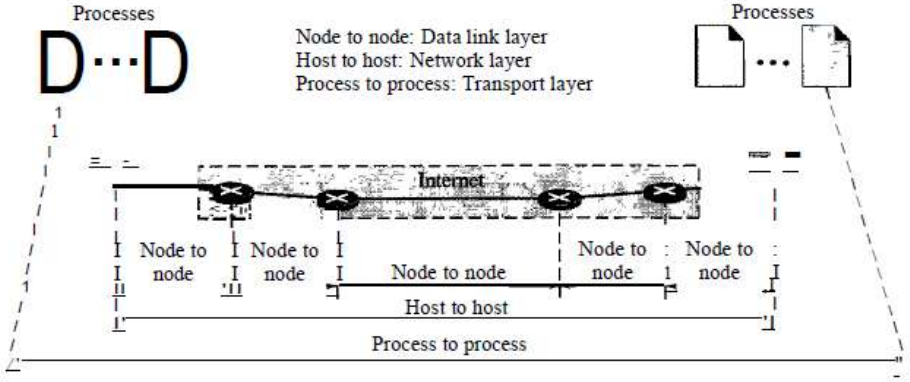
UNIT-4 TRANSPORT LAYER

Transport Layer: The data link layer is responsible for delivery of frames between two neighbouring nodes over a link. This is called node-to-node delivery. The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery. Communication on the Internet is not defined as the exchange of data between two nodes or between two hosts. Real communication takes place between two processes.

Types of data deliveries



Addressing

Whenever we need to deliver something to one specific destination among many, we need an address. At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply. In the Internet model, the port numbers are 16-bit integers between 0 and 65,535. The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral port number. The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number. Of course, one solution would be to send a special packet and request the port number of a specific server, but this requires more overhead. The Internet has decided to use universal port numbers for servers; these are called well-known port numbers. There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers. Every client process knows the well-known port number of the corresponding server process.

lANA Ranges

The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private).

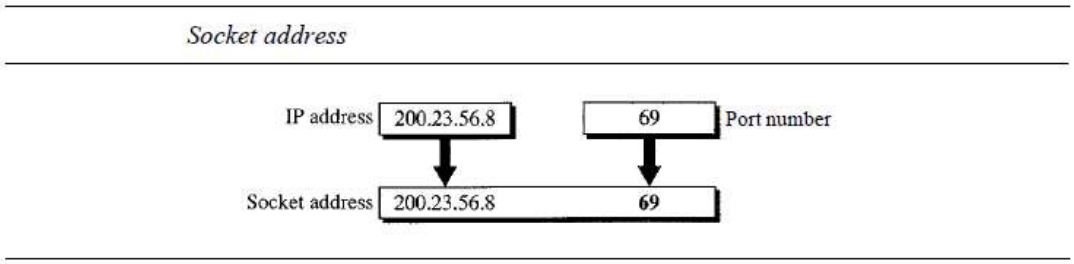
o Well-known ports. The ports ranging from 0 to 1023 are assigned and controlled by lANA. These are the well-known ports.

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

o Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.

Socket Addresses

Process-to-process delivery needs two identifiers, 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.



The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to process communication instead of host-to-host communication. Also, it performs very limited error checking. 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. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

Well-Known Ports for UDP

Well-known ports used with UDP (continued)

Port	*Protocol*	*Description*
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
III	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

UDP packets, called user datagrams, have a fixed-size header of 8 bytes.

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

Destination port number: This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a

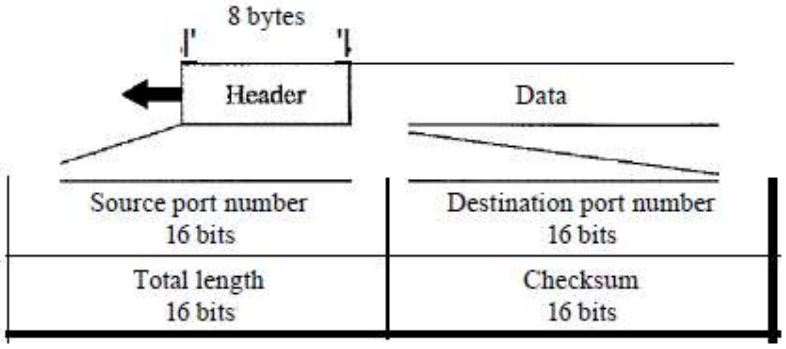
request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.

o Length: This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes. The length field in a UDP user datagram is actually not necessary. A user datagram is encapsulated in an IP datagram. There is a field in the IP datagram that defines the total length. There is another field in the IP datagram that defines the length of the header. So if we subtract the value of the second field from the first, we can deduce the length of a UDP datagram that is encapsulated in an IP datagram.

