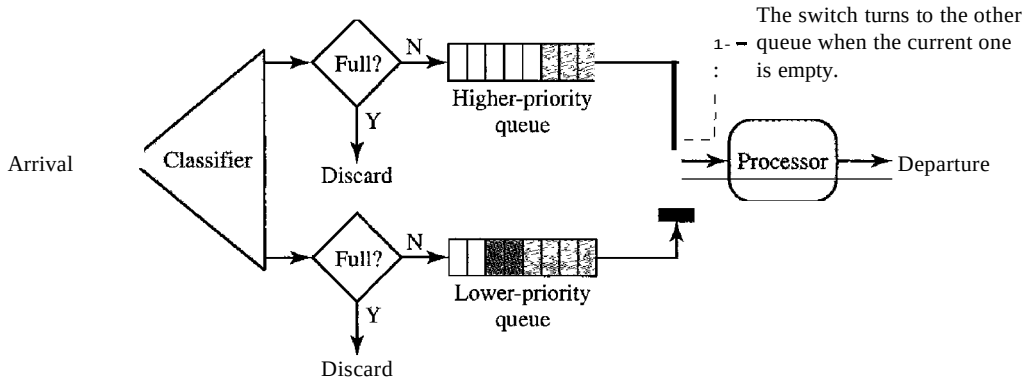


Priority Queuing

In priority queuing, packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving

a queue until it is empty. Figure 24.17 shows priority queuing with two priority levels (for simplicity).

Figure 24.17 *Priority queuing*



A priority queue can provide better QoS than the FIFO queue because higher-priority traffic, such as multimedia, can reach the destination with less delay. However, there is a potential drawback. If there is a continuous flow in a high-priority queue, the packets in the lower-priority queues will never have a chance to be processed. This is a condition called *starvation.*

Weighted Fair Queuing

A better scheduling method is weighted fair queuing. In this technique, the packets are still assigned to different classes and admitted to different queues. The queues, however, are weighted based on the priority of the queues; higher priority means a higher weight. The system processes packets in each queue in a round-robin fashion with the number of packets selected from each queue based on the corresponding weight. For example, if the weights are 3, 2, and 1, three packets are processed from the first queue, two from the second queue, and one from the third queue. If the system does not impose priority on the classes, all weights can be equaL In this way, we have fair queuing with priority. Figure 24.18 shows the technique with three classes.

Traffic Shaping

Traffic shaping is a mechanism to control the amount and the rate of the traffic sent to the network. Two techniques can shape traffic: leaky bucket and token bucket.

Leaky Bucket

If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate. Figure 24.19 shows a leaky bucket and its effects.

Figure 24.18 *Weighted fair queuing*

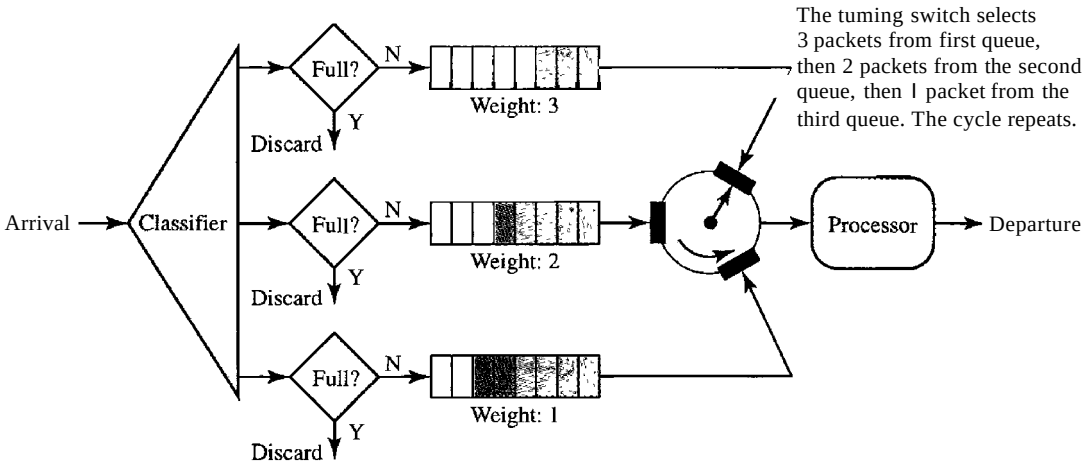
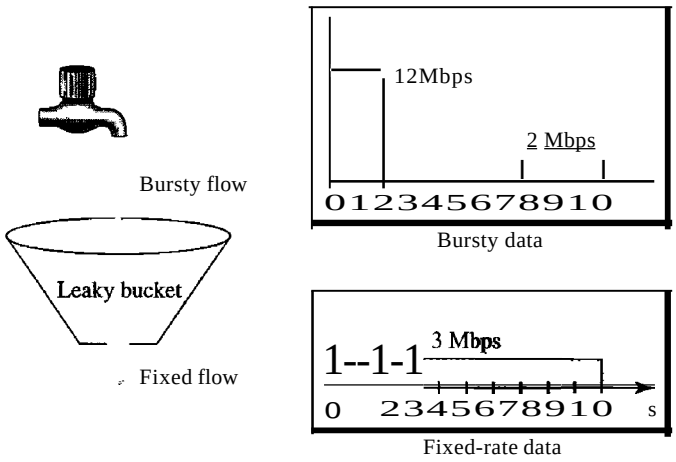


