

UNIT III

TRANSPORT AND APPLICATION LAYERS

Transport Layer Protocols – UDP and TCP Connection and State Transition Diagram – Congestion Control and Avoidance (DEC bit, RED)- QoS - Application Layer Paradigms – Client – Server Programming – Domain Name System – World Wide Web, HTTP, Electronic Mail.

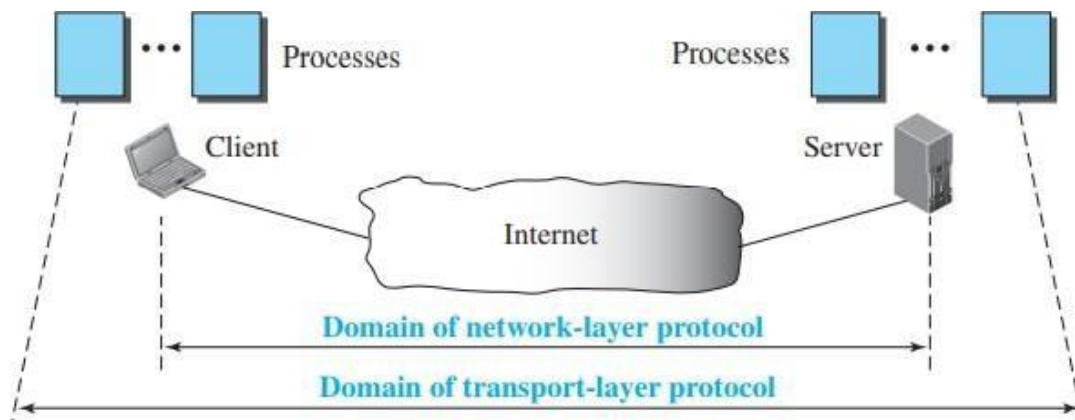
INTRODUCTION TO TRANSPORT LAYER

- The transport layer in the TCP/IP suite is located between the application layer and the network layer.
- It provides services to the application layer and receives services from the network layer.
- It provides a process-to-process communication between two application layers, one at the local host and the other at the remote host.

SERVICES AND FEATURES OF TRANSPORT LAYER

A. Process-to-Process Communication

The first duty of a transport-layer protocol is to provide process-to-process communication.



The network layer is responsible for communication at the computer level (host-to-host communication). A network-layer protocol can deliver the message only to the destination computer. However, this is an incomplete delivery. The message still needs to be handed to the correct process. This is where a transport-layer protocol takes over. A transport-layer protocol is responsible for delivery of the message to the appropriate process.

B. Addressing: Port Numbers

- Process to process communication is achieved through client-server paradigm.
- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- A remote computer can run several server programs at the same time, just as several local computers can run one or more client programs at the same time. For communication, we must define the local host, local process, remote host, and remote process. The local host and the remote host are defined using IP addresses.

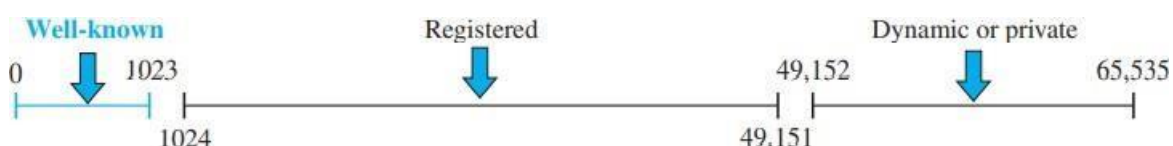
To define the processes, we need second identifiers, called port numbers.

- The client program defines itself with a port number, called the ephemeral port number. The word ephemeral means “short-lived” and is used because the life of a client is normally short.
- The server process must also define itself with a well-known port number. Internet Committee for Assigned Names and Numbers (ICANN) has divided the port numbers into three ranges: well-known, registered, and dynamic (or ephemeral).

❑ **Well-known ports.** The ports ranging from 0 to 1023 are assigned and controlled by ICANN. These are the well-known ports.

❑ **Registered ports.** The ports ranging from 1024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.

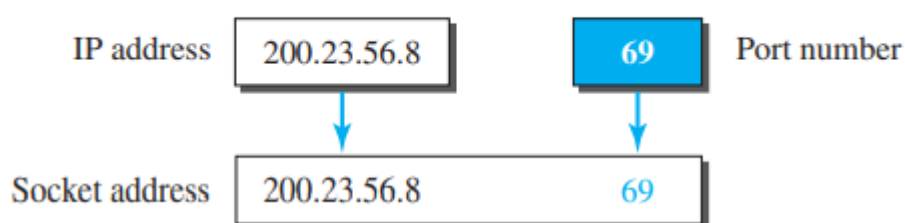
❑ **Dynamic ports.** The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.



C. Socket Addresses

A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection. The combination of an IP address and a port number is called a socket address.

Socket address

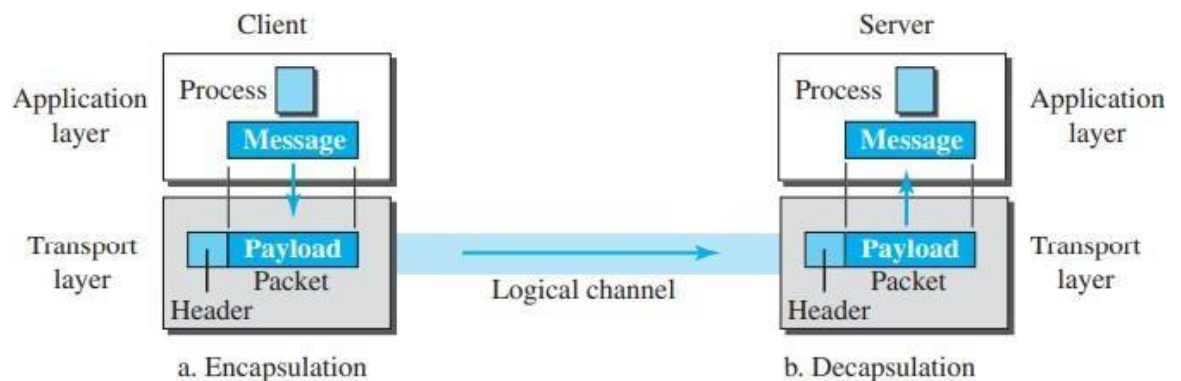


To use the services of the transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the network-layer packet header and the transport-layer packet header. The first header contains the IP addresses; the second header contains the port numbers.

D. Encapsulation and Decapsulation

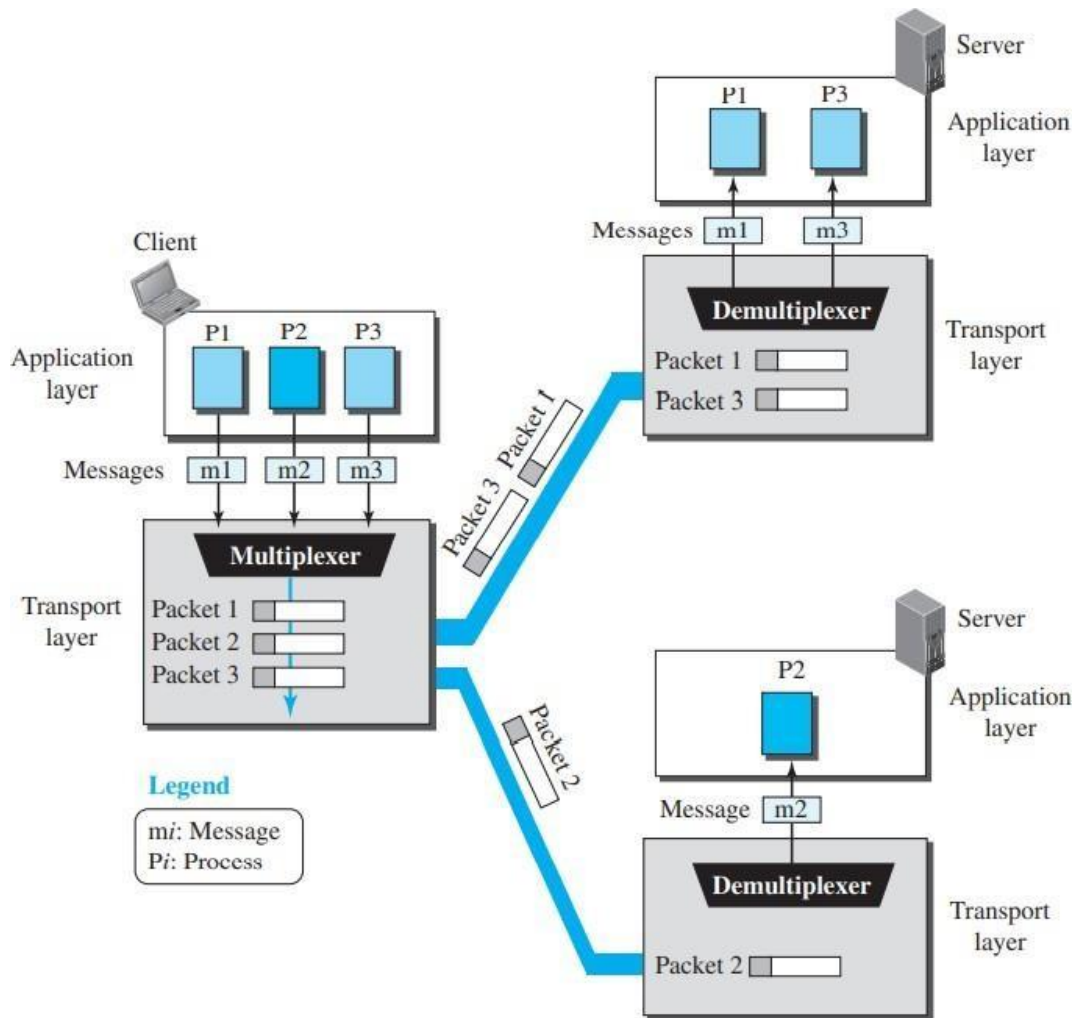
- To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages.
- Encapsulation happens at the sender site. When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol.

- The transport layer receives the data and adds the transport-layer header. The packets at the transport layer in the Internet are called user datagrams, segments, or packets, depending on what transport-layer protocol.
- Decapsulation happens at the receiver site. When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer. The sender socket address is passed to the process in case it needs to respond to the message received.



E. Multiplexing and Demultiplexing

- Whenever an entity accepts items from more than one source, this is referred to as multiplexing (many to one); whenever an entity delivers items to more than one source, this is referred to as demultiplexing (one to many).
- The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing.
- Figure shows communication between a client and two servers. Three client processes are running at the client site, P1, P2, and P3. The processes P1 and P3 need to send requests to the corresponding server process running in a server. The client process P2 needs to send a request to the corresponding server process running at another server.
- The transport layer at the client site accepts three messages from the three processes and creates three packets. It acts as a multiplexer. The packets 1 and 3 use the same logical channel to reach the transport layer of the first server. When they arrive at the server, the transport layer does the job of a demultiplexer and distributes the messages to two different processes.



- The transport layer at the second server receives packet 2 and delivers it to the corresponding process. Note that we still have demultiplexing although there is only one message.

F. Flow Control

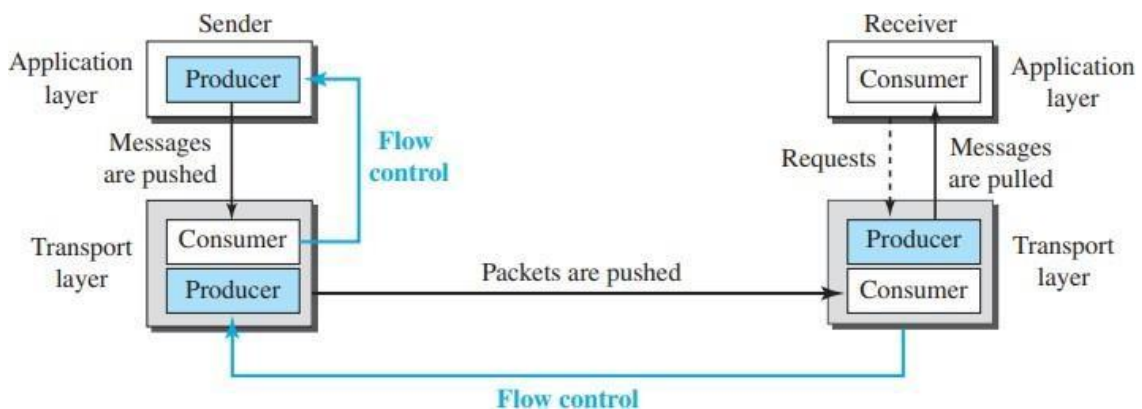


Figure shows that we need at least two cases of flow control: from the sending transport layer to the sending application layer and from the receiving transport layer to the sending transport layer.

Buffers

- Although flow control can be implemented in several ways, one of the solutions is

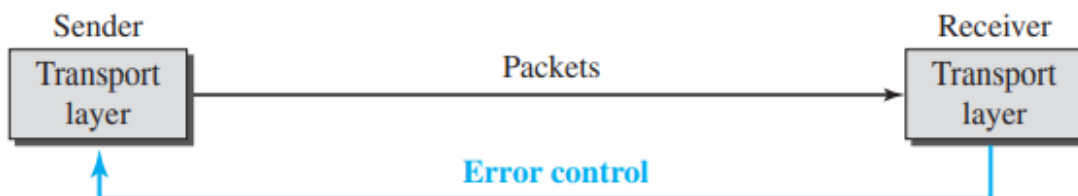
normally to use two buffers: one at the sending transport layer and the other at the receiving transport layer. A buffer is a set of memory locations that can hold packets at the sender and receiver.

- When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages; when there are some vacancies, it informs the application layer that it can pass message chunks again.
- When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets. When there are some vacancies, it informs the sending transport layer that it can send packets again.

G. Error Control

Reliability can be achieved by adding error control services to the transport layer. Error control at the transport layer is responsible for

1. Detecting and discarding corrupted packets.
2. Keeping track of lost and discarded packets and resending them.
3. Recognizing duplicate packets and discarding them.
4. Buffering out-of-order packets until the missing packets arrive.



Error control, unlike flow control, involves only the sending and receiving transport layers.

Sequence Numbers

Error control requires that the sending transport layer knows which packet is to be resent and the receiving transport layer knows which packet is a duplicate, or which packet has arrived out of order. This can be done if the packets are numbered. We can add a field to the transport-layer packet to hold the sequence number of the packet.

If the header of the packet allows m bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$. For example, if m is 4, the only sequence numbers are 0 through 15, inclusive.

Acknowledgment

The receiver side can send an acknowledgment (ACK) for each of a collection of packets that have arrived safe and sound. The receiver can simply discard the corrupted packets. The sender can detect lost packets if it uses a timer. When a packet is sent, the sender starts a timer. If an ACK does not arrive before the timer expires, the sender resends the packet.

H. Congestion control

An important issue in a packet-switched network, such as the Internet, is congestion. Congestion in a network may occur if the load on the network—the number of packet

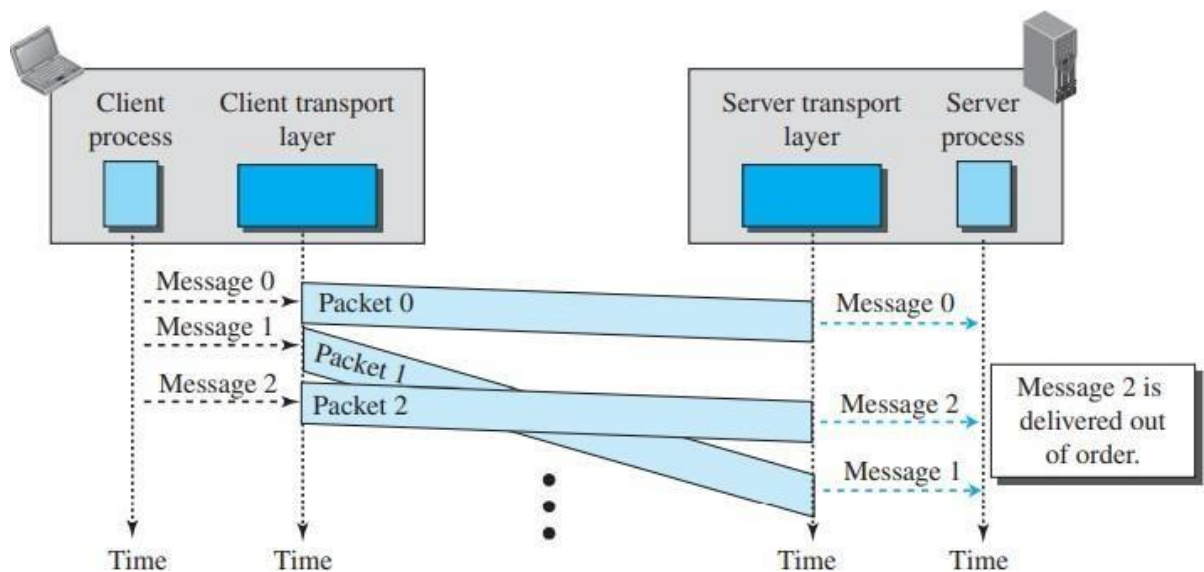
sent to the network—is greater than the capacity of the network—the number of packets a network can handle. Congestion control refers to the mechanisms and techniques that control the congestion and keep the load below the capacity.

CONNECTIONLESS AND CONNECTION-ORIENTED PROTOCOLS

A transport-layer protocol can provide two types of services: connectionless and connection-oriented.

Connectionless Service

- In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.



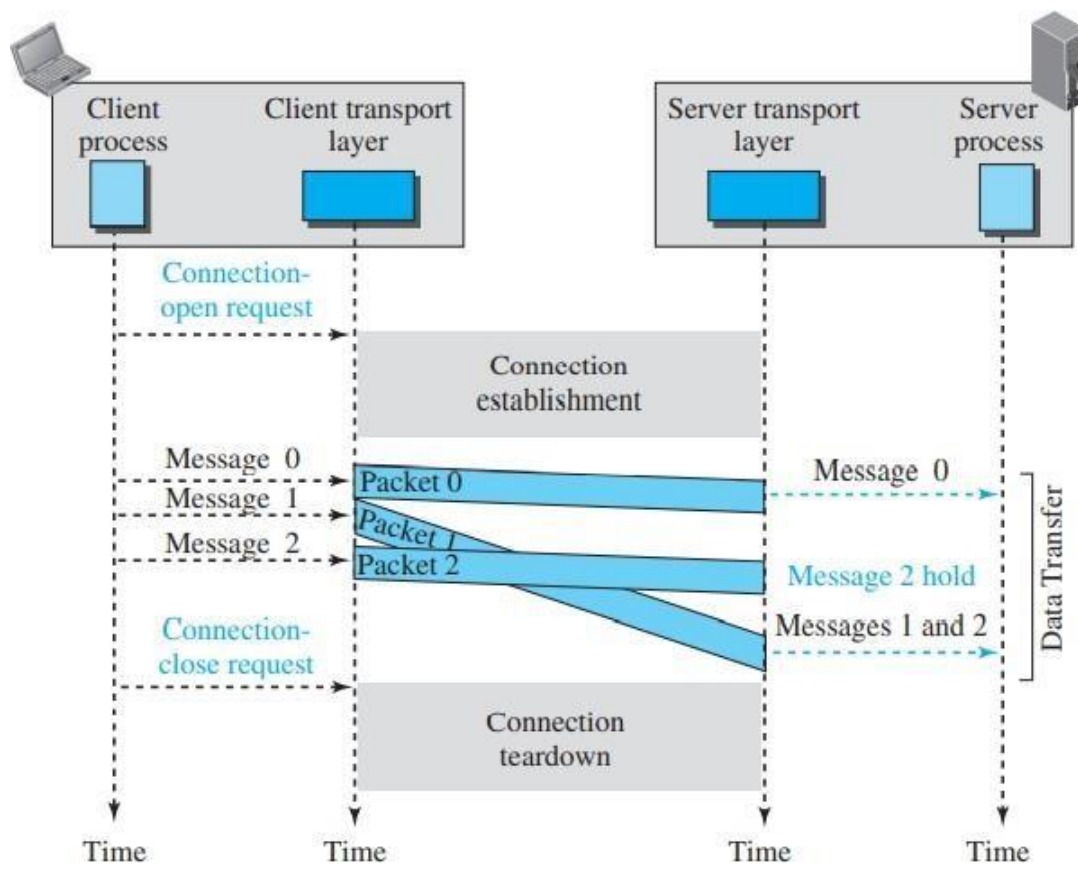
The chunks are handed over to the connectionless transport protocol in order. However, since there is no dependency between the packets at the transport layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process.