UDP Length=IP length-IP header length

Checksum: This 16 bit field is used to detect errors over the entire user datagram (header plus data).

User datagram format



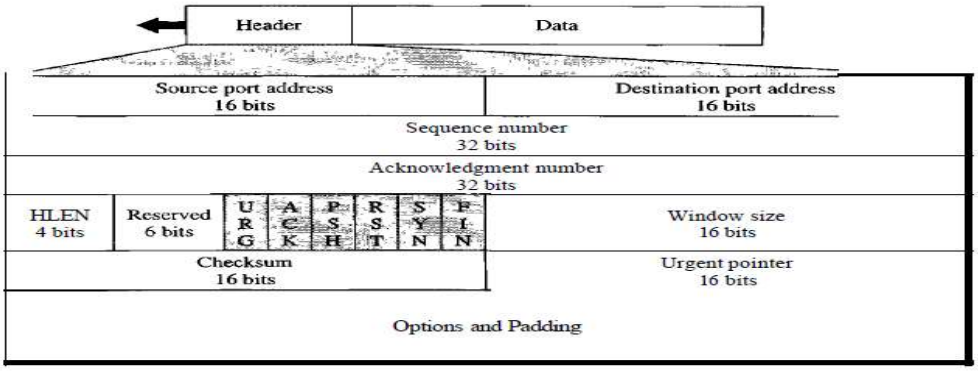
Transmission Control Protocol: The second transport layer protocol we discuss in this chapter is called Transmission Control Protocol (TCP). TCP, like UDP, is a process-to-process (program-to-program) protocol. TCP, therefore, like UDP, uses port numbers. Unlike UDP, TCP is a connection oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level. In brief, TCP is called a connection-oriented, reliable transport protocol. It adds connection-oriented and reliability features to the services of IP. Well known ports used by TCP are:

Well-known ports used by TCP

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FIP, Data	File Transfer Protocol (data connection)
21	FIP, Control	File Transfer Protocol (control connection)
23	TELNET	Tenninal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

The segment consists of a 20- to 60-byte header, 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.

TCP segment format



o 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. This serves the same purpose as the source port address in the UDP header.

o **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. This serves the same purpose as the destination port address in the UDP header.

o Sequence number: This 32-bit field defines the number assigned to the first byte of data contained in this segment. As we said before, TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence comprises the first byte in the segment. During connection establishment, each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction.

o 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.

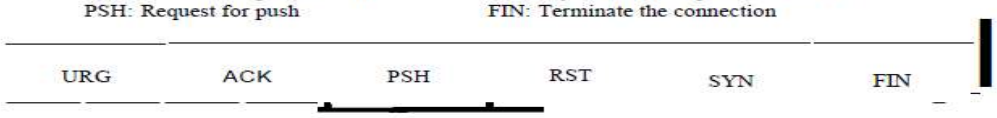
Header length: This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (5 x 4 =20) and 15 (15 x 4 =60).

Reserved: This is a 6-bit field reserved for future use.

Control: This field defines 6 different control bits or flags. One or more of these bits can be set at a time.

Control field

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



These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

Description of flags in the control field

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

Window size: This field defines the size of the window, in bytes, that the other party must maintain. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (rwnd) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

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 inclusion of the checksum in the UDP datagram is optional, whereas the inclusion of the checksum for TCP is mandatory. The same pseudo header, serving the same purpose, is added to the segment. For the TCP pseudo header, the value for the protocol field is 6.

Urgent pointer: This l6-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines the number that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment. This will be discussed later in this chapter.

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

Congestion control Algorithms:

Congestion control algorithms are designed for two kinds of systems open loop and closed loops systems.

Open loop systems decide when to accept new traffic and when to discard packets. Open loop algorithms work with source and destination.