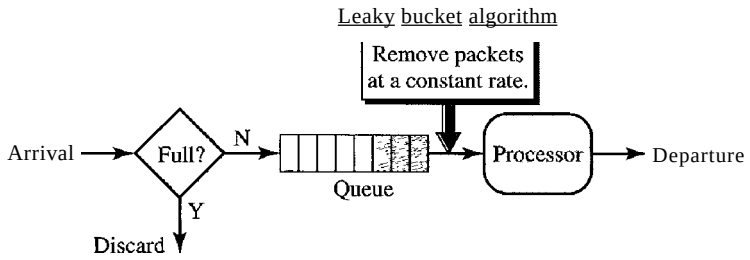
Figure 24.19 *Leaky bucket*



In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In Figure 24.19 the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in lOs. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s. Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion. As an analogy, consider the freeway during rush hour (bursty traffic). If, instead, commuters could stagger their working hours, congestion o'n our freeways could be avoided.

A simple leaky bucket implementation is shown in Figure 24.20. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM

Figure 24.20 *Leaky bucket implementation*



networks), the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

The following is an algorithm for variable-length packets:

1. Initialize a counter to *n* at the tick of the clock.
2. If *n* is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until *n* is smaller than the packet size.
3. Reset the counter and go to step 1.

A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full.

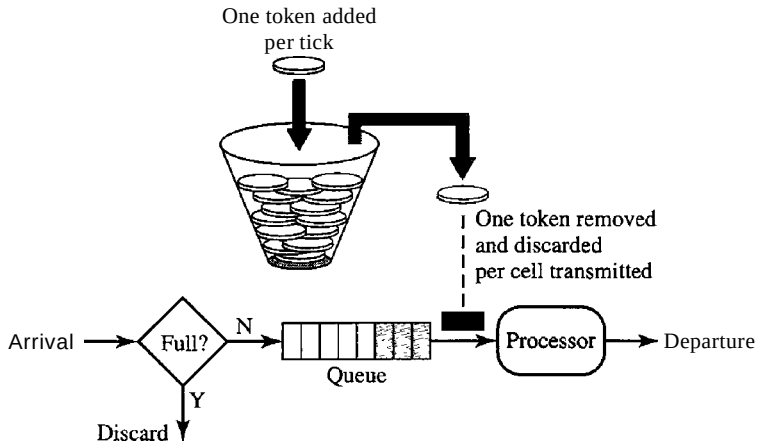
Token Bucket

The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account. On the other hand, the token bucket algorithm allows idle hosts to accumulate credit for the future in the form of tokens. For each tick of the clock, the system sends *n* tokens to the bucket. The system removes one token for every cell (or byte) of data sent. For example, if *n* is 100 and the host is idle for 100 ticks, the bucket collects 10,000 tokens. Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1000 ticks with 10 cells per tick. In other words, the host can send bursty data as long as the bucket is not empty. Figure 24.21 shows the idea.

The token bucket can easily be implemented with a counter. The token is initialized to zero. Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.

The token bucket allows bursty traffic at a regulated maximum rate.

Figure 24.21 *Token bucket*



Combining Token Bucket and Leaky Bucket

The two techniques can be combined to credit an idle host and at the same time regulate the traffic. The leaky bucket is applied after the token bucket; the rate of the leaky bucket needs to be higher than the rate of tokens dropped in the bucket.

Resource Reservation

A flow of data needs resources such as a buffer, bandwidth, CPU time, and so on. The quality of service is improved if these resources are reserved beforehand. We discuss in this section one QoS model called Integrated Services, which depends heavily on resource reservation to improve the quality of service.

Admission Control

Admission control refers to the mechanism used by a router, or a switch, to accept or reject a flow based on predefined parameters called flow specifications. Before a router accepts a flow for processing, it checks the flow specifications to see if its capacity (in terms of bandwidth, buffer size, CPU speed, etc.) and its previous commitments to other flows can handle the new flow.

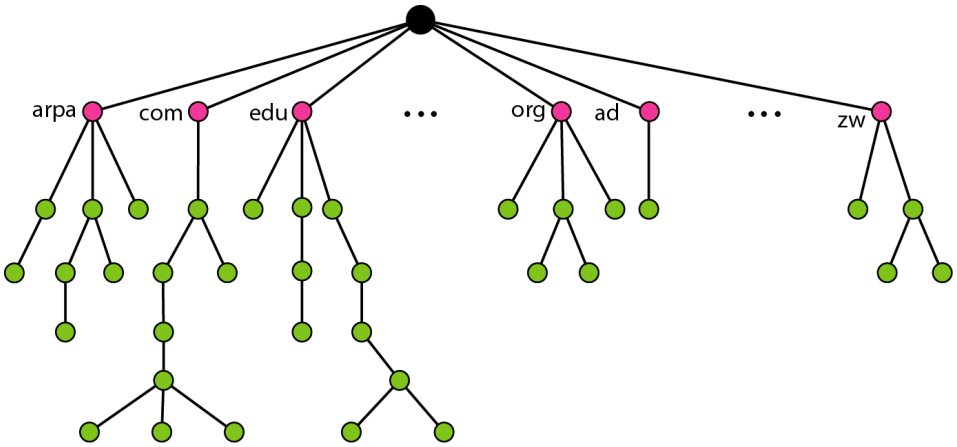
24.7 INTEGRATED SERVICES

Based on the topics in Sections 24.5 and 24.6, two models have been designed to provide quality of service in the Internet: Integrated Services and Differentiated Services. Both models emphasize the use of quality of service at the network layer (IP), although the model can also be used in other layers such as the data link. We discuss Integrated Services in this section and Differentiated Service in Section 24.8.

