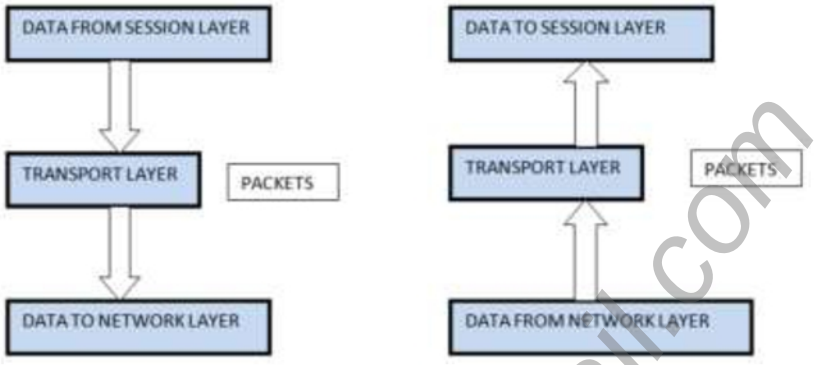
Chapter-4
Transport Layer Protocols

Introduction

- Transport layer resides between the session and network layer of OSI model.
- A transport layer protocols provides for logical communication between application processes running on different host.
- The TCP/IP and UDP protocols resides at this layer.



- The primary functions of this layer are
  i. Provision of connection oriented and connectionless service
  ii. Breaking large message into small segments called packets and reassembling them at destination.
  iii. Establishing and closing connections across the network
  iv. Packet sequencing
  v. Flow control
  vi. Multiplexing and DE multiplexing: End to end data transport
  vii. Congestion control.

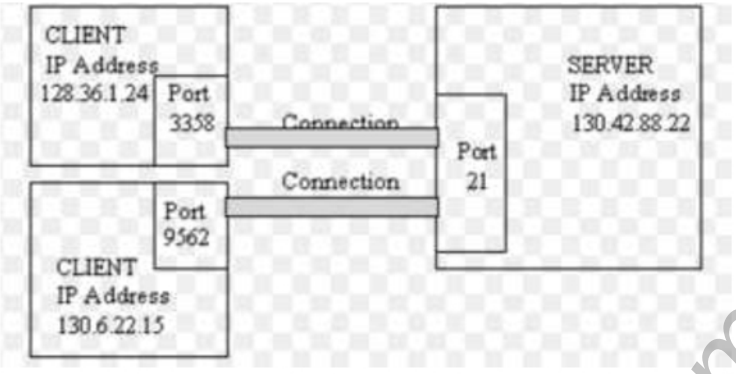
Relationship between transport layer and network layer

- Transport layer resides just above the network layer in protocol stack.
- A transport layer provides logical communication **between processes running on different host** whereas a network layer protocol provides logical communication between **hosts**.
- Transport layer protocols are implemented in the end systems but not in network routers. Network routers only act on the network-layer.
- A transport layer moves messages from application processes to the network edge and vice versa but it doesn't say how the message are moved within the network core.
- A transport layer protocol can offer reliable data transfer service while the underlying network layer protocol are unreliable.

Port number and port addressing overview

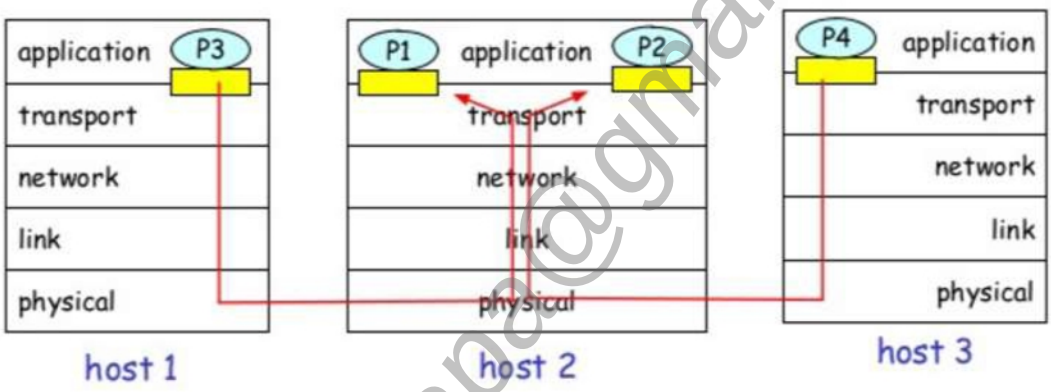
- A port is a type of programming related docking point through which information flows from a program on one computer to another computer in the network.
- So two programs running on different host connects via a port.
- Port are numbered 0 to 65535.
- Ports which are numbered from 0 to 1023 are also called as well-known port and are reserved. Example: HTTP protocol uses port number 80 and similarly FTP uses 20 and 21.
- Ports ranging from 1024 to 49151 are called registered port. They can be used under IANA (Internet Assigned Numbers Authority) registration for specific protocols by software corporations.
- Ports ranging from 49152 to 65535 can be used without any consideration. This range ports are also called as dynamic or private port.
- A port is identified for each address and protocol by a 16-bit number, commonly known as the port number.
- For example, an address may be "protocol: TCP, IP address: 192.168.31.4, port number: 80", which may be written 192.168.31.4:80 when the protocol is known from context.