Where as, closed loop systems based on the concept of a feedback loop. Closed loop algorithms are divided into two subcategories, explicit and implicit feed back. This approach has 3 steps

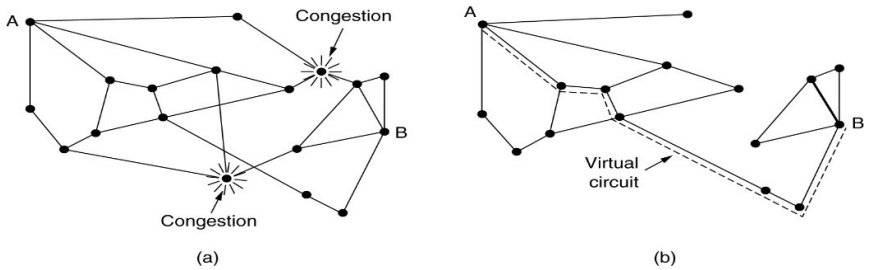
1. Monitor the system to detect when and where congestion occurs.
2. pass this information to places where action can be taken.
3. Adjust the system operation to correct the problem.

In explicit feedback, packets are sent back from the point of congestion to warn the source. In implicit algorithms, the source deduces the existence of congestion by making local observations, such as the time required to acks to come back.

Congestion Prevention Policies :

Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

Congestion control in Virtual circuit subnets : One of the widely used technique to avoid congestion is admission control mechanism. Means once congestion has been signaled, no more virtual circuits are set up until the problem has gone away. An alternative approach is to allow new virtual circuits, but carefully route all new virtual circuits around problem areas.



Consider the above subnet, suppose that a host attached to router A, wants to set up a connection to host attached to router B. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the subnet like in the 2nd diagram, by omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids congested routes.

Congestion control in Datagram Subnets : Each router can easily monitor the utilization of its output lines and other resources. It can associate a value with each line, a real variable 'u' whose value is in between 0.0 and 1.0 reflects the recent utilization of the line.

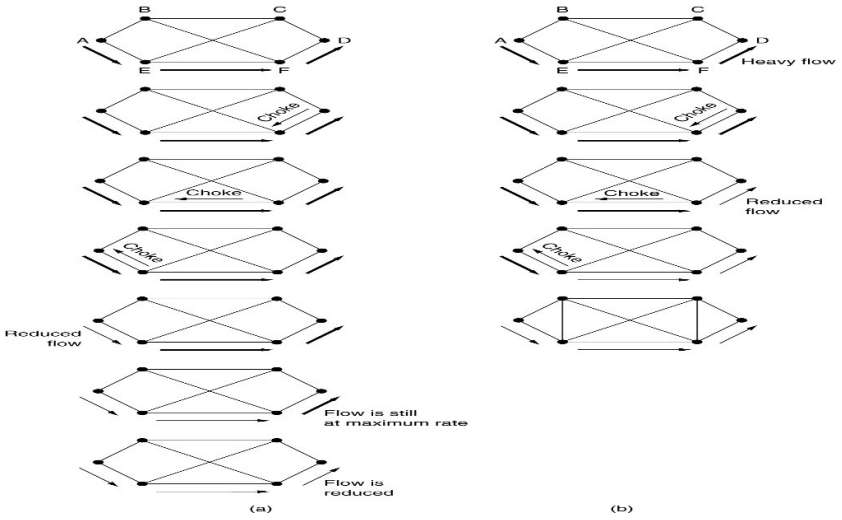
Whenever u moves above the threshold, the output line enters into a warning state. The newly arriving packet is checked to see if its output line is in warning state or not. If it is in warning state, an action is taken. The action can be taken based on the several alternatives.

1.Warning Bit : DECNET signals the warning state by setting a special bit in the packet's header. When the packet arrives at tits destination, the transport entity copied the bit into the next ack sent back to the source.

As long as the router was in the warning state, it continues to set the warning bit. This means that source continually gets acks with a warning bit set to 1. so source decrease its transmission rate.