As we learned in Chapter 20, IP was originally designed for *best-effort* delivery. This means that every user receives the same level of services. This type of delivery

Module 5

8. Explain the concept of **Domain Name Space**. **25-4**



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

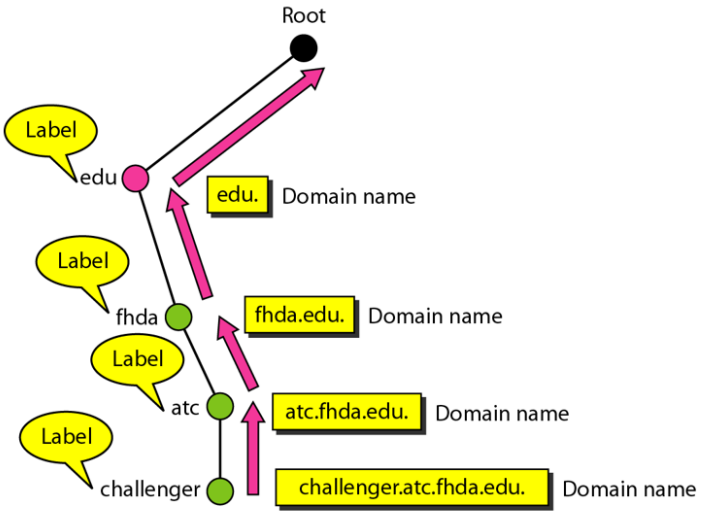
Label

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

Domain Name

Each node in the tree has a domain name. A full domain name is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null). This means that a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing. Figure 25.3 shows some domain names.

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



Fully Qualified Domain Name

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

challenger.ate.tbda.edu.

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

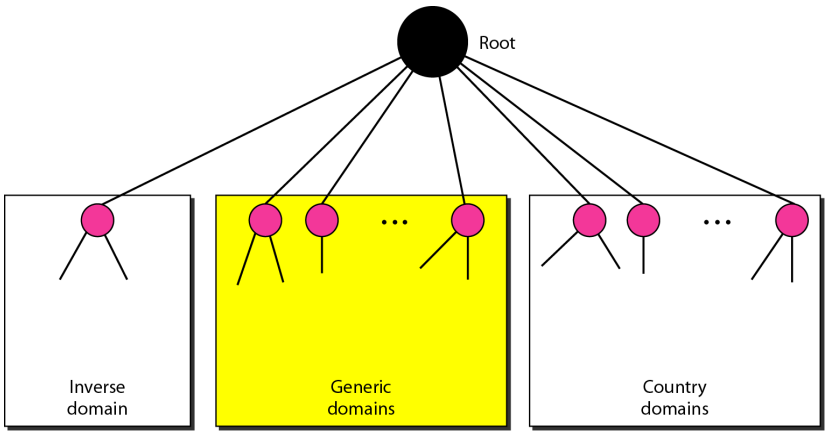
Partially Qualified Domain Name

If a label is not terminated by a null string, it is called a partially qualified domain name (PQDN). A PQDN starts from a node, but it does not reach the root. It is used when the name to be resolved belongs to the same site as the client. Here the resolver can supply the missing part, called the suffix, to create an FQDN. For example, if a user at the jhda.edu. site wants to get the IP address of the challenger computer, he or she can define the partial name

Challenger

The DNS client adds the suffix atc.jhda.edu. before passing the address to the DNS server. The DNS client normally holds a list of suffixes. The following can be the list of suffixes at De Anza College. The null suffix defines nothing. This suffix is added when the user defines an FQDN.

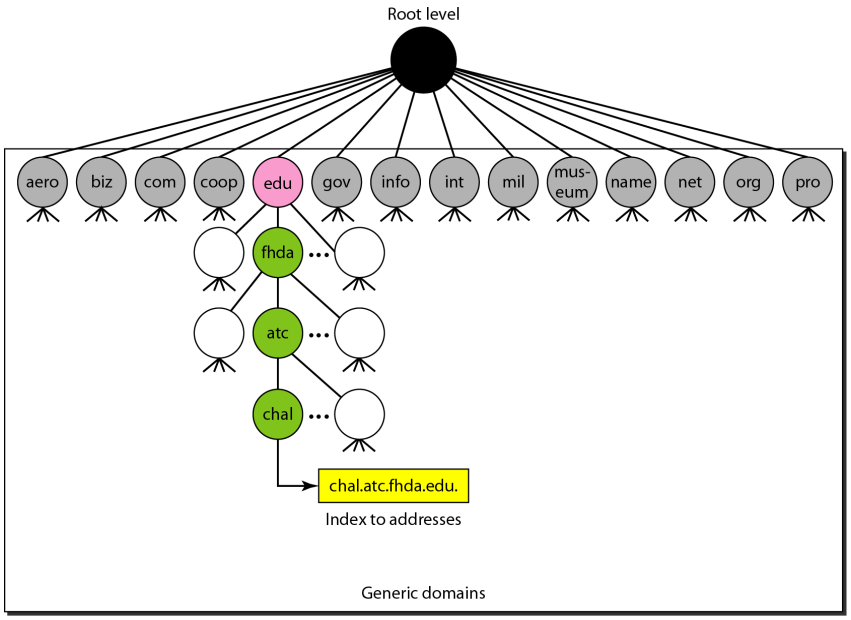
9. Describe **DNS in the internet**. **25-13**



