11.10.3 Path Attributes :

- The path is specified in terms of attributes. Each attribute gives some information about the path. Hence the list of attributes helps the receiving router to make a better decision about when to apply its policy.
- Attributes are of two types :
  1. A well known attribute
  2. An optional attribute
- An attribute is called as a well known attribute if it is recognised by every BGP router.
- An optional attribute is the one that need not be recognised by every BGP router.
- The well known attributes are further classified into two categories :
  1. Well known mandatory attributes
  2. Well known discretionary attributes.
- The optional attributes also are classified into two types :
  1. An optional transitive attribute
  2. An optional nontransitive attribute.

Review Questions

Q. 1 State and explain the various services provided by network layer.

Q. 2 What is packetizing ?

Q. 3 Write short note on : routing and forwarding.

Q. 4 Explain error control and flow control.

Q. 5 Write short note on : IPv4 addresses.

Q. 6 What do you mean by uniqueness of IP addresses.

Q. 7 Draw IPv4 address format.

Q. 8 Define classful addressing.

Q. 9 Draw class B IPv4 address format.

Q. 10 How to recognize IPv4 classes.

Q. 11 Write short note on : Two level addressing in classful addressing.

Q. 12 How information is extracted in classful addressing ?

Q. 13 Define default mask.

Q. 14 Write default masks for different classes.

Q. 15 Define subnetting.

Q. 16 Write down limitations of IPv4.

Q. 17 Who decides the IP addresses ?

Q. 18 State the types of routing.

Q. 19 Explain unicast and broadcast routing.

Q. 20 Write down desired properties of a routing algorithm.

Q. 21 Write short note on : optimality principle.

Q. 22 Explain shortest path routing.

Q. 23 Explain distance vector routing algorithms.

Q. 24 Write short note on : Link state routing.

Q. 25 Compare link state routing and distance vector routing.

Q. 26 Write short note on : path vector routing.

□□□

CHAPTER 12 Unit III

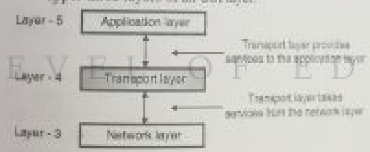
Introduction to Transport Layer

Syllabus :

Introduction to transport layer, Transport layer services, Connectionless and connection oriented protocols, Transport layer protocols, Services, Port number, User datagram protocol, User datagram, UDP services, UDP applications, Transmission control protocol, TCP services, TCP features, Segment.

12.1 Introduction :

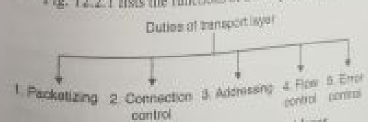
- The transport layer is the core of the Internet model. The application layer programs interact with each other using the services of the transport layer.
- Transport layer provides services to the application layer and takes services from the network layer.
- Fig. 12.1.1 shows the position of the transport layer in the 5-layer internet model. The transport layer is fourth layer in this model. It connects the lower three layers to upper three layers of an OSI layer.



(G-892) Fig. 12.1.1 : Position of transport layer

12.2 Transport Layer Duties and Functionalities :

- Transport layer is meant for the process to process delivery and it is achieved by performing a number of functions.
- Fig. 12.2.1 lists the functions of a transport layer.



(G-1407) Fig. 12.2.1 : Duties of transport layer

1. **Packetizing :**
   - The transport layer creates packets with the help of encapsulation on the messages received from the application layer. Packetizing is a process of dividing a long message into smaller ones.
   - These packets are then encapsulated into the data field of the transport layer packet. The headers containing source and destination address are then added.
   - The length of the message which is to be divided can vary from several lines (e-mail) to several pages.
   - But the size of the message can become a problem. The message size can be larger than the maximum size that can be handled by the lower layer protocols.
   - Hence the messages must be divided into smaller sections. Each small section is then encapsulated into a separate packet.
   - Then a header is added to each packet to allow the transport layer to perform its other functions.
2. **Connection control :**
   - Transport layer protocols are divided into two categories :
     1. Connection oriented.
     2. Connectionless.

   Connection oriented delivery :
   - A connection oriented transport layer protocol establishes a connection i.e. virtual path between sender and receiver.
   - This is a virtual connection. The packet may travel out of order. The packets are numbered consecutively and communication is bi directional.

   Connectionless delivery :

   A connectionless transport protocol will treat each packet independently. There is no connection between them. Each packet can take its own different route.
3. **Addressing :**

   The client needs the address of the remote computer it wants to communicate with. Such a remote computer has a unique address so that it can be distinguished from all the other computers.

4. **Flow and error control :**

For high reliability the flow control and error control should be incorporated.

- **Flow control :** We know that data link layer can provide the flow control. Similarly transport layer also can provide flow control. But this flow control is performed end to end and not across a single link.
- **Error control :** The transport layer can provide error control as well. But error control at transport layer is performed end to end and not across a single link. Error correction is generally achieved by retransmission of the packets discarded due to errors.

Congestion control and QoS :

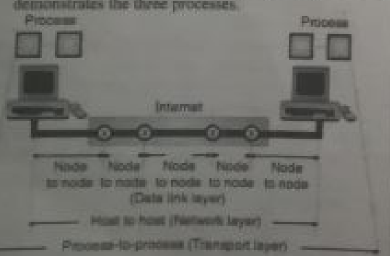
- The congestion can take place in the data link, network or transport layer. But the effect of congestion is generally evident in the transport layer.
- Quality of Service (QoS) can be implemented in other layers but its actual effect is felt in the transport layer.
- The transport layer enhances the QoS provided by the network layer.

12.3 Transport Layer Services :

In this section we are going to discuss the services provided by the transport layer.

12.3.1 Process-to-Process Communication :

- The data link layer performs a node to node delivery. The network layer carries out the datagram delivery between two hosts (host to host delivery).
- But the real communication takes place between two processes or application programs for which we need the **process-to-process delivery.**
- The transport layer takes care of the **process-to-process delivery.** In this a packet from one process is delivered to the other process.
- The relationship between the communicating processes is the client-server relationship. Fig. 12.3.1 demonstrates the three processes.



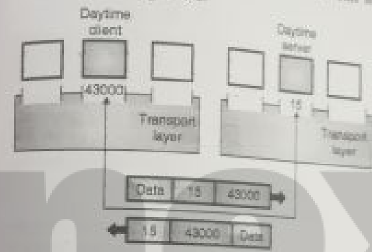
(G-594) **Fig. 12.3.1 : Types of data deliveries**

- There is a difference between host-to-host communication and process to process communication that we need to understand clearly.
- The host to host (computer to computer) communication is handled by the network layer. But this communication only ensures that the message is delivered to the destination computer. But this is not enough.
- It is necessary to handover this message to the correct process. The transport layer will take care of this.

12.3.2 Addressing : Port Number :

- There are several ways of achieving the process-to-process communication, but the most common method is using the client-server paradigm.
- **Client** is defined as the process on the local host. It needs services from another process called **server** which is on the other (remote) host.
- Both client and server have the same name. Some of the important terms related to the client-server paradigm are :
  1. Local host
  2. Remote host
  3. Local process
  4. Remote process
- We can use the IP addresses to define the local host and remote host. But this is not enough to define a process.
- In order to define a process, we have to use one more identifier called **Port Numbers.** In TCP/protocol suite, the port numbers are integers and they are numbered between 0 and 65,535.
- At the data link layer we need a MAC address, at the network layer we need to use an IP address. A datagram uses the destination IP address to deliver the datagram and uses the source IP address for the destination's reply.
- At the transport layer a transport layer address called a **port number** is required to be used to choose among multiple processes running on the destination host.
- The destination port number is required to make the packet delivery and the source port number is needed to return back the reply.
- In the Internet model, the port numbers are 16 bit integers. Hence the number of possible port numbers will be $2^{16} = 65,535$ and the port numbers range from 0 to 65,535.
- The client program identifies itself with a port number which is chosen randomly. This number is called as **ephemeral port number.** Ephemeral means short lived. It is used because life of a client is generally short.
- The server process should also identify itself with a port number but this port number can not be chosen randomly.

- The Internet uses universal port numbers for servers and these numbers are called as **well known port numbers.**
- Every client process knows the well known port numbers of the pre identified server process.
- For example, a Day time client process can use an ephemeral (temporary) port number 43000 for identifying itself, the Day time server process must use the well known (permanent) port number 15. This is illustrated in Fig. 12.3.2.



(G-595) **Fig. 12.3.2 : Concept of port numbers**

What is difference between IP Addresses and Port Numbers ?

- The IP addresses and port numbers have altogether different roles in selecting the final destination of data.
- The destination IP address is used for defining a particular host among the millions of hosts in the world.
- After a particular host is selected, the port number is used for identifying one of the processes on this selected host.

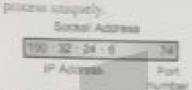
IANA Ranges :

- The port numbers are divided into three ranges by IANA (International Assigned Number Authority).
- The ranges are as follows :
  1. Well known ports
  2. Registered ports
  3. Dynamic or private ports.

1. **Well known ports :** The ports from 0 to 1023 are known as well known ports. They are assigned as well as controlled by IANA.
2. **Registered ports :** The ports from 1024 to 49,151 are neither controlled nor assigned by IANA. We can only register them with IANA to avoid duplication.
3. **Dynamic or private ports :** The ports from 49,152 to 65,535 are known as dynamic ports and they are neither controlled nor registered. They can be used by any process. Dynamic ports are also known as private ports and dynamic port are called as ephemeral ports.

Socket Address :

- Process to process delivery (transport layer communication) has to use two addresses, one is IP address and the other is port number at each end to make a connection. Hence a process to process delivery uses the combination of these two.
- The combination of IP address and port number is as shown in Fig. 12.3.3 and it is known as the socket address.
- The client socket address defines the client process uniquely whereas the server socket address defines the server process uniquely.



(G-596) **Fig. 12.3.3 : Socket address**

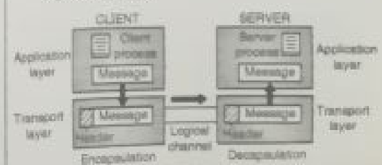
- A transport layer protocol requires the client socket address as well as the server socket address. These two addresses contain four pieces.
- These four pieces go into the IP header and the transport layer protocol header.
- The IP header contains the IP addresses while the UDP and TCP headers contain the port numbers.
- If we want to use the transport layer services in the Internet, then we have to use a pair of socket addresses namely the clients socket address and the server's socket address.

12.3.3 Encapsulation and Decapsulation :

- The transport layer carries out the **Encapsulation** of the message at the sending end and then **Decapsulation** at the receiving end when two computers communicate. This process has been illustrated in Fig. 12.3.4.

Encapsulation :

- At the sending end the process that has a message to send, will pass it to the transport layer alongwith a pair of socket addresses and some additional information.
- The transport layer adds its own header to this data. This packet at the transport layer in the Internet is known by different names such as **user datagram, segment or packet.**



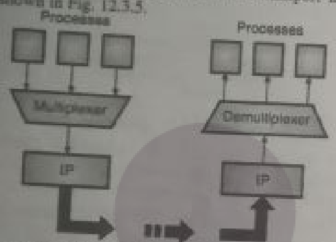
(G-2803) **Fig. 12.3.4 : Encapsulation and decapsulation**

Decapsulation :

- When the segment or datagram arrives at the receiving end, the header is isolated and destroyed, and the message is delivered to the process running at the application layer as shown in Fig. 12.3.4.
- The socket address of the sender process is then handed over to the destination process.

12.3.4 Multiplexing and Demultiplexing :

- The addressing mechanism allows multiplexing and demultiplexing taking place at the transport layer as shown in Fig. 12.3.5.



(G-897) **Fig. 12.3.5 : Multiplexing and demultiplexing**

Multiplexing :

- At the sending end, there are several processes that are interested in sending packets. But there is only one transport layer protocol (UDP or TCP). Thus it is a many processes-one transport layer protocol situation.
- Such a many-to-one relationship requires multiplexing.
- The protocol first accepts messages from different processes. These messages are separated from each other by their port numbers. Each process has a unique port number assigned to it.
- Then the transport layer adds header and passes the packet to the network layer as shown in Fig. 12.3.5.

Demultiplexing :

- At the receiving end, the relationship is one as to many. So we need a demultiplexer.
- First the transport layer receives datagrams from the network layer.
- The transport layer then checks for errors and drops the header to obtain the messages and delivers them to appropriate process based on the port number.