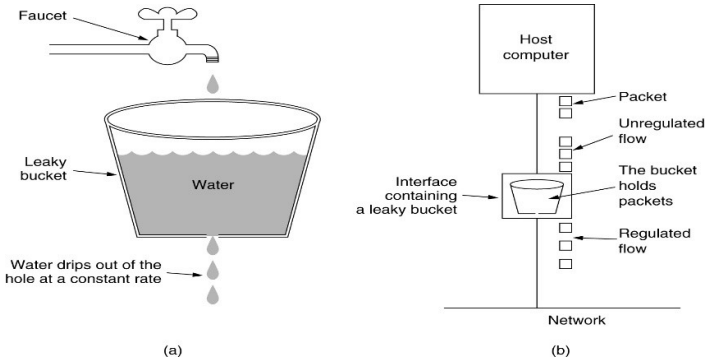
2. Choke Packets : The previous congestion control algorithm is fairly clever. This algorithm uses a choke packet to warn the source station. The original packet is added to this choke packet and is forwarded to the destination. When the source host gets the choke packet, it is required to reduce the traffic sent to the specific destination by X percent. Hosts can reduce traffic by adjusting their policy parameters, for example, window size or leaky bucket output rate.

3. Hop By Hop Choke Packets : when we are using choke packets to control congestion, on long distance lines, it may take several seconds to reach to the original source. Within this much time source can generate many packets, all of them will be discarded due to traffic failures. That is why an alternative approach to have choke packet that take effect at every hop it passes through.



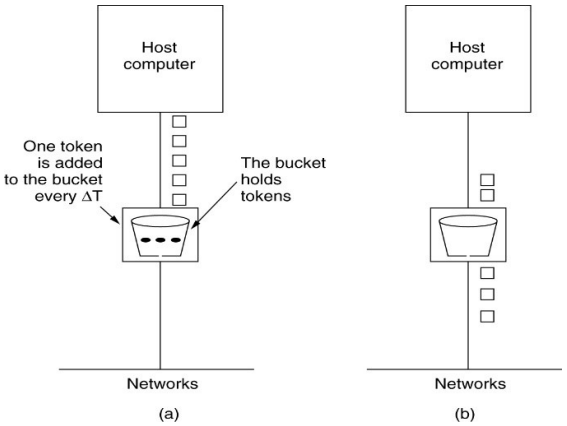
Here in the second diagram, as soon as the choke packet reaches F, F is required reduce the flow from F to D. Doing this, F can assign more buffers to flow. In the next step choke packet reaches to E, which tells E to reduce the flow to F. Finally choke packet reaches to A and flow will slow down. This helps us to get relief fastly from congested routers and traffic.

4. Leaky Bucket Algorithm : Imagine a bucket with a small whole in the bottom. It will not bother about the incoming data transmission rate, the outflow is at a constant rate, when there is any water in the bucket, and zero when the bucket is empty. Also once the bucket is full, any additional water entering it spills over the sides and is lost. The same idea can be applied to packets. Conceptually each host is connected to the network by an interface containing a leaky bucket. If a packet arrives at the queue, when it is full the packet is discarded. In other wards, if one or more processes within the host try to send a packet is discarded. It was first proposed by Turner. In fact it is a single server queuing system with constant service time.



The host is allowed to put one packet per clock tick on to the network. Again this can be enforced by the interface card or by the operating system. This mechanism turns an uneven flow of packets from the user processes inside the host into an even flow of packets on to the network. Implementing the original leaky bucket algorithm is easy. The leaky bucket consists of a finite queue. When a packet arrives, if there is a room on the queue it is appended to the queue; otherwise it is discarded.

5. Token Bucket Algorithm : The leaky bucket algorithm enforces a rigid output pattern at the averaged rate, without considering how burst the traffic is. For many applications, it is better to allow the output to speed up somewhat when large bursts arrive. So a more flexible algorithm is needed, preferably one that never loses data. One such algorithm is token bucket algorithm.



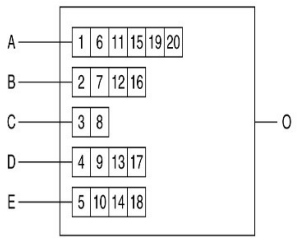
In this algorithm, the leaky bucket holds tokens, generated by a clock at the rate of the token every second. Here in the above diagram, we see a bucket which holds 3 tokens, with 5 packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. We see that 3 of 5 packets have passed through. But the other two are stuck waiting for two more tokens to be generated. Token bucket algorithm throws away token when the bucket is fills up but never discards packet.