We can say that no flow control, error control, or congestion control can be effectively implemented in a connectionless service.

Connection-Oriented Service

In a connection-oriented service, the client and the server first need to establish a logical connection between themselves. The data exchange can only happen after the connection establishment. After data exchange, the connection needs to be torn down. At the transport layer, connection-oriented service involves only the two hosts; the service is end to end.

We can implement flow control, error control, and congestion control in a connection-oriented protocol.



Transport layer Protocols

Name any two differences between Transmission Control Protocol and User Datagram Protocol. (2 marks) Nov 2021

Explain the methods of transport layer protocols. (13 marks) Nov 2021

UDP

UDP is an unreliable connectionless transport-layer protocol used for its simplicity and efficiency in applications where error control can be provided by the application-layer process.

TCP

TCP is a reliable connection-oriented protocol that can be used in any application where reliability is important.

SCTP

SCTP is a new transport-layer protocol that combines the features of UDP and TCP.

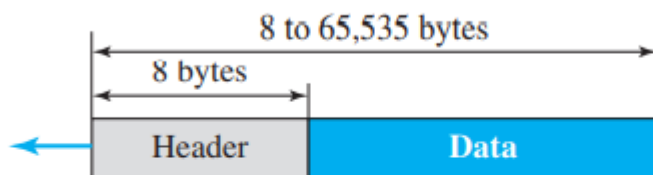
USER DATAGRAM PROTOCOL

- The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol.
- UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP.

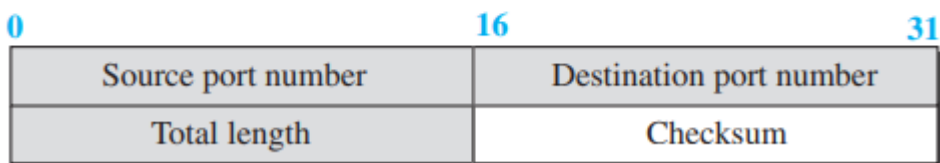
User Datagram

UDP packets are called user datagrams.

User datagram packet format



a. UDP user datagram



b. Header format

The first two fields define the source and destination port numbers. The third field defines the total length of the user datagram, header plus data.

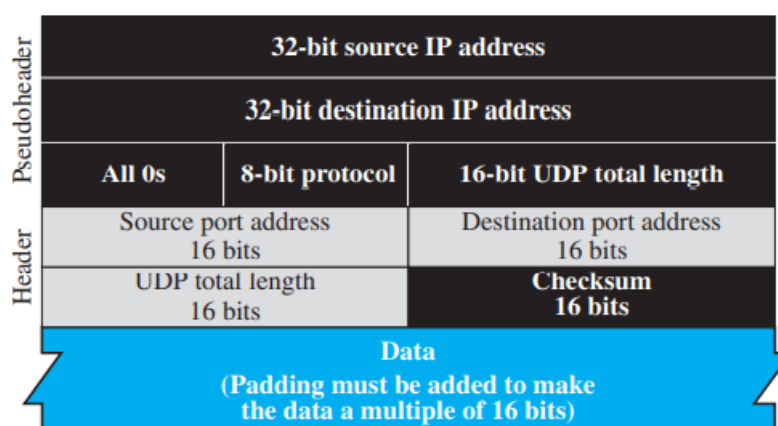
Checksum

UDP checksum calculation includes three sections: a pseudo header, the UDP header, and the data coming from the application layer.

The pseudo header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s.

The protocol field is added to ensure that the packet belongs to UDP, and not to TCP. The value of the protocol field for UDP is 17.

Pseudoheader for checksum calculation



The sender of a UDP packet can choose not to calculate the checksum. In this case, the checksum field is filled with all 0s before being sent.

UDP SERVICES:

1. Process-to-process communication:

UDP provides process-to-process communication using **socket addresses**, a combination of IP addresses and port numbers.

2. Connectionless Services

UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination, as is the case for TCP. This means that each user datagram can travel on a different path.

3. Flow and Error Control

UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages.

4. Error Control

There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

5. Checksum

UDP checksum calculation includes three sections: a pseudo header, the UDP header, and the data coming from the application layer. The pseudo header is the part of the header of the IP packet.

6. Congestion Control

Since UDP is a connectionless protocol, it does not provide congestion control.

7. Encapsulation and Decapsulation

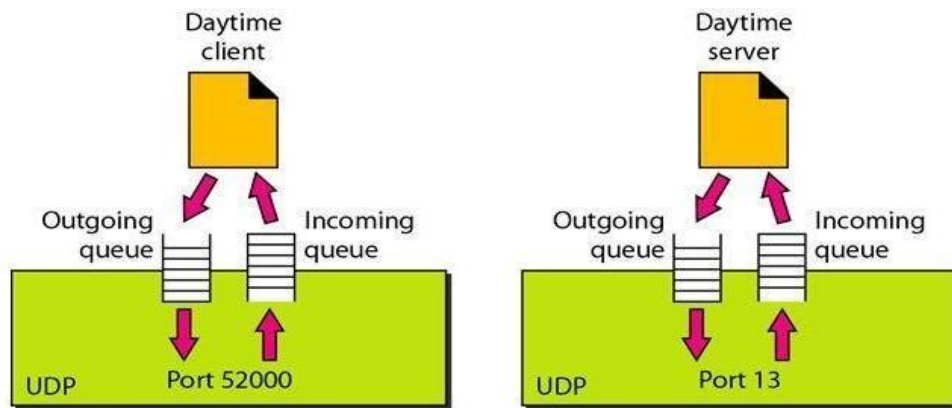
To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages.

8. Queuing

- 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.
- 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.
- 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.
- 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 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 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.



9. Multiplexing and Demultiplexing

In a host running a TCP/IP protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP. To handle this situation, UDP multiplexes and demultiplexes

Typical Applications

The following shows some typical applications that can benefit more from the services of UDP than from those of TCP.

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control.
- UDP is suitable for a process with internal flow- and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP.
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP).
- UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message.

TRANSMISSION CONTROL PROTOCOL

Explain in detail about TCP segment format (13 marks) Nov 2020

Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol. TCP explicitly defines connection establishment, data transfer, and connection tear down phases to provide a connection-oriented service.

TCP Services

The services offered by TCP to the processes at the application layer are:

1. Process-to-Process Communication:

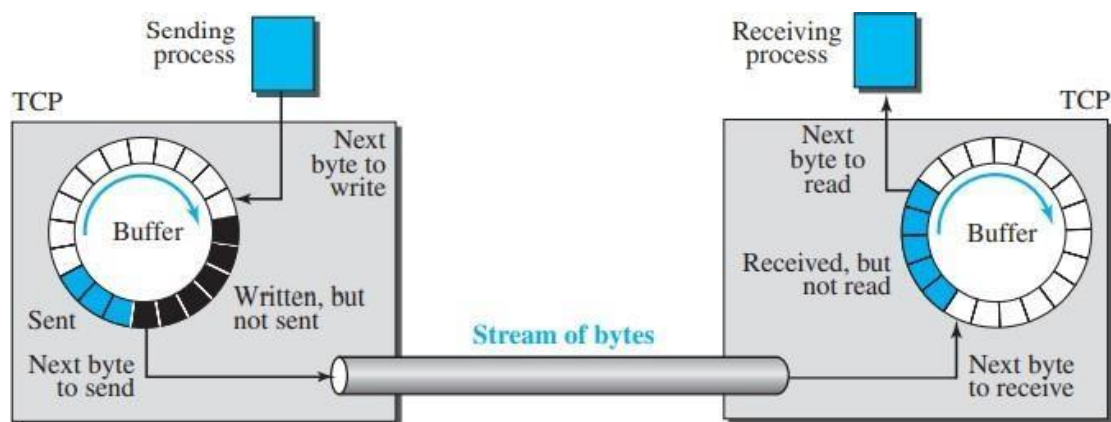
TCP provides process-to-process communication using port numbers.

2. Stream Delivery Service

TCP, unlike UDP, is a stream-oriented protocol. TCP allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes.

Sending and Receiving Buffers

- Because the sending and the receiving processes may not necessarily write or read data at the same rate, TCP needs buffers for storage.
- There are two buffers, the sending buffer and the receiving buffer, one for each direction.
- One way to implement a buffer is to use a circular array of 1-byte locations.

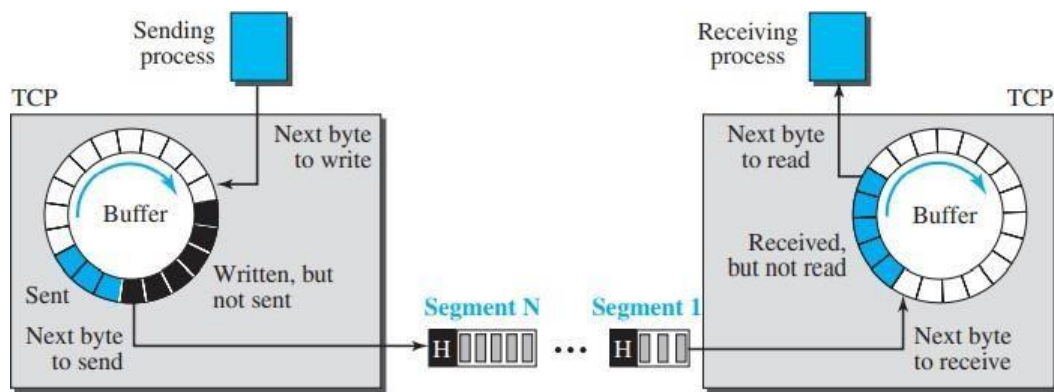


At the sender, the buffer has three types of chambers. The white section contains empty chambers that can be filled by the sending process (producer). The colored area holds bytes that have been sent but not yet acknowledged. The TCP sender keeps these bytes in the buffer until it receives an acknowledgment. The shaded area contains bytes to be sent by the sending TCP. After the bytes in the colored chambers are acknowledged, the chambers are recycled and available for use by the sending process.

The operation of the buffer at the receiver is simpler. The circular buffer is divided into two areas (shown as white and colored). The white area contains empty chambers to be filled by bytes received from the network. The colored sections contain received bytes that can be read by the receiving process. When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

Segments

The network layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes. At the transport layer, TCP groups a number of bytes together into a packet called a segment. TCP adds a header to each segment (for control purposes) and delivers the segment to the network layer for transmission. segments are not necessarily all the same size.



3. Full-Duplex Communication

TCP offers full-duplex service, where data can flow in both directions at the same time. Each TCP endpoint then has its own sending and receiving buffer, and segments move in both directions.

4. Multiplexing and Demultiplexing

Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver. However, since TCP is a connection-oriented protocol, a connection needs to be established for each pair of processes.

5. Connection-Oriented Service

TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:

1. The two TCPs establish a logical connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated. Note that this is a logical connection, not a physical connection.

6. Reliable Service

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data.

TCP Features

1. Numbering System

In each segment there are two fields, called the sequence number and the acknowledgment number. These two fields refer to a byte number and not a segment number.

Byte Number

- ☒ TCP numbers all data bytes (octets) that are transmitted in a connection. Numbering is independent in each direction.
- ☒ When TCP receives bytes of data from a process, TCP stores them in the sending buffer and numbers them. The numbering does not necessarily start from 0. Instead, TCP chooses an arbitrary number between 0 and $2^{32} - 1$ for the number

of the first byte.

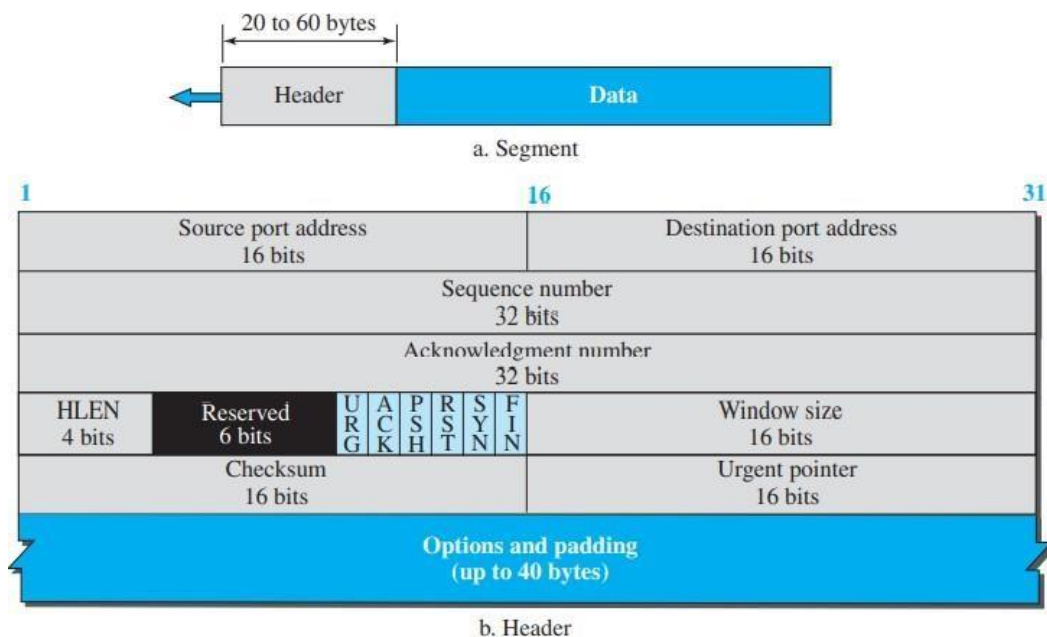
Sequence Number

The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.

Acknowledgment Number

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative. If a party uses 5643 as an acknowledgment number, it has received all bytes from the beginning up to 5642. Note that this does not mean that the party has received 5642 bytes, because the first byte number does not have to be 0.

TCP SEGMENT FORMAT



The segment consists of a header of 20 to 60 bytes, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

Source port address.

This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.

Destination port address.

This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.

Sequence number.

This 32-bit field defines the number assigned to the first byte of data contained in this segment.

Acknowledgment number.

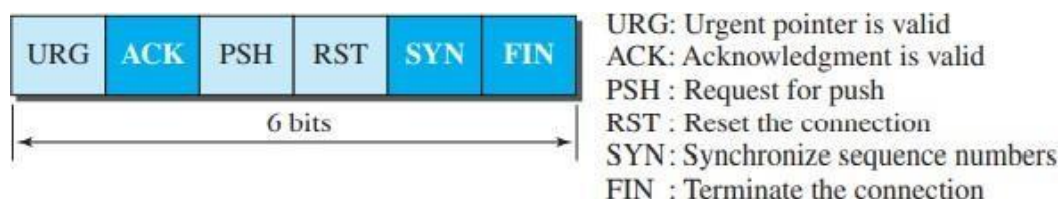
This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. Acknowledgment and data can be piggybacked together.

❑ Header length.

The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$).

❑ Control.

This field defines 6 different control bits or flags, as shown in Figure 24.8. One or more of these bits can be set at a time.

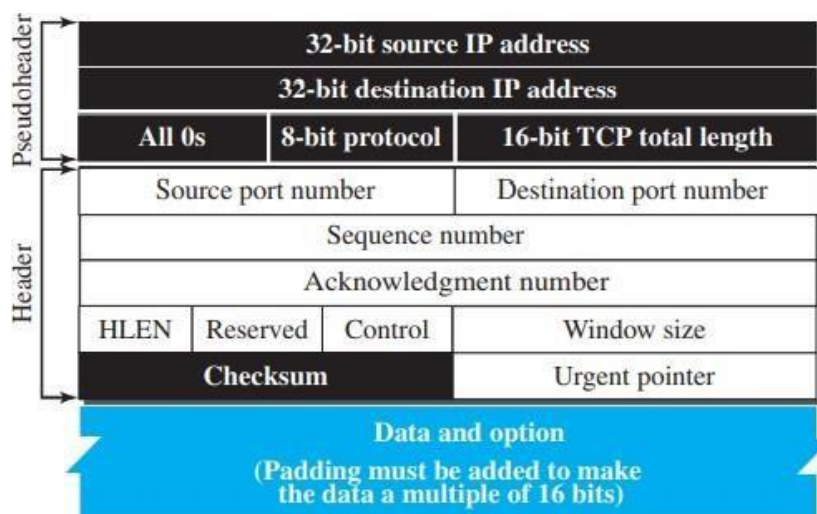


❑ Window size.

This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes.

❑ Checksum.

This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas the use of the checksum for TCP is mandatory.



❑ Urgent pointer.

This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines a value that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.

❑ Options.

There can be up to 40 bytes of optional information in the TCP header.

TCP CONNECTION

TCP is a connection-oriented protocol establishes a logical path between the source and destination. All of the segments belonging to a message are then sent over this logical path. In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

A. Connection Establishment

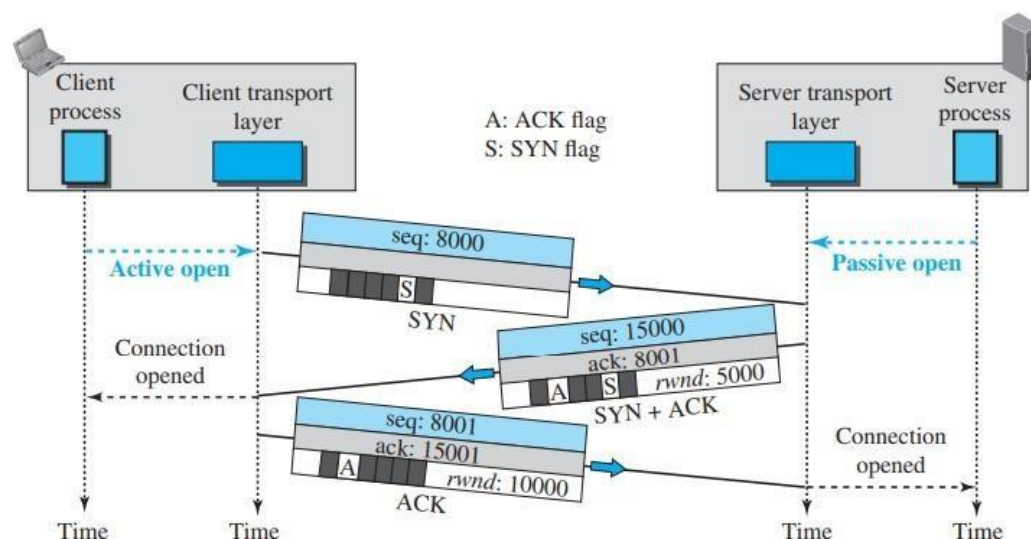
TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously.

