Unit VI

6 Transport Layer - Services and Protocols

6.1 : Transport Layer Duties and Functionalities

Q.1 Explain various transport layer services.

☞ [SPPU : May-17, End Sem, Marks 4]

Or Write a short note on quality of service parameter in transport layer.

☞ [SPPU : Dec.-19, Marks 4]

Ans. : • The transport protocol should provide to higher-level protocols. The transport entity that provides services to transport service users, which might be an application process. The hardware and software within the transport layer that does the work is called the transport entity. It can be in the operating system kernel, in a separate user process or on the network interface card. The relationship of the network, transport and application layers is shown in Fig. Q.1.1. (See Fig. Q.1.1 on next page)

• The following categories of service are useful for describing the transport service.

1. Type of service
2. Quality of service
3. Data transfer
4. User interface
5. Connection management
6. Expedited delivery
7. Status reporting
8. Security

1. Type of service :

• It provides two types of services connection-oriented and connectionless or datagram service. A connection-oriented service provides for the establishment, maintenance and termination of a logical connection between transport service users. The connection-oriented service generally implies that the service is reliable. The connection-oriented

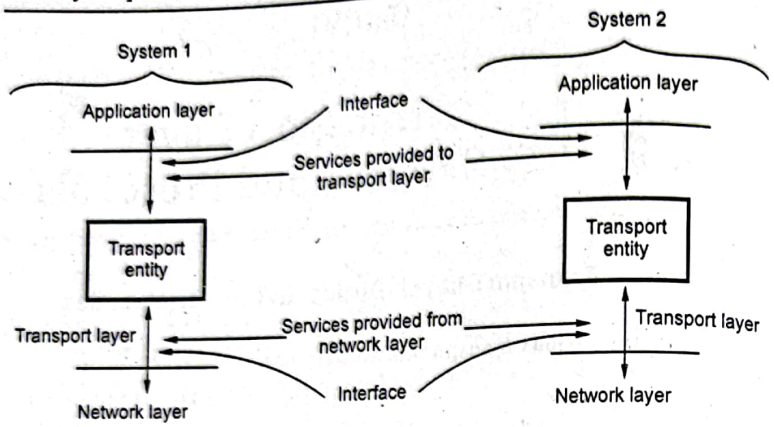


Fig. Q.1.1 Transport entity

service allows for connection-related features such as flow control, error control and sequenced delivery.

2. Quality of service :

- The transport protocol entity should allow the transport service user to specify the quality of transmission service to be provided. Following are the transport layer quality of service parameters
  - I) Error and loss levels
  - II) Desired average and maximum delay
  - III) Throughput
  - IV) Priority level
  - V) Resilience
- The error and loss level measures the number of lost or garbled messages as a fraction of the total sent.
- The desired average and maximum delay measures the time between a message being sent by the transport user on the source machine and its being received by the transport user on the destination machine. The throughput parameter measures the number of bytes of user data transferred per second, measured over some time interval.
- The priority level parameter provides a way for a transport user to indicate that some of its connection are more important than other ones.

The high priority connections get serviced before the low priority ones. Examples of applications that might request particular qualities of service are as follows :

a) A FTP might require high throughput.
b) A transaction protocol may require low delay.
c) An E-mail protocol may require multiple priority levels.

3. Data transfer :

- It transfers data between two transport entities. Both user data and control data must be transferred. Full duplex service must be provided. Half-duplex and simplex modes may also be offered.

4. User interface :

- There is not clear mechanism of the user interface to the transport protocol should be standardized.

5. Connection management :

- If connection-oriented service is provided, the transport entity is responsible for establishing and terminating connections. Symmetric connection procedure should be provided, which allows either TS user to initiate connection establishment.

6. Status reporting :

- It gives the following information.
  - a) Addresses
  - b) Performance characteristics of a connection
  - c) Class of protocol in use
  - d) Current timer values.

7. Security :

- The transport entity may provide a variety of security services. It provides encryption and decryption of data. The transport entity may be capable of routing through secure links or nodes if such a service is available from the transmission facility.

6.2 : Transmission Control Protocol (TCP)

Q.2 State and explain services provided by TCP.

☞ [SPPU : Oct.-16, In Sem, Marks 4]

Ans. : • TCP and UDP use the same network layer (IP), TCP provides totally different services. TCP provides a connection-oriented, reliable, byte stream service. There are exactly two end points communicating with each other on a TCP connection.

- TCP does not support multicasting and broadcasting. The application data is broken into what TCP considers the best sized chunks to send. The unit of information passed by TCP to IP is called a segment.
- When TCP sends a segment it maintains a timer, waiting for the other end to acknowledge reception of segment. If an acknowledgement isn't received in time, the segment is retransmitted.
- When TCP receives data from the other end of the connection, it sends an acknowledgement. TCP maintains a checksum on its header and data.
- TCP segments are transmitted as IP datagrams, and since IP datagrams can arrive out of order, TCP segments can arrive out of order. Since IP datagrams can get duplicated, a receiving TCP must discard duplicate data.
- TCP also provides flow control. Each end of a TCP connection has a finite amount of buffer space. A receiving TCP only allows the other end to send as much data as the receiver has buffers for. This prevents a fast host from taking all the buffers on a slower host.
- A TCP connection is a byte stream, not a message stream. A stream of 8-bit bytes is exchanged across the TCP connection between the two applications. There are no record markers automatically inserted by TCP. This is called a **byte stream service**.
- If the application on one end writes 20 bytes followed by a write of 40 bytes, followed by a write of 80 bytes, the application at the other end of the connection cannot tell what size the individual writes there. The

other end may read 140 bytes ones at a time or 140 bytes in two reads of 70 bytes at a time.

- TCP does not interpret the contents of the bytes at all. TCP has no idea if the data bytes being exchanged are binary data, ASCII character or any other.

Q.3 Explain TCP header format.

☞ [SPPU : Dec.-14, End Sem, Aug.-15, In Sem, Marks 4]

Or Explain TCP header format. ☞ [SPPU : May-19, Marks 4]

Ans. : The TCP data is encapsulated in an IP datagram as shown in the Fig. Q.3.1 (a).

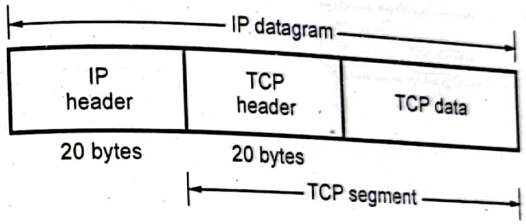


Fig. Q.3.1 (a) Encapsulation of TCP data in an IP datagram

- Fig. Q.3.1 (b) shows the format of the TCP header.
- Description of field in the TCP header as follows :

1. **Source port :** It specifies the application sending the segment. This is different from the IP address, which specifies an internet address.
2. **Destination port :** It identifies the receiving application port numbers below 256 are called well-known ports and have assigned to commonly used applications. For examples, port 23 corresponds to a Telnet function. Port 53 for DNS name server and port 21 assigned for FTP.
3. **Sequence number :** Each byte in the stream that TCP sends is numbered. The sequence number wraps back to 0 after $2^{32} - 1$.
4. **Acknowledgement number :** This field identifies the sequence number of the next data by the that the sender expects to receive if the ACK bit is set. If the ACK bit is not set, this field has no effect.

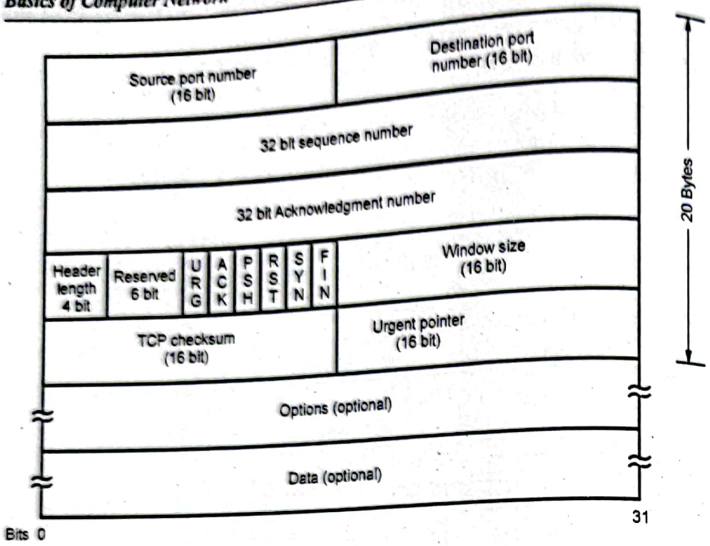


Fig. Q.3.1 (b) TCP header format

5. **Header length :** It specifies the length of the TCP header in 32-bit words. Because of option field, header length is used.
6. **Reserved :** This field is reserved for future use and must be set to 0 (zero).
7. TCP header contains six flag bits. One or more than one can be turned on at the same time. The function of each flag is as follows.
   a. **URG :** The Urgent pointer is valid if it set to 1.
   b. **ACK :** ACK bit is set to 1 to indicate that the acknowledgment number is valid.
   c. **PSH :** The receiver should pass this data to the application as soon as possible.
   d. **RST :** This flag is used to reset the connection. It is also used to reject an invalid segment.
   e. **SYN :** Synchronize sequence number to initiate a connection. The connection request has SYN = 1 and ACK = 0 to indicate that the piggyback acknowledgement field is not in use.
   f. **FIN :** The FIN bit is used to release a connection. It specifies that the sender is finished sending data.