12.3.5 Flow Control :

- If the packets produced by the sender are at a rate X and the receiver is receiving them at a rate Y, then for X = Y, there will be a perfect balance observed in the system.
- But if X is higher than Y (source is producing packets at a rate which is higher than the rate at which the receiver is accepting them), then the receiver can be overwhelmed and has to **discard** some packets.
- And if X is less than Y (i.e. source is producing packets at slower rate than the rate of acceptance at the receiver) then system becomes **less efficient.**
- Flow control is related to the situation in which X > Y because it is very important to prevent data loss (due to discarding of packets) at the receiver site.

Pushing and pulling for flow control :

- There are two different ways of delivering the packets produced by the sender to the receiver. They are pushing or pulling.

1. **Pushing :**

   If the sender is sending the packets soon as they are produced, without receiving any prior request from the receiver then this type of deliver is called as **pushing.** Fig. 12.3.6(a) illustrates this concept.

2. **Pulling :**

   If the sender sends the produced packets only when they are requested by the receiver then the delivery is called as **pulling.** Fig. 12.3.6(b) illustrates the principle of pulling.

(a) Concept of pushing (b) Concept of pulling

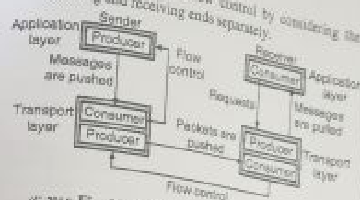
(G-2013) **Fig. 12.3.6**

- In case of **pushing** type delivery, if the packets are being sent at a higher rate than that of receiving, then the receiver will be **overwhelmed,** and some received packets will have to be discarded.
- In order to avoid discarding of packets, the **flow control** will have to be exercised. For this the receiver has to warn the sender to stop the delivery when it is overwhelmed and it has to inform the sender again to start delivery when it (receiver) is ready, to receive the packets.
- In case of **pulling type delivery,** the receiver is actually pulling the packets from the sender. It requests for the packets when it is ready. Therefore the flow control is not required in this case.

12.3.6 Flow Control at Transport Layer :

- The concept of flow control at transport layer has been illustrated in Fig. 12.3.7. It shows the communication taking place between a sender and a receiver.
- As shown in Fig. 12.3.7, there are four entities involved in this communication. They are as follows :
  1. Sender process.
  2. Sender transport layer.

  3. Receiver process.
  4. Receiver transport layer.
- We will discuss the flow control by considering the sending and receiving ends separately.



(G-2014) **Fig. 12.3.7 : Flow control at transport layer**

Sending end :

- The first entity on the sending end is the **sender process,** at the application layer. It works only as a **producer** which produces chunk of messages and pushes them to the transport layer on the sending end, as shown in Fig. 12.3.7.
- The second entity on the sending end is the **sender transport layer.** It has two different roles to play.
- First it acts as a **customer** and consumes all the messages produced and pushed by the producer. Then it **encapsulates** those messages into packets and pushes them to the receiver transport layer as shown in Fig. 12.3.7. Here it acts as a producer.

Receiving end :

- The first entity on the receiving end is the receiver **transport layer.** It also has two different roles to play.
- It acts as a **consumer** for the packets pushed by the senders transport layer and it also acts as the **producer.** It has decapsulate the messages and deliver them to the application layer as shown in Fig. 12.3.7.
- However the delivery of decapsulated messages to the application layer is a **pulling** type delivery. That means the transport layer waits till the application layer process requests for the decapsulated messages.

Flow control :

- As shown in Fig. 12.3.7, the flow control is needed for atleast two cases. First is from transport layer of sender to the application layer of sender.
- And secondly form the transport layer of receiver to the transport layer of sender.

Buffers :

- It is possible to implement the flow control in many different ways. One of the ways of implementation is to use two **buffers** one each at the sending and receiving transport layers.
- A **buffer** is nothing but a set of memory locations which can temporarily hold (store) packets.
- It is possible to exercise flow control communication by sending signals from the consumer to producer.
- The **flow control** at the **sending end** takes place as follows : As soon as the buffer at the transport layer becomes full it sends the stop message to its application layer in order to stop the chunk of messages that are being pushed into the buffer.
- The second flow control takes place at the receiver transport layer as follows : As soon as the buffer at receiver transport layer becomes full, it will inform the sender transport layer to stop pushing the packets.
- Whenever the buffer becomes partially empty, it again informs the sender transport layer to start sending the packets again.

12.3.7 Error Control :

Need of error control :

- In the Internet, the network layer protocol IP has the responsibility to carry the packets from the transport layer at the sending end to the transport layer at the receiving end.
- But IP is unreliable. Therefore transport layer should be made reliable, in order to ensure reliability at the application layer.
- We can make the transport layer reliable by adding the **error control service** to the transport layer.

Duties of error control mechanism :

- Following are the important responsibilities of the error control mechanism introduced at the transport layer :
  1. To find and discard the corrupted packets.
  2. To keep the track of lost and discarded packets and to resend them.
  3. Identify the duplicate packets and discard them.
  4. To buffer out of order packets until the missing packets arrive.
- In the error control process, only the sending and receiving transport layers are involved. That means it is assumed that the chunk of messages exchanged between the application layers and transport layers are error free.
- The concept of error control at the transport layer level is demonstrated in Fig. 12.3.8.
- The receiving transport layer manages the error control by communicating with the sending transport layer about the problem.

(G-2015) **Fig. 12.3.8 : Concept of error control at the transport layer**

Sequence numbers :

- In order to exercise the error control at the transport layer following two requirements should be satisfied :

1. The sending transport layer should know about the packet which is to be resent.
2. The receiving transport layer should know about the packets which are duplicate or the ones that have arrived out of order.

- The requirements can be satisfied only if each packet has a unique **sequence number.**
- If a packet is either corrupted or lost the receiving transport layer will somehow inform the sending transport layer about the sequence number of those packets and request it to resend those packets.
- Due to the unique sequence number assigned to each packet it is possible for the receiving transport layer to identify the duplicate packets received. The out of order packets can also be recognized by observing gaps in the sequence numbers of the received packets.
- Packet numbers are given sequentially. But the length of the sequence number cannot be too long because the sequence number is to be included in the header of the packets.
- If the header of a packet allows "m" bits per sequence number, then the range of sequence number will be from 0 to 2^m-1. For example if m = 3 then the range of sequence numbers will be from 0 to 7.
- Thus sequence numbers are modulo 2^m.

Acknowledgement :

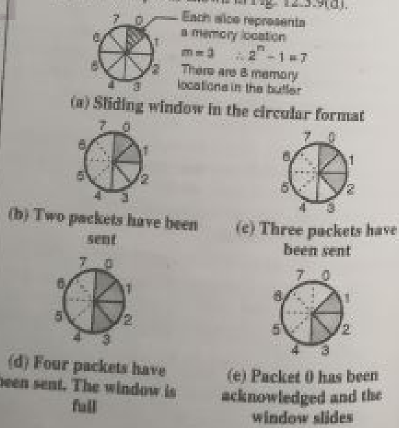
- The receiver side can send an acknowledgement (ACK) signal corresponding to each packet or each group of packets which arrived safe and sound.
- The question is what happens if a received packet is corrupted ? The answer is that the receiver simply discards the corrupted packet and does not send any ACK signal for it.
- The sender can detect a lost packet with the help of a timer. A timer is started at the sending end as soon as a packet is sent. If the ACK does not arrive before the expiry of the timer, then the sender treats the packet to be either lost or corrupted and resends it.
- The receiver silently discards the duplicate packets. It will either discard the out of order packets or stored until the missing packet is received.
- Note that every discarded packet is treated as a lost packet by the sender.

12.3.8 Combination of Flow and Error Control :

- Till now we have discussed the following important concepts :
  1. We need to use buffers at the sending and receiving ends for exercising the flow control.
  2. Also we have to use the sequence numbers and acknowledgements for exercising the error control.
- We can combine these two concepts together by using two numbered buffers one at the sender and the other at the receiver, in order to exercise a combination of flow and error control.
- At the sending end, when a packet, is prepared to be sent, the number of the next free location (x) in the buffer is used as the sequence number of that packet.
- As soon as the packet is sent, its copy is stored at location (x) in the sending end buffer and the sender waits for the acknowledgement from the receiver.
- On reception of the acknowledgement of the sent packet, the copy of that packet is purged to make the memory location (x) free again.
- At the receiver, when a packet having a sequence number "y" arrives, it is stored at the memory location "y" in the receiver buffer until the receiver application layer is ready to receive it. The receiver will send the ACK message back to sender to inform it that packet "y" has arrived.

Sliding window :

- As the sequence numbers are modulo 2^m, we can use a circle as shown in Fig. 12.3.9 to represent the sequence number from 0 to 2^m-1.
- We can represent the buffer as a set of slices, called as the **sliding window** which will occupy a part of the circle at any time.
- In Fig. 12.3.9, we have assumed that m = 3. Therefore $2^m - 1 = 7$ and the sequence numbers are from 0 to 7. Hence the number of memory locations in a buffer will also be 8 i.e. 0 to 7.
- The sliding windows will correspond to the sender as well as receiver.
- On the sending side, when a packet is sent we will mark the corresponding slice. Therefore when marking of all the slices is done, it means the **sending buffer is full,** and it cannot accept any further messages from the application layer as shown in Fig. 12.3.9(d).



(a) Sliding window in the circular format

(b) Two packets have been sent

(c) Three packets have been sent

(d) Four packets have been sent. The window is full

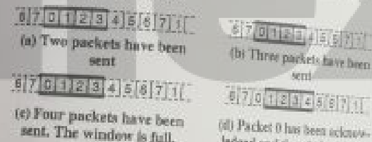
(e) Packet 0 has been acknowledged and the window slides

(G-2017) **Fig. 12.3.9**

- When the acknowledgement for segment "0" arrives at the sending end, the corresponding segment (segment 0) is unmarked and window slides ahead by one slice as shown in Fig. 12.3.9(e). The size of the **sending window** is 4.
- Note that the sliding window is just an abstraction. In actual practice, computer variables are used to hold the sequence number of the next packet to be sent and the last packet sent.

Sliding window in the linear format :

- This is another way to diagrammatically represent a sliding window. It is as shown in Fig. 12.3.10.
- The principle of this type of sliding window is same as that of the circular representation. The linear format is the most preferred format. It needs less space on paper.
- Fig. 12.3.10(a), (b), (c) and (d) are the sliding windows presented in the linear format corresponding to Figs. 12.3.9(b), (c), (d) and (e) respectively in the circular presentation.



(a) Two packets have been sent

(b) Three packets have been sent

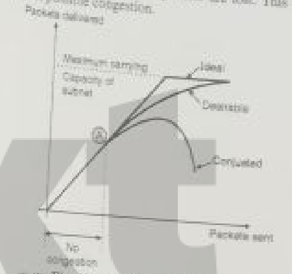
(c) Four packets have been sent. The window is full.

(d) Packet 0 has been acknowledged and the window slides

(G-2075) **Fig. 12.3.10 : Sliding windows presented in the linear format**

12.3.9 Congestion Control :

- An important issue in a packet switching network is congestion.
- If an extremely large number of packets are present in a part of a subnet, the performance degrades. This situation is called as congestion.
- Congestion in a network may occur when the load on the network i.e. the number of packets sent to the network is greater than the capacity of the network (i.e. the number of packets a network can handle).
- Fig. 12.3.11 explains the concept of congestion graphically.
- Upto point A in Fig. 12.3.11, the number of packets sent into the subnet by the host is within the capacity of the network. So all these packets are delivered. In short the number of packets delivered is proportional to number of packets sent and no congestion takes place.
- But after point A, the traffic increases too far. The routers cannot cope with the increased traffic and they begin to lose packets. The congestion begins here.
- As the traffic increases further, the performance degrades more and more packets are lost and congestion worsens.
- At very high traffic, the performance collapses completely and almost all packets are lost. This is the worst possible congestion.



(G-453) **Fig. 12.3.11 : Concept of congestion**

Need of congestion control :

- We may define the **congestion control** as the mechanisms and techniques to control the congestion and keep the load below the capacity.
- It is not possible to completely avoid the congestion but it is necessary to avoid it otherwise control it.
- Congestion will result in long queues, which results in buffer overflow and loss of packets.
- So congestion control is necessary to ensure that the user gets the negotiated QoS (Quality of Service).