Three-Way Handshaking

The connection establishment in TCP is called three-way handshaking.

- ☐ In our example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport-layer protocol.
- ☐ The server program tells its TCP that it is ready to accept a connection. This request is called a passive open.

The client program issues a request for an active open. A client that wishes to connect to an open server tells its TCP to connect to a particular server. TCP can now start the three-way handshaking process.



The three steps in this phase are as follows.

1. The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. The client in our example chooses a random number as the first sequence number and sends this number to the server. This sequence number is called the initial sequence number (ISN). Note that this segment does not contain an acknowledgment number. It does not define the window size either; a window size definition makes sense only when a segment includes an acknowledgment.
2. The server sends the second segment, a SYN + ACK segment with two flag bits set as: SYN and ACK. This segment has a dual purpose. First, it is a SYN segment for communication in the other direction. The server uses this segment to

initialize a sequence number for numbering the bytes sent from the server to the client. The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client. Because the segment contains an acknowledgment, it also needs to define the receive window size, `rwnd`.

3. The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. Note that the ACK segment does not consume any sequence numbers if it does not carry data.

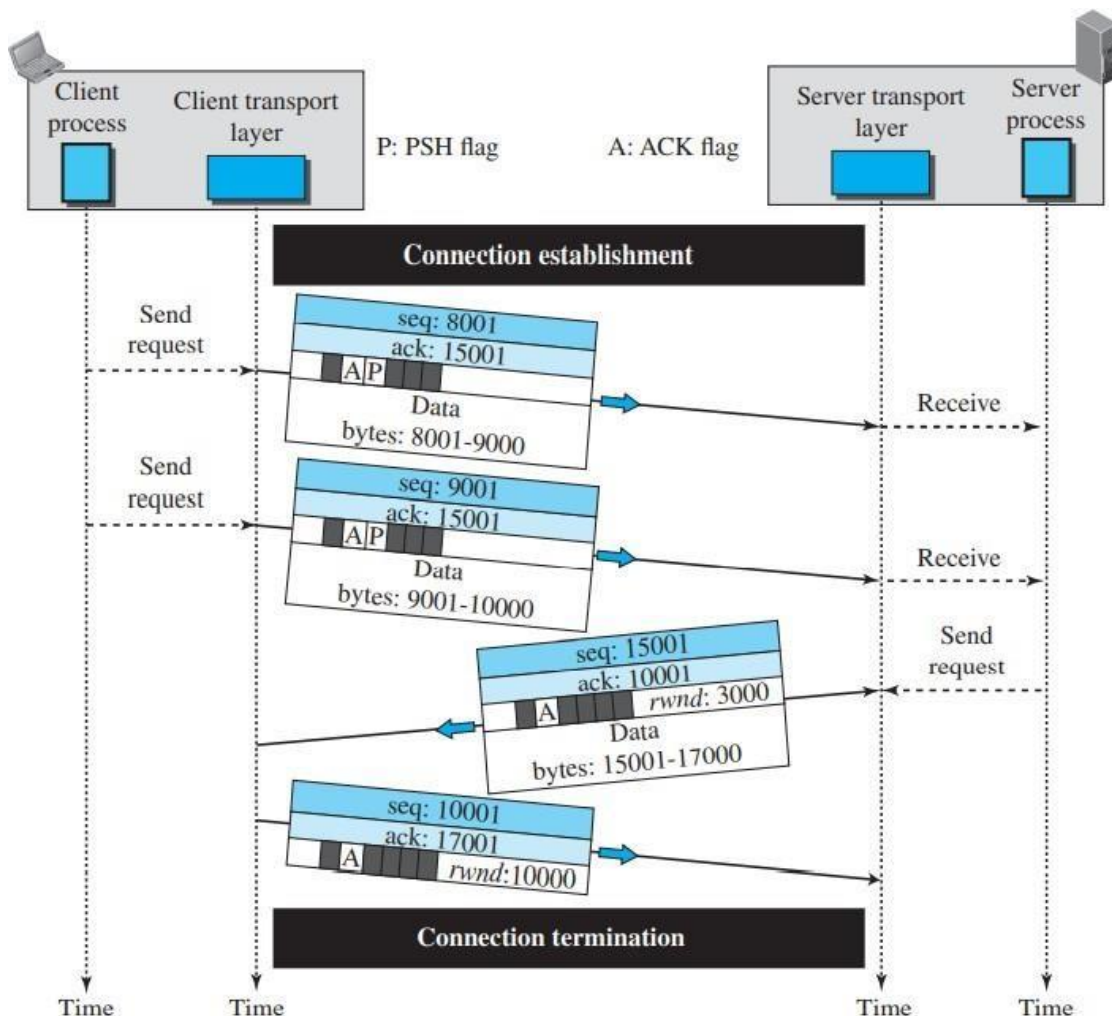
SYN Flooding Attack

The connection establishment procedure in TCP is susceptible to a serious security problem called SYN flooding attack. This SYN flooding attack belongs to a group of security attacks known as a denial-of-service attack, in which an attacker monopolizes a system with so many service requests that the system overloads and denies service to valid requests.

Some implementations of TCP have strategies to alleviate the effect of a SYN attack. Some have imposed a limit of connection requests during a specified period of time. Others try to filter out datagrams coming from unwanted source addresses. One recent strategy is to postpone resource allocation until the server can verify that the connection request is coming from a valid IP address, by using what is called a cookie.

B. Data Transfer

After connection is established, bidirectional data transfer can take place. The client and server can send data and acknowledgments in both directions.



In this example, after a connection is established, the client sends 2,000 bytes of data in two segments. The server then sends 2,000 bytes in one segment. The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there is no more data to be sent.

Pushing Data

The application program at the sender can request a push operation. This means that the sending TCP must not wait for the window to be filled. It must create a segment and send it immediately. The sending TCP must also set the push bit (PSH) to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come.

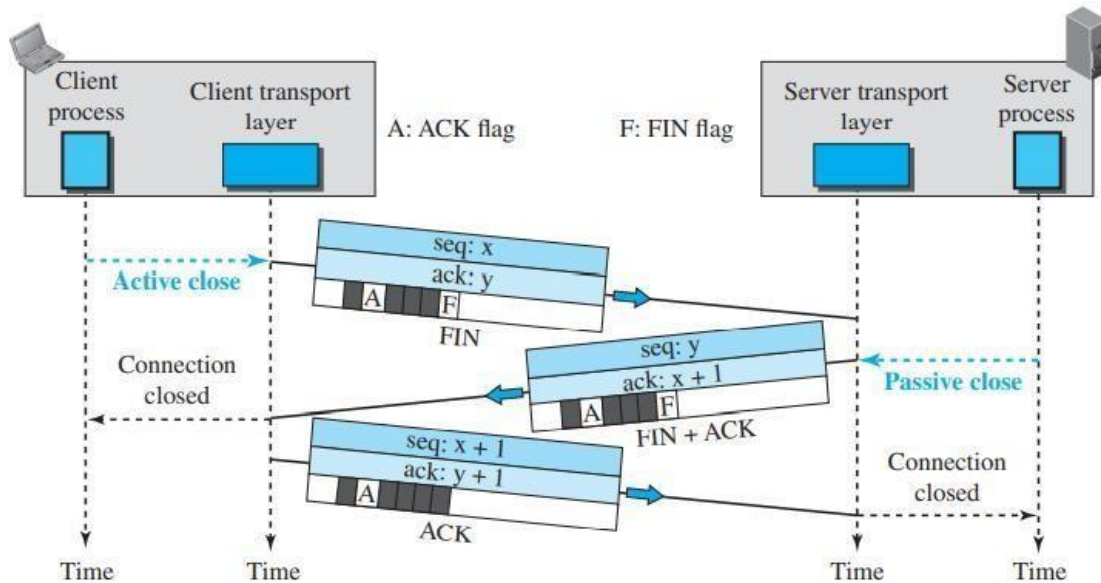
Urgent Data

There are occasions in which an application program needs to send urgent bytes, some bytes that need to be treated in a special way by the application at the other end. The solution is to send a segment with the URG bit set. The sending application program tells the sending TCP that the piece of data is urgent. The sending TCP creates a segment and inserts the urgent data at the beginning of the segment. The rest of the segment can contain normal data from the buffer. The urgent pointer field in the header defines the end of the urgent data (the last byte of urgent data). For example, if the segment sequence number is 15000 and the value of the urgent pointer is 200, the first byte of urgent data is the byte 15000 and the last byte is the byte 15200. The rest of the bytes in the segment (if present) are nonurgent.

C. Connection Termination

Either of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client. Most implementations today allow two options for connection termination: three-way handshaking and four-way handshaking with a half-close option.

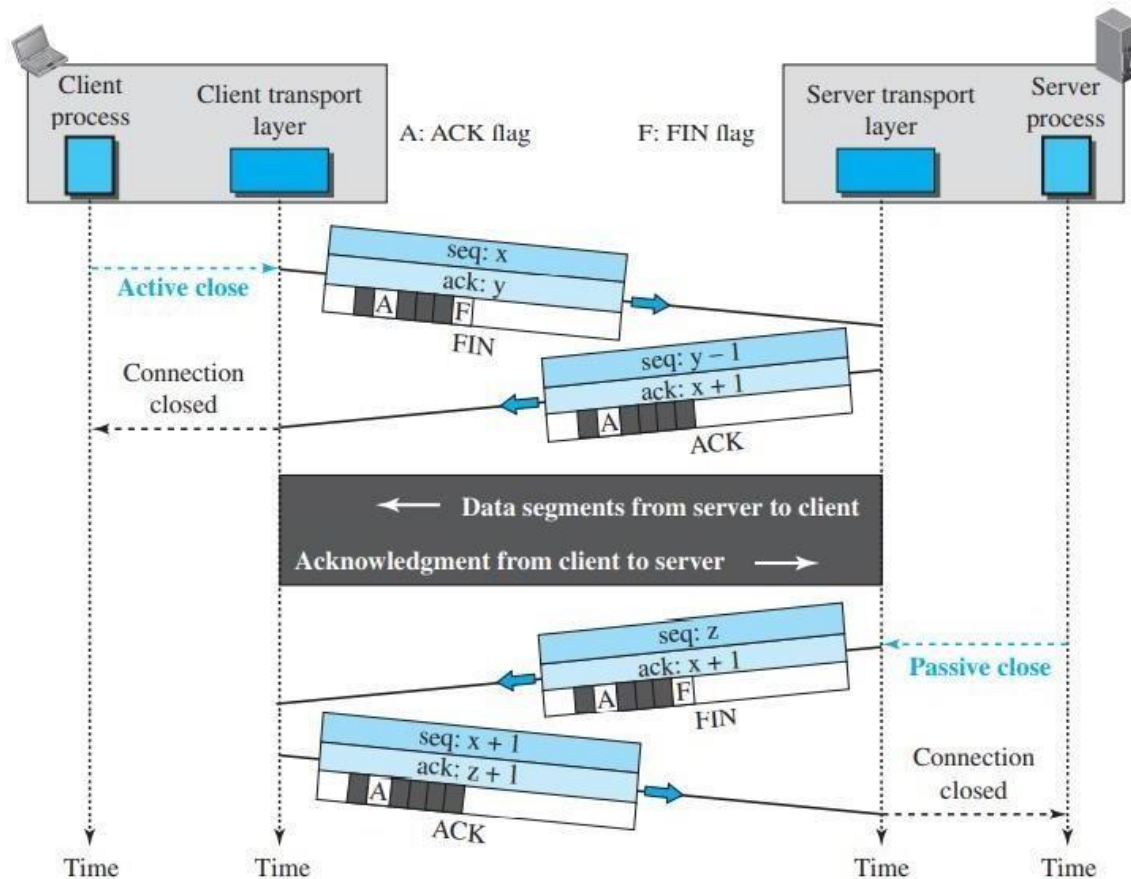
Three-Way Handshaking



1. In this situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set. Note that a FIN segment can include the last chunk of data sent by the client or it can be just a control segment as shown in the figure. If it is only a control segment, it consumes only one sequence number because it needs to be acknowledged.
2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN + ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from the server. If it does not carry data, it consumes only one sequence number because it needs to be acknowledged.
3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is one plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequencenumbers.

Half-Close

In TCP, one end can stop sending data while still receiving data. This is called a half-close.



Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin. A good example is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all data, can close the connection in the client-to-server direction. However, the server-to-client direction must remain open to return the sorted data.

The client half-closes the connection by sending a FIN segment. The server accepts the half-close by sending the ACK segment. The server, however, can still send data. When the server has sent all of the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.

After half-closing the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server.

TCP STATE TRANSITION DIAGRAM

To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the finite state machine (FSM).

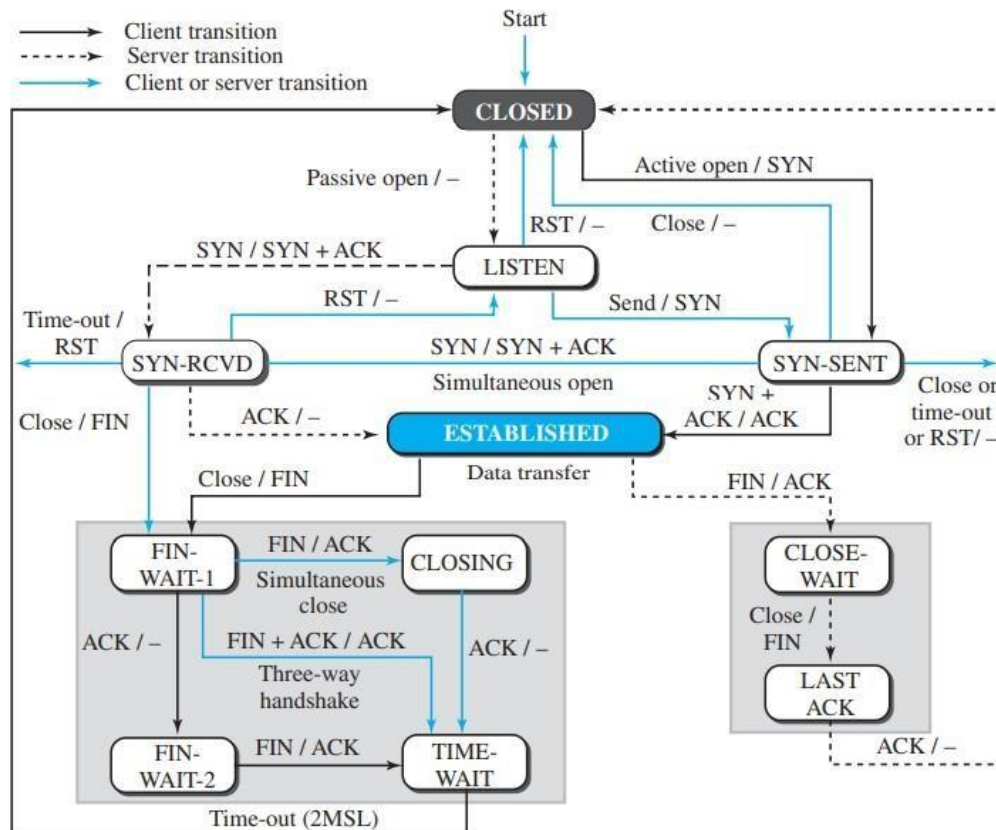


Table 24.2 States for TCP

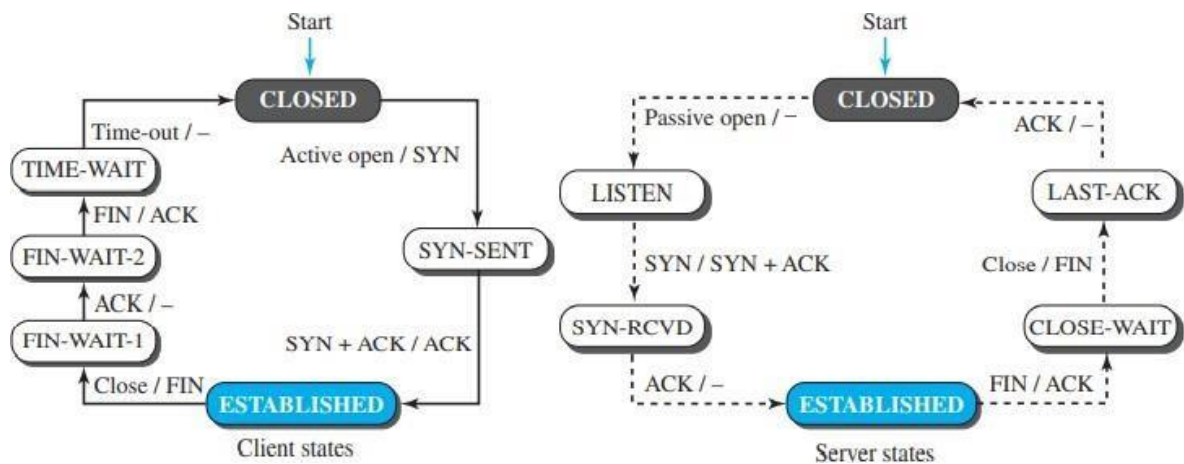
State	Description
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN + ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

- The figure shows the two FSMs used by the TCP client and server combined in one diagram.
- The rounded-corner rectangles represent the states. The transition from one state to another is shown using directed lines.
- Each line has two strings separated by a slash. The first string is the input,

what TCP receives. The second is the output, what TCP sends.

- The dotted black lines in the figure represent the transition that a server normally goes through; the solid black lines show the transitions that a client normally goes through. The colored lines show special situations.

A Half-Close Scenario



The client process issues an active open command to its TCP to request a connection to a specific socket address. TCP sends a SYN segment and moves to the SYN-SENT state. After receiving the SYN + ACK segment, TCP sends an ACK segment and goes to the ESTABLISHED state. Data are transferred, possibly in both directions, and acknowledged. When the client process has no more data to send, it issues a command called an active close. The TCP sends a FIN segment and goes to the FIN-WAIT-1 state. When it receives the ACK segment, it goes to the FIN-WAIT-2 state. When the client receives a FIN segment, it sends an ACK segment and goes to the TIME-WAIT state. The client remains in this state for 2 MSL seconds. When the corresponding timer expires, the client goes to the CLOSED state.

The server process issues a passive open command. The server TCP goes to the LISTEN state and remains there passively until it receives a SYN segment. The TCP then sends a SYN + ACK segment and goes to the SYN-RCVD state, waiting for the client to send an ACK segment. After receiving the ACK segment, TCP goes to the ESTABLISHED state, where data transfer can take place. TCP remains in this state until it receives a FIN segment from the client signifying that there are no more data to be exchanged and the connection can be closed. The server, upon receiving the FIN segment, sends all queued data to the server with a virtual EOF marker, which means that the connection must be closed. It sends an ACK segment and goes to the CLOSEWAIT state, but postpones acknowledging the FIN segment received from the client until it receives a passive close command from its process. After receiving the passive close command, the server sends a FIN segment to the client and goes to the LASTACK state, waiting for the final ACK. When the ACK segment is received from the client, the server goes to the CLOSE state.