8. **Window size :** It specifies the number of bytes the sender is willing to accept. This field can be used to control the flow of data and congestion.
9. **Checksum :** Used for transport layer error detection.
10. **Urgent pointer :** If the URG flag bit is set, the segment contains urgent data meaning the receiving TCP entity must deliver it to the higher layers immediately.
11. **Options :** Size of this field is variable options field may be used to provide other functions that are not covered by the header.
12. **Data :** Data field size is variable. It contains user data.

- TCP header normal size is 20 bytes, unless options are present. Each TCP segment contains the source and destination port number to identify the sending and receiving application.
- The port number alongwith the source and destination IP addresses in the IP header, uniquely identify each connection. The combination of an IP address and a port number is sometimes called a **socket**.
- Sequence number is a 32-bit unsigned number. Sequence number identifies the byte in the stream of data from sending TCP to the receiving TCP that the first byte of data in this segment represents.
- When a new connection is being established, the SYN flag is turned on. The sequence number of the first byte of data sent by this host will be the ISN plus one because, the SYN flag consumes a sequence number.
- Every byte that is exchanged is numbered, the acknowledgement number contains the next sequence number that the sender of the acknowledgement excepts to receive. Therefore the sequence number plus 1 of the last successfully received byte of data. This field is valid only if the ACK flag is on.
- TCP provides full duplex service. Therefore, each end of a connection must contain a sequence number of the data flowing in each direction.
- TCP can be described as a sliding window protocol without selective or negative acknowledgements.

- The TCP header length tells how many 32-bit words are contained in the TCP header. This information is needed because the options field is of variable length with a 4-bit field, TCP is limited to a 60-byte header. Without options, the normal size is 20 bytes.
- TCP's flow control is handled using a variable size sliding window. This is the number of bytes, starting with the one specified by the acknowledgement number field, that the receiver is willing to accept.
- This is a 16-bit field, limiting the window to 65535 bytes.
- The checksum covers the TCP segment, the TCP header and the TCP data. Checksum field must be calculated and stored by the sender and then verified by the receiver.
- The urgent pointer is valid only if the URG flag is set. This pointer is a positive offset that must be added to the sequence number field of the segment to yield the sequence number of the last byte of urgent data. Option field is the maximum segment size option, called the Maximum Segment Size (MSS). MSS is the largest chunk of data that TCP will send to the other end.

Q.4 Explain the three-way handshake algorithm for TCP connection establishment. List the fields in TCP header that are not part of UDP header. Give the reasons of each missing field.

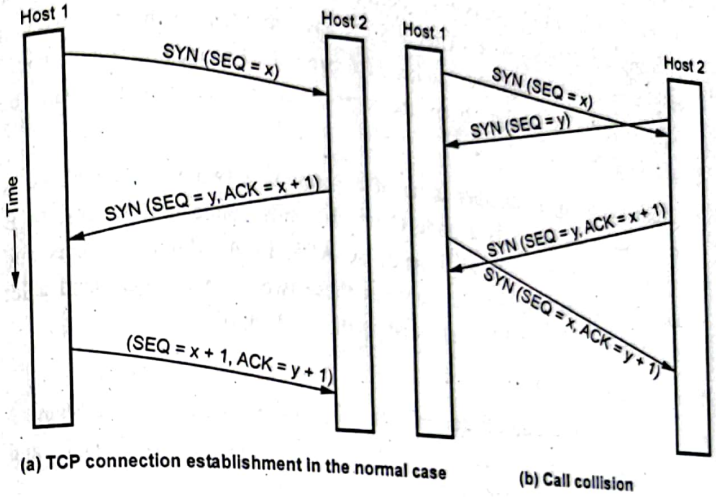
☞ [SPPU : June-22, Marks 9]

OR Explain the three way handshake algorithm for TCP connection establishment. ☞ [SPPU : May-19, April-18, In Sem, Marks 4]

Ans. : TCP Connection Establishment

- Connection establishment in a TCP session is initialized through a three-way handshake. To establish the connection, one side (server) passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives, either specifying a specific source.
- Other side (client) executes a CONNECT primitive specifying the IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data.

- Fig Q.4.1 shows the TCP connection establishment in the normal case and call collision.



(a) TCP connection establishment in the normal case (b) Call collision

Fig. Q.4.1

- A connection is established using a three-way handshake.
- The transmitter sends Connection Request (seq = x) to start a connection with transmitter message id x.
- The receiver replies Connection Accepted (seq = y, ACK = x+1), to acknowledge x and establish for its messages the identity y.
- Finally the transmitter confirms the connection with Connection Accepted (seq = x+1, ACK = y+1) to confirm its own identifier x and accept the receiver's identifier y.
- If the receiver wanted to reject x, it would send Reject(ACK = x).
- If the transmitter wanted to reject y it would send Reject(ACK = y).
- As part of the handshake the transmitter and receiver specify their MSS (Maximum Segment Size) that is the maximum size of a segment they can accept. A typical value for MSS is 1460.
- TCP connections are full duplex. The steps required establishing and release connections can be represented in a finite state machine.

Problem regarding 2-way handshake

- The only real problem with a 2-way handshake is that duplicate packets from a previous connection between the two nodes might still be floating on the network. After a SYN has been sent to the responder, it might receive a duplicate packet of a previous connection and it would regard it as a packet from the current connection which would be undesirable.
- Again spoofing is another issue of concern if a two way handshake is used. Suppose there is a node C which sends connection request to B saying that it is A. Now B sends an ACK to A which it rejects and asks B to close connection. Between these two events C can send a lot of packets which will be delivered to the application.

Three-way handshake

- The three-way handshake is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP.
- The three-way handshake involves the exchange of three messages between the client and the server.
- The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.
- The three-way handshake reduces the possibility of false connections. It is the implementation of a trade-off between memory and messages to provide information for this checking.
- Fig. Q.4.2 shows the three way handshake scenario for establishing a connection.

Normal Operation

- Host 1 select a sequence number, 'x' and sends a CONNECTION REQUEST TPDU containing it to Host 2. Host 2 replies with an ACK TPDU acknowledging 'x' and announcing its own initial sequence number 'y'.
- Host 1 acknowledges Host 2's choice of an initial sequence number in the first data TPDU that it sends.
- This is shown in Fig. Q.4.2 (a).

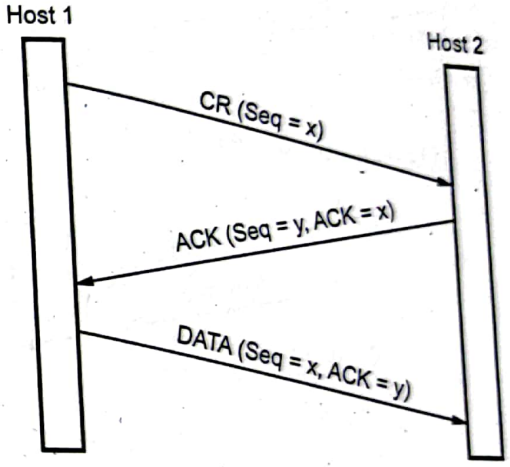


Fig. Q.4.2 (a) Normal operation

Old Duplicate

- The first TPDU is a delayed duplicate CONNECTION REQUEST from an old connection. This TPDU arrives at Host 2 without Host 1's knowledge.
- Host 2 sends ACK TPDU to Host 1 and ask for verification.
- When Host 1 rejects Host 2's attempt to establish a connection, Host 2 realizes that it was tricked by a delayed duplicate and abandons the connection.
- So, the delay duplicate does no damage.
- For this, refer Fig. Q.4.2 (b).