Causes of congestion :

- Congestion happens in any network due to waiting, and due to the abnormality in the flow.
- It also occurs due to the fact that routers and switches have queues at the buffers which store packet before and after their processing.

12.3.10 Connectionless and Connection Oriented Services :

- A transport layer protocol is capable of providing two types of services :
  1. Connectionless services.
  2. Connection oriented services.

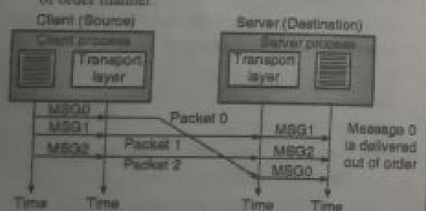
- The meaning of the words connectionless and connection oriented is different at the transport layer than that at the network layer.
- A connectionless service at the network layer means different datagrams of the same message following different paths.
- However at the transport layer, the meaning of connectionless service is independency between different packets.
- On the other hand a connection oriented service means the packets are interdependent.

Connectionless service :

- Refer Fig. 12.3.12 to understand the concept of connectionless service.
- The source process at the application layer first divides its message in chunks of data the size of which is acceptable to the transport layer.
- These data chunks are then delivered to the transport layer one by one. These chunks are treated as independent units by the transport layer.
- Every data chunk arriving from the application layer is encapsulated in a packet by the transport layer and sent to the destination transport layer as shown in Fig. 12.3.12.

Out of Order Delivery :

- In Fig. 12.3.12 we have considered three chunks of independent messages 0, 1 and 2. As the corresponding packets also are independent of each other and as they are free to follow their own path, these packets can arrive out of order at the destination as shown in Fig. 12.3.12.
- Naturally they are delivered to server process in an out of order manner.



(G-2010) **Fig. 12.3.12 : Concept of connectionless service**

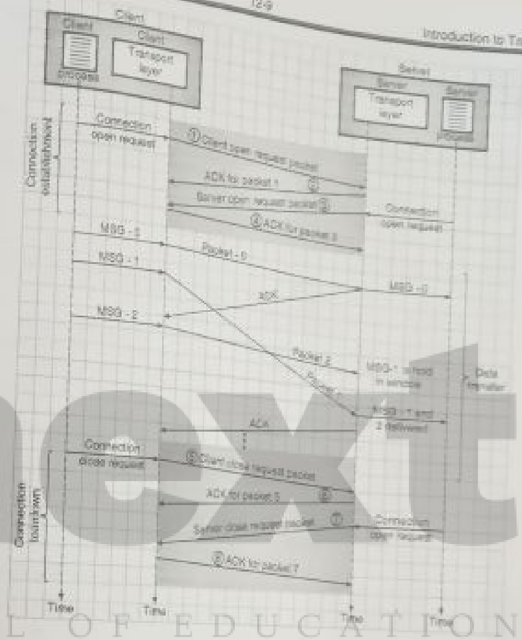
- As seen in Fig. 12.3.12, at the sending end (client) the three chunks of messages 0, 1 and 2 are delivered to the transport layer in the order 0, 1, 2.
- But packet 0 travels a longer path and undergoes an extra delay. Therefore the packets are not delivered in order at the destination (server) transport layer.
- Therefore the message chunks delivered to the server process will also be out of order (1, 2, 0).
- If these chunks are of the same message then due to their out of order delivery the server will receive a strange message.

One packet is lost :

- The UDP packets are not numbered. So if one of the packets is lost, then the receiving transport layer will not have any idea about the lost packet. It will simply deliver the received chunks of messages to the server process.
- The above problems arise due to **lack of coordination** between the two transport layers. Due to this lack of co-ordination it is not possible to implement flow control, error control or congestion control in the connectionless service.

Connection oriented service :

- As we know, there are three stages involved in the connection oriented service. They are :
  1. Connection establishment.
  2. Exchange of data.
  3. Connection teardown.
- The connection oriented service is present at the network layer as well, but it is different from that at the transport layer.
- At the network layer, the meaning of connection oriented service involves the co-ordination between the hosts on either sides and all the routers between them.
- But at the transport layer, the meaning of connection oriented service is the end to end service that involves only the two hosts.
- Refer Fig. 12.3.13 to understand the concept of connection oriented service at the transport layer.
- In Fig. 12.3.13, all the three stages namely connection establishment, data exchange and connection teardown have been shown.
- It is important to note that it is possible to implement the flow control, error control and congestion control in the connection oriented service.



(G-2016) Fig. 12.3.13 : Concept of connection oriented service

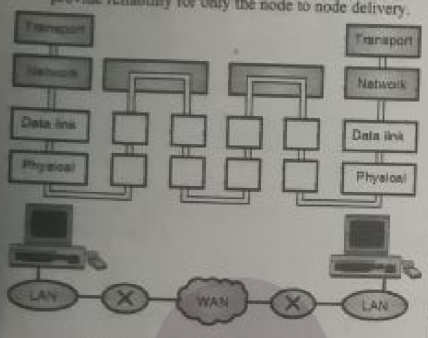
Comparison of Connection Oriented and Connectionless Services :

Sr. No.	Parameter	Connection oriented	Connectionless
1.	Reservation of resources	Necessary	Not necessary
2.	Utilization of resources	Less	Good
3.	State information	Lot of information required	Not much information is required to be stored
4.	Guarantee of service	Guaranteed	No guarantee
5.	Connection	Connection needs to be established	Connection need not be established
6.	Delays	More	Less
7.	Overheads	Less	More
8.	Packets travel	Sequentially	Randomly
9.	Congestion due to overloading	Not possible	Very much possible

12.3.11 Reliability at Transport Layer Versus Reliability at DLL :

- The transport layer services can be of two types :
  1. Reliable services 2. Unreliable services.
- If the application layer program needs reliability then the reliable transport layer protocol is used which implements the flow and error control at the transport layer. But this service will be slow and more complex.
- But some application layer programs do not need reliability because they have their own flow and error control mechanisms. Such programs use an unreliable service.
- UDP is connectionless and unreliable, but TCP is connection oriented and reliable protocol. Both these are the transport layer protocols.

- We need reliability at the transport layer eventhough data link layer is reliable because the data link can provide reliability for only the node to node delivery.



(G-898) Fig. 12.3.14 : Error control

- The error control at the data link layer does not guarantee error control at the transport layer. The network layer service in the Internet is unreliable. Hence reliability at the transport layer must be ensured independently.
- Therefore flow and error controls are implemented in TCP using the sliding window protocols. This is reliability assurance at the transport layer.

Note that the error is checked only upto the data link layer by the data link error control system.

12.3.12 Quality of Service (QoS) :

- As mentioned earlier, the QoS parameters are as follows :

1. **Connection establishment delay :**
   - The time difference between the instant at which a request for transport connection is made and the instant at which it is confirmed is called as **connection establishment delay.**
   - This delay should be as short as possible to ensure better service.
2. **Connection establishment failure probability :**
   - Sometimes the connection may not get established even after the maximum connection establishment delay.
   - This can be due to network congestion, lack of table space or some other problems.
3. **Throughput :**
   - It is defined as the number of bytes of user data transferred per second, measured over some time interval.
   - Throughput is measured separately for each direction.
4. **Transit delay :**

   It is the time duration between a message being sent by the transport user from the source machine and its being received by the transport user at the destination machine.
5. **Residual error ratio :**
   - It measures the number of lost or garbled messages as a percentage of the total messages sent.
   - Ideally the value of this ratio should be zero and practically it should be as small as possible.
6. **Protection :**

   This parameter provides a way to protect the transmitted data against reading or modifying it by some unauthorised parties.
7. **Priority :**
   - Using this parameter the user can show that some of its connections are more important (have higher priority) than the other ones.
   - This is important when congestions take place. Because the higher priority connections should get service before the low priority connections.
8. **Resilience :**

   Due to internal problem or congestion the transport layer spontaneously terminates a connection. The resilience parameter gives the probability of such a termination.

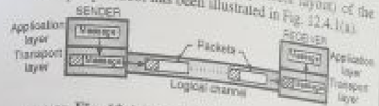
12.4 Transport Layer Protocols :

- We have discussed a few transport layer services in the previous section. By combining a set of these services as per requirement, we can create a transport layer protocol.
- It is important to understand the behavior of these general protocols, before we discuss the transport layer protocols such as UDP and TCP.
- In this section we will discuss the following protocols :
  1. Simplex protocol.
  2. Stop and wait protocol.
  3. Go back N (GBN) protocol.
  4. Selective repeat protocol.
  5. Bidirectional protocol. (Piggybacking).
- Initially we will discuss all these protocols as simplex i.e. **unidirectional** protocols and then we will see how to make them the **full duplex** i.e. **bidirectional** protocols.

12.4.1 Simplex Protocol :

- This is the simplest type of connectionless protocol which has the following characteristics :
  1. No flow control.
  2. No error control.
  3. The receiver does not get overwhelmed.

- Because the receiver does not get overwhelmed due to the incoming packets even at very high rate, the receiver can handle any packet immediately as soon as it is received.
- The principle of operation (or protocol layout) of the simple protocol has been illustrated in Fig. 12.4.1(a).



(G-2179) Fig. 12.4.1(a) : Layout of the simple protocol

Operation :

At the sender :

- The application layer at the sender, sends its message to the transport layer.
- The sender transport layer receives the message and makes a packet out of it.
- This packet is then sent over the logical channel between the transport layers on the two ends.

At the receiver :

- The network layer at the receiver (not shown in Fig. 12.4.1(a)) delivers the received packet to the transport layer.
- The receiver transport layer extracts the message from the packet (decapsulation) and sends the message to the application layer.

FSM :

- In this protocol, the sender should not send a packet as long as its application layer does not have a message to send.
- Whereas the receiving transport layer should not deliver a message to its application layer unless it receives a packet from the sender.
- These two requirements suggest, the sender and the receiver have only one state : Ready state.
- The sending machine remains in the **ready** state until a process in its application layer sends a request to send its message.
- As soon as the request comes, the sending machine will encapsulate the message and send it to the receiver.
- The receiving machine also remains in **ready state** until it receives a packet from the sender.
- On arrival of a packet, the receiver decapsulates it and delivers the extracted message to the application layer process.

(G-2180) Fig. 12.4.1(b) : FSM for the simple protocol

- Note that the UDP protocol is a slight modification of this protocol. The FSM (Finite State Machine) for this protocol has been shown in Fig. 12.4.1(b) and its flow diagram is as shown in Fig. 12.4.1(c).

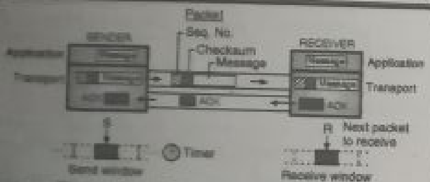
Flow Diagram :

- The communication between the sender and receiver using the simple protocol has been shown in Fig. 12.4.1(c).
- The sender keeps sending the packets, without taking the receiver into consideration at all.

(G-2181) Fig. 12.4.1(c) : Flow diagram for the simple protocol

12.4.2 Stop and Wait Protocol :

- The second transport layer protocol that we will discuss now is a **connection oriented** protocol called as **stop and wait protocol.**
- The operation of this protocol are as follows :
  1. It is a connection oriented protocol.
  2. It provides both flow and error control.
  3. Sender sends one packet at a time and waits for its acknowledgement from receiver before sending the next packet.
  4. A checksum is added to each data packet so as to detect a corrupted packet.
  5. At the receiver, the checksum in each packet is checked. If found incorrect, the receiver considers it as the corrupted packet and discards it silently. Such a packet is not acknowledged by the receiver.
  6. If the sender does not receive an acknowledgement for a packet within a predecided time, it understands that the packet is either corrupted or lost.
  7. The sender starts a timer everytime it sends out a packet. If it receives the acknowledgement for the packet before the expiry of the timer, it stops the timer, and sends the next packet. But if the timer expires before the arrival of acknowledgement, the sender resends the previous packet which was either corrupted or lost.
- Fig. 12.4.2(a) shows the principle of the stop and wait protocol. Note that at any given time there can be only one packet and one acknowledgement in the channel.