CONGESTION CONTROL

What is the need for congestion control? (2 marks) Nov 2021

- Congestion in a network may occur if the load on the network-the number of packets sent to the network-is greater than the capacity of the network-the number of packets a network can handle.
- Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.
- Congestion in a network or internetwork occurs because routers and switches have queues-buffers that hold the packets before and after processing.

Congestion Control in TCP

To control the number of segments to transmit, TCP uses another variable called a congestion window, *cwnd*, whose size is controlled by the congestion situation in the network.

The *cwnd* variable and the *rwnd* variable together define the size of the send window in TCP. The first is related to the congestion in the middle (network); the second is related to the congestion at the end. The actual size of the window is the minimum of these two.

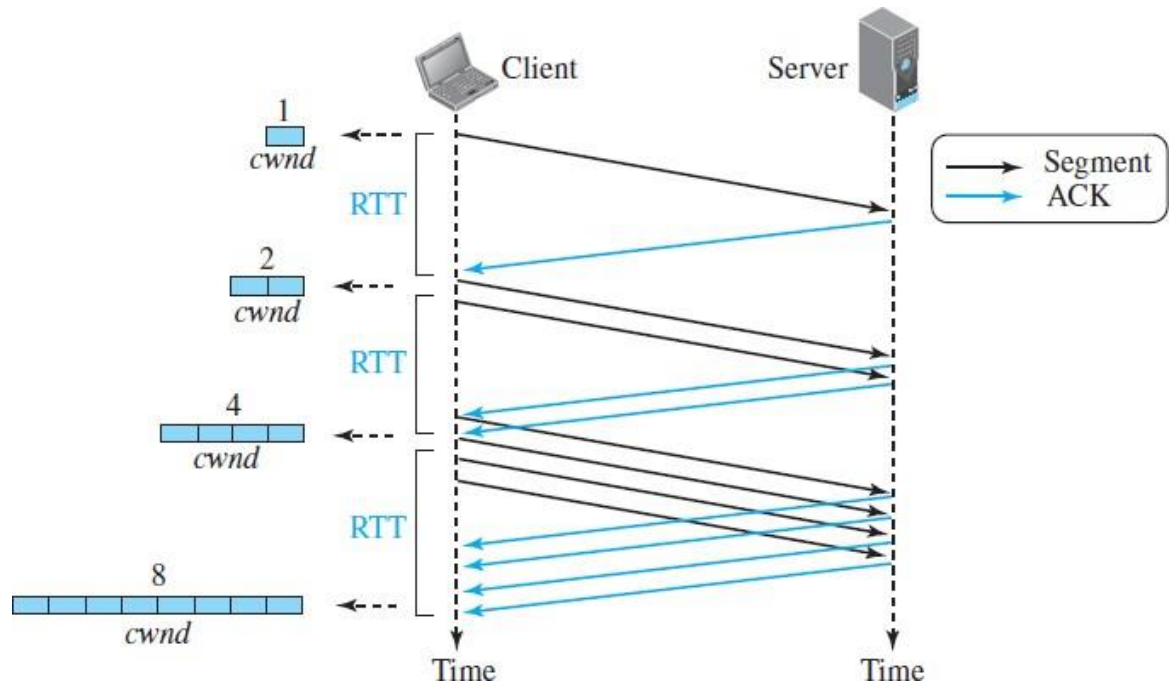
$$\text{Actual window size} = \text{minimum} (rwnd, cwnd)$$

Congestion Policies

TCP's general policy for handling congestion is based on three algorithms: **slow start, congestion avoidance, and fast recovery.**

1. Slow Start: Exponential Increase

- The slow-start algorithm is based on the idea that the size of the congestion window (*cwnd*) starts with one maximum segment size (MSS), but it increases one MSS each time an acknowledgment arrives.
- The algorithm starts slowly, but grows exponentially. To show the idea, let us look at Figure below. Assume that *rwnd* is much larger than *cwnd*, so that the sender window size always equals *cwnd*. We also assume that each segment is of the same size and carries MSS bytes. For simplicity, we also ignore the delayed-ACK policy and assume that each segment is acknowledged individually.
- The sender starts with *cwnd*=1. This means that the sender can send only one segment. After the first ACK arrives, the acknowledged segment is purged from the window, which means there is now one empty segment slot in the window. The size of the congestion window is also increased by 1 because the arrival of the acknowledgment is a good sign that there is no congestion in the network. The size of the window is now 2. After sending two segments and receiving two individual acknowledgments for them, the size of the congestion window now becomes 4, and so on.



- The size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined as follows.

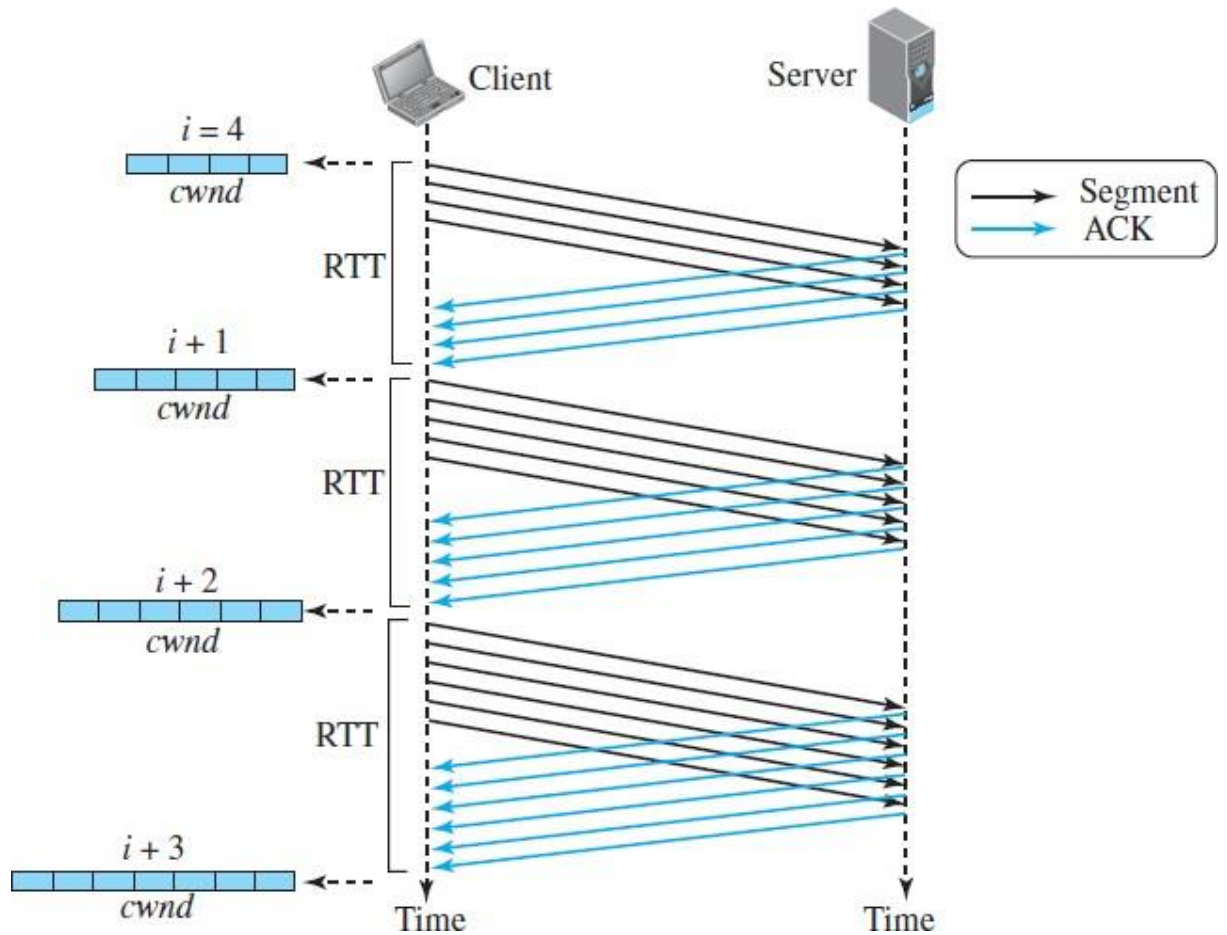
If an ACK arrives, $cwnd = cwnd + 1$

Start	→	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	→	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	→	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	→	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

A slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named ssthresh (slow-start threshold). When the size of the window in bytes reaches this threshold, slow start stops and the next phase starts.

2. Congestion Avoidance: Additive Increase

If we continue with the slow-start algorithm, the size of the congestion window increases exponentially. To avoid congestion before it happens, we must slow down this exponential growth. TCP defines another algorithm called congestion avoidance, which increases the cwnd additively instead of exponentially. When the size of the congestion window reaches the slow-start threshold in the case where $cwnd = i$, the slow-start phase stops and the additive phase begins. In this algorithm, each time the whole "window" of segments is acknowledged, the size of the congestion window is increased by one. A window is the number of segments transmitted during RTT. Figure below shows the idea.



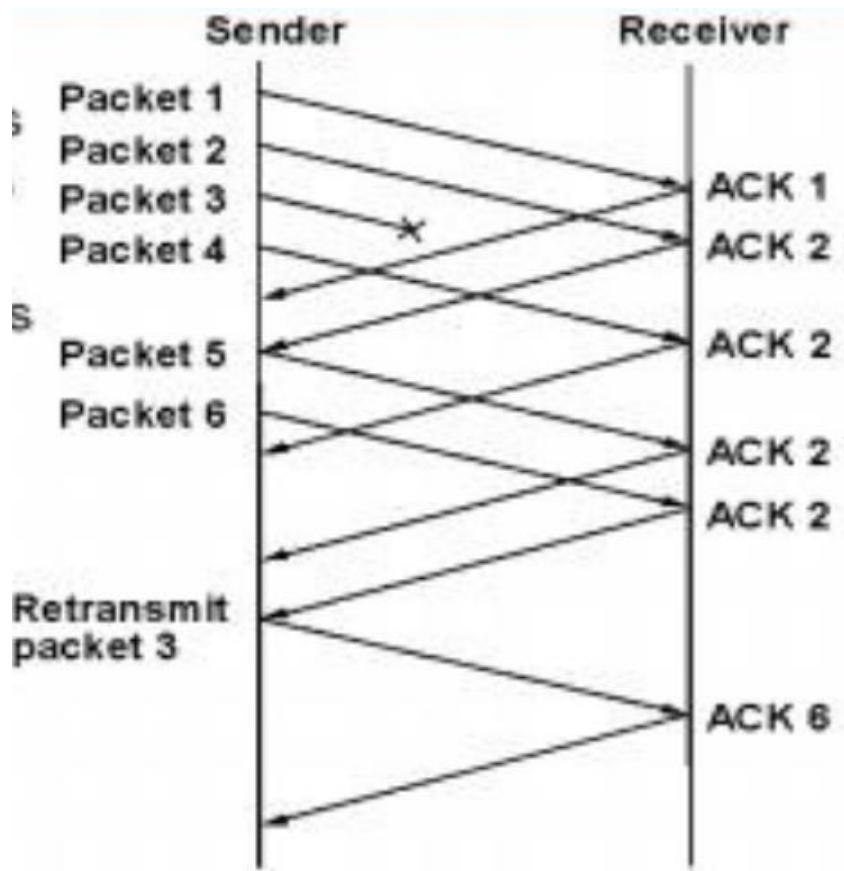
If an ACK arrives, $cwnd = cwnd + (1/cwnd)$

Start	→	$cwnd = i$
After 1 RTT	→	$cwnd = i + 1$
After 2 RTT	→	$cwnd = i + 2$
After 3 RTT	→	$cwnd = i + 3$

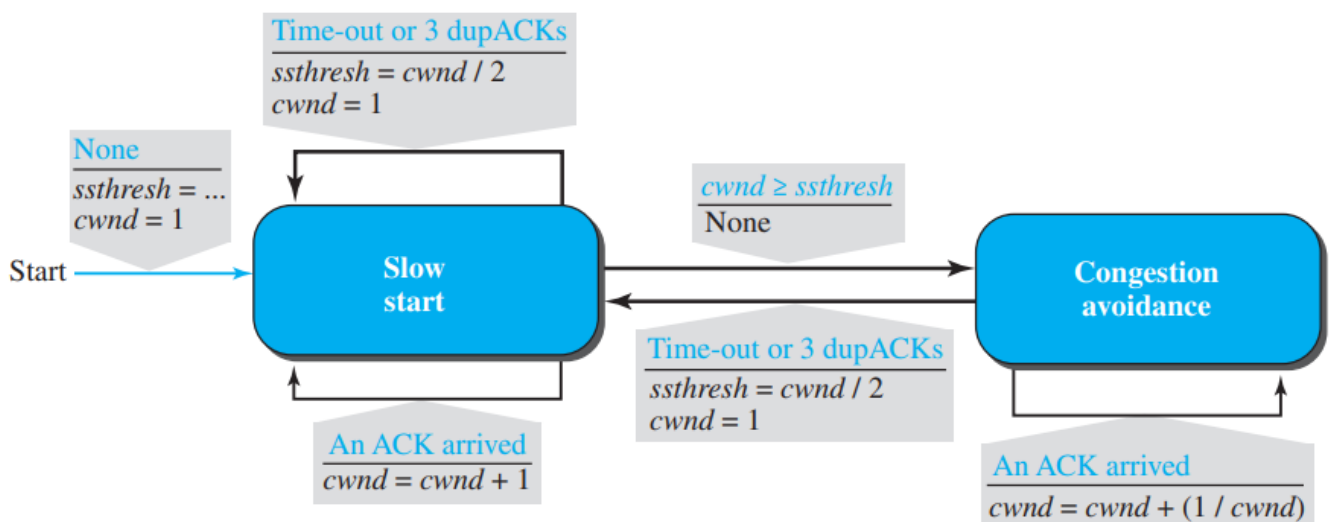
3. Fast Recovery

The fast-recovery algorithm is optional in TCP. The old version of TCP did not use it, but the new versions try to use it. It starts when three duplicate ACKs arrive, which is interpreted as light congestion in the network. Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives (after the three duplicate ACKs that trigger the use of this algorithm). We can say

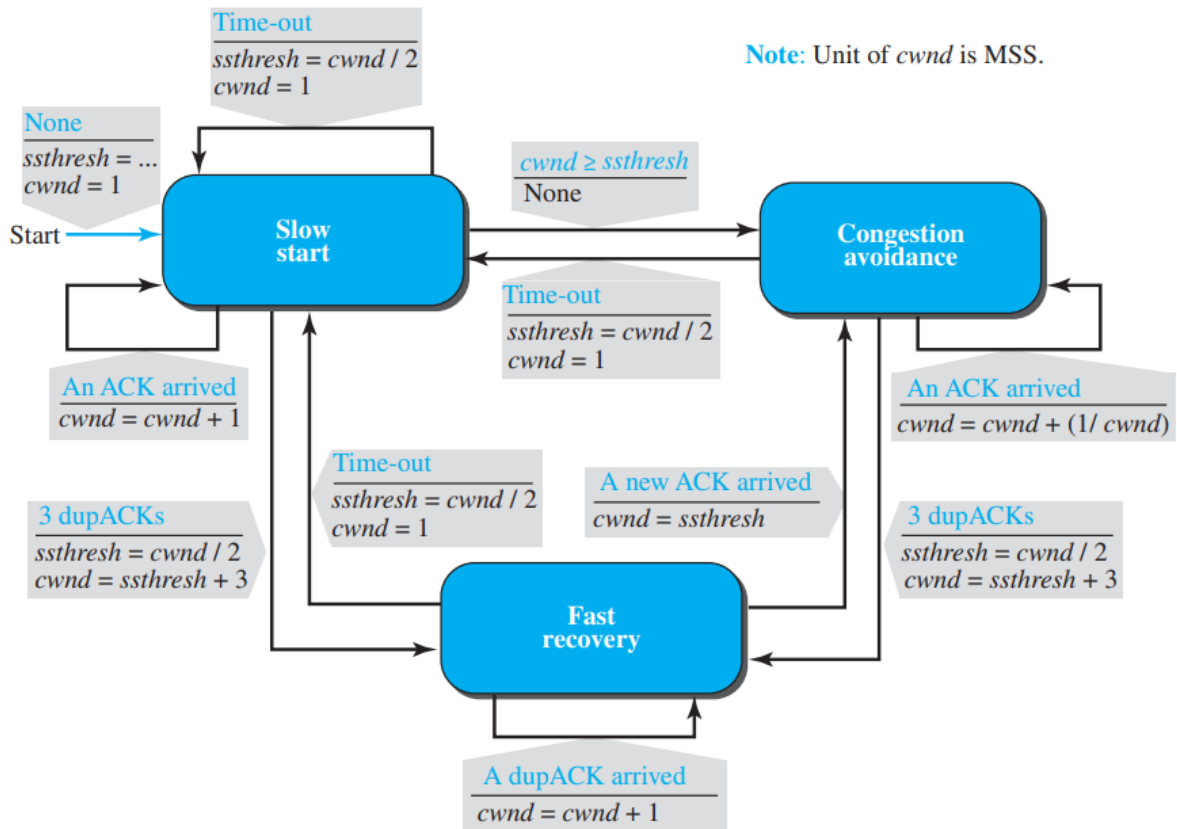
If a duplicate ACK arrives, $cwnd = cwnd + (1/cwnd)$



FSM for Tahoe TCP (older version)



FSM for Reno TCP (Newer Version)



CONGESTION AVOIDANCE (DECBIT, RED)

Discuss in length the applications of congestion avoidance. (13 maks) Nov 2021

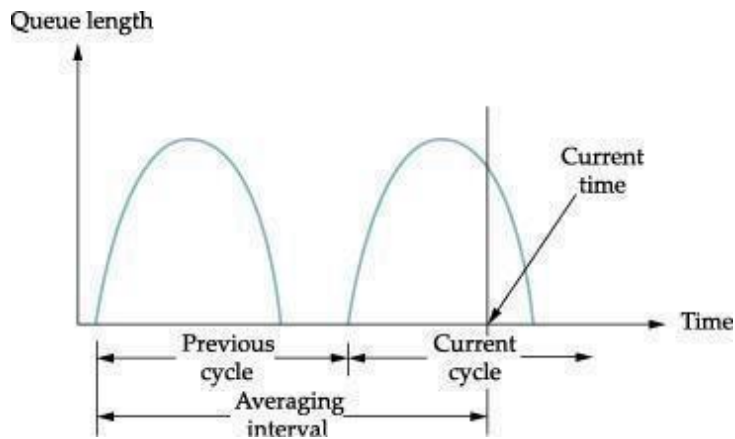
Congestion avoidance mechanisms prevent congestion before it actually occurs. When congestion is likely to occur, TCP decreases load on the network. TCP creates loss of packets in order to determine bandwidth of the connection. The two congestion-avoidance mechanisms are:

DECbit

DECbit was developed for use on Digital Network Architecture. In DECbit, each router monitors the load it is experiencing and explicitly notifies the end node when congestion is about to occur by setting a binary congestion bit called DECbit in packets that flow through it. The destination host copies the DECbit onto the ACK and sends back to the source. Eventually the source reduces its transmission rate and congestion is avoided.