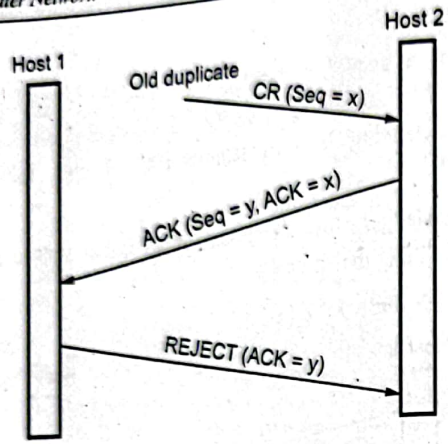


Fig. Q.4.2 (b) Old duplicate CR

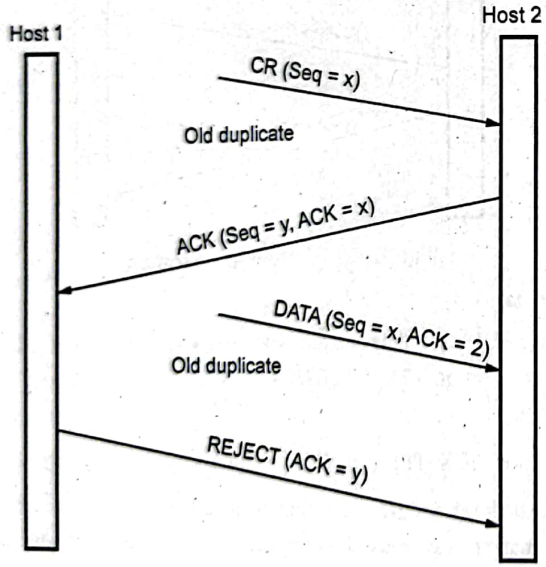


Fig. Q.4.2 (c) Duplicate CR and duplicate ACK

Fig. Q.4.2 Connection establishing using three-way handshake

Duplicate CR and Duplicate ACK

- When both a delayed CONNECTION REQUEST and an ACK are floating around in the subnet.
- Host 2 gets a delayed CONNECTION REQUEST and replies to it.
- When the second delayed, TPDU arrives at Host 2, the fact that 'Z' has been acknowledged rather than 'y' tells Host 2 that this, too, is an old duplicate.

TCP Connection Release

- Any of the two parties involved in exchanging data can close the connection. When connection in one direction is terminated, the other party can continue sending data in the other direction.
- Four steps are required to close the connection in both direction. Fig. Q.4.3 shows four step connection termination.

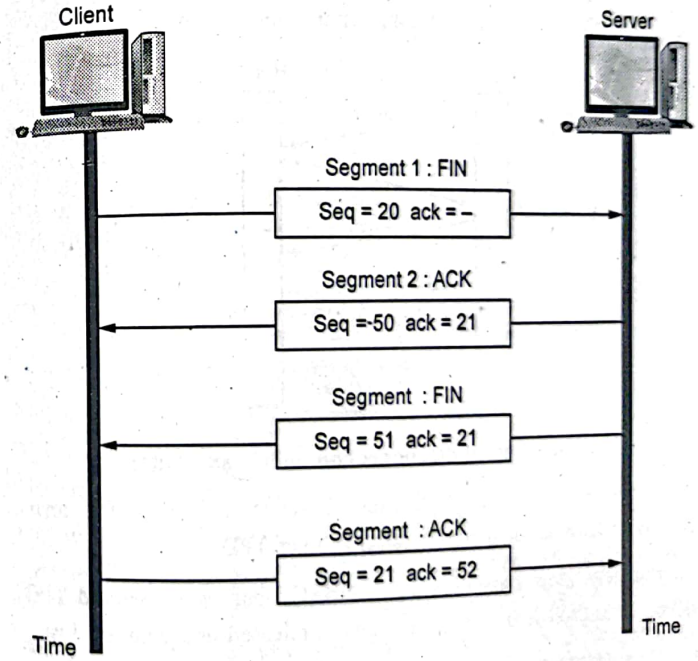


Fig. Q.4.3 Four steps connection termination

- Steps are as follows
  1. The client TCP sends the first segment, a FIN segment.
  2. The server TCP sends the second segment, an ACK segment, to confirm the receipt of the FIN segment from the client.
  3. The server TCP can continue sending data in the server client direction. When it does not have any more data to send, it sends the third segment.
  4. The client TCP sends the fourth segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server.
- Connection release is easier than connection establishing. Connection releases are of two types : Symmetric release and Asymmetric release.
- Asymmetric release is abrupt and may result in data loss. One way to avoid data loss is to use symmetric release, in which each direction is released independently of the other one.
- Fig. Q.4.4 shows the abrupt disconnection with loss of data.

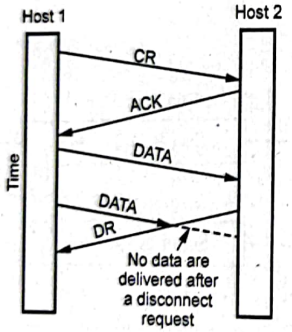


Fig. Q.4.4 Abrupt disconnection with loss of data

- After connection is established, Host 1 sends a TPDU that arrives properly at Host 2. Then Host 1 sends another TPDU.
- Unfortunately, Host 2 issues a DISCONNECT before the second TPDU arrives. The result is that the connection is released and data are lost.
- A more sophisticated release protocol is required to avoid data loss.

says:"I am done. Are you done too ?"

If responds: "I am done too. Goodbye."

This way does not always work.

- One way to avoid data loss is to use symmetric release, in which each direction is released independently of the other one.

Two army problem

- A white army is encamped in a valley.
- On both of the surrounding hillsides are blue armies.
- The white army is larger than either of the blue armies alone, but together they are larger than the white army.
- If either blue army attacks by itself, it will be defeated, but if the two blue armies attack simultaneously, they will be victorious.
- The communication medium between the two blue armies is to send messengers on foot down into the valley, where they might be captured and the message lost.
- Fig. Q.4.5 shows the two army problem.

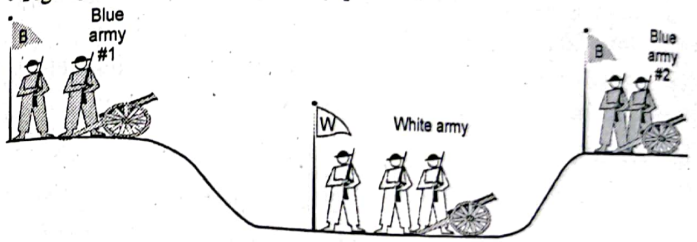


Fig. Q.4.5 Two army problem

- The question is, does a protocol exist that allows the blue armies to win ?
- The answer is there is **NO** such protocol exists.
- Just substitute "disconnect" for "attack". If neither side is prepared to disconnect until it is convinced that the other side is prepared to disconnect too, the disconnection will never happen.
- In practice, one is usually prepared to take more risks when releasing connections than attacking white armies, so the situation is not entirely hopeless.

- Fig. Q.4.6 shows four protocol scenarios for releasing a connection.

a) Normal case of three way handshake

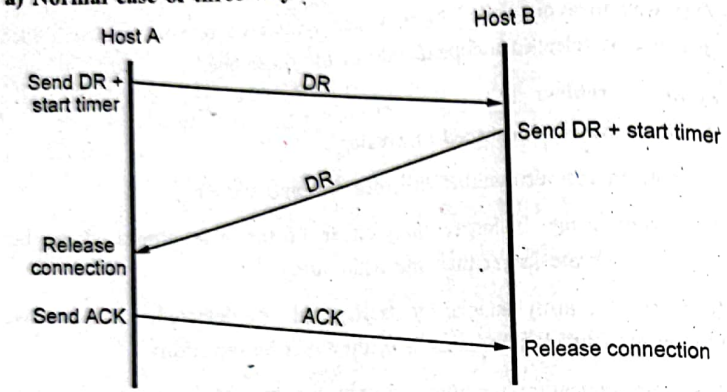


Fig. Q.4.6 (a) Releasing connection

- One of the user sends a DR (DISCONNECTION REQUEST) TPDU to initiate the connection release. When it arrives, the receipent sends back and DR TPDU and start a timer. When this DR arrives, the original sender sends back an ACK TPDU and releases the connection. Finally, when the ACK TPDU arrives, the receiver also releases the connection.

b) Final ACK lost

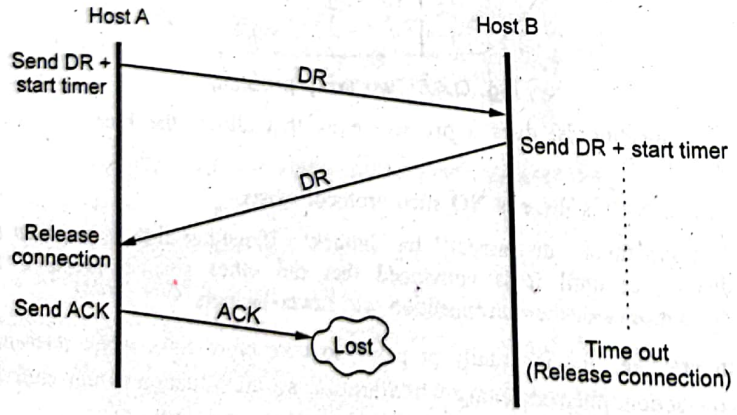


Fig. Q.4.6 (b) Releasing connection

- If the final ACK TPDU is lost, the situation is saved by the timer. When the timer expires, the connection is released anyway.