(G-2182) **Fig. 12.4.2(a) : Principle of stop and wait protocol**

Sequence number :

- In this protocol, sequence numbers and acknowledgement numbers are used for preventing duplicate packets.
- As shown in Fig. 12.4.2(b), an additional field is created in the packet header of each packet to hold its sequence number.

(G-2183) **Fig. 12.4.2(b) : Packet**

- A very important consideration about the sequence number is the **range of sequence numbers.**
- In order to provide an unambiguous communication with the minimum packet size, we look for the smallest range of sequence numbers.
- Let x be the sequence number of a packet, then the next sequence number should be (x + 1). There is no need for (x + 2). We can show it using the following discussion.
- Suppose that a packet with the sequence number x has been sent by the sender. Then the following three things can possibly happen.

1. Everything is normal :

- The first possibility is that the packet reaches its destination safe and sound without getting corrupted or lost. The receiver sends the acknowledgement for it.
- The acknowledgement reaches the sender safe and sound.
- The sender sends the next packet having a sequence number of (x + 1).

2. Packet corrupted or lost :

- The second possibility is that the sent packet either gets corrupted or gets lost and does not reach the receiving end at all.
- The receiver discards the corrupted packet silently. In either case (corrupted or lost packet), the acknowledgement is not sent back.
- The sender waits for the timer to expire and resends the packet numbered x. The receiver sends back the acknowledgement for it.

3. The acknowledgement is corrupted or lost :

- The packet (numbered x) arrives safe and sound at the receiving end for which it sends an acknowledgement back to the sender.
- However the acknowledgement either get corrupted or gets lost on its way back. Therefore the sender resends the packet (numbered x) again after the expiry of the timer.
- Thus packet x has a duplicate now. The receiver will understand this fact because it was expecting packet numbered (x + 1) to arrive but instead it received the packet numbered x again.

Conclusions :

- From the above discussion we can conclude that sequence numbers x and x + 1 are required so that the receiver can distinguish between cases 1 and 3 discussed above. But it is not necessary to number the packet as (x + 2).
- In case 1, we can number the packet as x again because both the packets (x and x + 1) are acknowledged by the receiver and neither the sender nor the receiver has any ambiguity about it.
- Finally in the case 2 and 3, the new packet is (x + 1) and not (x + 2). Therefore we conclude that only two sequence numbers x and x + 1 are needed and x +2 is not needed.
- So let x = 0 then (x + 1) = 1. Thus there will be only two sequence numbers 0 and 1 and the packet sequence would be 0, 1, 0, 1, 0... and so on. Due to the presence of only two distinct sequence numbers, this is called as modulo-2 arithmetic.

Acknowledgement numbers :

- For both types of packets i.e. data packets and acknowledgements, the same sequence numbers should be suitable.
- For this to happen successfully the following convention is used.
- The acknowledgement number always indicates the sequence number of the **next packet** that the receiver is expecting to receive.
- For example, the packet with a sequence number 0 arrives at the receiver safe and sound. Then the corresponding ACK sent by the receiver will have a number 1 on it which means that the next expected packet to be received is packet 1.
- Similarly if packet 1 arrives safe and sound then ACK with acknowledgement 0 is sent back which means that packet – 0 is the next expected packet at the receiver.
- The **control variable** at the sender is called as the **sender (s)** and it points to the only slot present in the send window as shown in Fig. 12.4.2(a).
- Similarly the **control variable** at the receiving end called as the **Receiver (R)** and it points to the only slot present in the receive window as shown in Fig. 12.4.2(a).

FSMs of stop and wait protocol :

- This protocol is a connection oriented protocol. Therefore a connection between the two ends should be established before transferring the data.

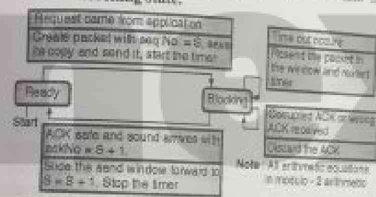
- In other words both sender and receiver must be in the **established state** before the beginning of data exchange.

1. Sender FSM :

- The sender's FSM is shown in Fig. 12.4.2(c). Initially it is in the **ready** state. However it can move between the ready and blocking state.
- The initial value of variable "s" is set to 0.

1. Ready state :

- The sender, when in the ready state waits only for one event to happen, that is the request coming from application layer.
- As soon as such a request comes from the application layer, the sender makes a packet with the sequence number same as "s".
- It stores a copy of this packet and sends the packet. The sender starts the timer and moves into its **blocking state.**



(G-2184) **Fig. 12.4.2(c) : Sender's FSM for stop and wait protocol**

2. Blocking state :

- When the sender is in the blocking state as shown in Fig. 12.4.2(c), the following three possible events can happen :

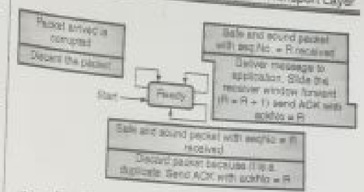
a. An error free ACK is received by the sender. It's ackNo is also correct i.e. (S + 1). The sender then stops the timer, slides the sending window to S = (S + 1) modulo - 2 and moves to the ready state.

b. The ACK received by the sender is either corrupted or a wrong ACK i.e. the one having the ackNo other than (S + 1). The sender discards the ACK.

c. In case if the timer expires (time out condition), the sender resends the only outstanding packet with it. It then restarts the timer as shown in Fig. 12.4.2(c).

2. Receiver FSM :

- The receiver's FSM is shown in Fig. 12.4.2(d). Note that there is no blocking state in the receiver's FSM. There is only the ready state.
- At the receiver also there is a possibility of following three events happening after the arrival of a packet.



(G-2185) **Fig. 12.4.2(d) : FSM of receiver for stop and wait protocol**

a. A safe and sound packet (without corruption) is received with seq.No = R. Then the messages is extracted (decapsulation) and delivered to the application layer. The receive window slides forward to (R = R + 1) modulo - 2 and the receiver sends an ACK with ackNo = R.

b. A safe and sound packet (with any error) arrives, but its seq.No ≠ R. This shows that it is a duplicate packet. The receiver will discard this packet but sends an ACK with ackNo = R.

c. The received packet is corrupted. The receiver silently discards it. No ACK is sent back.

Efficiency of stop and wait protocol :

- The efficiency of the stop and wait protocol is very very low. This is because it sends a packet and simply waits for its ACK before sending the next packet.
- This is a gross underutilization of the communication channel especially if the channel is **thick and long.**
- A channel is thick if it has a large bandwidth and it is **long** if it has a long round trip time.
- The product of these two parameters is called as **bandwidth delay product.**
- A channel is equivalent to a pipe. If it is underutilized, then it will be called inefficient.
- The number of bits a sender can transmit through the channel can be measured from the value of bandwidth delay product.
- On all these accounts the stop and wait protocol proves to be extremely inefficient.

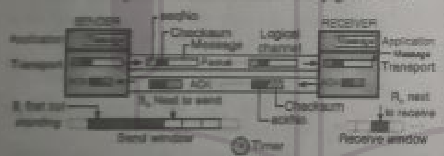
Pipelining :

- In networking and even other areas, a task is started before the ending of previous task. This is known as **pipelining.**
- In the stop and wait protocol, the senders sends a packet and waits for its acknowledgement before sending the next packet.
- This shows that there is no pipelining in the stop and wait protocol.
- But in the other protocols that we are going to discuss after the concept of pipelining will be used.

- Therefore it is possible for the sender to send several packets before it receives only acknowledgements for the previously sent packets.
- The process of pipelining improves the efficiency of the protocol.

12.4.3 Go Back-N Protocol (GBN) :

- The efficiency of transmission can be improved by transmitting multiple packets while the sender is waiting for acknowledgment.
- That means we should allow more than one outstanding packets even when the sender is waiting for acknowledgement because this will keep the channel busy.
- A protocol which can achieve this goal is our next protocol called Go Back-N (GBN) protocol.
- The most important part in the operation of GBN protocol is that we can send several packets before receiving acknowledgement. But the receiver can buffer only one packet.
- A copy of every sent packet is kept by the sender until it receives the acknowledgement of that packet.
- Fig. 12.4.3(a) shows the outline of GBN protocol which explains its principle of operation. Note the simultaneous presence of multiple packets and multiple acknowledgements in the channel at any given time.



(G-2186) **Fig. 12.4.3(a) : Principle of Go Back-N (GBN) protocol**

Sequence numbers :

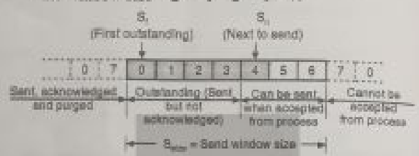
In GBN protocol, the sequence numbers are modulo 2^m, where m denotes the size of sequence number field in bits.

Acknowledge numbers :

- In the GBN protocol, the acknowledgement number is cumulative and it carries the sequence number of the next packet that is expected to be received at the receiver.
- If the ackNo = 6, its an indication that the receiver has received all the packets having sequence number upto 5 safe and sound. Hence the receiver is expecting the packet with seq.No = 6 to arrive next.

Send window :

- We can define the send window as an imaginary box, which covers the sequence numbers of the data packets that can be sent.
- The **maximum size** of the send window is $(2^m - 1)$ for the reasons discussed later on in the chapter.
- In each send window position (it can slide), some sequence numbers indicate the packets that have been already sent whereas the other sequence numbers indicate the data packet that are to be sent.
- In this chapter we assume that the send window size is fixed and has been set to its maximum possible value. But in some protocols the send window size is variable.
- The structure of a send window for the GBN protocol with m = 3 has been shown in Fig. 12.4.3(b). Note that the window size $= 2^m - 1 = 2^3 - 1 = 7$.



(G-2187) **Fig. 12.4.3(b) : Format of the send window of GBN**

- At any given time, the send window divides the possible sequence numbers into four regions.
- As shown in Fig. 12.4.3(b), the first region corresponds to the portion to the left of the send window. It consists of the sequence numbers which belong to the packet which are already acknowledged. The sender does not keep any copy of these packets.
- The second region which is shaded in Fig. 12.4.3(b) contains the sequence numbers belonging to the packets that are already sent but not acknowledged by the receiver. That means the exact status of these packets is not known.
- These packets are called as **outstanding packets.**
- The third range, which is not shaded in Fig. 12.4.3(b), contains the sequence numbers belonging to the packets which the sender can send. But the corresponding data is yet to be received from the application layer.
- And finally the fourth range, which is at the right of the send window in Fig. 12.4.3(b), consists of the sequence numbers that cannot be used by the sender until the send window slides to the right hand side.

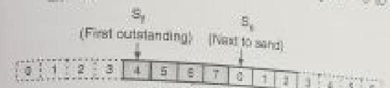
Size and location of send window :

- There are three variables that define the size and location of the send window at any given time. They are :
  1. S_f **:** Send window, the first outstanding packet.
  2. S_n **:** Send window ; the next packet to be sent.
  3. S_{size} **:** Send window, size.
- The sequence number of the first (oldest) outstanding packet is defined by the variable S_f.

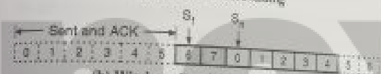
- The sequence number, that will be assigned to the next packet to be sent is defined by the variable S_n.
- And finally the size of the send window which is fixed in GBN protocol is defined by the variable S_{size}.

Sliding of send window :

- A send window will slide right on the arrival of acknowledgements.
- Fig. 12.4.4 shows the send window before sliding and after the arrival of an acknowledgement with ackNo = 6. This means that all packets upto seq.No = 5 have reached safe and sound and the receiver is expecting the packet with seq.No = 6 to arrive.



(a) Window before sliding



(b) Window after sliding (an ACK with ackNo = 6 has arrived)

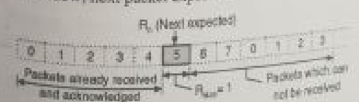
(G-2188) **Fig. 12.4.4 : Sliding of send window**

Conclusion :

From all this discussion we conclude that the send window will slide by one or more slots when the sender receives an errorfree ACK whose ackNo is greater than or equal to S_f and less than S_n.

Receive window :

- The receive window has two tasks : First it has to ensure that correct data packets are received and second is to make sure that correct acknowledgements are sent.
- The size of receive window in the GBN protocol is always 1. Therefore, the receiver is always expecting a specific packet to arrive.
- That means the receiver will discard any packet which arrives out of order and the sender has to resend the discarded packet.
- The receive window for the GBN protocol is shown in Fig. 12.4.5. It has only one variable R_n, i.e. receive window, next packet expected.