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

Generic Domains

Generic domains Country 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

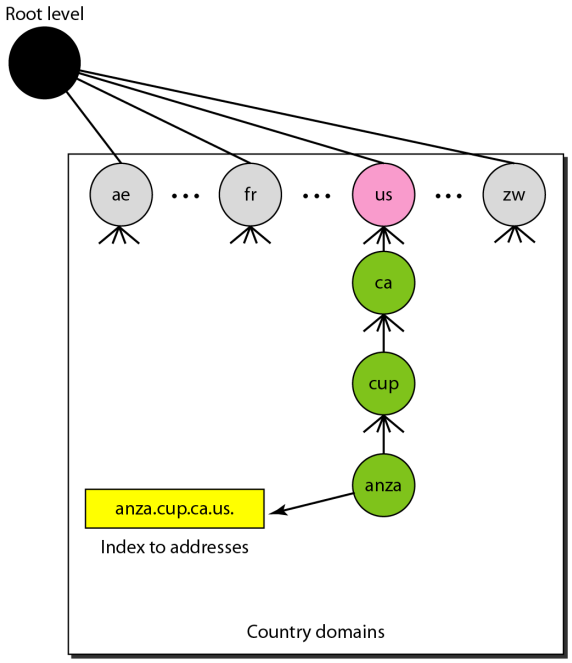


Looking at the tree, we see that the first level in the generic domains section allows 14 possible labels. These labels describe the organization types as listed in Table

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

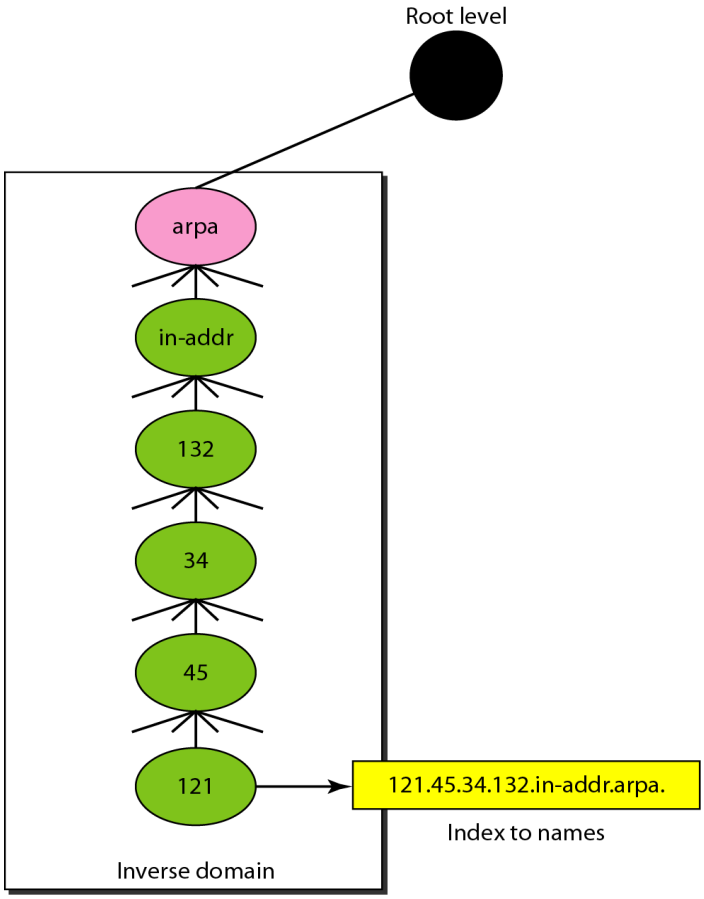
Country Domains

The country domains section uses two-character country abbreviations (e.g., us for United States). Second labels can be organizational, or they can be more specific, national designations. The United States, for example, uses state abbreviations as a subdivision of us (e.g., ca.us.). Figure 25.10 shows the country domains section. The address anza.cup.ca.us can be translated to De Anza College in Cupertino, California, in the United States.



Inverse Domain

The inverse domain is used to map an address to a name. This may happen, for exam ple, when a server has received a request from a client to do a task. Although the server has a file that contains a list of authorized clients, only the IP address of the client (extracted from the received IP packet) is listed. The server asks its resolver to send a query to the DNS server to map an address to a name to determine ifthe client is on the authorized list. This type of query is called an inverse or pointer (PTR) query. To handle a pointer query, the inverse domain is added to the domain name space with the first-level node called arpa (for historical reasons). The second level is also one single node named in-addr (for inverse address). The rest of the domain defines IP addresses. The servers that handle the inverse domain are also hierarchical. This means the netid part ofthe address should be at a higher level than the subnetid part, and the subnetid part higher than the hostid part. In this way, a server serving the whole site is at a higher level than the servers serving each subnet. This configuration makes the domain look inverted when compared to a generic or country domain. To follow the convention of reading the domain labels from the bottom to the top, an IF address such as 132.34.45.121 (a class B address with netid 132.34) is read as 121.45.34.132.in-addr. arpa. See Figure 25.11 for an illustration of the inverse domain configuration.



10. Explain the Email architectures elaborating on the different scenarios in detail. **26-15**

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

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

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

26.2 ELECTRONIC MAIL

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

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

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

Architecture

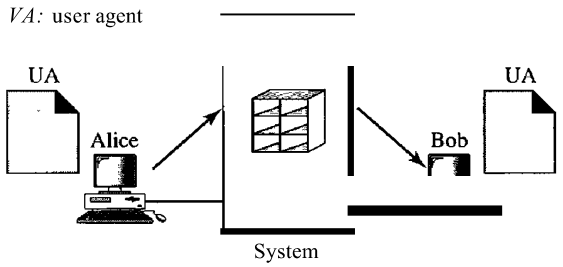
To explain the architecture of e-mail, we give four scenarios. We begin with the simplest situation and add complexity as we proceed. The fourth scenario is the most common in the exchange of email.