Algorithm

A single congestion bit is added to the packet header. A router sets this bit in a packet if its average queue length is greater than or equal to 1 at the time packet arrives. The average queue length is measured over a time interval that spans the last busy + last idle cycle + current busy cycle. Router calculates average queue length by dividing the curve area by time interval.



The source computes how many ACK has DECBIT set for the previous window packets it has sent. If it is less than 50% then source increases its congestion window by 1 packet. Otherwise, source decrease the congestion window by 87.5%.

Random Early Detection (RED)

Proposed by Floyd and Jackson. In RED, router implicitly notifies the source that congestion is likely to occur by dropping one of its packets. The source is notified by timeout or duplicate ACK. The router drops a few packets earlier before it runs out of space, so that it need not drop more packets later. Each incoming packet is dropped with a probability known as drop probability when the queue length exceeds drop level.

Algorithm

The RED algorithm contains the following components:

- Calculation of average queue length.
- Calculation of drop probability, whether the packet will be enqueued or dropped depends on this probability.
- Decision-making logic (helps to decide whether the incoming packet should be enqueued or dropped).

Calculation of Average Queue Length:

On arrival of every packet, RED calculates the average queue length using Eq. (1). This mathematical model is known as 'Exponential Weighted Moving Average' or EWMA.

$$\Rightarrow \text{newavg} = (1 - w_q) \times \text{oldavg} + w_q \times \text{current_queue_len} \text{ Eq. (1)}$$

\Rightarrow where, newavg = new average queue length being calculated in this sample

\Rightarrow oldavg = old average queue length obtained during the previous sample

\Rightarrow current_queue_len = 'instantaneous' queue length at the router

\Rightarrow w_q = weight associated with the 'current_queue_len'. Default value: 0.002

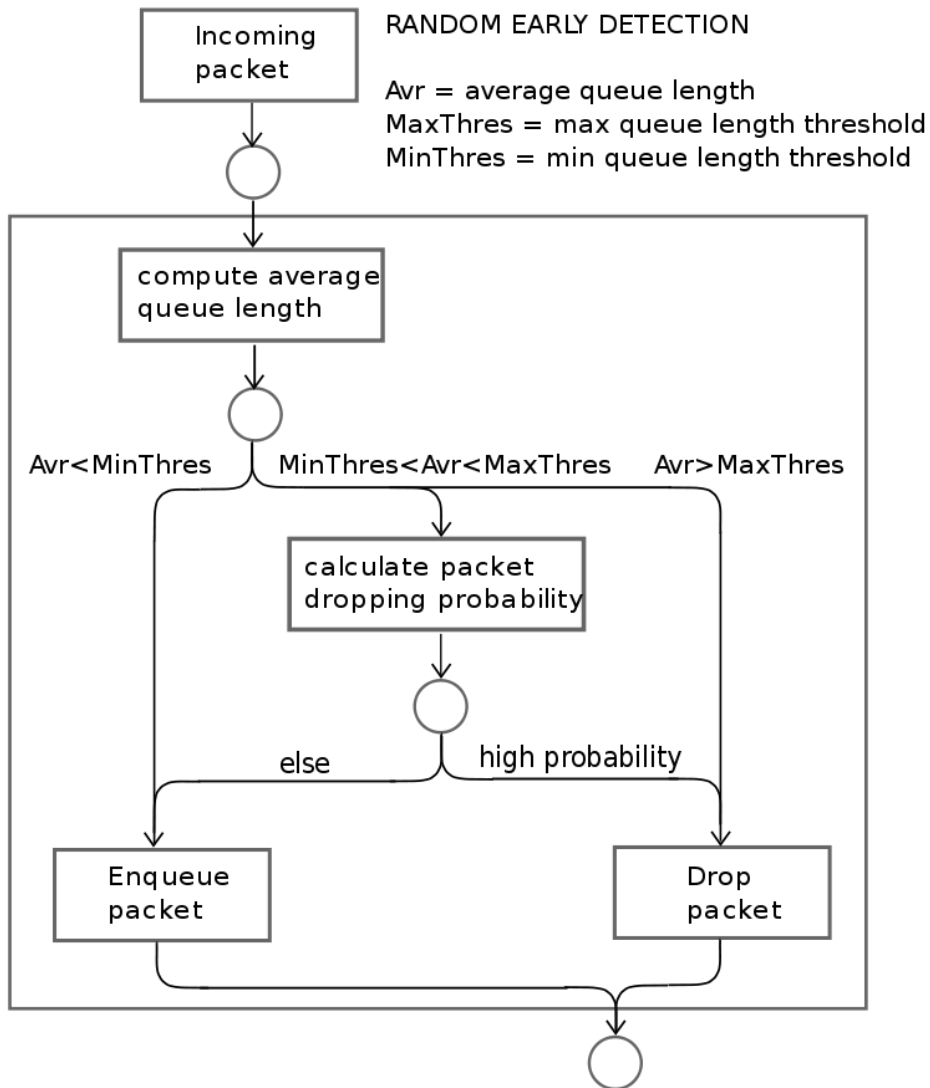
Calculation of drop probability

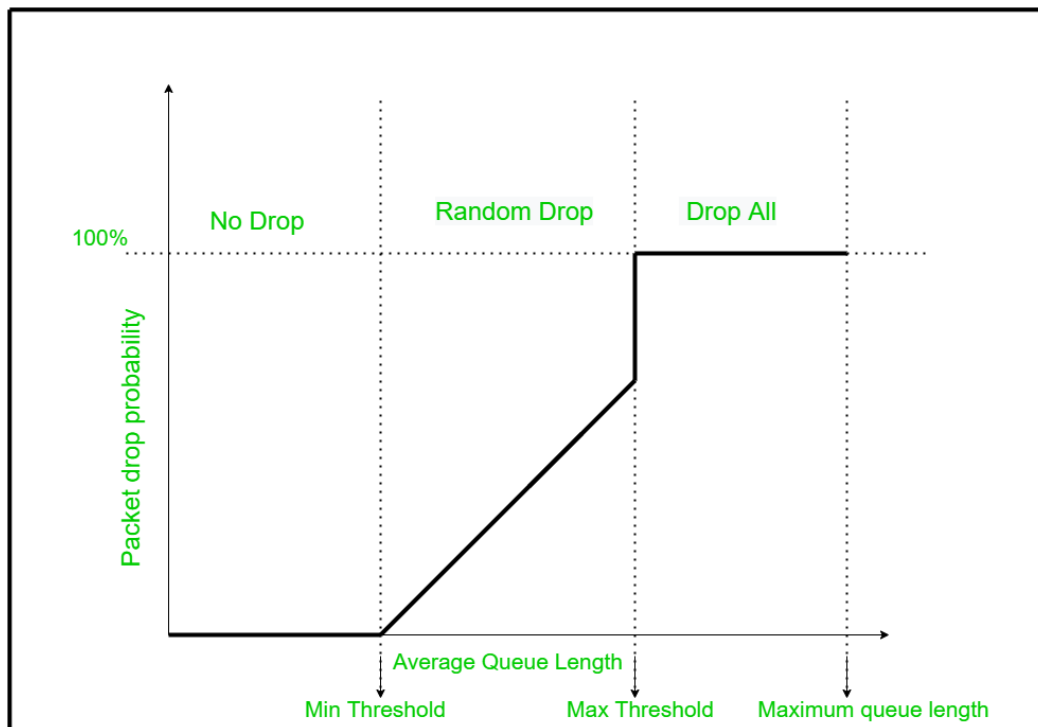
$$P_d = \max_p \times \left[\frac{(\text{newavg} - \text{min}_{th})}{(\text{max}_{th} - \text{min}_{th})} \right]$$

where, \max_p = maximum drop probability. Default: 0.5

min_{th} represents the minimum threshold for 'average queue length'

max_{th} represents the maximum threshold for 'average queue length'.





The left part is the “No drop” region. If the average queue length is less than equal to the minimum threshold value then the packet will be enqueued. The right part is the “drop all” region. If the average queue length is greater than the maximum threshold value then the packet will be dropped. If the average queue length is in between min and max threshold values then the drop probability of that packet will be calculated. Based on that value, the final decision will be made about enqueue or discard.

QOS : QULAITY OF SERVICE

What are the techniques to control QoS or Write in detail the principle of establishment of QoS through differentiated services?

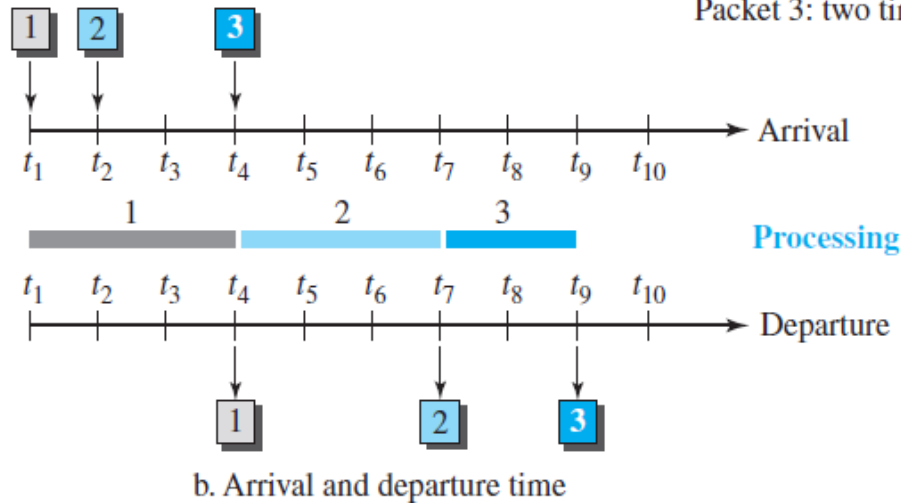
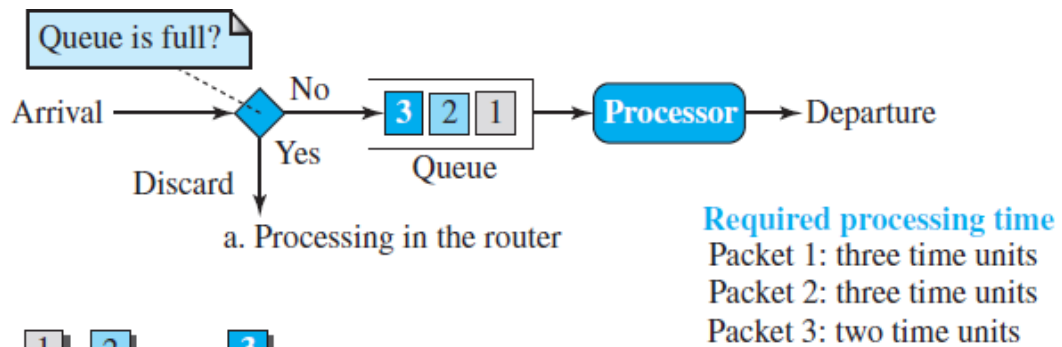
Some techniques that can be used to improve the quality of service. The four common methods: scheduling, traffic shaping, admission control, and resource reservation.

a. 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.

i. 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.

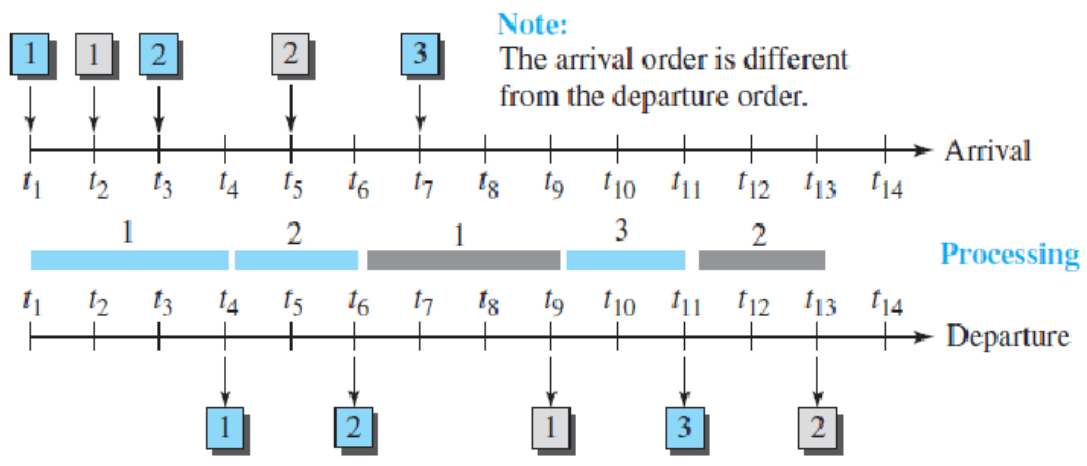
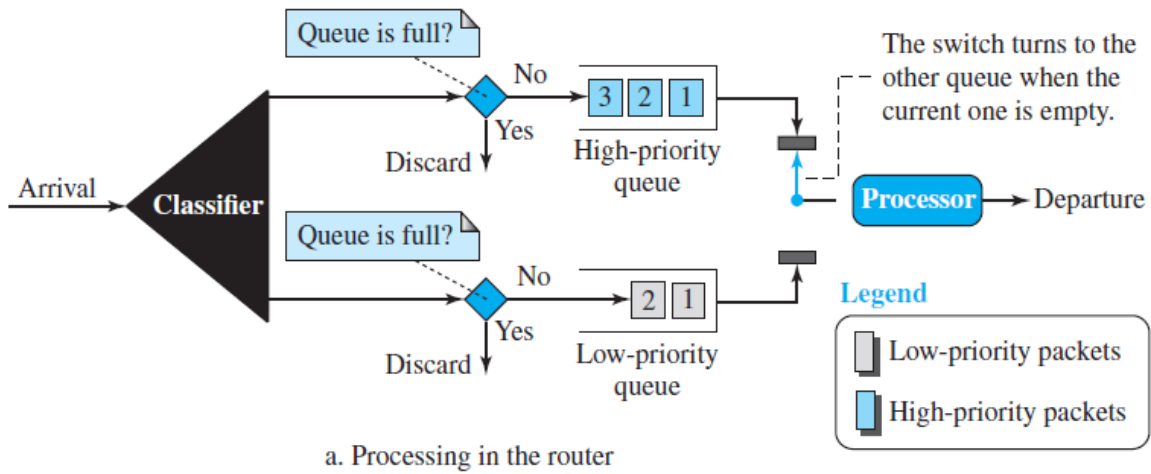


Packets from different applications (with different sizes) arrive at the queue, are processed, and depart. A larger packet definitely may need a longer processing time. In the figure, packets 1 and 2 need three time units of processing, but packet 3, which is smaller, needs two time units. This means that packets may arrive with some delays but depart with different delays.

ii. Priority Queuing

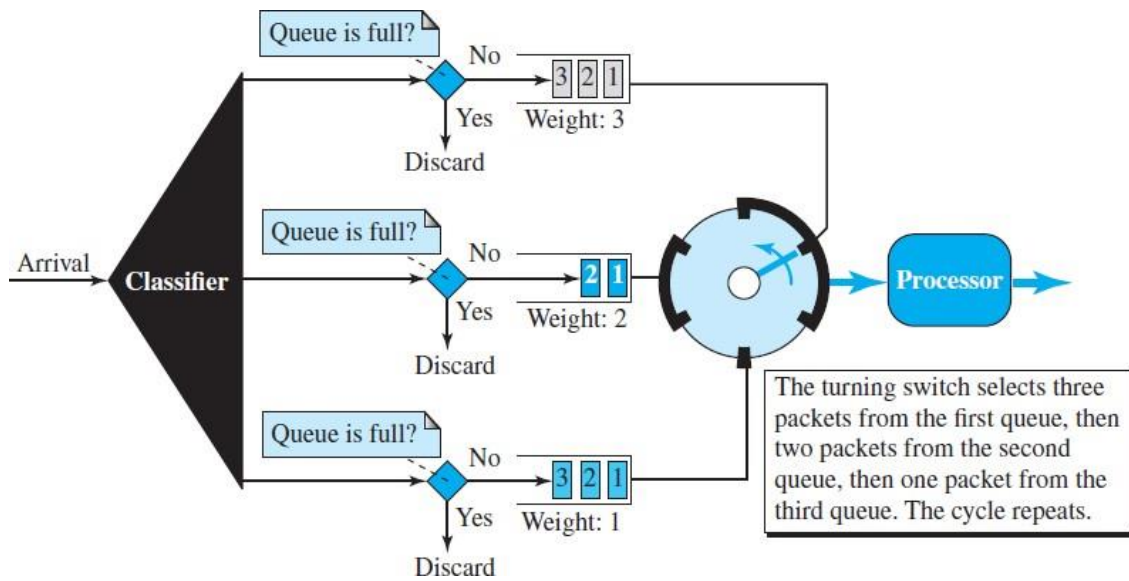
In priority queuing, packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving a queue until it is empty. Figure shows priority queuing with two priority levels (for simplicity).

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



iii. 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 shows the technique with three classes.

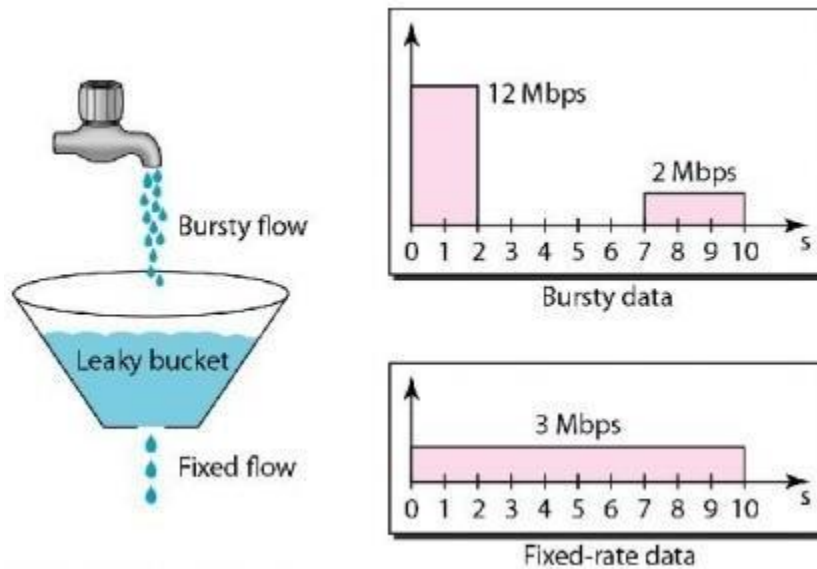


b. Traffic Shaping

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

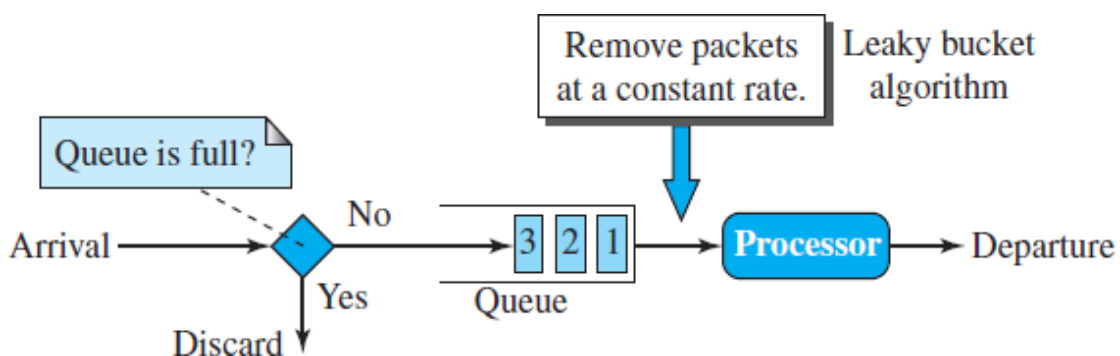
i. Leaky Bucket

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 4.34 shows a leaky bucket and its effects.