(G-2189) **Fig. 12.4.5 : Structure of receive window of GBN**

- The sequence numbers to the left of the receive window correspond to the already received and acknowledged packets.
- The sequence numbers to the right of receive window correspond to the packets which cannot be received.
- The receiver discards any packet that belongs to these two ranges. It will only accept that packet whose sequence number exactly matches with the value of R_n.
- Like the sliding window, the receive window also slides but only by one slot at a time. On reception of a correct packet, the receive window slides to $R_n = (R_{n+1})$ modulo 2^m.
- If a corrupted packet is received, the receive window does not slide at all.

Timers :

- Ideally there should be one timer per packet, which is sent. In GBN protocol only one timer is used.
- The reason for this is that the timer for the first outgoing packet will always expire first. If so, then all the outstanding packets will be resent by the sender.

Resending the packets :

- As stated earlier, on the expiry of the only timer (also called as time out), all the outstanding packets will be resent.
- As an example, let us assume that the sender has already sent the packet having seq.No 6 (S_n = 7) but the time out takes place (that means the only timer in GBN has expired).
- If S_f = 3, then it is an indication that the packets 3, 4, 5 and 6 are all outstanding packets i.e. they are sent but not acknowledged.
- Hence, as soon as the timer expires, the sender will go back and resend all the outstanding packets i.e. packets 3, 4, 5 and 6.
- This is the reason behind the name of this protocol which Go Back N. The sender goes back by N slots and resends all the packets from there as soon as the timer expires.

Send window size :

- Now we are going to discuss, why in GBN protocol the size of send window should be less than 2^m.
- Let m = 2. Therefore the size of the send window will be $2^m - 1 = 3$. With this send window size if all the acknowledgements are lost and the timer expires, then the sender resends all packets.
- As the receiver is expecting packet 3 and not 0, it will successfully identify the resent packet 0 as the duplicate and discard it.
- But if the send window size is $2^m = 4$, and if all the acknowledgement are lost and the timer expires, then the sender will retransmit packet 0.
- But this time, the receiver also is expecting packet 0 to arrive (next cycle). Hence it won't treat the resent packet 0 as the duplicate packet and won't discard it.
- In fact the duplicate packet 0 is accepted as the legitimate packet 0 of the next cycle. This is an error.

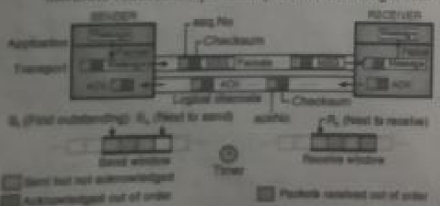
- From this example we conclude that the size of send window in GBN protocol should be **less than 2^m**.

Comparison of GBN with stop and wait :

- The GBN and stop and wait protocols are somewhat similar to each other.
- The stop and wait protocol is actually a GBN protocol with only two sequence numbers (0 and 1) and send window size of 1.
- In stop and wait protocol, the modulo 2 arithmetic is used whereas in GBN protocol, modulo 2^m arithmetic is said to have been used.
- Thus stop and wait protocol is a GBN protocol with m = 1.

12.4.4 Selective Repeat Protocol :

- The process at the receiving end is simplified in the GBN protocol to a great extent. This is because R_n is the only variable which is to be tracked by the receiver and the out of order received packets need not be buffered. They are to be simply discarded.
- But the problem with this protocol is its **inefficiency** if the underlying protocol tends to loose a lot of packets.
- This is because everytime with the loss of a packet the sender has to send all the outstanding packets.
- It is possible that some of these packets may have been received without any error but out of order.
- If the network congestion is already existing, then it will become worse due to these frequently resent packets. The worsened network congestion will result in the loss of more packets which leads to retransmission on of more packets and so on.
- This is called as an **avalanche effect** which may eventually cause total collapse of the network.
- In order to overcome these problems of the GBN protocol, a new protocol has been deviced which is called as the **Selective Repeat Protocol.**
- This new protocol, as the name suggests, resends only **selected packets**, that are actually corrupted or lost. It does not resend all the outstanding packets like the GBN protocol.
- This will reduce the number of resent packets and therefore reduces the possibility of network congestion.



(G-2190) **Fig. 12.4.6 : Outline of selective repeat protocol**

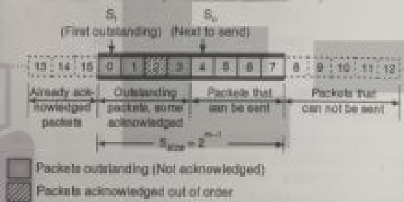
- The principle of selective repeat protocol has been illustrated in Fig. 12.4.6.

Windows :

- In the selective request protocol also there are two windows used : a send window and a receive window.
- However these windows are different from those in the GBN protocol. In this protocol the maximum size of send window is (2^{m-1}). This size is much smaller than that in the GBN protocol. Also the size of receive window is same as that of the send window.

Send and receive windows :

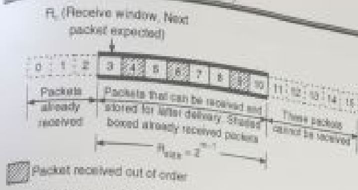
- If m = 4, then the maximum size of the send window is $2^{m-1} = 2^3 = 8$ (It is 15 in the GBN protocol). Fig. 12.4.6(a) shows the structure of the send window.
- Fig. 12.4.6(b) shows the structure of receive window in the selective repeat protocol. Note that it is totally different from that in the GBN protocol.
- The receive window here has the same size as that of the send window (Maximum size = 2^{m-1}).



(G-2191) **Fig. 12.4.6(a) : Send window for selective repeat protocol**

Principle :

- In the selective repeat protocol, the packets equal to the size of the receive window are allowed to arrive out of order.
- The receiver is allowed to keep them until it has a set of consecutive packets which can be delivered to the application layer.
- As the send and receive windows are of the same size, all the packets in the send window can arrive out of order at the receiver and the receiver is allowed to store them until it can deliver them to the application layer.
- However the selective repeat is a reliable protocol. Therefore the receiver is not expected to deliver packets out of order to the application layer.
- The structure of the receiver window for selective repeat protocol is as shown in Fig. 12.4.6(b). It shows that there are packets received out of order. These packets have to wait for the earlier transmitted packets to arrive before all of them are finally delivered to the application layer.



(G-2192) **Fig. 12.4.6(b) : Receive window for selective repeat protocol**

Timer :

- Theoretically in SR protocol a timer is assigned to each outstanding packet in the send window. When a timer expires, only the corresponding packet is resent.
- This is totally different from the GBN protocol which has only one timer for a group of outstanding packets.
- But practically, almost all the transport layer protocols which are based on selective repeat principle use only one timer.

Acknowledgements :

- In GBN protocol, the ackNo is cumulative. It carries the number of the next expected packet to be received. It also confirms that all the previous packets have been received safe and sound.
- But in the SR protocol it is totally different. In SR the ackNo defines the sequence number of only one packet which is received safe and sound. It does not give any feedback about the other packets.

Window sizes :

- The maximum size of send and receive windows in the SR protocol is 2^{m-1} that means $2^m/2$ i.e. half of 2^m.
- If m = 2, all the acknowledgements are lost and if the time out takes place (i.e. timer expires) then sender retransmits packet 0.
- But the receiver window is expecting packet 2 and not packet 0. Hence the receiver will identify packet 0 as the duplicate packet and will discard it. (The sequence number 0 is not in the window).
- Now imagine that the window size is 3, all acknowledgements lost and the timer expires. Now the sender will resend packet 0.
- At this time the receiver is also expecting packet 0 of the next cycle to arrive (0 is the part of the window). Therefore the receiver cannot recognize that packet 0 is a duplicate packet. This is an error.
- That is why in S.R. protocol, the maximum size of the send and receive windows is 2^{m-1} or half of 2^m.

12.4.5 Bidirectional Protocols : Piggybacking :

- Note that in all the protocols discussed so far the data packets flow in only one direction and acknowledgements travel in the opposite direction. Therefore all these four protocols are said to be **unidirectional protocols.**
- However in reality the data packets are travelling in both the directions, client to server and vice versa. The acknowledgements also are travelling in both the directions.
- Thus all the transport layer protocols in real life are bidirectional. We can improve the efficiency of these bidirectional protocols with a technique called **piggybacking.**
- In piggybacking, the data packet going from A to B can also carry acknowledgement for the data packet arrived from B to A.
- Similarly a data packet sent by B to A can carry acknowledgement for the data packet arrived from A to B.

12.5 The Internet Transport Protocols (TCP and UDP) :

- The Internet has two main protocols in the transport layer. One of them is connection oriented and the other one supports the connectionless service.
- TCP (Transmission Control Protocol) is a connection oriented protocol and UDP (User's Data Protocol) is the connectionless protocol.
- UDP is basically just IP with an additional short header.

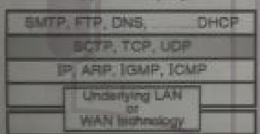
12.6 User Datagram Protocol (UDP) :

- The User Datagram Protocol is a very simple protocol. It adds little to the basic functionality of IP. Like IP, it is an unreliable, connectionless protocol.
- You do not need to establish a connection with a host before exchanging data with it using UDP, and there is no mechanism for ensuring that data sent is received.
- A unit of data sent using UDP is called a Datagram. UDP adds four 16-bit header fields (8 bytes) to whatever data is sent.
- These fields are : a length field, a checksum field, and source and destination port numbers. "Port number", in this context, represents a software port, not a hardware port.
- The concept of port numbers is common to both UDP and TCP. The port numbers identify which protocol module sent (or is to receive) the data.

- Most protocols have standard ports that are generally used for this. For example, the Telnet protocol generally uses port 23. The Simple Mail Transfer Protocol (SMTP) uses port 25. The use of standard port numbers makes it possible for clients to communicate with a server without first having to establish which port to use.
- The port number and the protocol field in the IP header duplicate each other to some extent, though the protocol field is not available to the higher-level protocols. IP uses the protocol field to determine whether data should be passed to the UDP or TCP module.
- UDP or TCP use the port number to determine which application-layer protocol should receive the data.
- Although UDP isn't reliable, it is still a preferred choice for many applications. It is used in real-time applications like Net audio and video where, if data is lost, it's better to do without it than send it again out of sequence. It is also used by protocols like the Simple Network Management Protocol (SNMP).

Relationship with other protocols :

- The relationship of UDP with the other protocols and layers of TCP/IP suite is as shown in Fig. 12.6.1. As shown, UDP is located between IP and application layer. It therefore works as an intermediary between application program and the network layer.



(G-3819) **Fig. 12.6.1 : Relation between UDP and other protocols**

12.6.1 Responsibilities of UDP :

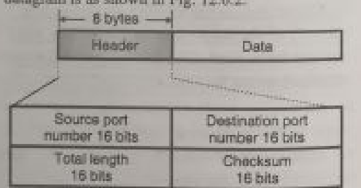
- Being a transport layer protocol, the UDP has the following responsibilities :
  1. To create a process to process communication. UDP uses port numbers to accomplish this.
  2. To provide control mechanisms at the transport layer. UDP does not provide flow control or acknowledgements. It provides error detection. The erroneous packet is discarded.
  3. UDP does not add anything to the services of IP except for providing process to process communication.

12.6.2 Advantages of UDP :

- UDP, despite all its simplicity and powerlessness is still used because it offers the following advantages :
  1. UDP has minimum overheads.
  2. UDP can be easily used if the sending process is not too bothered about reliability.
  3. UDP reduces interaction between sender and receiver.

12.6.3 User Datagram :

- User Datagram Protocol (UDP) provides a connectionless packet service that offers unreliable 'best effort' delivery. This means that the arrival of packets is not guaranteed, nor is the correct sequencing of delivered packets.
- Applications that do not require an acknowledgement of receipt of data, for example, audio or video broadcasting uses UDP.
- UDP is also used by applications that typically transmit small amounts of data at one time, for example, the Simple Network Management Protocol (SNMP).
- UDP provides a mechanism that application programs use to send data to other application programs. UDP provides protocol port numbers used to distinguish between multiple programs executing on a single device.
- That is, in addition to the data sent, each UDP message contains both a destination port number and a source port number. This makes it possible for the UDP software at the destination to deliver the message to the correct application program, and for the application program to send a reply.
- UDP packets are called as **user datagrams.** They have a fixed-size header of 8-bytes. The format of user datagram is as shown in Fig. 12.6.2.