First Scenario

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

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

Figure 26.6 *First scenario in electronic mail*



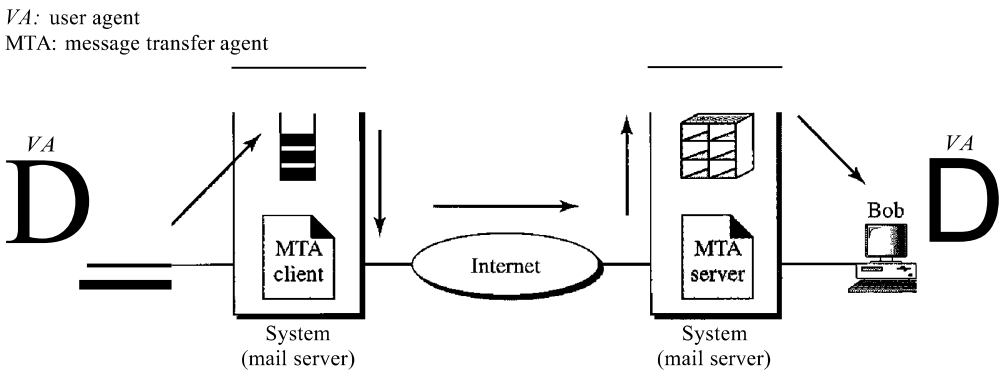
When Alice needs to send a memo to Bob, she writes the memo and inserts it into Bob's mailbox. When Bob checks his mailbox, he finds Alice's memo and reads it.

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

Second Scenario

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

Figure 26.7 *Second scenario in electronic mail*



Alice needs to use a user agent program to send her message to the system at her own site. The system (sometimes called the mail server) at her site uses a queue to store messages waiting to be sent. Bob also needs a user agent program to retrieve messages stored in the mailbox of the system at his site. The message, however, needs to be sent through the Internet from Alice's site to Bob's site. Here two message transfer agents are needed: one client and one server. Like most client/server programs on the Internet, the server needs to run all the time because it does not know when a client will ask for a

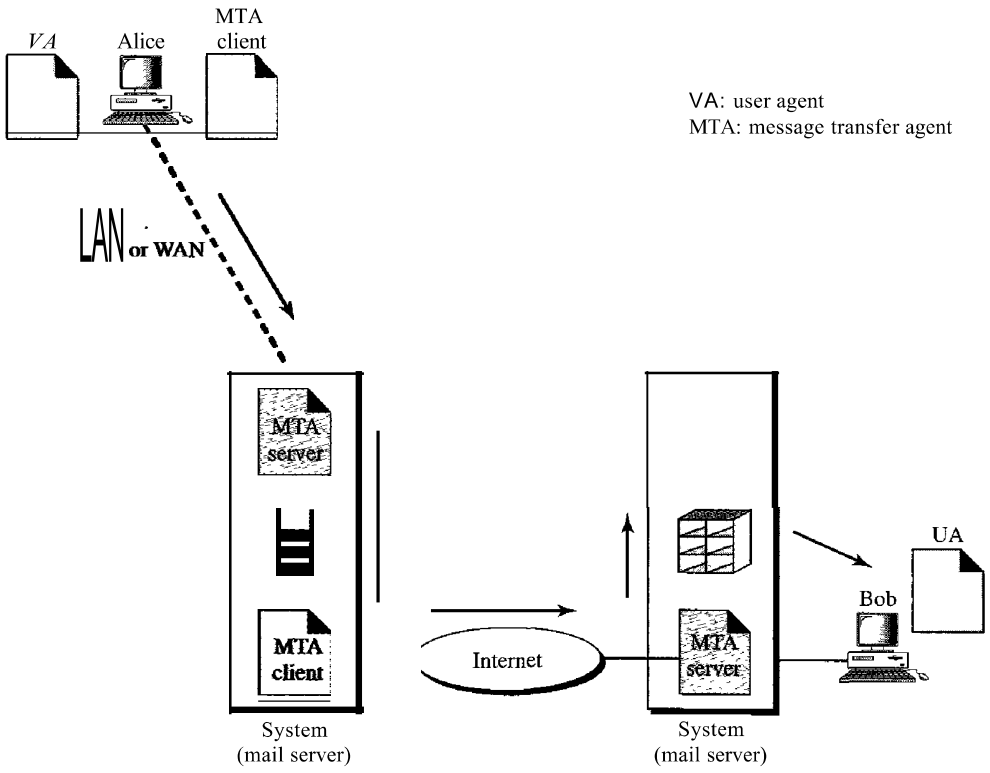
connection. The client, on the other hand, can be alerted by the system when there is a message in the queue to be sent.