c) Response lost

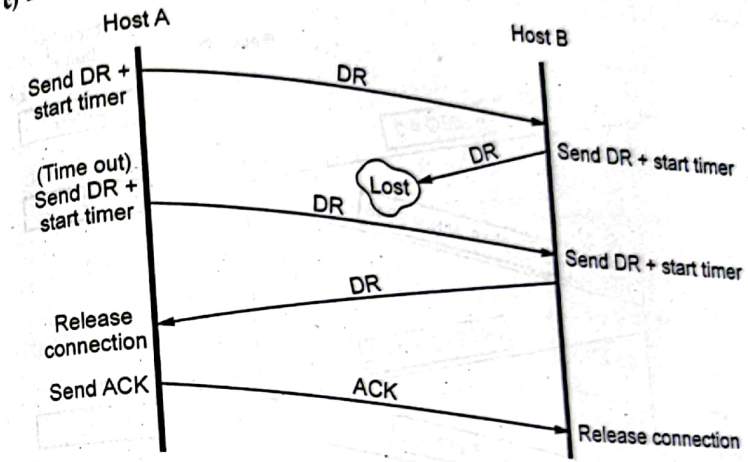


Fig. Q.4.6 (c) Releasing connection

- If the second DR lost then the user initiating the disconnection will not receive the expected response, will time out. The second time no TPDU are lost and all TPDUs are delivered correctly and on time.

d) Response lost and subsequent DR lost

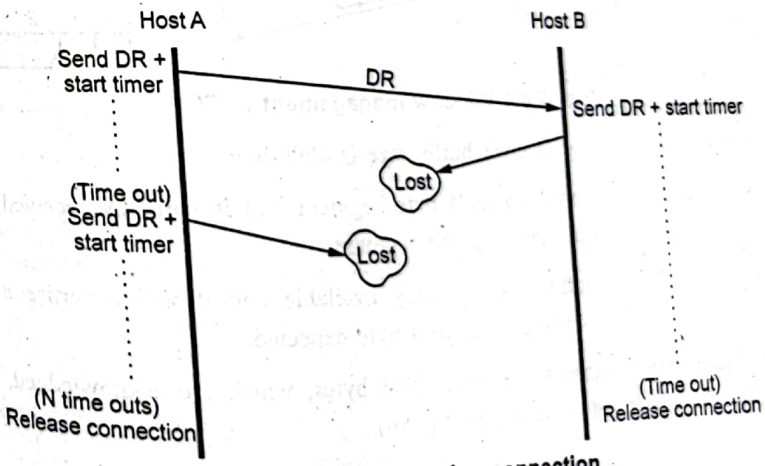


Fig. Q.4.6 (d) Releasing connection

Q.5 How Nagle algorithm helps in TCP transmission policy ? Explain the Clark's solution to overcome the silly window syndrome.

☞ [SPPU : April-18, In Sem, Marks 4, May-19, Marks 6]

Ans. : • Fig. Q.5.1 shows window management in TCP.

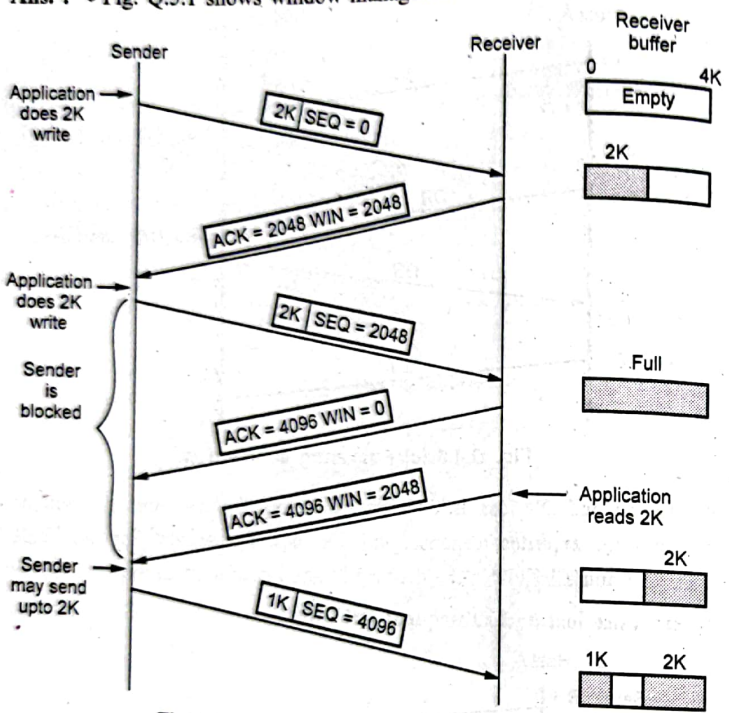


Fig. Q.5.1 Window management in TCP

- Let us assume that receiver buffer size is 4096-byte.
- If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
- 2048 bytes of buffer space is only available and it will advertise a window of 2048 starting at the next byte expected.
- Again sender transmit one more 2048 bytes, which are acknowledged, but the advertised window size 0 (zero).

- Sender must stop until the application process has removed some data from the buffer.
- When the window is 0, the sender may not normally send segments because of two reasons :
  1. Urgent data may be sent.
  2. Sender may send a 1-byte segment to make the receiver reannounce the next byte excepted and window size.

Q.6 Explain different timers used in TCP.

☞ [SPPU : May-18, Dec.-18, Marks 4]

Or Why three timers are required in TCP timer management.

☞ [SPPU : Dec.-16, End Sem, Marks 6]

Ans. : • TCP manages four different timers for each connection.

a) A **retransmission timer** is used when excepting an acknowledgement from the other end.

b) A **persist timer** keeps window size information flowing even if the other end closes its receiver window.

c) A **keep alive timer** detects when the other end on an otherwise idle connection crashes.

d) A 2 **maximum segment lifetime** (2 MSL) timer measures the time a connection has been in the TIME_WAIT state.

- Fundamental to TCP timeout and retransmission is the measurement of the Round-Trip Time (RTT) experienced on a given connection. The TCP must measure the RTT between sending byte with a particular sequence number and receiving an acknowledgement that covers that sequence number. For each connection, TCP maintains a variable RTT, that is the best current estimate of the round-trip time to the destination. When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long.
- If the acknowledgement get back before the timer expires, TCP measures how long the acknowledgement took i.e. M. The original TCP specification had TCP update a smoothed RTT estimator (R) using low-pass filter.

$$R \leftarrow \alpha R + (1-\alpha) M$$

where α is a smoothing factor with a recommended value 0.9. This smoothed RTT is updated every time when a new measurement is made. For given this smoothed estimator, which changes as the RTT changes, the retransmission timeout value (RTO) be set to

$$RTO = R\beta$$

where β = Delay variance factor with a recommended value 2.

- Unnecessary retransmission add to the network load, when the network is already loaded. Calculating the RTO based on both the mean and variance provide much better response to wide fluctuation in the round-trip time, than just calculating the RTO as a constant multiple of the mean. As described by Jacobson the mean deviation is a good approximation to the standard deviation, but easier to compute. This leads to the following equations that are applied to each RTT measurement M.

$$E_{rr} = M - A$$

$$A \leftarrow A + g E_{rr}$$

$$D \leftarrow D + h(|E_{rr}| - D)$$

$$RTO = A + 4D$$

where
- A = Smoothed RTT (estimator of average)
- D = Smoothed mean deviation
- E_{rr} = Difference between the measured value just obtained and the current RTT and estimator.
- g = Gain
- h = Gain of deviation

- Both A and D are used to calculate the next Retransmission Time Out (RTO). The gain (g) is for the average and is set to 0.125 and h is set to 0.25. The larger gain for the deviation makes the RTO go up faster when the RTT changes.

a) Karn's algorithm :

- A problem occurs when a packet is retransmitted. If the packet is retransmitted, a timeout occurs, the RTO is backed off. The packet is

retransmitted with the longer RTO and an acknowledgement is received. The received acknowledgement is whether the first transmission or the second. This is called the retransmission ambiguity problem.

- Karn's algorithm specify that when a timeout and retransmission occur, we cannot update the RTT estimator when the acknowledgement for the retransmitted data finally arrives. Since the data was retransmitted, and the exponential back off has been applied to the RTO, we reuse this backed off RTO for the next transmission. Do not calculate a new RTO until an acknowledgement is received for a segment that was not retransmitted.

Q.7 What is silly window syndrome ? How to overcome it ?