(G-424) **Fig. 12.6.2 : User datagram format**

- The UDP header is divided into the following four 16-bit fields :
  1. Source port
  2. Destination port
  3. Total length
  4. Checksum.

Source Port Number :

- Source port is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

- This is a 16 bit field. That means the port numbers can range from 0 to 65,535.
- If the source host is a client, means if a client is sending a request using UDP, then generally a **ephemeral (temporary)** port number is requested by the process and chosen by the UDP.
- If the source host is a server that means if a server is sending a response message, mostly the **well known port** number is used.

Destination Port Number :

- The destination port number also is a 16 bit number and this port number is used by the process running on the destination host.
- If the destination host is a server that means if a client is sending a request to it, then a **well known port** number is used in most cases.
- However if the destination host is a client than means if a server is sending its response to it, then the chosen port number is generally an **ephemeral port** number.

Length :

- It is also a 16 bit field which is used for defining the total length of the UDP datagram including header as well as data. Due to 16 bit length it can define a total length of the datagram upto 65,535 bytes.
- However practically the total length of a UDP datagram is much smaller than 65,535 bytes. This is because the UDP datagram is to be stored in an IP datagram which itself has a length of 65,535 bytes.
- The **length** field in the UDP datagram is actually not necessary, because this UDP datagram is actually encapsulated in an IP datagram and the IP datagram has its own length field.
- So without using the length field in UDP datagram, we can obtain the length of the UDP datagram as follows :

UDP length = IP length – IP header length

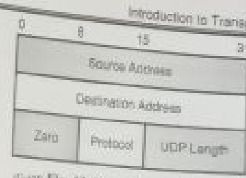
- Note that while delivering the UDP datagram to UDP layer, the IP software drops the IP header.

UDP Checksum :

This is used to verify the integrity (i.e. to detect errors) of the UDP header. The checksum is performed on a "pseudo header" consisting of information obtained from the IP header (source and destination address) as well as the UDP header.

12.6.4 UDP Pseudo Header :

- The purpose of using a pseudo-header is to verify that the UDP packet has reached its correct destination.
- The correct destination consists of a specific machine and a specific protocol port number within that machine.



(G-425) **Fig. 12.6.3 : UDP pseudo header**

- The UDP header itself specifies only the protocol port number. Thus, to verify the destination, UDP on the sending machine computes a checksum that covers the destination IP address as well as the UDP packet.
- At the ultimate destination, UDP software verifies the checksum using the destination IP address obtained from the header of the IP packet that carried the UDP message.
- If the checksum agrees, then it must be true that the packet has reached the intended destination host as well as the correct protocol port within that host.

User Interface :

A user interface should allow the creation of new receive ports, receive operations on the receive ports that return the data octets and an indication of source port and source address, and an operation that allows a datagram to be sent, specifying the data, source and destination ports and addresses to be sent.

IP Interface :

- The UDP module must be able to determine the source and destination Internet addresses and the protocol field from the Internet header.
- One possible UDP/IP interface would return the whole Internet datagram including the entire Internet header in response to a receive operation. Such an interface would also allow the UDP to pass a full Internet datagram complete with header to the IP to send.
- The IP would verify certain fields for consistency and compute the Internet header checksum.

Protocol Application :

The major uses of this protocol are the Internet Name Server, and the Trivial File Transfer.

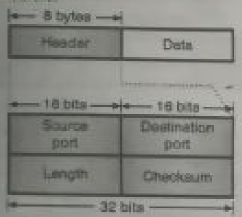
Protocol Number :

This is protocol 17 (21 octal) when used in the Internet Protocol.

Ex. 12.6.1 : The dump of a UDP header in hexadecimal format is as follows :
B C 8 2 0 0 0 D 0 0 2 B 0 0 1 D Obtain the following from it :
1. Source port number
2. Destination port number
3. Total length
4. Length of the data
5. Packet direction.
6. Name of client process.

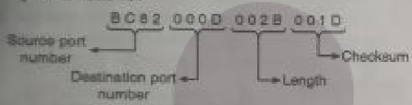
Soln. :

- The standard format of UDP header has been shown in Fig. P. 12.6.1.



(G-2020) **Fig. P. 12.6.1 : UDP header format**

- Therefore we can split the given UDP header in 4 equal parts as follows :



(G-2021)

1. Source port number = $(BC82)_{16}$ ...Ans.
2. Destination port number = $(000D)_{16}$ ...Ans.
3. Total length of UDP packet = $(002B)_{16}$ = $(43)_{10}$ bytes ...Ans.
4. Length of data = Total length – Length of the header = 43 – 8 = 35 bytes ...Ans.
5. Destination port number is $(000D)_{16} = (13)_{10}$

It is a well known port. Hence the direction of UDP packet travel is from client to server.

6. The client process can be obtained from Table 3.6.1 which shows that for well known port number 13, the corresponding client process is "Daytime".

12.7 UDP Services :

In this section we are going to discuss the following important services provided by the UDP :

1. Process to process communication.
2. Connectionless services.
3. Flow control.
4. Error control.
5. Checksum.
6. Congestion control.
7. Encapsulation and decapsulation.
8. Queuing.
9. Multiplexing and demultiplexing.

12.7.1 Process to Process Communication :

- We have already discussed the process to process communication in a general sense, earlier in this chapter.
- UDP also does it with the help of sockets which is a combination of IP address and port numbers. Table 12.7.1 shows different port numbers used by UDP.

Some of these ports can be used by UDP as well as TCP.

Table 12.7.1 : Well known ports used with UDP

Port	Protocol	Description
7	Echo	The received datagram is echoed back to sender.
9	Discard	Any received datagram is discarded.
11	Users	Active users.
13	Daytime	Return the day and the current time.
17	Quote	Return the quote of the day.
19	Chargen	To return a string of characters.
53	Nameserver	Domain Name Service (DNS).
67	BOOT PS	This is the server port to download the bootstrap information.
68	BOOT PC	This is the client port to download bootstrap information.
69	TFTP	Trivial File Transport Protocol.
111	RPC	Remote Procedure Call.
123	NTP	Network Time Protocol.
161	SNMP	Simple Network Management Protocol.
162	SNMP	Simple Network Management Protocol (Trap).

12.7.2 Connectionless Services :

- As UDP is a connectionless, unreliable protocol, each user datagram sent using UDP is an independent datagram.
- Different user datagrams sent by the UDP have absolutely no relationship between them. This is true even for those datagrams which are originating from the same process and being sent to the same destination. The user datagrams **do not have any number.**
- Also the connection establishment and release are not at all required. So each datagram is free to travel any path.
- Only those processes which are sending very short messages can successfully use the UDP.

12.7.3 Flow and Error Control :

- Being a connectionless protocol, UDP is a simple, unreliable protocol. It does not provide any flow control, hence the receiver can overflow with incoming messages.

- UDP does not support any other error control mechanism, except for the checksum.
- There are no acknowledgements sent from destination to sender. Hence the sender does not know if the message has reached, lost or duplicated. If the receiver detects any error using the checksum, then that particular datagram is discarded.

12.7.4 Checksum :

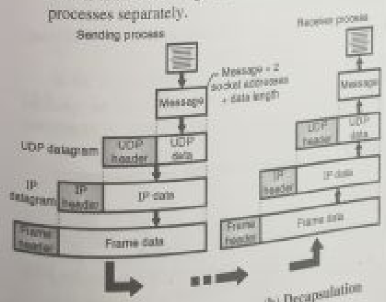
- The calculation of checksum for UDP is different than that for IP. In UDP the checksum is calculated by considering the following three sections :
  1. A pseudoheader
  2. The UDP header.
  3. The data coming from the application layer.
- The checksum in UDP is **optional.** That means the sender can make a decision of not calculating the checksum. If so, then the checksum field is filled with all zeros before sending the UDP packet.
- In case if the calculated checksum is all zeros (when the sender decides to send checksum) then an all 1 checksum is sent.
- This solution works without any problem because, a checksum will never have an all 1 value.

12.7.5 Congestion Control :

- UDP does not provide any congestion control. It assumes that the UDP packets being small, will not create any congestion.
- But this assumption may not always be correct.

12.7.6 Encapsulation and Decapsulation :

- The UDP encapsulates and decapsulates messages in an IP datagram in order to exchange the message between two communicating processes.
- This is as shown in Fig. 12.7.1. We will discuss the two processes separately.



(a) Encapsulation (b) Decapsulation

(G-2022) **Fig. 12.7.1**

Encapsulation :

- Refer Fig. 12.7.1(a). The message produced by a process is to be sent with the help of UDP. The process passes the message and two socket addresses alongwith the length of data to UDP.
- UDP receives this data and adds the UDP header to it as shown. This is called as UDP datagram which is passed to IP with the socket address.
- IP adds its own header to UDP datagram as shown. It enters value 17 into the protocol field. This is an indication that UDP is being used. The IP datagram is then passed on to the data link layer.
- The DLL adds its own header and possibly a trailer to create a frame and sends it to the physical layer.
- Finally the physical layer converts these bits into electrical or optical signals and sends them to the destination machine.

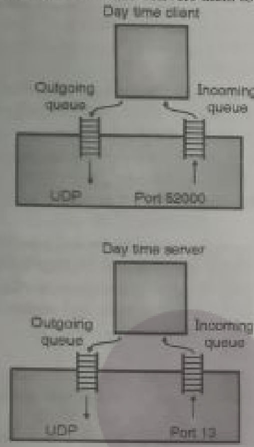
Decapsulation :

- Refer Fig. 12.7.1(b) for understanding of the decapsulation process. The encoded message arrives at the destination physical layer where it decodes the electrical/optical signals into bits and passes them to the DLL.
- The DLL checks the data using header and trailer. The header and trailer are discarded if no errors are found, and the datagram is passed to IP.
- The IP carries out its checking to find the errors and if none are found, the datagram is passed on to UDP, after dropping the IP header.
- The datagram from IP to UDP also contains the sender and receiver IP addresses. This entire user datagram is checked by the UDP with the help of checksum.
- If there is no error detected, then the UDP header is dropped and the application data plus senders socket address are handed over to the process.
- The process can use this senders socket address if it wants to respond to the message received.

12.7.7 Queuing :

- The queues in UDP are related with ports as shown in Fig. 12.7.2.
- A process starts at the client site by requesting a port number from the operating system. In some implementations both incoming and outgoing queues are created in association with each process.
- Every process gets only one port number and hence it can create one outgoing and another incoming queue. The queues function only when the process is running. They are destroyed as soon as the process is terminated.
- The client process uses the source port number mentioned in the request to send message to its outgoing queue.

- UDP removes the queue messages one by one by adding the UDP header and delivers them to IP.

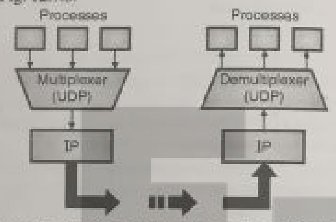


(G-626) **Fig. 12.7.2 : Queues in UDP**

- If the outgoing queue overflows, then operating system tells that client process to wait before sending the next message.
- When the client receives a message, UDP checks if the incoming queue has been created or not. If the queue has been created, then the UDP sends the received datagram to the end of the queue.
- If the queue is not present then UDP will simply discard the user datagram. If the incoming queue overflows, then UDP discards the user datagram and arranges to send the port unavailable message to the server.
- The mechanism to create the server queue is different. The server creates the incoming and outgoing queues using its well known port as soon as it starts running. The queues exist as long as the server is running.
- When a message is received at the server, the UDP checks if the incoming queue has been created or not.
- If the queue is not present, the UDP discards the user datagram. If the queue is present then UDP sends the datagram at the end of the queue.
- If the incoming queue overflows, then UDP drops the user datagram and arranges to send the port unavailable message to the client.
- When the server wants to send a message to client it sends that message to the outgoing queue. These messages are then removed one by one after adding the UDP header. They are delivered to IP.
- If the outgoing queue overflows then the operating system will ask the server to wait before it sends the next message.

12.7.8 Multiplexing and Demultiplexing :

- We have discussed the general principle of multiplexing and demultiplexing in the transport layer.
- Now let us see how to apply the same principle to UDP. Imagine that a host is running a TCP/IP protocol suite and that there is only one UDP and a number of processes which would like to use the services of UDP.
- UDP handles such a situation by using the principle of multiplexing and demultiplexing as shown in Fig. 12.7.3.