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 4.34 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 10s. The leaky bucket smooth's the traffic by sending out data at a rate of 3 Mbps during the same 10 s.

A simple leaky bucket implementation is shown in Figure



ii. 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.

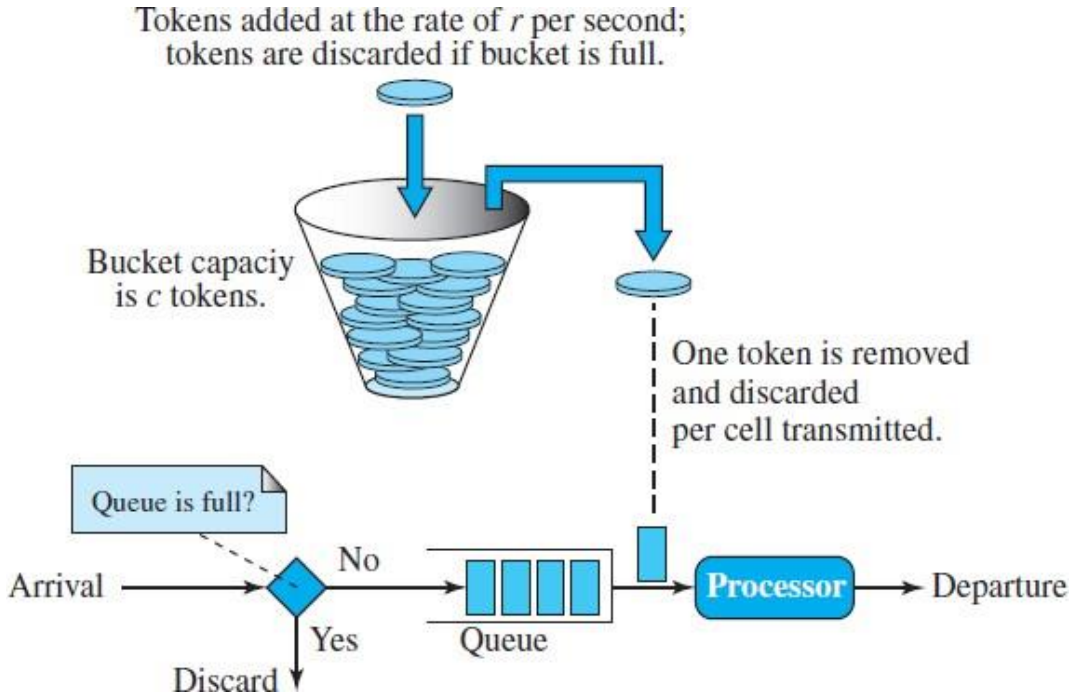
Assume the capacity of the bucket is c tokens and tokens enter the bucket at the rate of r tokens per second. The system removes one token for every cell of data sent. The maximum number of cells that can enter the network during any time interval of length t is shown below.

$$\text{Maximum number of packets} = r \times t + c$$

The maximum average rate for the token bucket is shown below.

$$\text{Maximum average rate} = (r \times t + c)/t \text{ packets per second}$$

This means that the token bucket limits the average packet rate to the network.



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.

c. 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.

d. 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 can handle the new flow. It takes into account bandwidth, buffer size, CPU speed, etc., as well as its previous commitments to other flows.

Application Layer

Introduction

- The application layer provides services to the user. 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.
- The application layer, however, is somewhat 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.

APPLICATION LAYER PARADIGMS

- To use 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.
- Two paradigms have been developed based on how the application programs request and provide services. They are client-server paradigm and 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.
- Normally few server processes are available 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.
- For example, a telephone directory center in any area can be a server; a subscriber that calls and asks for a specific telephone number can be thought of as a client. The directory center must be ready and available all the time; the subscriber can call the center for a short period when the service is needed.
- Figure below shows an example of a client-server communication in which three clients communicate with one server to receive the services provided by this server

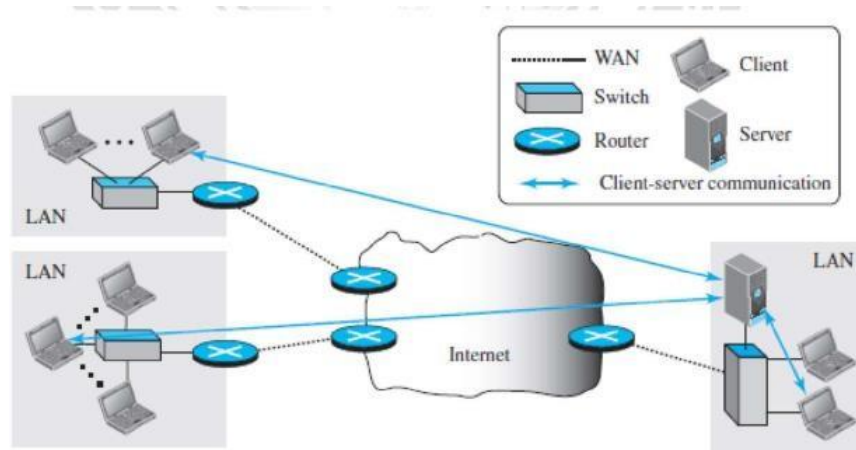


Fig. Client Server Paradigm

Peer-to-Peer paradigm

- 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 below shows an example of communication in this paradigm.

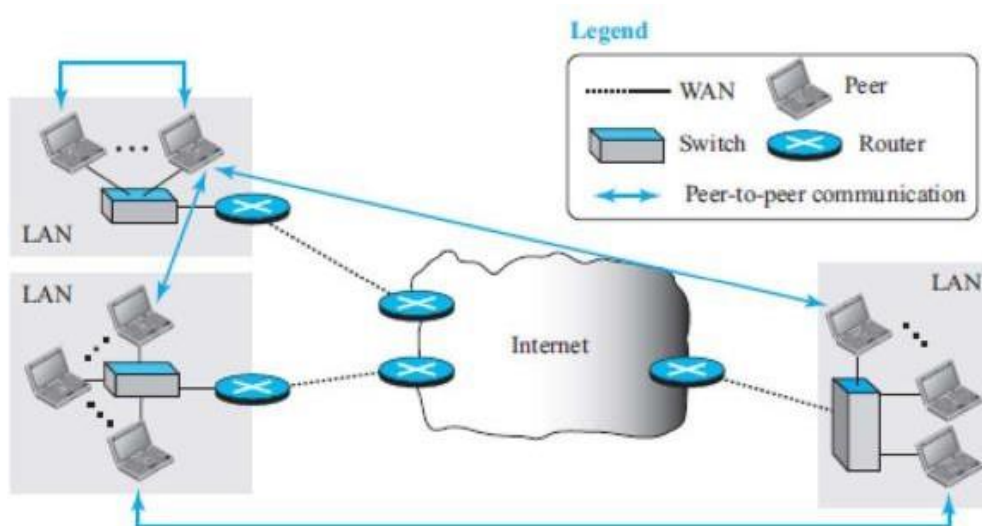


Fig. Peer to peer paradigm

Communication by phone is a peer-to-peer activity; no party needs to wait for the other party to call. The peer-to-peer paradigm can be used in a situation, when some computers connected to the Internet have something to share with each other. For example, if an

Internet user has a file available to share with other Internet users, there is no need for the file holder to become a server and run a server process all the time waiting for other users to connect and to get the file

CLIENT-SERVER PROGRAMMING

State the need for client side programming. (2 marks) Nov2021

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.

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 sends a finite number of requests to the corresponding server, receives the responses, and stops.

APPLICATION PROGRAMMING INTERFACE

- 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 type is called 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.
- Several APIs have been designed for communication. Three are common: socket interface, Transport Layer Interface (TLI), and STREAM.

SOCKET INTERFACE

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. It is a set of instructions that can be used by a process to communicate with another process.

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).

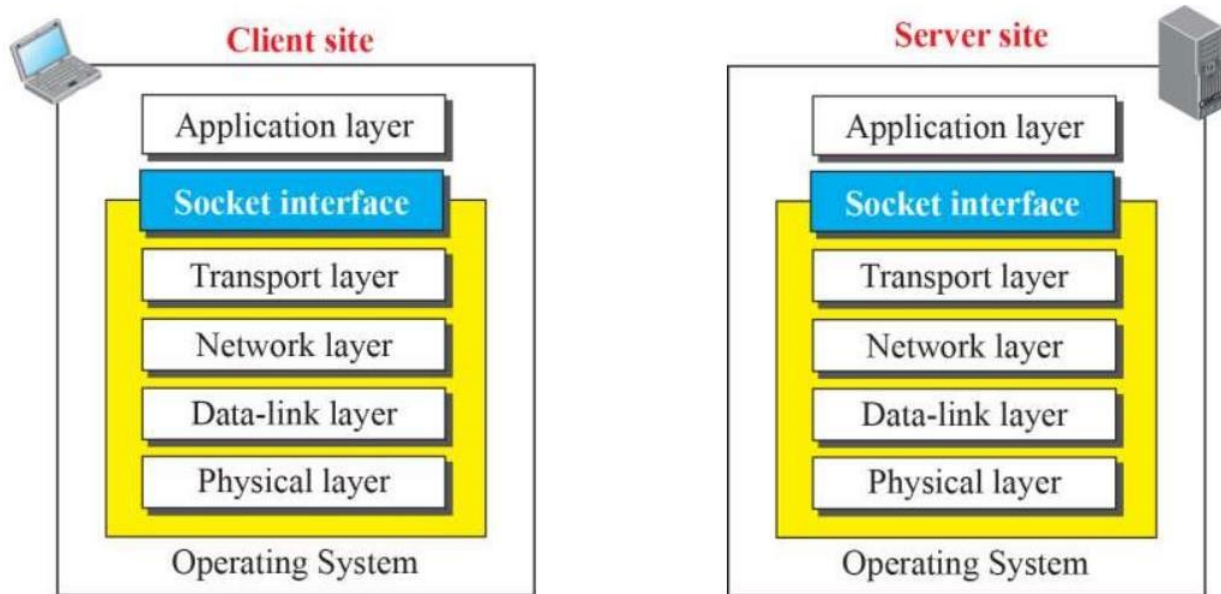


Fig. Position of socket interface

- The idea of sockets allows us to use the set of all instructions already designed in a programming language for other sources and sinks.
- We can use the same instructions to read from or write to sockets. In other words, we are adding only new sources and sinks to the programming language without changing the way we send data or receive data. Figure 5.4 shows the idea and compares the sockets with other sources and sinks.
- Although a 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.

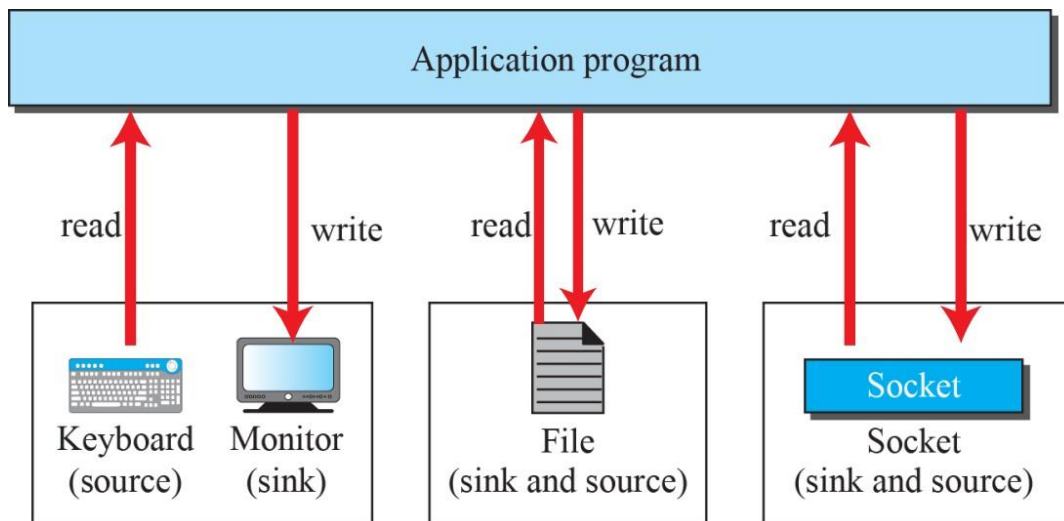


Fig Socket Format

In application layer, communication between a client process and a server process is the communication between two sockets. 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 5.5. 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.

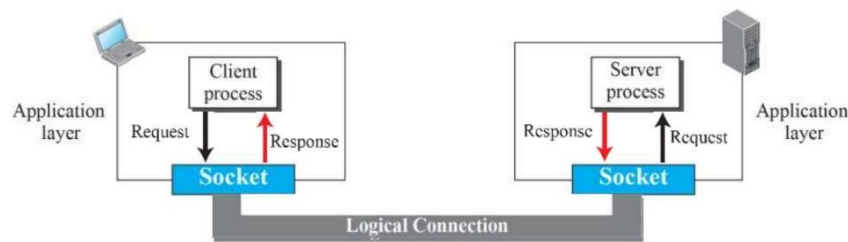


Fig. Socket in process-to-process communication.

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. A socket address should first define the computer on which a client or a server is running. A computer in the Internet is defined by its IP address. a socket address should be a combination of an IP address and a port number as shown below:



Fig. A Socket address.

FINDING SOCKET ADDRESSES

To find socket address, the situation is different for each site.

Server Site

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. 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.

Client Site

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.

Remote Socket Address

When a client process starts, it should know the socket address of the server it wants to connect to. Although each standard application has a well-known port number, most of the time, we do not know the IP 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). DNS maps the server name to the IP address of the computer running that server.

WORLD WIDE WEB

Discuss the evolution of world wide web with its current status. (8 marks) Nov 2021

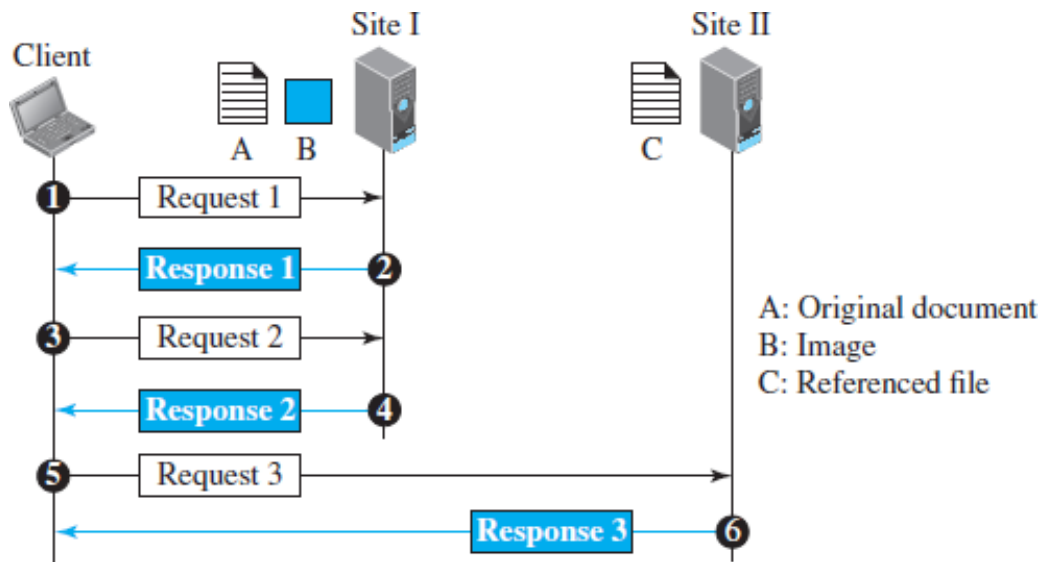
- The Web is a repository of information in which the documents, called web pages, are distributed all over the world and related documents are linked together.
- Each web server in the world can add a new web page to the repository and announce it to all Internet users without overloading a few servers.
- 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. The Web implemented this idea electronically to allow the linked document to be retrieved when the link was clicked by the user.

Architecture

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

links to other web pages; a composite web page has one or more links to other web pages.

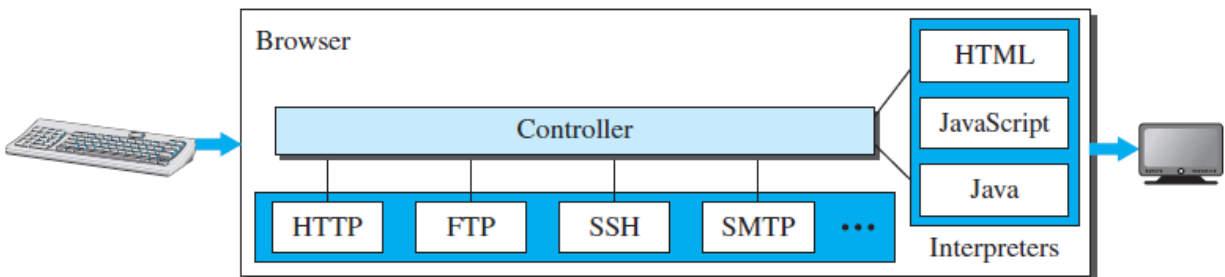
- Each web page is a file with a name and address.



The main document and the image are stored in two separate files (file A and file B) in the same site; the referenced text file (file C) is stored in another site. We need three transactions if we want to see the whole document. The first transaction (request/response) retrieves a copy of the main document (file A), which has references (pointers) to the second and third files. When a copy of the main document is retrieved and browsed, the user can click on the reference to the image to invoke the second transaction and retrieve a copy of the image (file B). If the user needs to see the contents of the referenced text file, she can click on its reference (pointer) invoking the third transaction and retrieving a copy of file C.

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



The controller receives input from the keyboard or the mouse and uses the client programs to access the document. After the document has been accessed, the controller uses one of the interpreters to display the document on the screen. The client protocol can be one of the protocols described later, such as HTTP or FTP. The interpreter can be HTML, Java, or

JavaScript, depending on the type of document. Some commercial browsers include Internet Explorer, Netscape Navigator, and Firefox.

Web Server

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

Uniform Resource Locator (URL)

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

protocol://host/path

Used most of the time

protocol://host:port/path

Used when port number is needed

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), we can also use other protocols such as FTP (File Transfer Protocol).

Host. The host identifier can be the IP address of the server or the unique name given to the server.

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.

Web Documents

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

Static Documents

Static documents are fixed-content documents that are created and stored in a server. The client can get a copy of the document only. In other words, the contents of the file are determined when the file is created, not when it is used and the contents in the server can be changed, but the user cannot change them.

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.

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.

HYPERTEXT TRANSFER PROTOCOL(HTTP)

Describe in detail about HTTP with neat diagram

Explain the complexities in HTTP with suitable examples. (5) Nov 2021

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. HTTP uses the services of TCP, which, as discussed before, is a connection-oriented and reliable protocol. This means that, before any transaction between the client and the server can take place, a connection needs to be established between them. After the transaction, the connection should be terminated.