☞ [SPPU : April-18, In Sem, Marks 4]

Ans. :

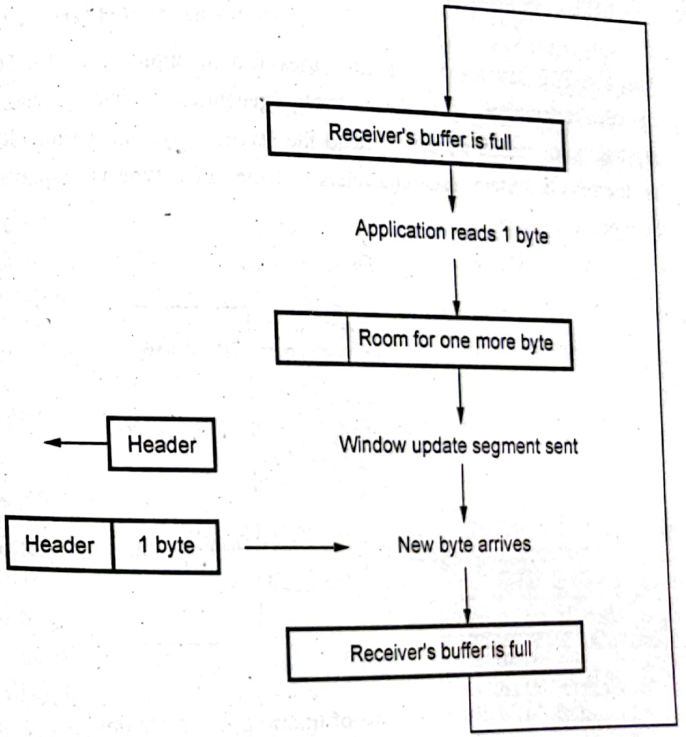


Fig. Q.7.1 Silly window syndrome

- When large block of data is passed from sender but the receiver reads data one byte at a time. Receiving side, the TCP buffer is full and the sender know the condition. The interactive application reads one character from the TCP stream.
- Receiving TCP tells to the sender to send the only 1 byte. Sender send 1 byte. Now buffer is full and receiver send acknowledgement the 1-byte segment and set the window 0. This operation is continuous. Fig. Q.7.1 shows these steps.
- Nagle algorithm and Clark's solution to the silly window syndrome are complementary. Clark solution is to prevent the receiver from sending a window update for 1 byte. Instead it is forced to wait until it has a decent amount of space available.

Q.8 Explain in detail how TCP provides flow control.

☞ **[SPPU : May-17, End Sem, Marks 4]**

Ans. : • TCP interactive data flow uses Rlogin application. In TCP/IP the each interactive keystroke normally generates a data packet. The keystrokes are sent from the client to the server 1 byte at a time. Rlogin has the remote system; each characters that the client type is displayed on the other side (server).

• Fig. Q.8.1 shows the flow of data.

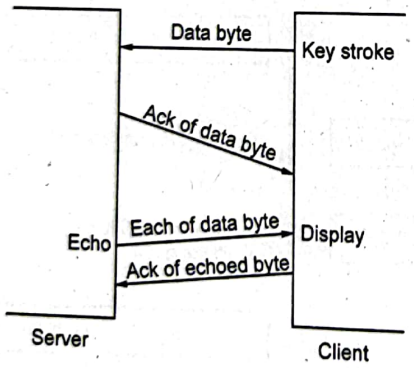


Fig. Q.8.1 Remote echo of interactive keystroke

- The TCP acknowledgements operates as follows.
- Line 1 sends the data byte with the sequence number 0. Line 2 ACKs this by setting the acknowledgement sequence number to 1, the sequence number of the last successfully received byte plus one. This is also called the sequence number of the next excepted byte.
- Line 2 also sends the data byte with a sequence number of 1 from the server to the client. This is ACKed by the client in line 3 by setting the acknowledgement sequence number to 2. Normally TCP does not send an ACK the instant it receives data. Instead, it delays the ACK, hoping to have data going in the same direction as the ACK, so the ACK can be sent alongwith the data. TCP will delay an ACK upto 200 ms to see if there is data to send with the ACK.

NAGLE Algorithm

- One byte at a time normally flows from the client to the server across a Rlogin connection. This generates 41-byte packet 20 bytes for the IP header and 20 bytes for TCP header and 1 byte of data. These small packets called as tinygrams. These tinygrams can add to congestion on WAN.
- Most LANs are not congested because tinygrams are not a problem on LANs. To solve the problem of congestion of WAN, the Nagle algorithm is used.
- The Nagle algorithm say that when TCP connection has outstanding data that has not yet been acknowledged, small segments cannot be sent until the outstanding data is acknowledged. Instead, small amounts of data are collected by TCP and sent in a single segment when the acknowledgement arrives.
- Nagle algorithm is self- clocking. The faster the ACKs come back, the faster the data is sent. But on a slow WAN, where it is desired to reduce the number of tinygrams, fewer segment are sent. Nagles algorithm is widely used by TCP implementations, but there are times when it is better to disable it. The example is the X window system server. Mouse movements must be delivered without delay to provide

real time feedback for interactive users doing certain operations for bulk data flow.

- TCP uses a different form of flow control called a sliding window protocol. This sliding window protocol working is same as Data link layer sliding window protocol.

6.3 : Congestion Control

Q.9 Explain choke packets and hop by hop choke packets.

☞ [SPPU : May-16, End Sem, Marks 6]

Ans. : • Closed loop control try to alleviate congestion after it happens.

End-to-End versus Hop-by-Hop

- In **end-to-end closed loop control**, the feedback information about state of the network is propagated back to the source that can regulate the packet flow rate. Fig. Q.9.1 shows end-to-end closed loop control.

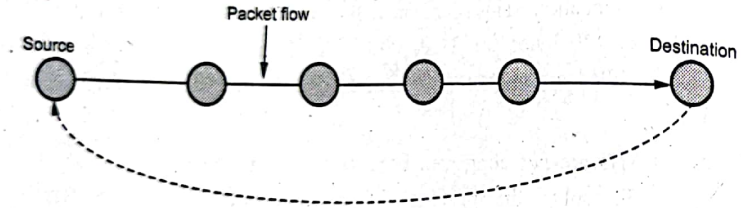


Fig. Q.9.1 End-to-end closed loop control

- The feedback information may be forwarded directly by a node that detects congestion, or it may be forwarded to the destination first which then relays the information to the source.
- With hop-by-hop closed loop control, the state of the network is propagated to the upstream node. Fig. Q.9.1 shows hop-by-hop control.
- When a node detects congestion on its outgoing link, it can tell its upstream neighbor to slow down its transmission rate.

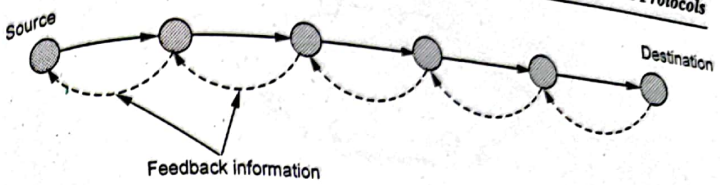
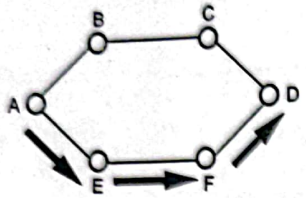


Fig. Q.9.2 Hop-by-hop loop control

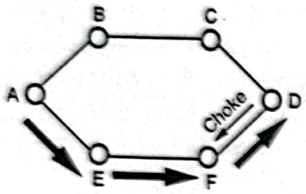
Choke Packets

- Another mechanism for congestion control is by using choke packets. This choke packet will have the effect of stopping or slowing down the rate of transmission from sources and hence limit the total number of packets in the networks. This approach requires additional traffic on the network during a period of congestion. This can be applicable to both virtual circuit and datagram subnets.
- When line utilization increases above some specific value called threshold, the line enters a 'alarming' situation. Each newly arriving packet is checked to see if its output line is in alarming state. If so, the router sends the said choke packet back to the source. This choke packet contains the destination address, so the source will not generate any more packets along the path.
- The traffic is reduced by adjusting parameters window size or leaky bucket output rate. Typically, the first choke packet causes the data rate to 50 % of its previous value the choke packet reduces the traffic to 25 % and so on.
- Congestion control using choke packets can be done by two ways. In first type the choke packet affects only source and in the second type the choke packet affects each hop it passes through. Fig. Q.9.3 shows choke packet that affects only the source.
- Fig. Q.9.3 shows a choke packet that affects each hop it passes through.
  a) A subnet with six nodes A, B, C, D, E and F is shown in Fig. Q.9.3 (a). Here the source node is A and destination node is D.

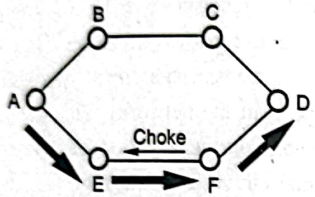


(a)

b) When link utilization increased above its threshold, destination node D starts sending choke packets towards source node A.



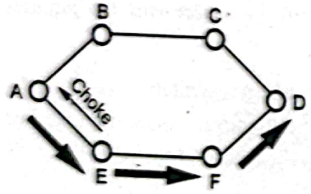
(b)



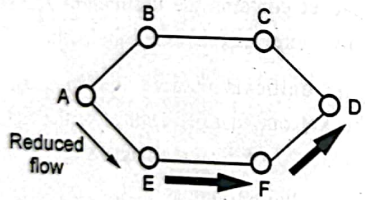
(c)

c) The choke packets travels through the shortest or same path as that of packets.

d) The choke packet reaches to the source node A.



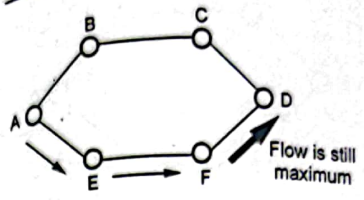
(d)



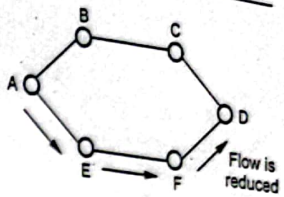
(e)

e) After receiving first chock packet source node A reduces its flow towards destination.

f) The reduced packet flow follows the same reversed path i.e. the path of choke packets through various nodes.