(G-2023) **Fig. 12.7.3 : Multiplexing and demultiplexing**

Multiplexing :

- At the sending end, there are several processes that are interested in sending packets. But there is only one transport layer protocol (UDP or TCP). Thus it is a many processes-one transport layer protocol situation.
- Such a many-to-one relationship requires multiplexing.
- The UDP first accepts messages from different processes. These messages are separated from each other by their port numbers. Each process has a unique port number assigned to it.
- Then the UDP adds header and passes the packet to IP as shown in Fig. 12.7.3.

Demultiplexing :

- At the receiving end, the relationship is one as to many. So we need a demultiplexer.
- First the UDP layer receives datagrams from the IP.
- The UDP then checks for errors and drops the header to obtain the messages and delivers them to appropriate process based on the port number.

12.7.9 Comparison of UDP and Generic Simple Protocol :

- In this section we will compare UDP with a simple connectionless transport layer protocol.
- The only difference between the two is that the UDP provides an **optional checksum.**

- If the checksum is added to the UDP packet then at the destination, the receiving UDP can check the packet for any error with the help of the checksum.
- If any error is detected, the receiving UDP will discard that packet, without sending any feedback to the sender.

12.8 UDP Applications :

- Despite being connectionless, unreliable, no flow control, no error control, UDP is still preferred for some applications.
- This is because UDP has some advantages too. An application designer has to sometimes compromise between advantages and drawbacks to get the optimum.
- Here we will discuss some important features of UDP that are useful in designing an application program.

12.9 UDP Features :

12.9.1 Connectionless Service :

- The feature of UDP is that it is a connectionless protocol and that each UDP packet is independent from the other packets, can be considered as an advantage or a disadvantage depending on the requirements of an application.
- In an application, if we want to send only short messages to server and receive short messages from the server. Then the above mentioned feature becomes an advantage.
- The feature of being connectionless is an advantage if request and respond each can fit in one single user datagram.
- The overhead (number packets to be exchanged) required to establish and close a connection is zero in case of UDP. This can be a very important advantage for some applications.
- Similarly the delay involved with the connectionless delivery is very short as compared to that with the connection oriented delivery. Hence the connectionless service provided by UDP is preferred for the applications in which delay is important.

12.9.2 Lack of Error Control :

- UDP is an unreliable protocol which does not provide any error control. Now this is actually a disadvantage but it becomes an advantage for some applications as explained below.
- If TCP is used for reliable service and if a packet is lost, then TCP will resend it. So the receiver transport layer is unable to deliver that part of the message to the application immediately. Due to this an uneven delay is introduced between different parts of the messages which is undesirable for some delay sensitive applications.
- This delay is actually a side effect of the reliable operation of TCP.
- Some applications are not affected by this delay but for some others it is very crucial.

12.9.3 Lack of Congestion Control :

- We know that there is no provision for congestion control in UDP. But this disadvantage can become an advantage for some applications.
- A good side effect of lack of congestion control is that UDP does not create any additional traffic that is created by TCP for congestion control.
- Hence the UDP is preferred fro some congestion prone networks.

12.9.4 Typical Applications of UDP :

1. UDP is suitable for the applications (processes) that have the following requirements :
   (a) A simple response to request is to be made.
   (b) Flow and error controls not essential.
   (c) Bulk data is not to be sent (like FTP).
2. UDP is used for RIP (Routing Information Protocol).
3. UDP is used for management processes such as SNMP.
4. UDP is suitable for the processes having inbuilt flow and error control mechanisms, such as TFTP.
5. UDP is suitable for the multicasting applications.
6. UDP is also used in the real time applications which do not tolerate the uneven delays.

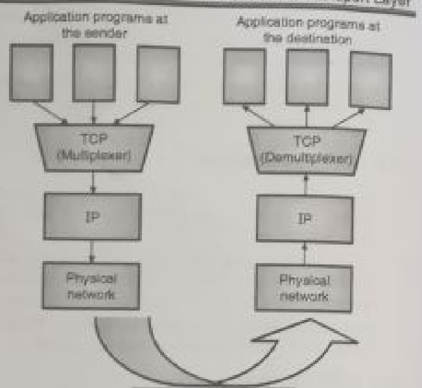
12.10 Transmission Control Protocol (TCP) :

- The TCP provides reliable transmission of data in an IP environment. TCP corresponds to the transport layer (Layer 4) of the OSI reference model.
- Among the services TCP provides are stream data transfer, reliability, efficient flow control, full-duplex operation, and multiplexing.
- TCP is the layer 4 protocol in the TCP/IP suite and it is a very important and complicated protocol. TCP has been revised multiple times in last few decades.
- With stream data transfer, TCP delivers an unstructured stream of bytes identified by sequence numbers.
- This service benefits applications because they do not have to chop data into blocks before handing it off to TCP. Instead, TCP groups bytes into segments and passes them to IP for delivery.
- TCP offers reliability by providing connection-oriented, end-to-end reliable packet delivery through an internetwork.

- It does this by sequencing bytes with a forwarding acknowledgment number that indicates to the destination the next byte the source expects to receive.
- Bytes not acknowledged within a specified time period are retransmitted.
- The reliability mechanism of TCP allows devices to deal with lost, delayed, duplicate, or misread packets. A time-out mechanism allows devices to detect lost packets and request retransmission.
- TCP offers efficient flow control, which means that, when sending acknowledgments back to the source, the receiving TCP process indicates the highest sequence number that it can receive without overflowing its internal buffers.
- TCP supports a full-duplex operation means that TCP processes can both send and receive at the same time.
- Finally, TCP's multiplexing means that numerous simultaneous upper-layer conversations can be multiplexed over a single connection.

12.10.1 Relationship Between TCP and IP :

- The relationship between TCP and IP is very interesting. Each TCP message gets encapsulated or inserted in an IP datagram and then this datagram is sent over the Internet to the destination.
- IP transports this datagram from sender to destination, without bothering about the contents of the TCP message.
- At the final destination the IP hands over the message to the TCP software running on the destination computer.
- IP acts like a postal service and transfers the datagrams from one computer to the other.
- Thus TCP deals with the actual data to be transferred and IP takes care of transfer of that data.
- Many applications such as FTP, Remote login TELNET etc. keep sending data to TCP software on the sending computer.
- The TCP software acts as a multiplexer at the sending computer. It receives data from various applications, multiplexes the data and hands it over to the IP software at the sending end as shown in Fig. 12.10.1.
- IP adds its own header to this TCP packet and creates an IP packet out of it. Then this packet is sent to its destination.
- At the destination exactly opposite process will take place. The IP software hands over the multiplexed data to the TCP software.
- The TCP software at the destination computer then demultiplexes the multiplexed data and gives it to the corresponding applications as shown in Fig. 12.10.1.

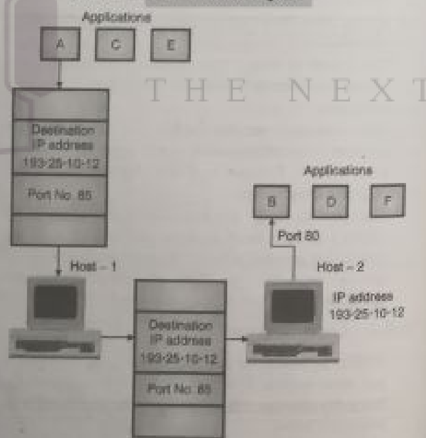


(G-1440) Fig. 12.10.1 : Multiplexing and demultiplexing using TCP

12.10.2 Ports and Sockets :

1. **Ports :**

- Applications running on different hosts communicate with TCP with the help of ports. Every application has been allotted a unique 16 bit number which is known as a **port**.



(G-1437) Fig. 12.10.2 : Use of port numbers

- When an application on one computer wants to communicate using a TCP connection to another application on some other computers these ports prove to be very helpful.

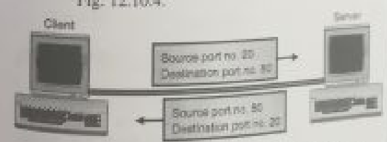
- Let an application A on host 1 wants to communicate with an application B on host 2. So the process takes place as shown in Fig. 12.10.2 and explained below.
- Application A running on computer 1 provides the IP address of computer 2 and the port number corresponding to application B as shown in Fig. 12.10.2.
- Computer 1 communicates with computer 2 using the IP address and computer 2 uses the port number to direct the message to application B.

2. **Sockets :**

- A port is a 16 bit unique number used for identification of a single application.
- But socket address or simply socket would identify the combination of the IP address and the port number concatenated together as shown in Fig. 12.10.3.
- For example if the IP address = 193.25.10.12 and the port number is 85. Then this port of this computer will have the following socket address.

(G-1438) Fig. 12.10.3

- So a pair of sockets is required to identify a TCP connection between two applications on two different hosts. These two socket addresses specify the end points of the connection as shown in Fig. 12.10.4.



(G-1436) Fig. 12.10.4 : Source and destination port numbers

- Generally the server port numbers are known as the **well known ports**. Some of the well known port numbers have already been mentioned for UDP and TCP earlier in this chapter.
- Multiple TCP connections between different applications or same applications on two hosts exist in practice. Here the IP addresses of the two hosts are same but the port numbers are different.
- The communication using port numbers is illustrated in Fig. 12.10.4.

12.11 TCP Services :

Following are some of the services offered by TCP to the processes at the application layer :

1. Stream delivery service
2. Sending and receiving buffers
3. Bytes and segments
4. Full duplex service
5. Connection oriented service
6. Reliable service.
7. Process to process communication.

12.11.1 Process to Process Communication :

- The TCP uses port numbers a transport layer addresses. Table 12.11.1 shows some well known port numbers used by TCP.
- Note that if an application can use both UDP and TCP, the same port number is assigned to this application.

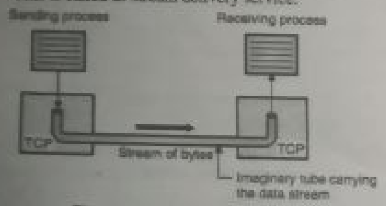
Table 12.11.1 : Well known ports used by TCP

Port	Protocol	Description
7	Echo	Sends received datagram back to sender
9	Discard	Discards any received packet
11	Users	Active users
13	Daytime	Sends the date and the time
17	Quote	Sends a quote of the day
19	Chargen	Sends a string character
20	FTP, Data	File Transfer protocol for data
21	FTP, Control	File Transfer protocol for control
23	TELNET	Terminal 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

12.11.2 Stream Delivery Service :

- TCP is a stream oriented protocol. The sending process delivers data in the form of a stream of bytes and the receiving process receives it in the same manner.
- TCP creates a working environment in such a way that the sending and receiving processes seem to be connected by an imaginary "tube" as shown in Fig. 12.11.1.

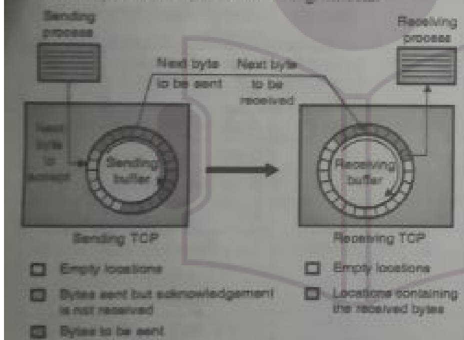
- This is called as stream delivery service.



(G-621) Fig. 12.11.1 : Stream delivery service

12.11.3 Sending and Receiving Buffers :

- The sending and receiving processes may not produce and receive data at the same speed.
- Hence TCP needs buffers for storage of data at both the ends. There are two types of buffers used in each direction :
  1. Sending buffer
  2. Receiving buffer.
- A buffer can be implemented by using a circular array of 1 byte locations as shown in Fig. 12.11.2.

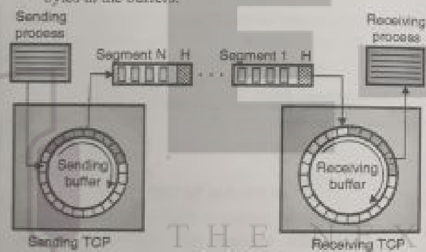


(G-622) Fig. 12.11.2 : Sending and receiving buffers

- Fig. 12.11.2 shows the direction of movement of data. The sending buffer has three types of locations :
  1. Empty locations.
  2. Locations containing the bytes which have been sent but not acknowledged. These bytes are kept in the buffer till an acknowledgement is received.
  3. The locations containing the bytes to be sent by the sending TCP.
- In practice, the TCP may be able to send only a part of data which is to be sent, due to slowness of the receiving process or congestion in the network.
- The buffer at the receiver is divided into two parts :
  1. The part containing empty locations.
  2. The part containing the received bytes which can be consumed by the sending process.