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

Third Scenario

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

Figure 26.8 *Third scenario in electronic mail*



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

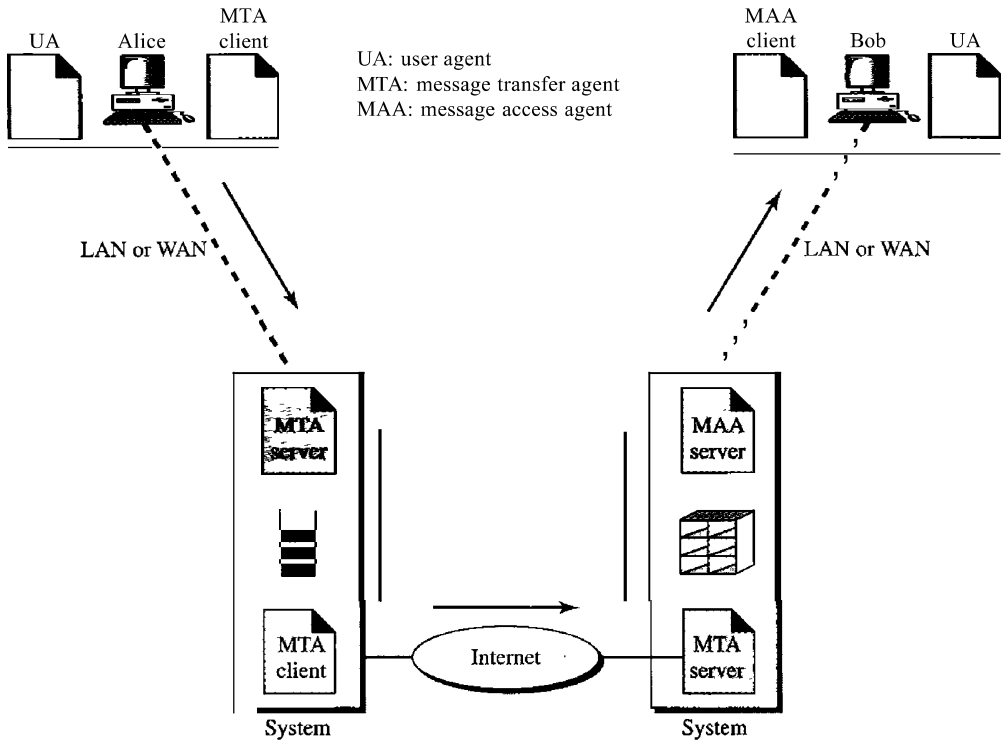
At his convenience, Bob uses his user agent to retrieve the message and reads it. Note that we need two pairs of MTA client/server programs.

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

Fourth Scenario

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

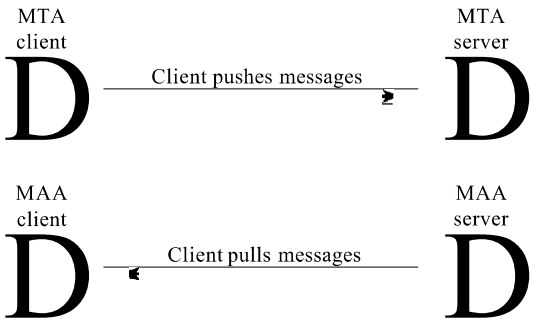
Figure 26.9 *Fourth scenario in electronic mail*



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

Second, note that Bob needs another pair of client/server programs: message access programs. This is so because an MTA client/server program is a *push* program: the client pushes the message to the server. Bob needs a *pull* program. The client needs to pull the message from the server. Figure 26.10 shows the difference.

Figure 26.10 *Push versus pull in electronic email*



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

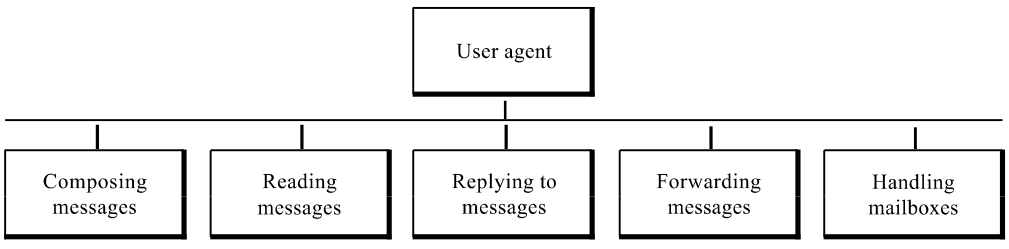
User Agent

The first component of an electronic mail system is the user agent *(VA).* It provides service to the user to make the process of sending and receiving a message easier.

Services Provided by a User Agent

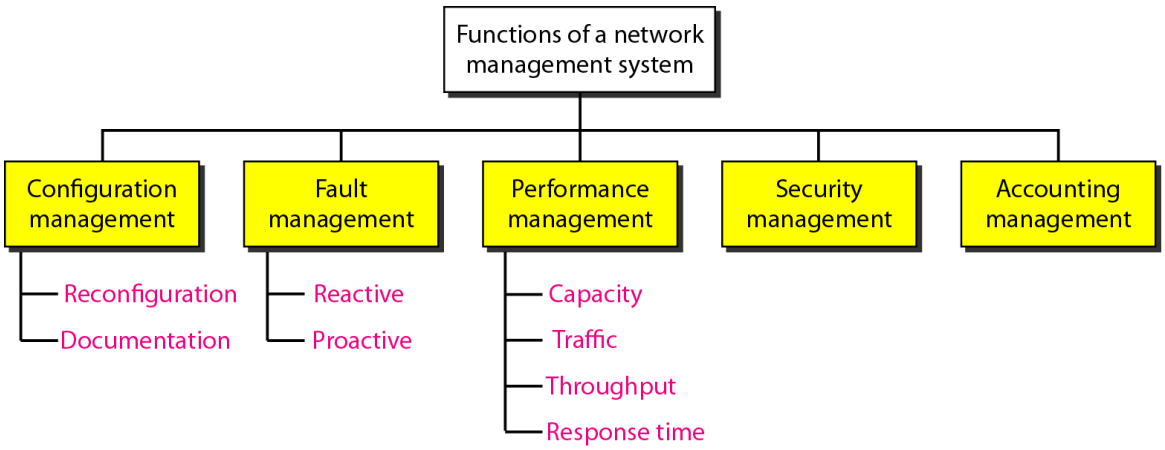
A user agent is a software package (program) that composes, reads, replies to, and forwards messages. It also handles mailboxes. Figure 26.11 shows the services of a typical user agent.