(f)



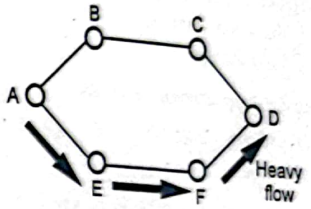
(g)

Fig. Q.9.3 A choke packet that affects only the source

g) The reduced flow reaches to destination node D.

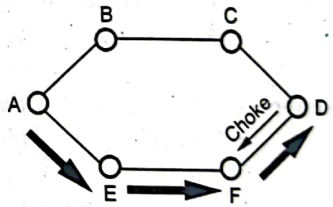
Fig. Q.9.4 shows a choke packet that affects each hop it passes through.

a) For the same subnet having nodes A, B, C, D, E and F source node and destination node D.

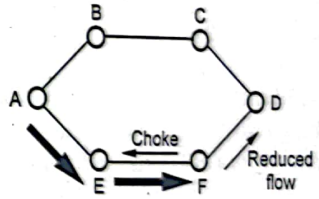


(a)

b) When link utilization increased above its threshold destination node D starts sending choke packets towards source node A.



(b)



(c)

c) The chock packets follows the exactly reversed path of traffic flowing packets. Here the chock packet reaches to node F. Immediately after reaching choke packet at node F, the traffic flow towards node D reduces.

d) As choke packet crosses node E, the traffic flow between nodes E, F, and F, D is reduced.

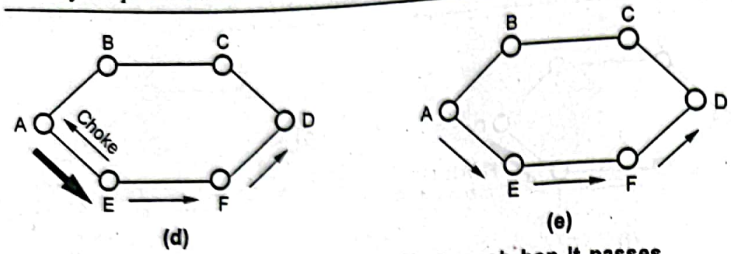


Fig. Q.9.4 A choke packet that affects each hop it passes

e) After reaching choke packet to source node A, the traffic flow between node A and node E and hence up to the destination node D is reduced.

Hop-by-hop choke packets

- Over long distances or at high speeds, the choke packets are not very effective.
- A more efficient way is to send choke packets hop-by-hop.
- A congested node would again generate a choke pocket, but each hop would be needed to reduce its transmission even before the choke packet arrives at the source.

Q.10 What is congestion control ? Explain leaky bucket and token bucket algorithm ? ☞ **[SPPU : April-19, Marks 5]**

Or What is the purpose of leaky bucket and token bucket algorithms ? Describe working of token bucket algorithm with reference to CBR, VBR and bursty traffic.

☞ **[SPPU : Dec.-18, Marks 6, June-22, Marks 9]**

Or Explain leaky bucket and token bucket algorithm.

☞ **[SPPU : May-18, Marks 6]**

Ans. : Congestion control : Congestion control is a process of maintaining the number of packets in a network below a certain level at which performance falls off. Congestion control makes sure that subnet is able to carry the offered traffic. So congestion control is different process than flow control.

Leaky Bucket Algorithm

Traffic shaping

- Traffic shaping is about regulating the average rate of data transmission.
- Traffic shaping smooths out the traffic on the server, rather than on the client side.
- Monitoring a traffic flow is called traffic policing. Agreeing to a traffic shape and policing it afterward are easier with virtual circuit subnets than with datagram subnets.
- Traffic shaping is an open loop method of congestion control.
- Two types of algorithm are used for traffic shaping.

1. Leaky bucket algorithm
2. Token bucket algorithm

Leaky bucket algorithm

- Leaky bucket i.e. a bucket with a small hole in the bottom is used to store the water. The outflow from hole is at constant rate and irrespective of rate of entering water. 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. This is similar to a single server queueing system with constant service time.
- Each host is connected to network with a finite internal queue. The host is allowed to put one packet per second on to the network. If a packet arrives at the queue when it is full, the packet is discarded. This mechanism turn an unregulated traffic of the host regulated traffic on the network. Thus bursty traffic is smoothen and chances of congestion is reduced. Fig. Q.10.1 illustrates this algorithm.
- A leaky bucket regulator allows to control the average rate, largest burst from a source. A leaky bucket regulator has both a packet bucket and a data buffer. Packets that arrive to the regulator that cannot be sent immediately are delayed in the data buffer.

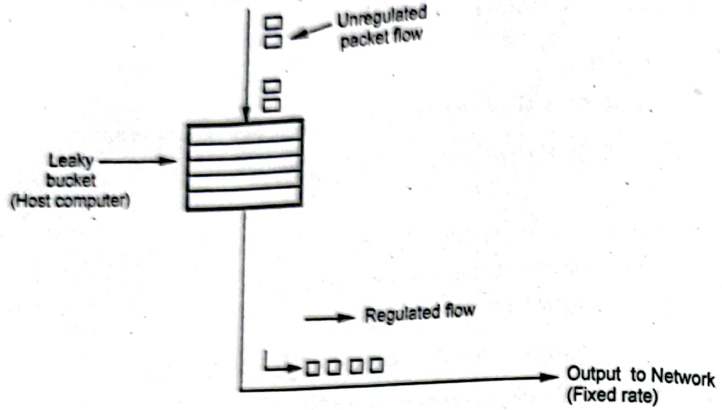


Fig. Q.10.1 Leaky bucket regulator

- The main drawback of leaky bucket algorithm is that its output pattern cannot be modified i.e. if the bursty traffic arrives the output should speed up, so that no packets will be lost.
- Fig. Q.10.2 (See Fig. Q.10.2 on next page) shows the leaky bucket algorithm that can be used to police the traffic flow.
- At the arrival of the first packet, the content of the bucket X is set to zero and the last conforming time is set to the arrival time of the first packet. The depth of the bucket is L+I where L typically depends on the traffic burstiness.

Token Bucket Algorithm

- Token bucket algorithm eliminates drawback of leaky bucket algorithm. In this, the leaky bucket holds tokens. These tokens are generated by a clock at the rate of one token for every ΔT sec. In token bucket bursts of upto n packets can be sent at once, which gives faster response to sudden bursts of input.
- The regulator collects tokens in a bucket, which fills-up at steady drip rate by packets. When a packet arrives at the regulator, the regulator sends the packet if the bucket has enough tokens. Otherwise, the packet waits either until the buckets has enough tokens. If the bucket is already fill of tokens, incoming tokens overflow and are not available to future

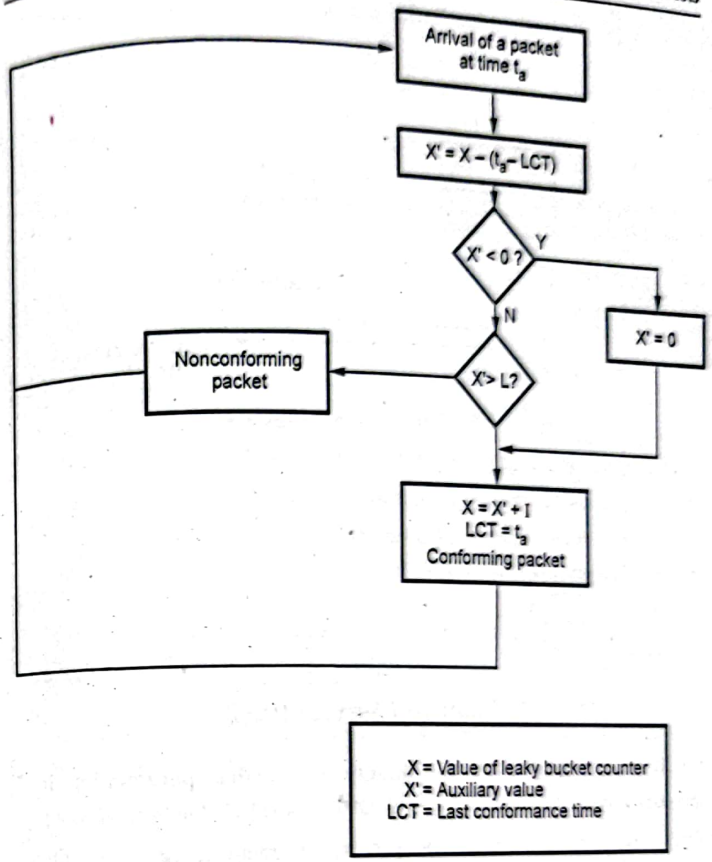


Fig. Q.10.2 Leaky bucket algorithm for policing

packets. Thus, at anytime, the largest burst a source can send into the network is roughly proportional to the size of the leaky bucket.

- The regulator delays a packet if does not have sufficient number of tokens for transmission. A counter keeps track of tokens, the counter is incremented by one every ΔT and decremented by one whenever a packet is sent. When the counter hits zero, no packets may be sent. Smoother traffic can be obtained by putting a leaky bucket after the token bucket.