12.11.4 Bytes and Segments :

- Buffering is used to handle the difference between the speed of data transmission and data consumption.
- But only buffering is not enough. We need one more step before sending the data.
- The IP layer, which provides service to TCP, has to send data in the form of packets instead of stream of bytes.
- At the transport layer, TCP groups a number of bytes to form a packet called a segment.
- A header is added to each segment for the purpose of exercising control.
- The segments are then inserted in an IP datagram and transmitted. The entire operation is transparent to the receiving process.
- The segments may be received out of order, lost or corrupted when it reaches the receiving end.
- Fig. 12.11.3 shows the creation of segments from the bytes in the buffers.



(G-623) Fig. 12.11.3

- The segments are not of the same size. Each segment can carry hundreds of bytes.

12.11.5 Full Duplex Service :

- TCP offers full duplex service where the data can flow in both the directions simultaneously.
- Each TCP will then have a sending buffer and receiving buffer. The TCP segments can travel in both the directions, therefore TCP provides a full duplex service.

12.11.6 Connection Oriented Service :

- TCP is a connection oriented protocol. When process – 1 wants to communicate (send and receive) with another process (process – 2), the sequence of operations is as follows :
  1. TCP of process – 1 informs TCP of process – 2 and create a connection between them.
  2. TCP of process – 1 and TCP of process – 2 exchange data in both the directions.
  3. After completing the data exchange, when buffers on both sides are empty, the two TCPs destroy their buffers to terminate the connection.

- The type of connection in TCP is not physical, it is virtual. The TCP segment is encapsulated in an IP datagram and these packets can be transmitted without following the sequence.
- These segments can get lost or corrupted and may have to be resent.
- Each segment may take a different path to reach the destination.

12.11.7 Reliable Service :

TCP is a reliable transport protocol and not unreliable like UDP. Different acknowledgements are used by the receiver to convey sender the status of data.

12.12 Features of TCP :

In order to provide the services mentioned in the previous section, TCP has a number of features as follows :

12.12.1 Numbering System :

- The TCP software keeps track of the segments being transmitted or received. However in the segment header there is no field for a segment number value.
- But there are fields called **sequence number** and the **acknowledgement number.**
- Note that these fields correspond to the byte number and not the segment number.

Byte numbers :

- TCP give numbers to all the data bytes which are transmitted. The numbering is independent of the direction of data travel.
- The numbering does not always start from 0, but it can start with a randomly generated number between 0 and $2^{32} - 1$.

Sequence number :

- After numbering the bytes, the TCP assigns a sequence number to each segment that is being transmitted.
- The sequence number for each segment is same as the number assigned to the first byte present in that segment.

Acknowledgement number :

- The TCP communication is duplex. So both the communicating processes can send and receive data at the same time.
- Each process will give numbers to the bytes with a different starting byte number.
- Each party also uses an ackNo to confirm the reception of bytes.
- The acknowledgement number is cumulative i.e. the receiver takes the number of the last byte received, adds 1 to it and uses this sum as the acknowledgement number.

12.12.2 Flow Control :

- TCP provides flow control (UDP does not). The receiver will control the amount of data to be sent by the sender.
- This will avoid data overflow at the receiver. The TCP uses byte oriented flow control.

12.12.3 Error Control :

- The error control mechanism is inbuilt for TCP. This allows TCP to provide a reliable service.
- The error control mechanism considers a segment as the unit of data for error correction however the byte oriented error control is provided.

12.12.4 Congestion Control :

- TCP takes the congestion in network into account. UDP does not do this.
- The amount of data sent by the sender depends on the following factors :
  1. The receiver's decision (flow control).
  2. The network congestion.

Summary of TCP features :

1. TCP is a process-to-process protocol.
2. TCP uses port numbers.
3. It is a connection oriented protocol (creates a virtual connection).
4. It uses flow and error control mechanisms.
5. TCP is a reliable protocol.

12.13 The TCP Protocol :

- Let us take a general overview of the TCP protocol.
- Every byte on a TCP connection has its own 32-bit sequence number. These numbers are used for both acknowledgement and for window mechanism.

Segments :

The sending and receiving TCP entities exchange data in the form of segments. A segment consists of a fixed 20 byte header (plus and optional part) followed by zero or more data bytes.

Segment size :

The segment size is decided by the TCP software. Two limits restrict the segment size as follows :

1. Each segment including the TCP header, must fit in the 65535 byte IP payload.
2. Each segment must fit in the **MTU (Maximum Transfer Unit)**. Each network has a maximum transfer unit. Practically an MTU which is a few thousand bytes defines the upper limit on the segment size.

Fragmentation :

- If a segment is too large, then it should be broken into small segments. Using fragmentation by a router.

- Each new segment gets a new IP header. So the fragmentation by router will increase the overhead.

Timer :

- The basic protocol used by TCP entities is the sliding window protocol. A sender starts a timer as soon as a sender transmits a segment.
- When the segment is received by the destination, it sends back acknowledgement alongwith data if any. The acknowledgement number is equal to the next sequence number it expects to receive.
- If the timer at the sender goes out before the acknowledgement reaches back, it will **retransmit** that segment again.

Possible problems :

- As the segments can be fragmented, a part of the transmitted segment only may reach the destination with the remaining part lost.
- Segments can arrive out of order.
- Segments can get delayed so much that timer is out and unnecessary retransmission will take place.
- If a retransmitted segment takes a different route than the original segment is fragmented then the fragments of original and retransmitted segments can reach the destination in a sporadic way. So a careful administration is required to achieve reliable byte stream.
- There is a possibility of congestion or broken network along the path.
- TCP should be able to solve these problems in an efficient manner.

12.13.1 TCP Segment :

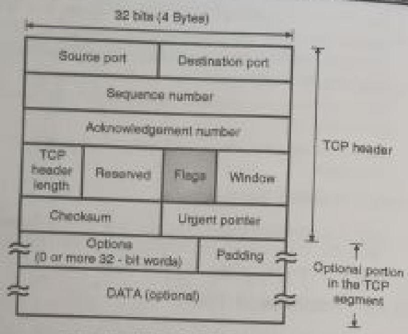
The TCP segment as shown in Fig. 12.13.1 consists of two parts :

1. Header
2. Data

(G-1423)**Fig. 12.13.1 : TCP segment**

12.13.2 The TCP Segment Header :

- Fig. 12.13.2 shows the layout of a TCP segment. Every segment begins with a 20 byte fixed format header.
- The fixed header may be followed by header options.
- After the options, if any, upto 65535 – 20 – 20 = 65495 data bytes may follow. Note that the first 20 bytes correspond to the IP header and the next 20 correspond to the TCP header.
- The TCP segment without data are used for sending the acknowledgements and control messages.



(G-611)**Fig. 12.13.2 : TCP header format**

Source port : A 16-bit number identifying the application the TCP segment originated from within the sending host. The port numbers are divided into three ranges, well-known ports (0 through 1023), registered ports (1024 through 49,151) and private ports (49,152 through 65,535). Port assignments are used by TCP as an interface to the application layer.

Destination port : A 16-bit number identifying the application the TCP segment is destined for on a receiving host. Destination ports use the same port number assignments as those set aside for source ports.

Sequence number : A 32-bit number identifying the current position of the first data byte in the segment within the entire byte stream for the TCP connection. After reaching $2^{32} - 1$, this number will wrap around to 0.

Acknowledgement number :

A 32-bit number identifying the next data byte the sender expects from the receiver. Therefore, the number will be one greater than the most recently received data byte. This field is only used when the ACK control bit is turned on.

Header length or offset :

A 4-bit field that specifies the total TCP header length in 32-bit words (or in ultiples of 4 bytes if you prefer). Without options, a TCP header is always 20 bytes in length. The largest a TCP header may be is 60 bytes. This field is required because the size of the options field(s) cannot be determined in advance. Note that this field is called "data offset" in the official TCP standard, but header length is more commonly used.

Reserved :

A 6-bit field currently unused and reserved for future use.

Control bits or flags :

1. **Urgent pointer (URG) :** If this bit field is set, the receiving TCP should interpret the urgent pointer field.
2. **Acknowledgement (ACK) :** If this bit field is set, the acknowledgement field described earlier is valid.

3. **Push function (PSH) :** If this bit field is set, the receiver should deliver this segment to the receiving application as soon as possible. An example of its use may be to send a Control-BREAK request to an application, which can jump ahead of queued data.
4. **Reset the connection (RST) :** If this bit is present, it signals the receiver that the sender is aborting the connection and all queued data and allocated buffers for the connection can be freely relinquished.
5. **Synchronize (SYN) :** When present, this bit field signifies that sender is attempting to "synchronize" sequence numbers. This bit is used during the initial stages of connection establishment between a sender and receiver.
6. **No more data from sender (FIN) :** If set, this bit field tells the receiver that the sender has reached the end of its byte stream for the current TCP connection.

Window :

A 16-bit integer used by TCP for flow control in the form of a data transmission window size. This number tells the sender how much data the receiver is willing to accept. The maximum value for this field would limit the window size to 65,535 bytes, however a "window scale" option can be used to make use of even larger windows.

Checksum : A TCP sender computes a value based on the contents of the TCP header and data fields. This 16-bit value will be compared with the value the receiver generates using the same computation. If the values match, the receiver can be very confident that the segment arrived intact.

Urgent pointer :

In certain circumstances, it may be necessary for a TCP sender to notify the receiver of urgent data that should be processed by the receiving application as soon as possible. This 16-bit field tells the receiver when the last byte of urgent data in the segment ends.

Options :

In order to provide additional functionality, several optional parameters may be used between a TCP sender and receiver. Depending on the option(s) used, the length of this field will vary in size, but it cannot be larger than 40 bytes due to the size of the header length field (4 bits). The most common option is the Maximum Segment Size (MSS) option. A TCP receiver tells the TCP sender the maximum segment size it is willing to accept through the use of this option. Other options are often used for various flow control and congestion control techniques.

Padding :

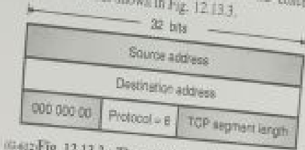
Because options may vary in size, it may be necessary to "pad" the TCP header with zeros so that the segment ends on a 32-bit word boundary as defined by the standard.

Data :

Although not used in some circumstances (e.g. acknowledgement segments with no data in the reverse direction), this variable length field carries the application data from TCP sender to receiver. This field coupled with the TCP header fields constitutes a TCP segment.

12.13.3 Checksum :

- A checksum is provided to ensure extreme reliability. It checksums the header, the data and the conceptual pseudo header shown in Fig. 12.13.3.



(G-612)**Fig. 12.13.3 : The pseudo header included in the TCP checksum**

- When the checksum is being computed, the TCP checksum field is set to zero, and the data field is padded out with an additional zero byte if its length is an odd number.
- Then all the 16 bit words are added in 1's complement and then 1's complement of the sum is taken to get the checksum.
- When a receiver performs the calculation on the entire segment including the checksum field, the result has to be zero.
- The pseudo header contains the 32 bit IP address of the source and destination machines, the protocol number for TCP i.e. 6 and the TCP segment length as shown in Fig. 12.13.3.

12.13.4 Encapsulation :

- The data coming from the application layer is encapsulated in a TCP segment. This TCP segment is then encapsulated in an IP datagram.
- The IP datagram is encapsulated in a frame at the data link layer. The process of encapsulation is shown in Fig. 12.13.4.

(G-2872) **Fig. 12.13.4 : Encapsulation**

Review Questions

Q. 1 Write down duties of transport layer.

Q. 2 What are the services provided by transport layer ?

Q. 3 Write short note on port numbers.

Q. 4 Explain the concept of socket address.

Q. 5 State the difference between IP addresses and port numbers.

Q. 6 Write short note no encapsulation and decapsulation.

Q. 7 Define multiplexing and demultiplexing.

Q. 8 Explain flow control at transport layer.