Figure 26.11 *Services of user agent*



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

11. Discuss the functions of a network management system. **28-3**

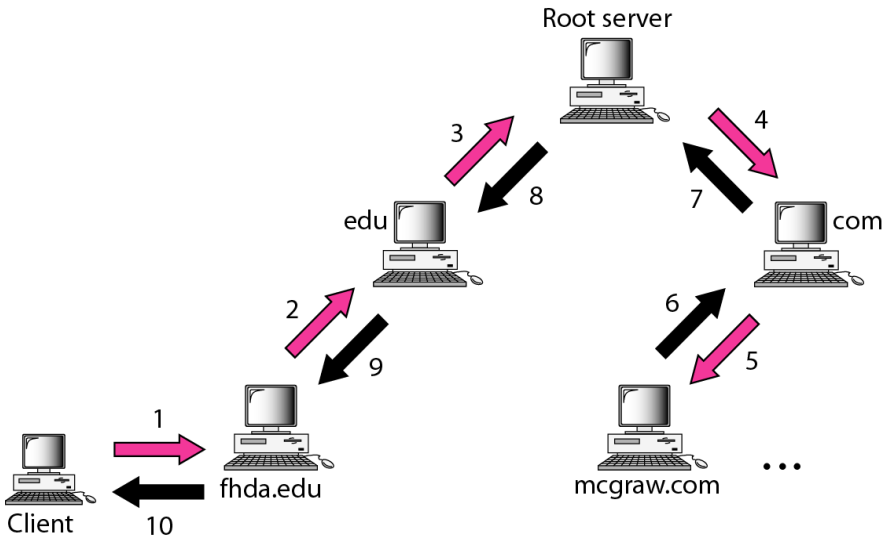


- The configuration management should know, at any time, the status of each entity and its relation to other entities.
- **Hardware Reconfiguration.**
- A desk computer may need to be replaced, router may need to be moved to another part of the network, a subnetwork may be added or removed from the network. This type of reconfiguration cannot be automated and must be manually handled case by case.
- **Software Reconfiguration:**
- New software may need to be installed on servers or clients. An operating system may need updating, an update for an application on some or all clients can be electronically download from the server.
- User-Account Reconfiguration:
- User-Account reconfiguration is not simply adding or deleting users on a system.
- **ACCOUNTING MANAGEMENT.**
- This controls the users access to network resources through charges.
- It prevents users from monopolizing limited network resources.
- It prevents users from using the system inefficiently.
- Network managers can do sort and long term planning based on the demand for network use.
- We must also consider the user privileges, both as a individual and as a member of group.
- **Fault Management**
- **Reactive fault management:**
- A reactive fault management system is responsible for detecting, isolating, correcting and recording faults.
- **Performance Management:**
- Performancemanagementtries to Quantify performance using some measurable quantity, such as capacity,
- Traffic, throughput or response time. Some protocols, such as SNMP can be used in performance management.
- **SECURITY MANAGEMENT.**
- Security management is responsible for controlling access to the network based on predefined policy. Encryption allows privacy for users: authentication forces the users to identify themselves.