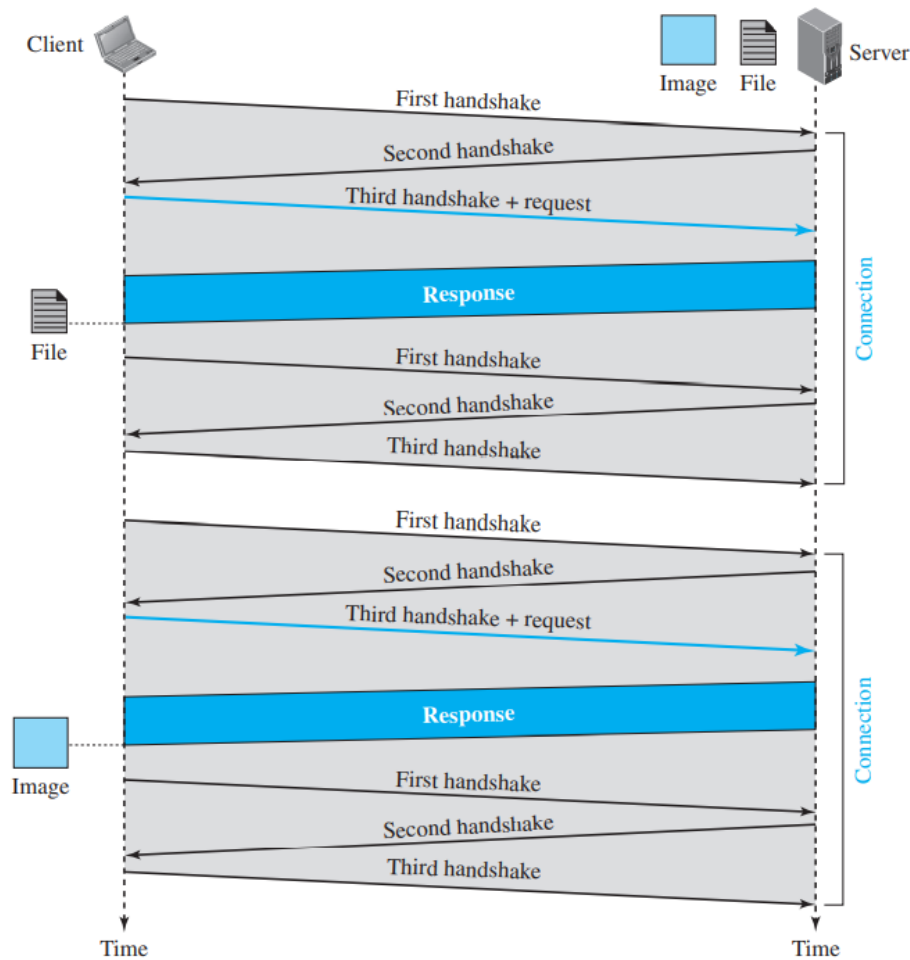
Nonpersistent versus Persistent Connections

If the web pages, objects to be retrieved, are located on different servers, we do not have any other choice than to create a new TCP connection for retrieving each object. However, if some of the objects are located on the same server, we have two choices: to retrieve each object using a new TCP connection or to make a TCP connection and retrieve them all.

Nonpersistent Connections

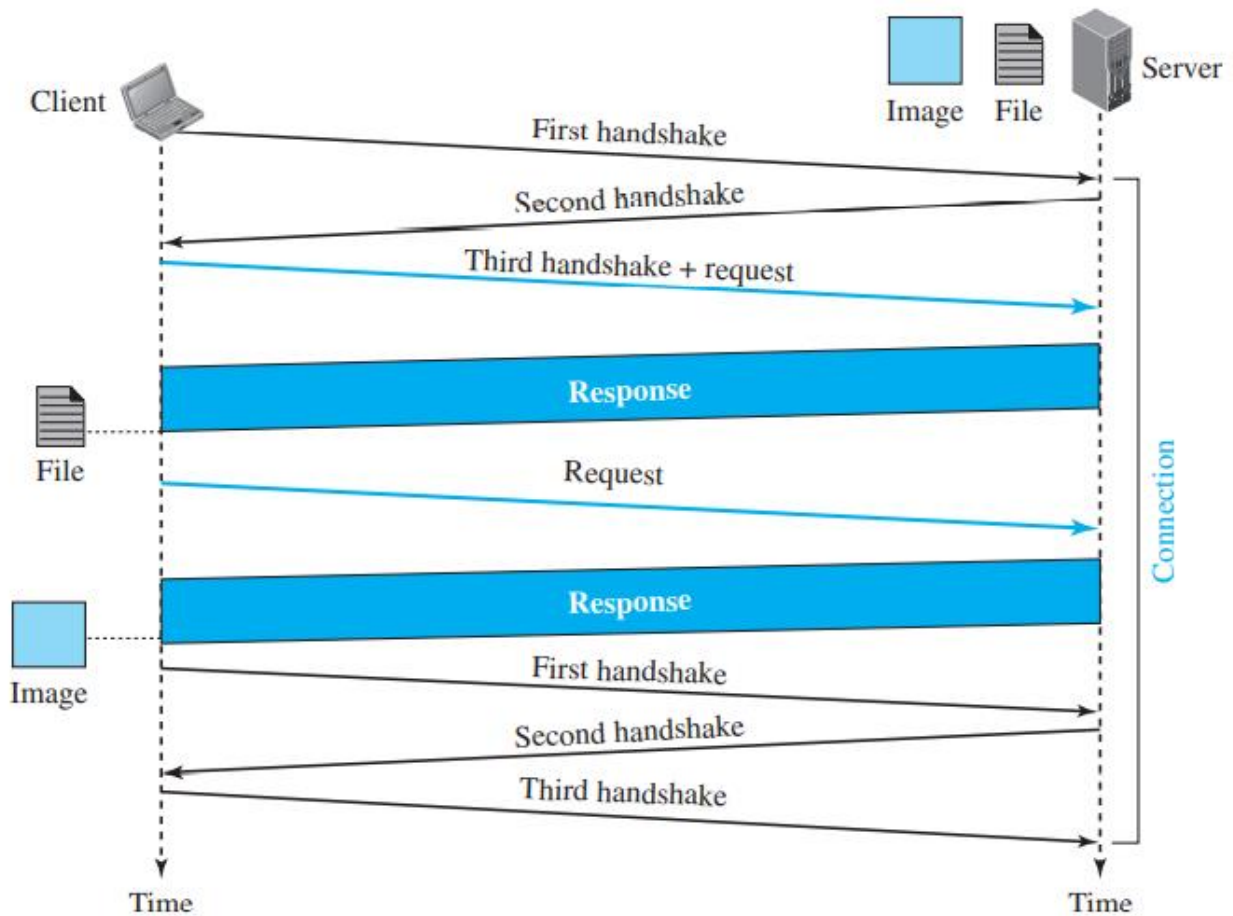
In a nonpersistent connection, one TCP connection is made for each request/response. The following lists the steps in this strategy:

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



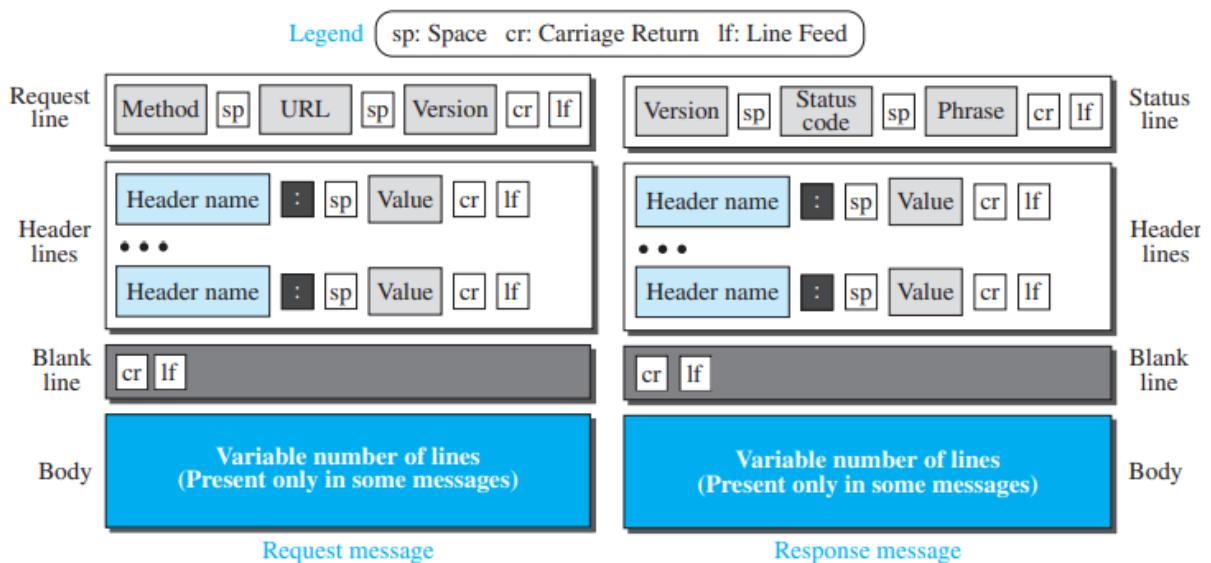
Persistent Connections

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. 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

Each message is made of four sections. The first section in the request message is called the request line; the first section in the response message is called the status line. The other three sections have the same names in the request and response messages. However, the similarities between these sections are only in the names; they may have different contents.



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). First field is method and the list of methods are given below.

<i>Method</i>	<i>Action</i>
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

- 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.
- 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.

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.
- 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. 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.

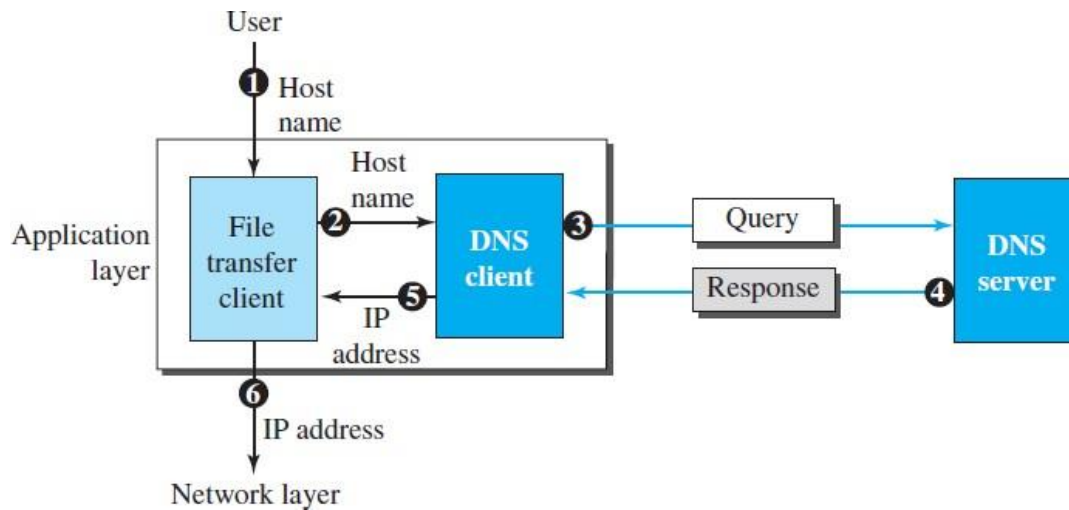
DOMAIN NAME SYSTEM (DNS)

Explain the role of a DNS on a computer network.

- TCP/IP protocols use the IP address to identify an entity. However, people prefer to use names instead of numeric addresses. Therefore, the Internet needs to have a directory system that can map a name to an address.
- 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. This

method is used by the Domain Name System (DNS).

TCP/IP uses a DNS client and a DNS server to map a name to an address.



The following six steps map the host name to an IP address:

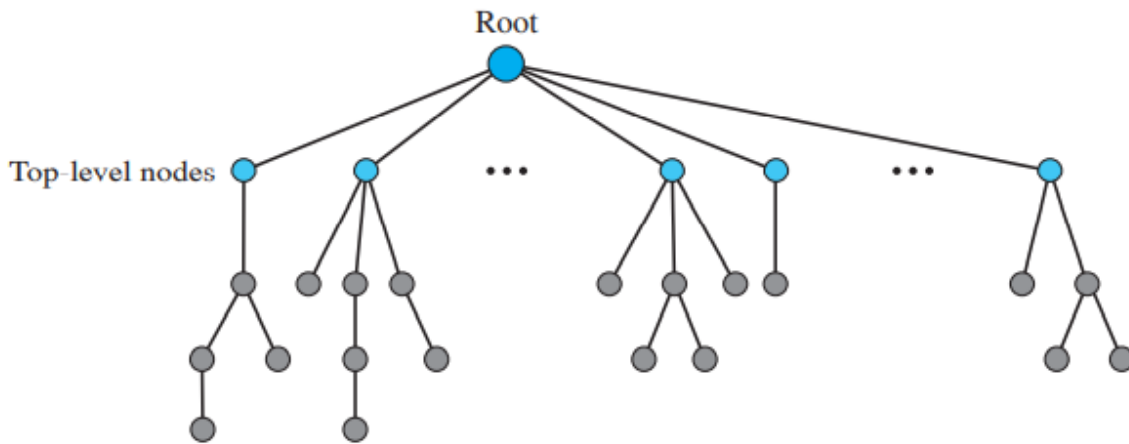
1. The user passes the host name to the file transfer client.
2. The file transfer client passes the host name to the DNS client.
3. Each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server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

Name Space

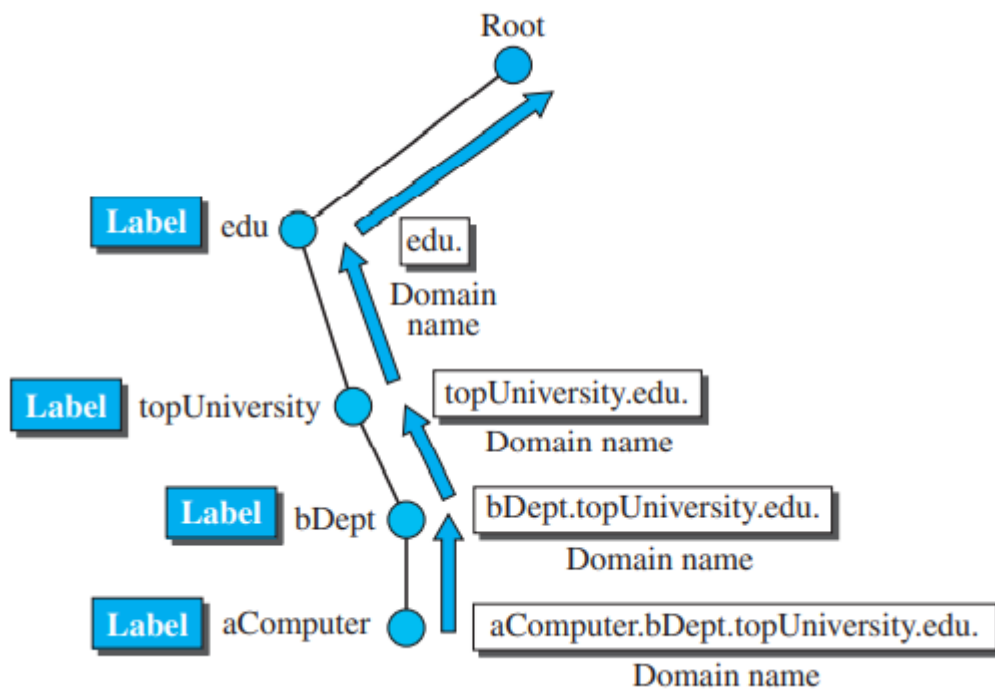
- The names must be unique because the addresses are unique. A name space that maps each address to a unique name can be organized in two ways: flat or hierarchical.
- In a flat name space, a name is assigned to an address. A name in this space is a sequence of characters without structure.
- The main 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. Even if part of an address is the same, the whole address is different.

Domain Name Space

- Domain name space uses hierarchical name space. 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.

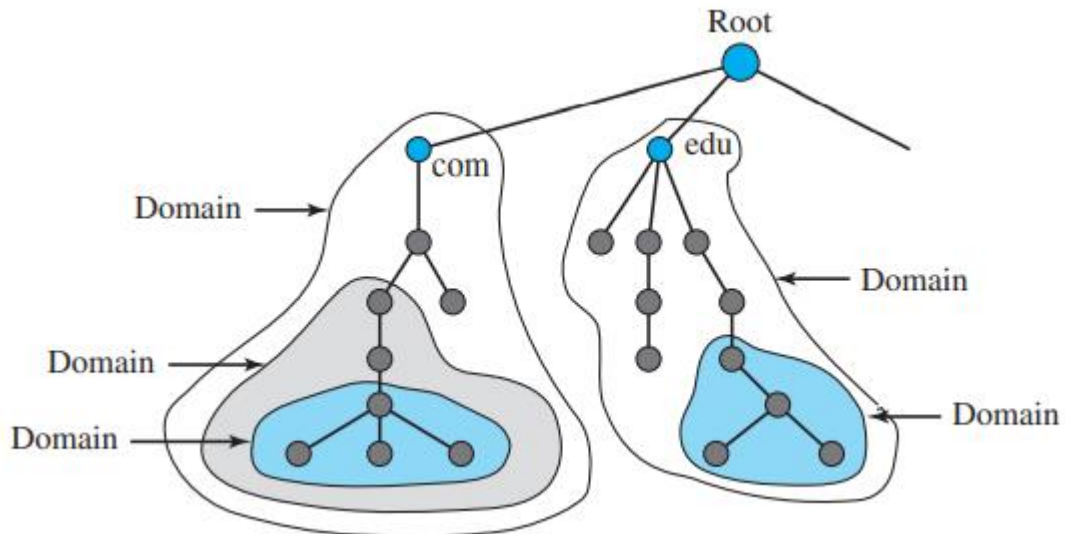


- Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.
- Each node in 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.
- If a label is terminated by a null string, it is called a fully qualified domain name (FQDN). If a label is not terminated by a null string, it is called a partially qualified domain name (PQDN).



Domain

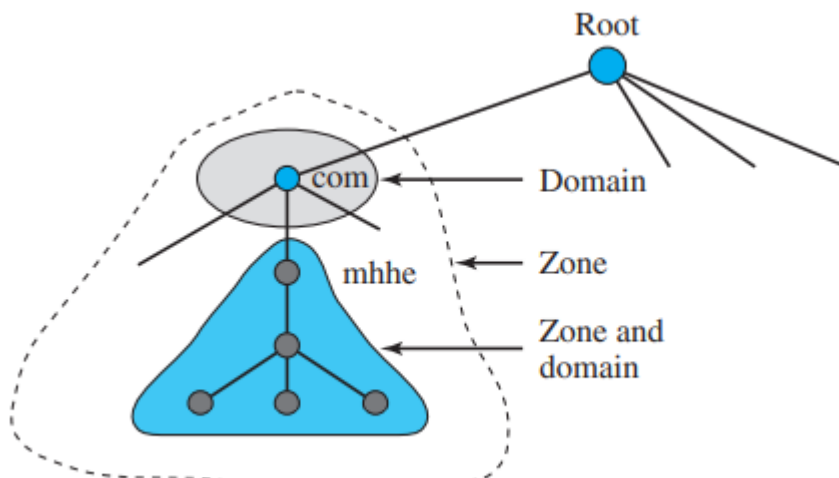
- A domain is a subtree of the domain name space. The name of the domain is the name of the node at the top of the subtree. A domain may itself be divided into domains.