• Fig. Q.10.3 illustrates token bucket regulator.

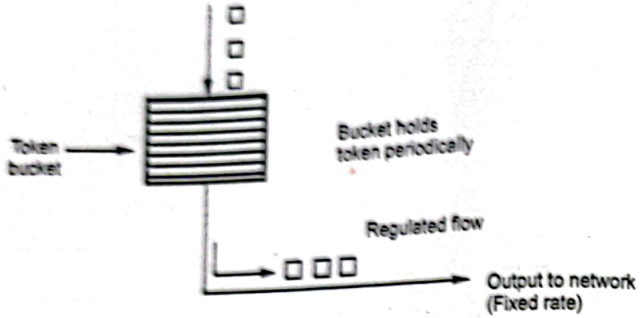


Fig. Q.10.3 Token bucket regulator

Let Token bucket capacity = C bytes

Token arrival rate = ρ bytes/sec

Maximum output rate = M bytes/sec

Then the maximum burst rate S,

$$S = \frac{C}{M-\rho}$$

6.4 : Quality of Service (QoS)

Q.11 Write a short note on quality of service parameters in transport layer. ☞ [SPPU : May-17,18, End Sem, Marks 4]

Ans. : • Quality of service (QoS) is an internetworking issue. QoS mainly defines flow characteristics and flow classes.

A] Flow Characteristics

• Four types of characteristics are attributed to a flow :

1. Bandwidth 2. Jitter 3. Delay 4. Reliability.

1. Bandwidth :

• Bandwidth is a characteristic of network. Bandwidth can be measured in hertz and in bits per seconds.

• **Bandwidth in Hertz :** Bandwidth in hertz refers to the range of frequencies in a composite signal or the range of frequencies that a transmission channel can pass.

• **Bandwidth in bps :** Bandwidth in bps refers to speed of bit transmission in a channel or link.

2. Jitter :

• Jitter is a parameter related to delay. Jitter is introduced since different packets of data encounter different delays. The data packets reaching at receiver at different times causing jitter.

3. Delay :

• Latency is also termed as delay. Latency is time required for a message to completely arrive at the destination from source. It has four components propagation time, transmission time, queuing time and processing delay.

4. Reliability :

• Loss of packet or acknowledgement is due to lack of reliability.

B] Flow Classes

• Based on the flow characteristics, we can classify flows into groups, with each group having similar levels of characteristics.

6.5 : User Datagram Protocol (UDP)

Q.12 List out key features of UDP protocol. Explain how flow control is different than congestion control in TCP ?

☞ [SPPU : June-22, Marks 9]

Or Draw and explain UDP header format.

☞ [SPPU : Oct.-14, In Sem, Marks 6]

Ans. : • Fig. Q.12.1 (a) shows the encapsulation of a UDP datagram as an IP datagram.

• Fig. Q.12.1 (b) shows the format of the UDP header. The port number identify the sending process and the receiving process.

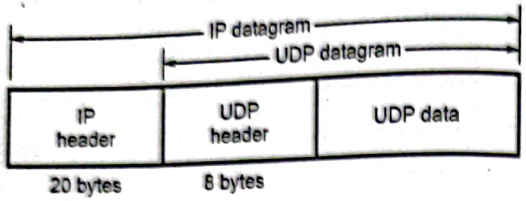


Fig. Q.12.1 (a) UDP encapsulation

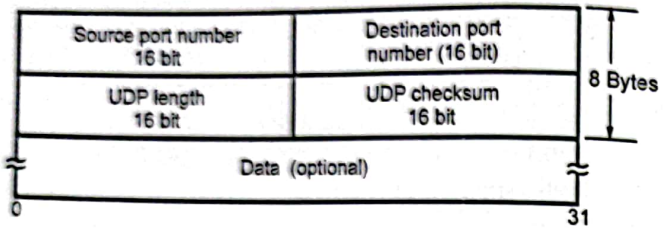


Fig. Q.12.1 (b) UDP header

- The UDP datagram contains a source port number and destination port number. Source port number identifies the port of the sending application process. The destination port number identifies the receiving process on the destination host machine.
- The UDP length field is the length of the UDP header and the UDP data in bytes. The minimum value for this field is 8 bytes.
- UDP checksum covers the UDP header and the UDP data. Both UDP and TCP include a 12 byte pseudo-header with the UDP datagram just for the checksum computation. This pseudo_header includes certain fields from the IP header. The purpose is to let UDP double check that the data has arrived at the correct destination.
- UDP checksum is end-to-end checksum. It is calculated by the sender, and then verified by receiver. It is designed to catch any modification of the UDP header or data anywhere between sender and receiver.
- Goal of the UDP checksum is to detect "errors" in transmitted segment. Function performed by sender and receiver is as follows :

Sender :

1. Treat segment contents as sequence of 16-bit integers
2. Checksum : addition (1's complement sum) of segment contents
3. Sender puts checksum value into UDP checksum field

Receiver :

1. Compute checksum of received segment
2. Check if computed checksum equals checksum field value : NO - error detected and YES - no error detected.

Why is there a UDP ?

1. No connection establishment (which can add delay)
2. Simple : no connection state at sender, receiver
3. Small segment header
4. No congestion control : UDP can blast away as fast as desired
5. Often used for streaming multimedia apps

6.6 : Socket

Q.13 What is socket ? Explain various socket primitives used in client server interaction.

☞ [SPPU : June-22, Marks 9, April-18,19, In Sem, Dec.-18, Marks 4]

Ans. : Socket : • Socket interface is a protocol independent interface to multiple transport layer primitives. In order to write applications which need to communicate with other applications.

- Socket is an abstraction that is provided to an application programmer to send or receive data to another process.
- Data can be sent to or received from another process running on the same machine or a different machine.
- It is like an endpoint of a connection. It exists on either side of connection and identified by IP Address and Port number.
- Sockets works with UNIX I/O services just like files, pipes and FIFO.
- API stands for Application Programming Interface. It is an interface to use the network. Socket API defines interface between application and transport layer.

- The API defines function calls to create, close, read and write to/from a socket.

Advantages of using socket interface

- Syntax of the API functions is independent of the protocol being used. Ex:- TCP/IP and UNIX domain protocols can be used by applications using a common set of functions.
- Gives way to better portability of applications across protocol suites.
- Hides the finer details of the protocols from application programs thereby yielding faster and bug free application development
- Sockets are referenced through socket descriptors which can be passed directly to UNIX system I/O calls. File I/O and socket I/O are exactly similar from the programmer perspective.

Sockets versus file I/O

- Working with sockets is very similar to working with files. The socket( ) and accept( ) functions both return handles (file descriptor) and reads and writes to the sockets requires the use of these handles (file descriptors).
- In Linux, sockets and file descriptors also share the same file descriptor table. That is, if you open a file and it returns a file descriptor with value say 8, and then immediately open a socket, you will be given a file descriptor with value 9 to reference that socket.
- Even though sockets and files share the same file descriptor table, they are still very different. Sockets have addresses associated with them whereas files do not; notice that this distinguishes sockets form pipes, since pipes do not have addresses with which they associate.
- You cannot randomly access a socket like you can a file with lseek( ). Sockets must be in the correct state to perform input or output.

Socket abstraction

- Socket is the basic abstraction for network communication in the socket API. Socket defines an endpoint of communication for a process.

- Operating system maintains information about the socket and its connection. Fig. Q.13.1 shows the socket and process.

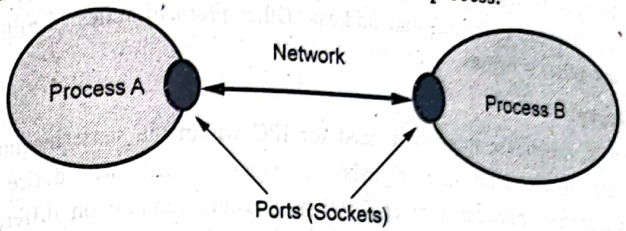


Fig. Q.13.1 Socket and process

Socket creation

int socket (int family, int type, int protocol);

Parameters :

1. family : AF_INET or PF_INET (These are the IP4 family)
2. type : SOCK_STREAM (for TCP) or SOCK_DGRAM (for UDP)
3. protocol : IPPROTO_TCP (for TCP) or IPPROTO_UDP (for UDP) or use 0

- If successful, socket ( ) returns a socket descriptor, which is an integer, and – 1 in the case of a failure.
- An example call :

```
if ((sd = socket(AF_INET, SOCK_DGRAM, 0) < 0)
{
        printf(socket() failed.);
        exit(1);
}
```

- Creating a socket is in some ways similar to opening a file. This function creates a file descriptor and returns it from the function call. You later use this file descriptor for reading, writing and using with other socket functions.

- Remember that the sockets API are generic. There must be a generic way to specify endpoint addresses. TCP/IP requires an IP address and port number for each endpoint address. Other protocol suites (families) may use other schemes.