6. Flow Specifications: Traffic shaping is most effective when the sender, receiver and the subnet all agree to it. To get agreement it is necessary to specify the traffic pattern in a good way. Such an agreement is called a flow specification. It consists of data structure that describes both the pattern of the injected traffic and the quality of service desired by the application. Flow specification can apply either to the packets sent on a virtual circuit, or to a sequence of data grams sent between source and destination.

Parameter	Unit
Token bucket rate	Bytes/sec
Token bucket size	Bytes
Peak data rate	Bytes/sec
Minimum packet size	Bytes
Maximum packet size	Bytes

7. Weighted fair Queuing : A problem with using choke packets is that the action to be taken by the source hosts is voluntary. Suppose that a router is being swapped by packets from four sources, and it sends choke packets to all four stations. One of them reduces its data transfer rate, but the other 3 just keep the same speed. The result is that the honest host gets an even smaller share of the bandwidth than it has before.

To get around this problem, Nagle proposed a fair queuing algorithm. The importance of the algorithm is routers have multiple queues for each output lien, one for each source. When a line becomes idle, the router scans the queues round robbin, taking the first packet on the queue.



(a)

Packet	Finishing time
C	8
B	16
D	17
E	18
A	20

(b)

In the above example, we have 2 to 6 bytes of information for each station. At first clock tick, first byte of the packet on line A is sent. Then first byte of the packet from line B and so on. The first station that finish, data transmission is C.

8. Load shedding : When none of the above methods help in avoiding congestion, then we use load shedding. It is a fancy way of saying that when routers are being inundated by packets that they can not handle, they just throw them away. This term comes from the word electrical power generation, where it refers to the practice of utilities intentionally blocking out certain areas to save the entire grid from collapsing on hot summer days when the demand for electricity greatly exceeds the supply.

A router drowning in packets can just pick packets at random to drop, but usually it can do better than that. The packet discarding policy will depend on the application. Suppose for file transfer, an old packet is more important than the new one. Always packet retransmission is necessary in these applications.

In contrast for multimedia, a new packet is more important than the old one. Here always late data is more worst than the bad data. Sometimes retransmission can cause serious problems.

9. Random Early detection : According to this, it discards packets before all the buffer space is really exhausted. Routers drop packets before the situation has become hopeless. To determine when to discard routers maintain a running average of their queue lengths. When the average queue length of some line exceeds a threshold, the line is said to be congested and the action is taken.

10. Jitter Control : For applications such as audio and video streaming, it is not a matter that packets take 20 msec or 30msec to deliver. The variation in the packet arrival time is called jitter. The jitter can be bounded by computing the expected transit time for each hop along the path. When a packet arrives at a router, the router checks to see how much the packet is behind or ahead of its schedule. This information is stored and updated at each hop. If the packet is ahead of the schedule, it is held just long enough to get back on schedule. If it is behind schedule, the router tries to get it out quickly.

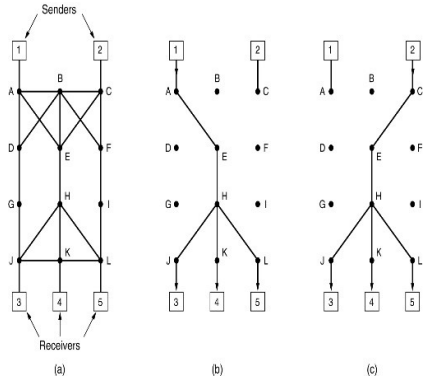
Quality of Service :

Requirements : A stream of packets from a source to a destination is called a flow. In a connection oriented network, all the packets belonging to a flow follow the same route. In a connectionless network, they may follow different routes.