- Ports become necessary only when computers are executing more than one program at a time and are connected to modern packet-switched networks.



Process to Process Delivery: Multiplexing and De-Multiplexing

- The transport layer provides multiplexing and de-multiplexing service i.e. it extends the host to host delivery service provided by network layer to a process to process delivery service for applications running on different host.
- Each process can have one or more sockets i.e. a door through which data passes from the process to the network.



- At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time. This is a many-to-one relationship and requires multiplexing. The protocol accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, the transport layer passes the packet to the network layer.
- The job of gathering data chunks at the source host from different socket, encapsulating/enveloping each data chunk with header information (needed for de-multiplexing) to create segments and passing the segments to the network layer is called multiplexing.
- At the receiver site, the relationship is one-to-many and requires de-multiplexing. After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.
- The job of delivering the data in a transport layer segment to the correct socket is called de-multiplexing.
- When host receives IP datagram. Each datagram has source IP address, destination IP address. Each segment has source, destination port number. Host uses IP addresses & port numbers to direct segment to appropriate socket
- Types
  i. Connection less Multiplexing, De-multiplexing
  ii. Connection oriented Multiplexing, Demultiplexing

Connectionless multiplexing/ De-multiplexing

- In connectionless Mux/De-mux, the socket used is UDP socket.
- A UDP socket is created as below

```
DatagramSocket mySocket1 = new DatagramSocket(12534);
DatagramSocket mySocket2 = new DatagramSocket(12535);
```

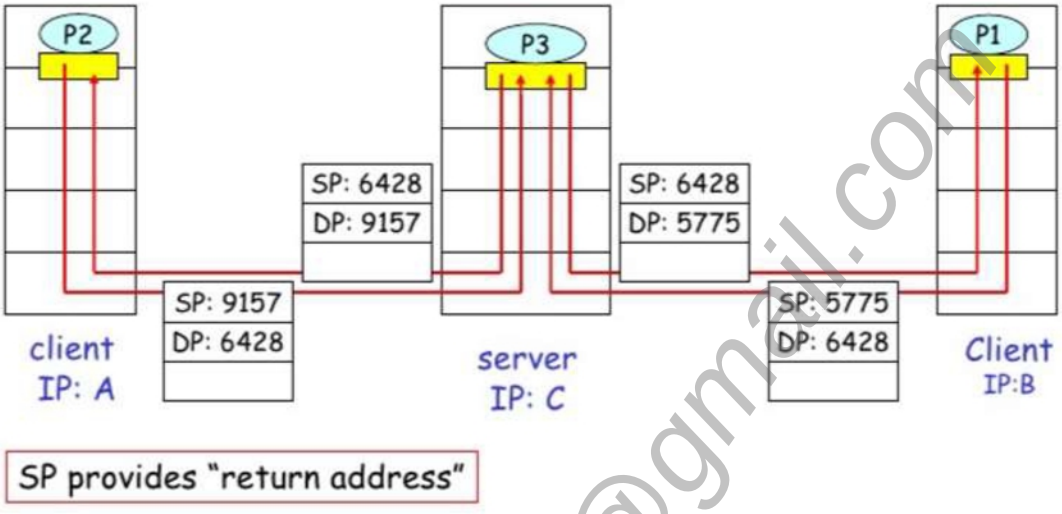
Where, 12354 and 12535 are the port number.

- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:
  i. Checks destination port number in segment
  ii. Directs UDP segment to socket with that port number

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



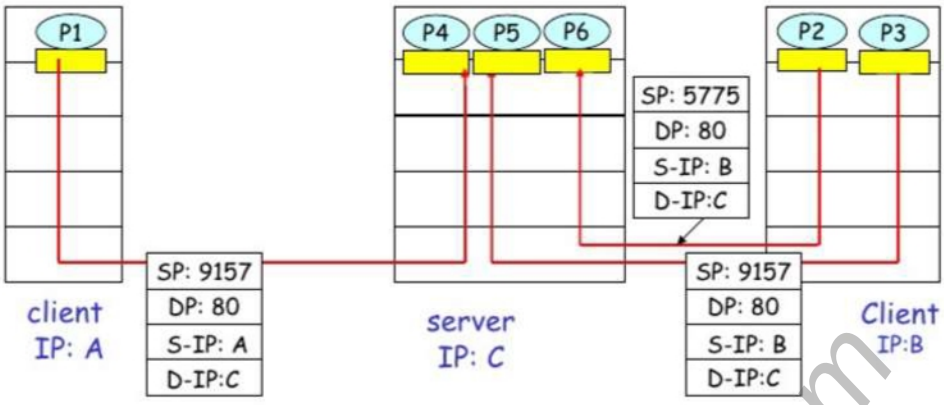
- If a process in host A with UDP port X, wants to send a chunk of data to a UDP port y in host C, then the UDP socket is identified by two tuples destination IP address and Destination port.
- Host A creates a transport layer segments that consists of application data, source port number X and destination port number Y.
- The transport layer then passes the resulting segments to network layer.
- The network layer **encapsulates** the segment in an IP datagram.
- When the segment arrives at receiving host C, the transport layer examines the destination port number and deliver the segment to its socket identified by the port number.
- Since host C might be running multiple processes with their own UDP socket and associated port number, Host C directs i.e. de-multiplex each segment to appropriate socket by examining the destination port number in received segments.

Connection oriented Multiplexing and De-multiplexing

- In connection oriented Mux/De-mux, the socket used is TCP socket.
- A TCP socket is created as

  Socket cs=new Socket("Server_Address", Port_Number)

- TCP socket is identified by four tuples
  i. Source IP address
  ii. Destination IP address
  iii. Source Port number
  iv. Destination Port number



- Receiver host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets. Each socket identified by its own 4-tuple
- So whenever a TCP segment arrives at receiving host(server), it uses all four tuples to direct i.e. de-multiplex the segment to appropriate socket.
- Example: Web servers have different sockets for each connecting client

Connectionless and Connection Oriented Service

A transport layer protocol can either be connectionless or connection-oriented.

i. Connectionless Service
   - In connection less service, when one side of an application wants to send packets to another side of an application, the sending application simply sends the packet without handshaking.
   - Since there in no handshaking procedure prior to the transmission of packets, data can be delivered faster.
   - But there is no acknowledgement either, so a source never knows for sure which packets arrive in destination.
   - This service also has no provision for flow control, congestion control.
   - The connectionless service is provided by transport layer protocol called UDP i.e. user datagram protocol.

ii. Connection Oriented Service
   - When an application uses the connection oriented service, the client and server residing in different end system sends control packet to each other before sending packets with real data.
   - The procedure of sending control packet is also called as handshaking that alert the client and server to be read for transmission of packets.
   - Once handshaking is finished, a connection is established between two end system hence called as connection oriented.
   - The connection oriented service provides other service like reliable data transfer, flow control, congestion control.
   - The connection oriented service is provided b transport layer protocol called TCP i.e. transmission control protocol.

Transmission control protocol(TCP)

i. Introduction

- TCP corresponds to the transport layer of OSI.
- The main goal of TCP protocol is to see that data is reliably received and sent, that the packet data reaches the proper program in the application layer, and that the data reaches the program in the right order.
- TCP is connection oriented service because before an application data to another, the two processes must first handshake with each other.
- TCP connection is always point to point connection between sender and receiver.

- The connection establishment procedure in TCP is also called as 3 way handshake because here first client sends a special TCP segment, then server responds with second special TCP segment and finally client responds again with third segment which consists of application layer data.

ii. Features of TCP

a. **Reliable Data transfer:** A reliable protocols ensures that data sent from one machine to another will eventually be communicated correctly. Reliability is achieved through the use of acknowledgement and retransmission policy. When a source send packet to some destination then destination acknowledge the source. If source receive acknowledgement then it knows that the packet has definitely received but if no acknowledge is received then sender assume that the packet it sent was not received by destination, so it retransmits the packet.