TCP Socket

- In UNIX, whenever there is a need for IPC within the same machine, we use mechanism like signals or pipes. When we desire a communication between two applications possibly running on different machines, we need **sockets.**
- Sockets are treated as another entry in the UNIX open file table.
- Sockets provide an interface for programming networks at the transport layer.
- Network communication using Sockets is very much similar to performing file I/O. In fact, socket handle is treated like file handle.
- Socket-based communication is programming language independent.
- To the Kernel, a socket is an endpoint of communication. To an application, a socket is a file descriptor that lets the application read/write from/to the network.
- A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request.
- To review, there are five significant steps that a program which uses TCP must take to establish and complete a connection. The server side would follow these steps :
  1. Create a socket.
  2. Listen for incoming connections from clients.
  3. Accept the client connection.
  4. Send and receive information.
  5. Close the socket when finished, terminating the conversation.

- In the case of the client, these steps are followed :
  1. Create a socket.
  2. Specify the address and service port of the server program.
  3. Establish the connection with the server.
  4. Send and receive information.
  5. Close the socket when finished, terminating the conversation.

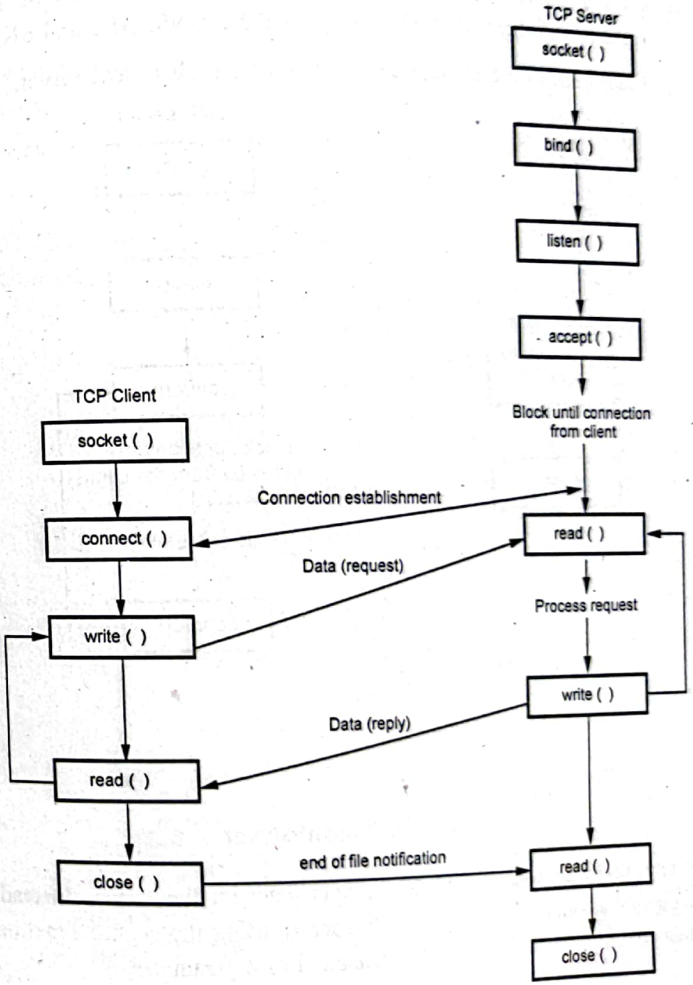


Fig. Q.13.2 Socket function for elementary TCP client server

- Only steps two and three are different, depending on if it's a client or server application.
- Fig. Q.13.2 shows a timeline of the typical scenario that takes place between a TCP client and server.

UDP Socket

- There are some instances when it makes to use UDP instead of TCP. Some popular applications built around UDP are DNS, NFS and SNMP.
- Fig. Q.13.3 shows the interaction between a UDP client and server.

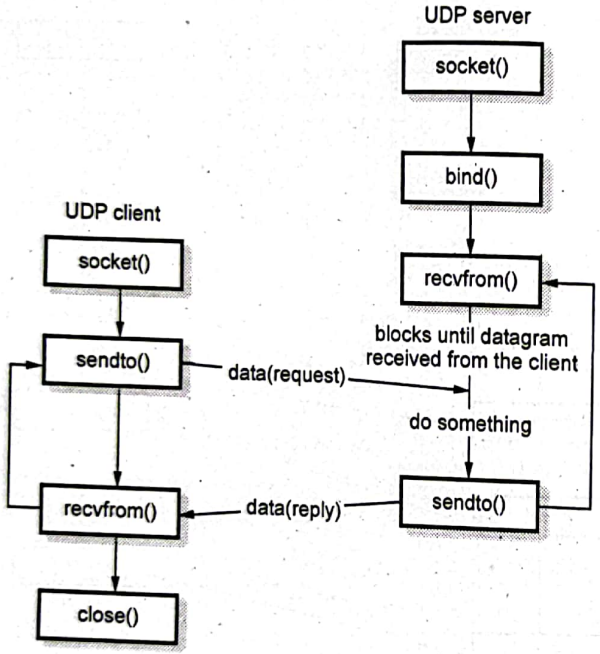


Fig. Q.13.3 UDP client-server

- Initially client does not establish a connection with the server. Instead, the client just sends a datagram to the server using the sendto function which requires the address of the destination as a parameter.

- Similarly, the server does not accept a connection from a client. Instead, the server just calls the recvfrom function, which waits until data arrives from some client.
- *recvfrom* returns the protocol address of the client, along with the datagram, so the server can send a response to the client.

Steps on the client side are as follows :

1. Create a socket using the socket( ) function;
2. Send and receive data by means of the recvfrom( ) and sendto( ) functions.

Steps on the server side are as follows :

1. Create a socket with the socket() function;
2. Bind the socket to an address using the bind( ) function;
3. Send and receive data by means of recvfrom( ) and sendto( ).

END...

JUNE - 2022 (END SEM) [5869] - 285 **Solved Paper**

Course 2019

Time : $2\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Instructions to the candidates :

1) *Neat diagrams must be drawn wherever necessary.*
2) *Figures to the right side indicate full marks.*
3) *Use of calculator is allowed.*
4) *Assume suitable data if necessary.*

Q.1 a) *Write a note on channelization techniques (Any Two)*
i) FDMA ii) TDMA iii) CDMA.
(Refer Q.6 of Chapter - 3) **[8]**

b) *Compare IEEE 802.3, IEEE 802.4, IEEE 802.5 in a tabular format.* **(Refer Q.11 of Chapter - 3)** **[9]**

OR

Q.2 a) *Explain CSMA / CA and CSMA / CD random access technique with suitable diagram / flowchart. Also comment on efficiency of each.*
(Refer Q.4 of Chapter - 3) **[8]**

b) *Write a note on*
i) Standard ethernet ii) Fast ethernet iii) Gigabit ethernet
(Refer Q.15 of Chapter - 3) **[9]**

Q.3 a) *What is subnetting ? A company is granted a site address 172.16.10.33/19 design the subnets and answer following questions :*
i) How many subnets does the chosen subnet mask produce ?

ii) How many valid hosts per subnet are available ?
iii) What are the valid subnets ?
iv) What's the broadcast address of each subnet ?
v) What are the valid hosts in each subnet ?
(Refer Q.7 of Chapter - 4) **[8]**

b) *What is the need of IPv6 ? Explain types of IPv6 address.* **(Refer Q.11 of Chapter - 4)** **[9]**

OR

Q.4 a) *Draw and explain IPv4 header format. List out special IP addresses and private IP addresses.*
(Refer Q.3 of Chapter - 4) **[8]**

b) *List the network layer services and define subnetting, supernetting, classful addressing, classless addressing.*
(Refer Q.9 of Chapter - 4) **[9]**

Q.5 a) *What is BGP ? What are the characteristics of BGP routing protocol ? What are the advantages and disadvantages of BGP routing protocol ?*
(Refer Q.13 of Chapter - 5) **[9]**

b) *Draw the router architecture. Explain the difference between RIP, EIGRP, OSPF in tabular format.*
(Refer Q.12 of Chapter - 5) **[9]**

OR

Q.6 a) *What are the problems in RIP ? How to overcome the problems ? Compare RIPv1 and RIPv2.*
(Refer Q.4 of Chapter - 5) **[9]**

b) *Explain following routing.*
i) Static routing ii) Dynamic routing iii) Default routing. [9]
(Refer Q.1 of Chapter - 5)

Q.7 a) *What is the purpose of Leaky bucket and token bucket algorithms ? Describe working of Leaky bucket algorithm with reference to CBR, VBR and bursty traffic.* [9]
(Refer Q.10 of Chapter - 6)

b) *What is socket ? Explain various socket primitives used in client server interaction.* **(Refer Q.13 of Chapter - 6)** [9]

OR

Q.8 a) *Explain the three-way handshake algorithm for TCP connection establishment. List the fields in TCP header that are not part of UDP header. Give the reasons of each missing field.* **(Refer Q.4 of Chapter - 6)** [9]

b) *List out key features of UDP protocol. Explain how flow control is different than congestion control in TCP ?* [9]
(Refer Q.12 of Chapter - 6)

END...