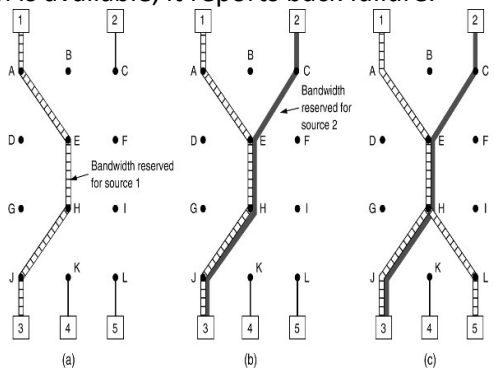
Application	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Remote login	High	Medium	Medium	Low
Audio on demand	Low	Low	High	Medium
Video on demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High	High

Congestion Control for multicasting :

Resource Reservation Protocol (RSVP) : It allows multiple senders to transmit to multiple groups or receivers. It permits individual receivers to switch channels freely, and optimizes bandwidth, and also eliminates congestion. Simple saying it uses multicast routing using spanning trees. Each group is assigned a group address. To send a packet to group, sender puts the group address in the packet. The algorithm then builds the spanning tree covering all group members.



Hosts 1 and 2 are multicast senders, and hosts 3,4, and 5 are multicast receivers. Here the senders and receivers are disjoint, but in general the two sets may overlap. The multicast trees for host1 and 2 are shown as separately. To get better reception and eliminate congestion, any of the receivers in a group can send a reservation message up the tree to the sender. The message is propagated using reverse path forwarding algorithm. At each hop, the router notes the reservation information and reserves the necessary bandwidth. If sufficient bandwidth is available, it reports back failure.

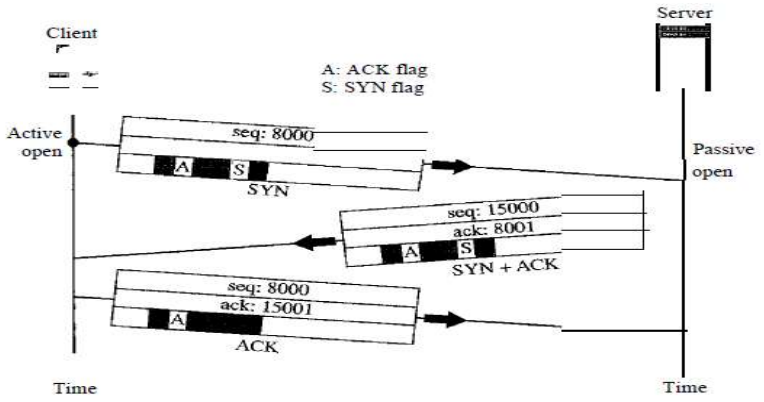


An example of such a reservation is shown above. Here host 3 has requested a channel from channel 1. Once it has been established, packets can flow from 1 to 3 without congestion. Now consider, if host 3 next reserves the other channel 2 also, then the user can watch two programs at once. A second path is reserved like a second diagram. Two separate channels are needed host 3 to router E, because two independent streams are being transmitted. Same process happens for host 5 also.

TCP Connection Establishment TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination. All the segments belonging to a message are then sent over this virtual path. Using a single virtual pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames. 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. This implies that each party must initialize communication and get approval from the other party before any data are transferred.

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 process starts with the server. The server program tells its TCP that it is ready to accept a connection. This is called a request for a passive open. Although the server TCP is ready to accept any connection from any machine in the world, it cannot make the connection itself. The client program issues a request for an active open. A client that wishes to connect to an open server tells its TCP that it needs to be connected to that particular server.