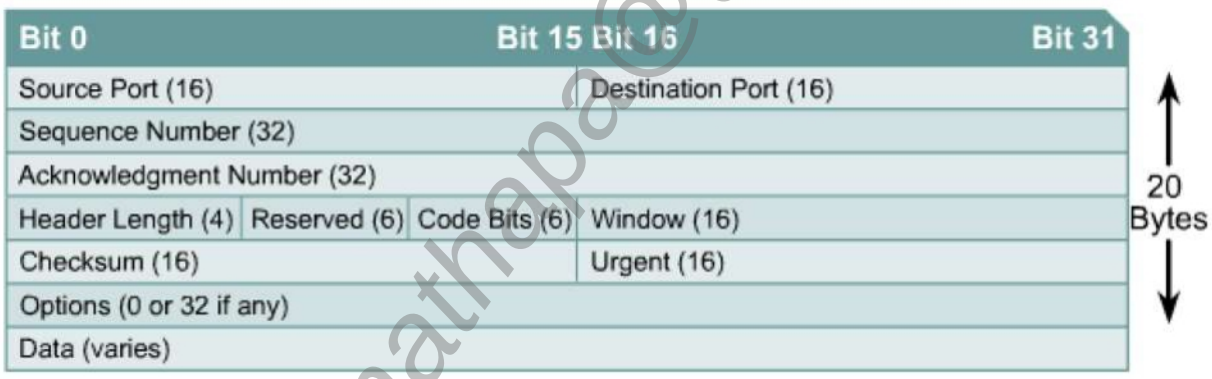
b. **Flow control:** Flow control makes sure that neither side of a connection overwhelms the other side by sending too many packets too fast. The flow control service forces the sending end system to reduce its pumping rate whenever there is such risk.

c. **Congestion control:** Congestion control service helps to prevent the network from entering a state of grid lock. If every pair of communication end system continue to pump as fast as they can, grid lock set in and few packets are delivered to destination because packet loss occur as router buffer overflows. So congestion control forces the end system to reduce the rate at which they send packet into the network during the period of congestion.

d. **Full duplex:** It provides bidirectional mode of communication i.e. both side can send and receive cconcurrently.

iii. <u>TCP header segment</u>

The TCP header segment is shown in the figure below. It consists of header field and data field



- Source port – Number of the port that sends data. Basically used for multiplexing and de-multiplexing.
- Destination port – Number of the port that receives data. Basically used for multiplexing and de-multiplexing.
- Sequence number – the 32 bit sequence number is used to ensure the data arrives in the correct order
- Acknowledgment number – Next expected TCP segment.
- Header length – The four bit header length specify the length of TCP header as TCP header is of variable length due to TCP option field.
- Reserved – Set to zero
- Code bits or flag bit – Control functions, such as setup and termination of a session. It consists of 6 bits each for URG(urgent), ACK(acknowledgement), PSH (push), RST(reset), SYN(synchronization) and FIN
- Window – The 16-bit Window field is used for flow control.
- Checksum – Calculated checksum of the header and data fields for error detection.
- Options: The variable length options field is used when sender and receiver negotiate to maximum segment size for use in high speed network.
- Urgent pointer – indicates a location in the data field where urgent data resides.

User Datagram Protocol(UDP)

i. Introduction

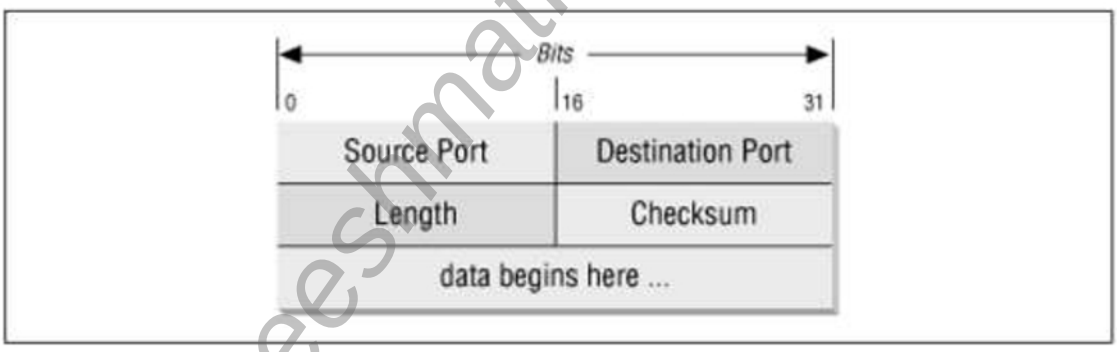
- UDP is an unreliable, connectionless datagram protocol.
- It is called connectionless because there is no handshaking between the sending and receiving entity.
- It is useful for application that do not require TCP's sequencing or flow control.
- It is used for one shot, request reply applications where prompt delivery is important.
- Example of these application includes DNS and transmission of speech, video.
- UDP is also commonly used today with multimedia applications, such as Internet phone, real-time video conferencing, and streaming of stored audio and video.