12. How does Recursive Resolution different from Iterative Resolution. **25-19**

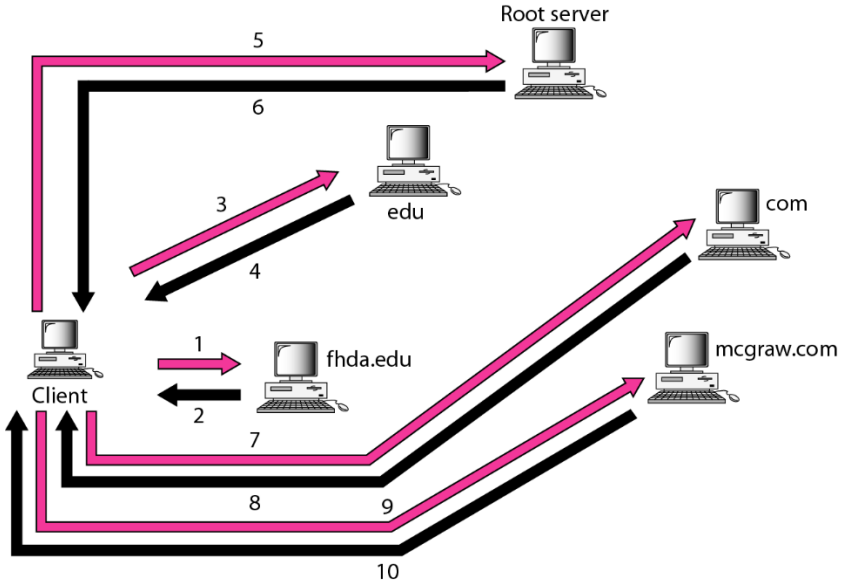
Recursive Resolution

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



Iterative Resolution

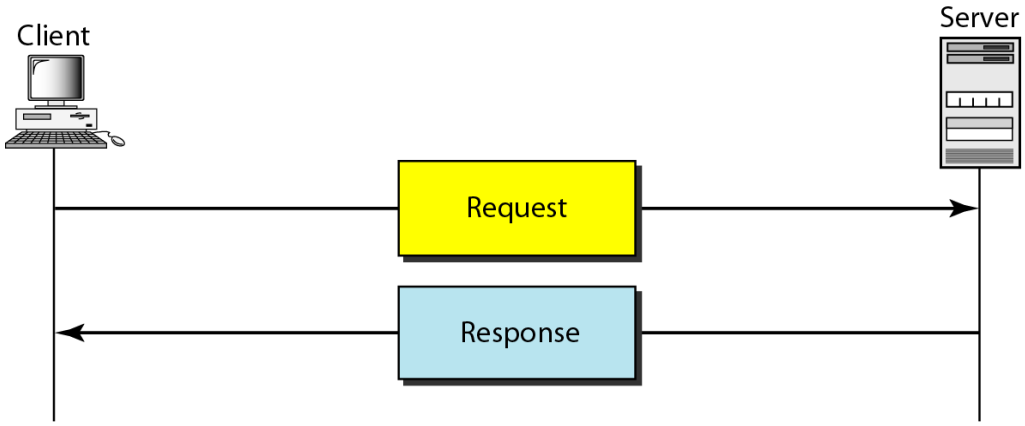
If the client does not ask for a recursive answer, the mapping can be done iteratively. If the server is an authority for the name, it sends the answer. If it is not, it returns (to the client) the IP address of the server that it thinks can resolve the query. The client is responsible for repeating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



13. Define HTTP. Illustrate the HTTP transaction between the client and server.**27-33**

The Hypertext Transfer Protocol (HTTP) is a protocol used mainly to access data on the World Wide Web. HTTP functions as a combination of FTP and SMTP.

- HTTP
- The HTTP is used to define how the client-server programs can be written to retrieve the web pages.
- 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.
- The client and server, however do not worry about errors in messages exchanged or loss of any message, because the TCP is reliable and will take care of this matter.
- **HTTP uses the services of TCP on well-known port 80.**

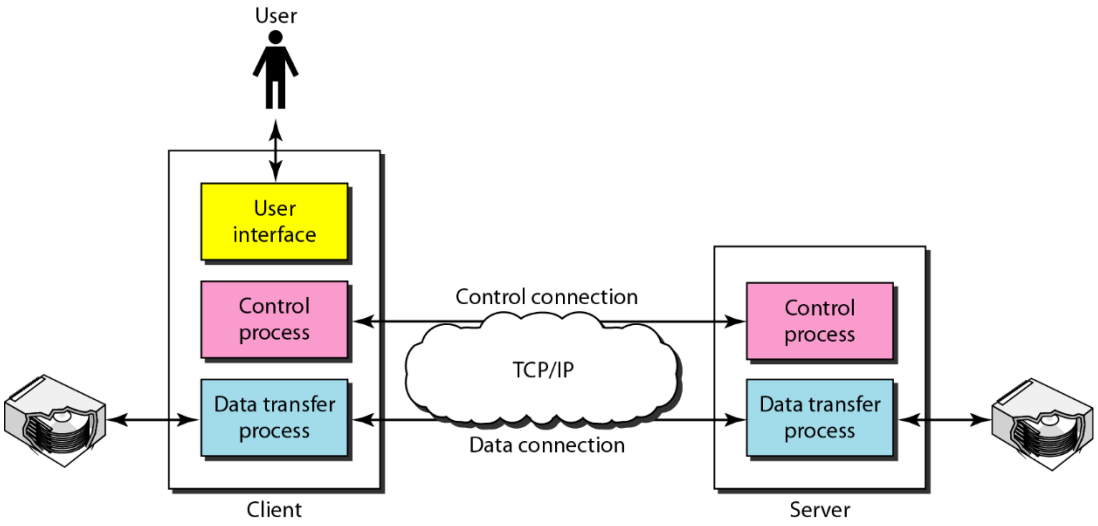


14. Describe FTP in detail.**26-47**

File Transfer Protocol (FTP) File Transfer Protocol (FTP) is the standard mechanism provided by TCP/IP for copying a file from one host to another. Although transferring files from one system to another seems simple and straightforward, some problems must be dealt with first. For example, two systems may use different file name conventions. Two systems may have different ways to represent text and data. Two systems may have different directory structures. All these problems have been solved by FTP in a very simple and elegant approach. FTP differs from other client/server applications in that it establishes two connec tions between the hosts. One connection is used for data transfer, the other for control information (commands and responses). Separation of commands and data transfer makes FTP more efficient. The control connection uses very simple rules of communi cation. Wc need to transfer only a line of command or a line of response at a time. The data connection, on the other hand, needs more complex rules due to the variety of data types transferred. However, the difference in complexity is at the FTP level, not TCP. For TCP, both connections are treated the same. FTP uses two well-known TCP ports: Port 21 is used for the control connection, and port 20 is used for the data connection.

Figure 26.21 shows the basic model of FTP. The client has three components: user interface, client control process, and the client data transfer process. The server has two components: the server control process and the server data transfer process. The control connection is made between the control processes. The data connection is made between the data transfer processes.

The control connection remains connected during the entire interactive FTP ses sion. The data connection is opened and then closed for each file transferred. It opens each time commands that involve transferring files are used, and it closes when the file is transferred. In other words, when a user starts an FTP session, the control connection opens. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.



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