- 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.
- 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.

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.
- If a server accepts responsibility for a domain and does not divide the domain into smaller domains, the “domain” and the “zone” refer to the same thing. The server makes a database called a zone file and keeps all the information for every node under that domain.
- However, if a server divides its domain into subdomains and delegates part of its authority to other servers, “domain” and “zone” refer to different things. The information about the nodes in the subdomains is stored in the servers at the lower levels, with the original server keeping some sort of reference to these lower-level servers.



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.

Primary and Secondary Servers

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

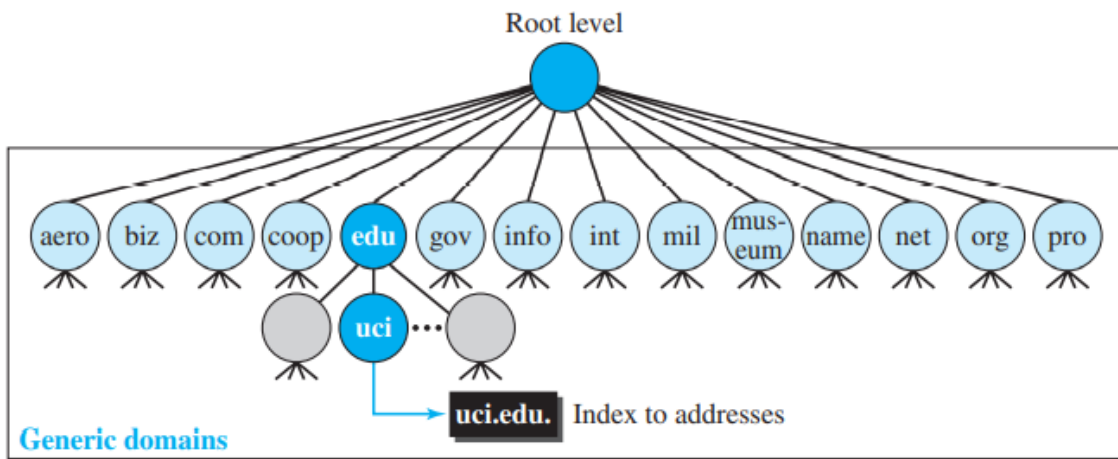
A secondary server is a server that transfers the complete information about a zone from another server (primary or secondary) and stores the file on its local disk. The secondary server neither creates nor updates the zone files.

DNS in the Internet

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

Generic Domains

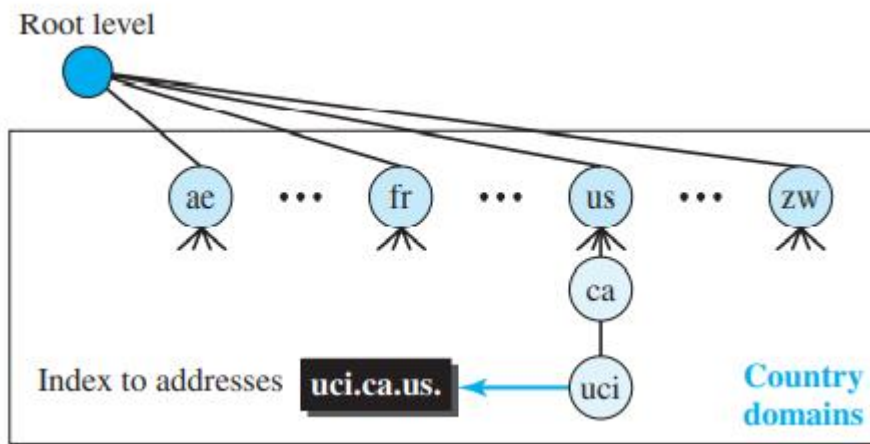
The generic domains define registered hosts according to their generic behavior. Each node in the tree defines a domain, which is an index to the domain name space database.



<i>Label</i>	<i>Description</i>	<i>Label</i>	<i>Description</i>
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.



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.

ELECTRONIC MAIL (SMTP, POP3, IMAP, MIME)

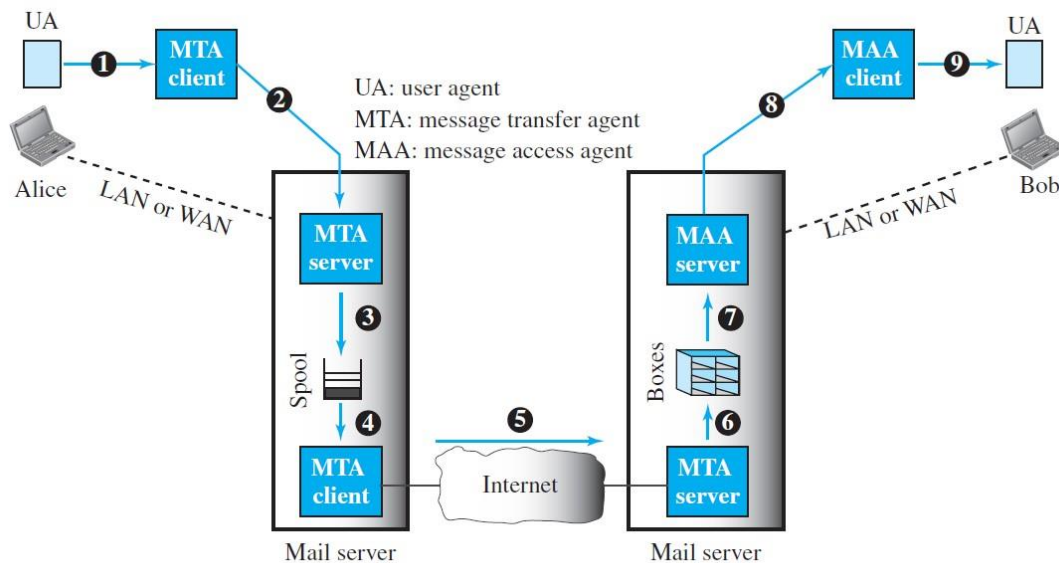
Describe the message format, the message transfer and the underlying protocol involved in the working of an electronic mail. (13)

Discuss the protocols used for Electronic mail. (13) Nov 2021

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 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:



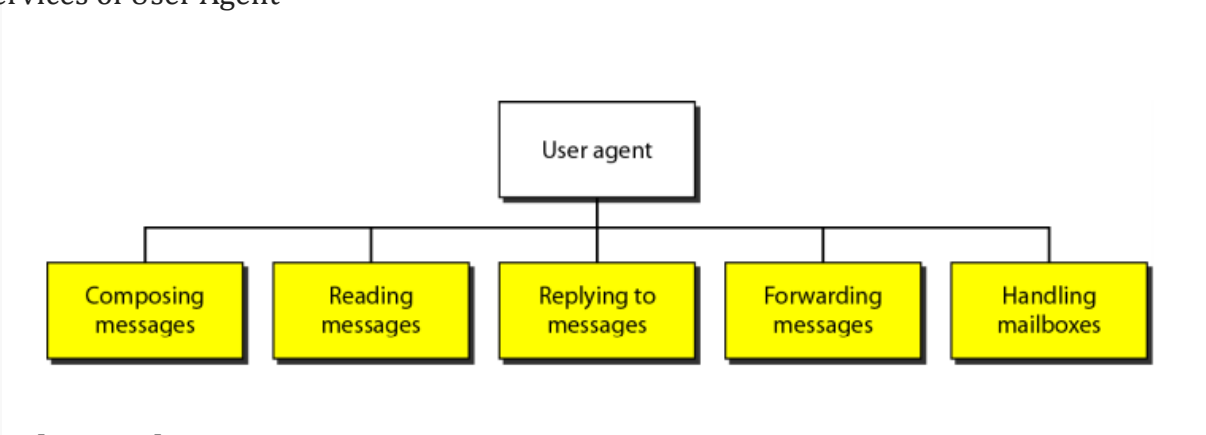
- In the common scenario, the sender and the receiver of the e-mail, Alice and Bob respectively, are connected via a LAN or a WAN to two mail servers.
- The administrator has created one mailbox for each user where the received messages are stored. A mailbox is part of a server hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it.
- The administrator has also created a queue (spool) to store messages waiting to be sent.
- A simple e-mail from Alice to Bob takes nine different steps, as shown in the figure.
- Alice and Bob use three different agents: a user agent (UA), a message transfer agent (MTA), and a message access agent (MAA).
- When Alice needs to send a message to Bob, she runs a UA program to prepare the message and send it to her mail server.
- The mail server at her site uses a queue (spool) to store messages waiting to be sent.
- The message, however, needs to be sent through the Internet from Alice's site to Bob's site using an MTA. Here two message transfer agents are needed: one client and one server. Like most client-server programs on the Internet, the server needs to run all the time because it does not know when a client will ask for a connection. The client, on the other hand, can be triggered by the system when there is a message in the queue to be sent.
- The user agent at the Bob site allows Bob to read the received message. Bob later uses an MAA client to retrieve the message from an MAA server running on the second server.

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.

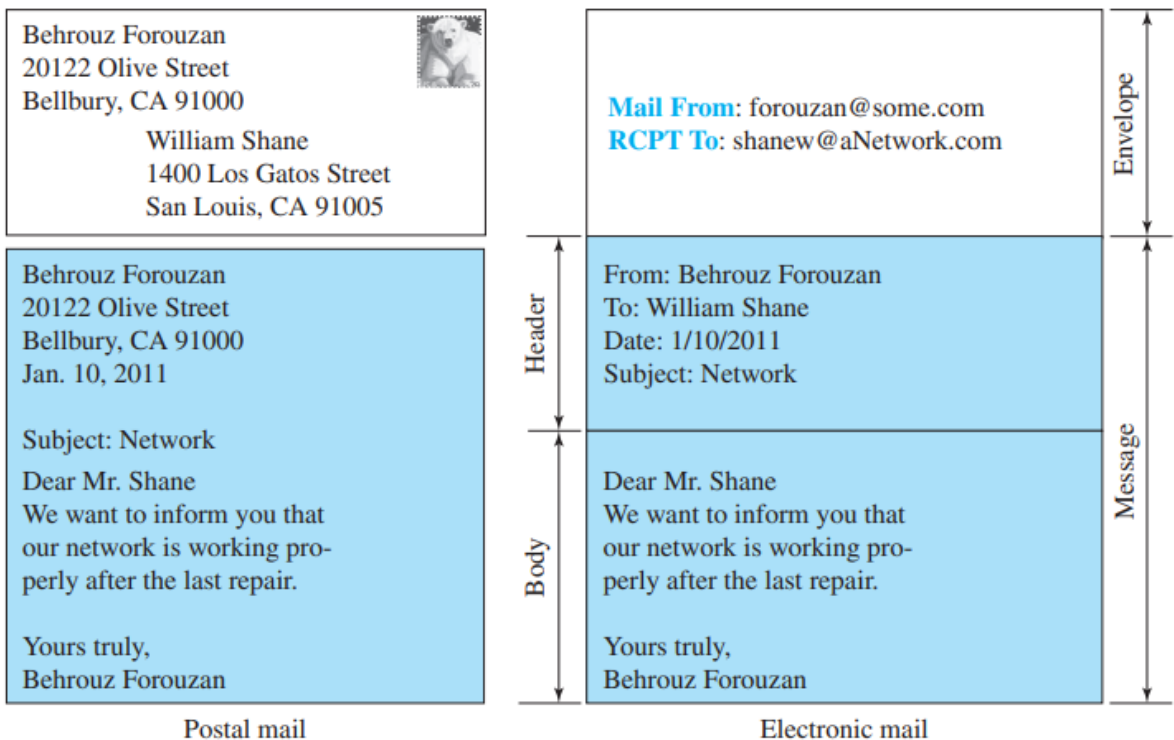
There are two types of user agents: command-driven and GUI-based. Command driven user agents belong to the early days of electronic mail.

Services of User Agent



Sending Mail

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



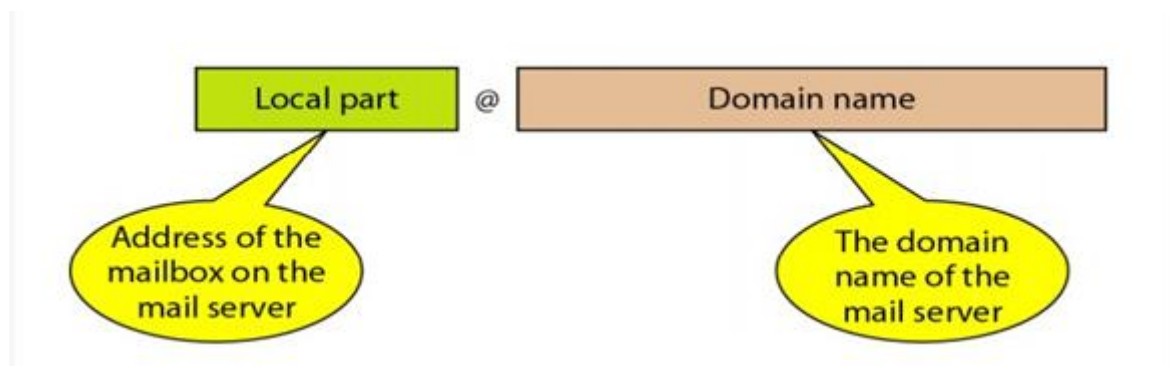
Receiving Mail

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.

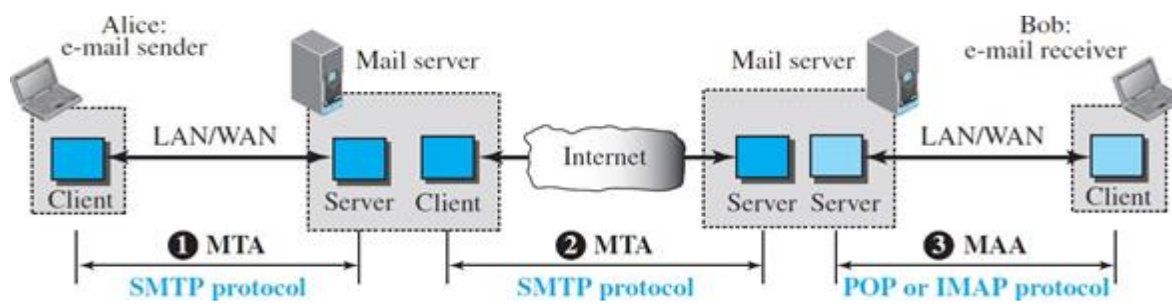
The summary usually includes the sender mail address, the subject, and the time the mail was sent or received. The user can select any of the messages and display its contents on the screen.

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.



E-mail is one of those applications that need three uses of client-server paradigms to accomplish its task. We refer to the first and the second as Message Transfer Agents (MTAs), the third as Message Access Agent (MAA).



Message Transfer Agent: SMTP

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 SMTP simply defines how commands and responses must be sent back and forth.

Commands and Responses

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

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

Keyword: argument(s)

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.

Mail Transfer Phases

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

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

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

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

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

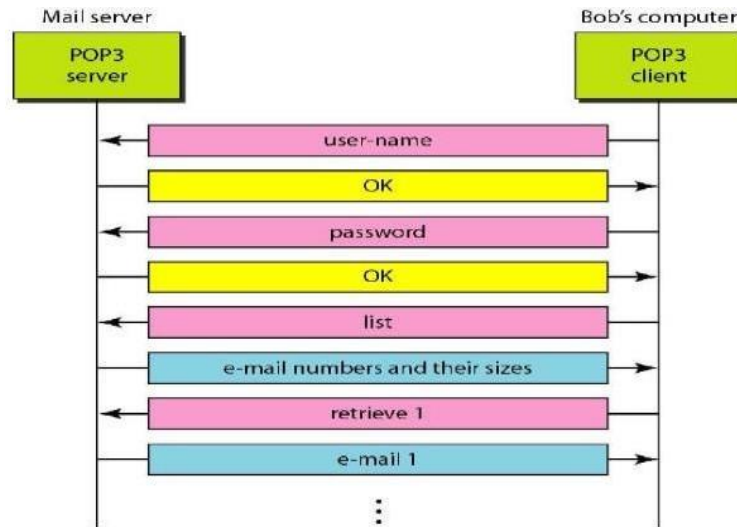
MESSAGE ACCESS AGENT- POP3&IMAP

- 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).

Post Office Protocol, version 3 (POP3)]

Post Office Protocol 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.

POP3 has two modes: the delete mode and the keep mode. In the delete mode, the mail is deleted from the mailbox after each retrieval. In the keep mode, the mail remains in the mailbox after retrieval.

IMAP 4

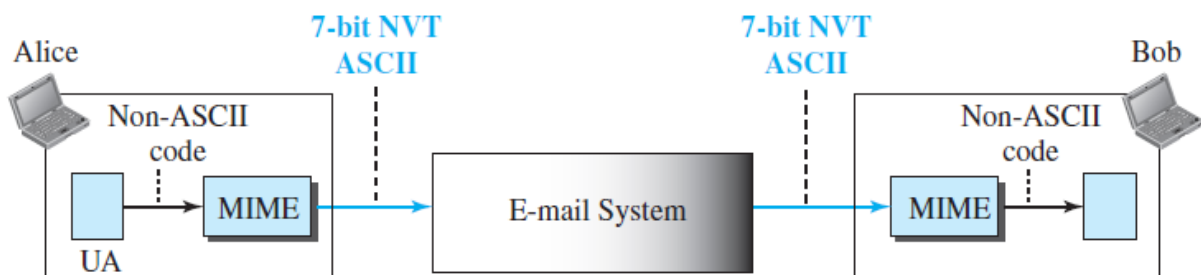
Another mail access protocol is **Internet Mail Access Protocol, version 4 (IMAP4)**. IMAP4 is similar to POP3, but it has more features; IMAP4 is more powerful and more complex.

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.

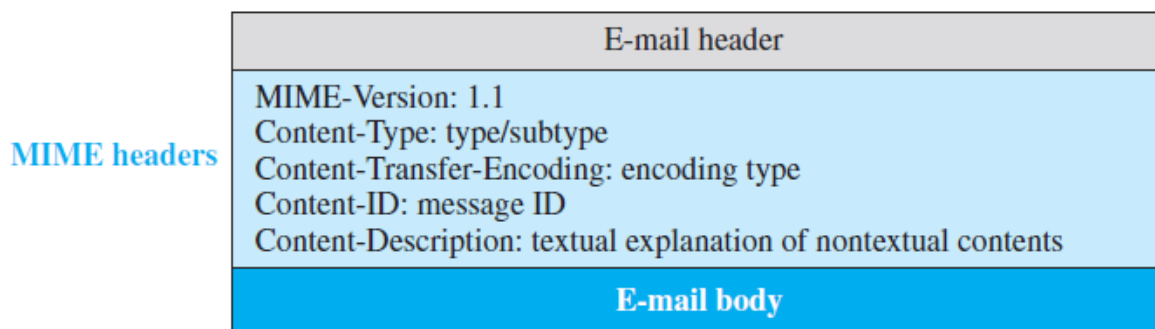
MIME: (Multipurpose Internet Mail Extensions)

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



MIME Headers

MIME defines five headers 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.

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

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.