Connection establishment using three-way handshaking



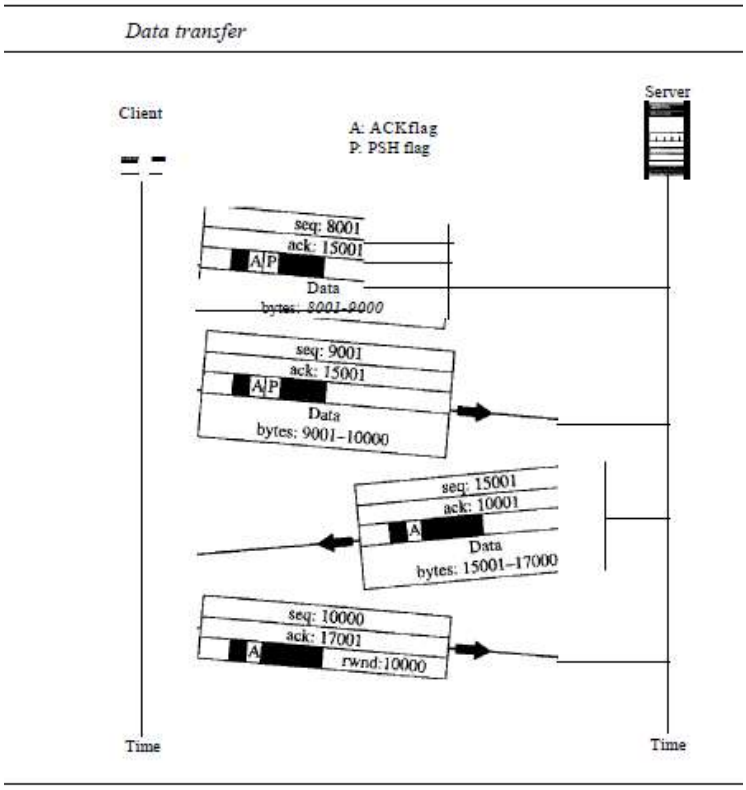
1. The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. It consumes one sequence number. When the data transfer starts, the sequence number is incremented by 1. SYN does not carry any data but it consumes a sequence number.
2. The server sends the second segment, a SYN +ACK segment, with 2 flag bits set: SYN and ACK. This segment has a dual purpose. It is a SYN segment for communication in

the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number.

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 sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers.

Simultaneous Open: A rare situation, called a simultaneous open, may occur when both processes issue an active open. In this case, both TCPs transmit a SYN + ACK segment to each other, and one single connection is established between them.

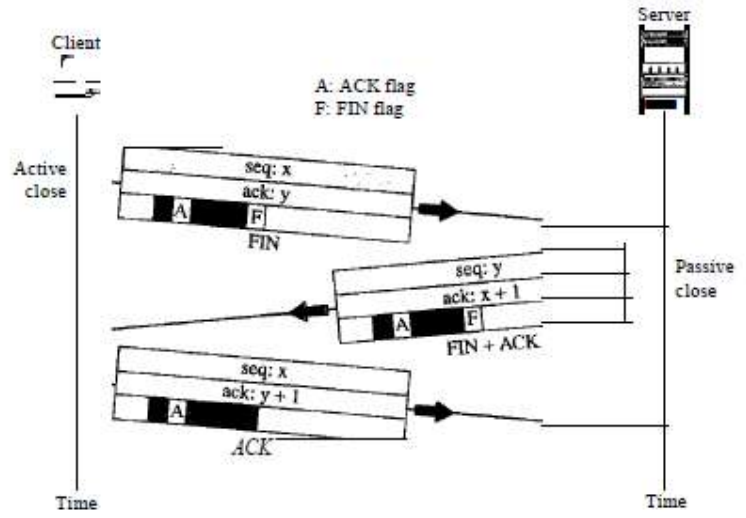
Data Transfer: After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments. We will study the rules of acknowledgment later in the chapter; for the moment, it is enough to know that data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is piggybacked with the data. Figure 23.19 shows an example. In this example, after connection is established (not shown in the figure), the client sends 2000 bytes of data in two segments. The server then sends 2000 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 are no more data to be sent. Note the values of the sequence and acknowledgment numbers. The data segments sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received. We discuss the use of this flag in greater detail later. The segment from the server, on the other hand, does not set the push flag. Most TCP implementations have the option to set or not set this flag.



TCP Connection Termination: Any 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.

1. In a normal 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 Figure. If it is only a control segment, it consumes only one sequence number.

Connection termination using three-way handshaking



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.
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 1 plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

Half-Close: In TCP, one end can stop sending data while still receiving data. This is called a half-close. Although either end can issue a half-close, it is normally initiated by the client. 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 the data, can close the connection in the outbound direction. However, the inbound direction must remain open to receive the sorted data. The server, after receiving the data, still needs time for sorting; its

outbound direction must remain open. The client half-closes the connection by sending a FIN segment. The server accepts the half-close by sending the ACK segment. The data transfer from the client to the server stops. The server, however, can still send data. When the server has sent all the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client. After half-closing of 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. Note the sequence numbers we have used. The second segment (ACK) consumes no sequence number. Although the client has received sequence number y - 1 and is expecting y, the server sequence number is still y - 1. When the connection finally closes, the sequence number of the last ACK segment is still x, because no sequence numbers are consumed during data transfer in that direction.

Half-close