ii. Features

- **Connectionless paradigm:** A process using UDP doesn't need to establish a connection between the sending and receiving entity. So it is faster than TCP.
- **Robustness:** UDP gives best effort delivery which means that segments can be lost, duplicated, or corrupted during transmission. This is why UDP is suitable for real time data transfer like audio/video transmission and unsuitable for transmission of text/character.
- **Unreliability:** UDP is fast, but unreliable in nature. This means when data bits are transferred via UDP, their reception acknowledgment cannot be attained in an automated fashion, unlike TCP. This feature of UDP prevents it from being used for text or character transmission/reception on computer networks
- **Disarrangement:** Data packets, when sent over a protocol like TCP, arrive in an arranged and assembled manner at the receiver's end. This property is also missing in UDP, since it takes no guarantee of transferring data bits or packets in an arranged manner
- **Reduced Overhead:** If the amount of data being transmitted is small, the overhead of creating connections and ensuring reliable delivery may be greater than the work of re-transmitting the entire data set. So UDP is better option as transport layer protocol

iii. UDP header format

- The format of a UDP header is shown in figure below



i. **Source port** – Number of the port that sends data
ii. **Destination port** – Number of the port that receives data
iii. **Length** – Number of bytes in header and data
iv. **Checksum** – Calculated checksum of the header and data fields
v. **Data** – Upper-layer protocol data

TCP vs UDP

S.no	TCP - Transmission Control Protocol	UDP - User Datagram Protocol
1	connection-oriented, reliable (virtual circuit)	connectionless, unreliable, does not check message delivery
2	Divides outgoing messages into segments	sends "datagrams"
3	reassembles messages at the destination	does not reassemble incoming messages
4	re-sends anything not received	Does-not acknowledge.
5	provides flow control	provides no flow control
6	more overhead than UDP (less efficient)	low overhead - faster than TCP
7	Examples:HTTP, NFS, SMTP	Eg. VOIP,DNS,TFTP

Principle of reliable data transfer protocol

- By the term reliable data transfer, it means that none of the transferred bits are corrupted, lost and all are delivered in the order as they were sent.
- It is responsible of RDT protocol to implement this service.
- TCP is reliable data transfer protocol implemented on the top of network layer protocol i.e. IP.

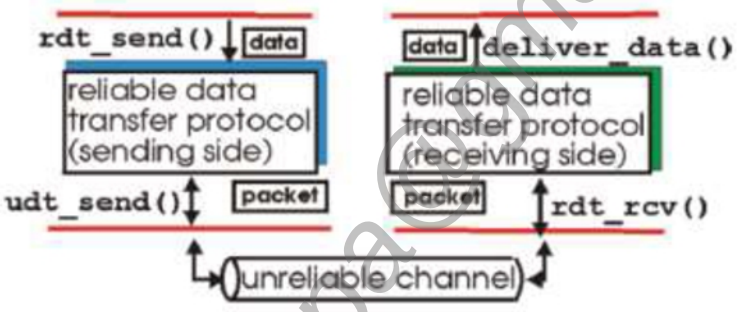


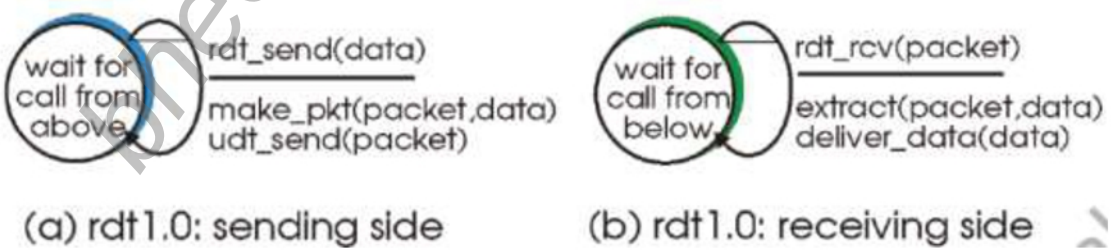
Fig: reliable data transfer service implementation

- In the above figure rdt_send() function is called by application layer for passing the data to the receiver applications layer.
- Udt_send() function is called by reliable data transfer protocol of sending side.
- Rdt_receive() function is called in receiving side, when a packet is received on channel of receiving side.
- Deliver_data() function is called by reliable data transfer protocol of receiving side to deliver data to application layer.

Building a reliable data transfer protocol

i. Reliable data transfer over a perfectly reliable channel: RDT 1.0

- The sender and receiver FSMs each have just only one state
- The finite state machine (FSM) definitions for the rdt1.0 sender and receiver are shown in Figure below.



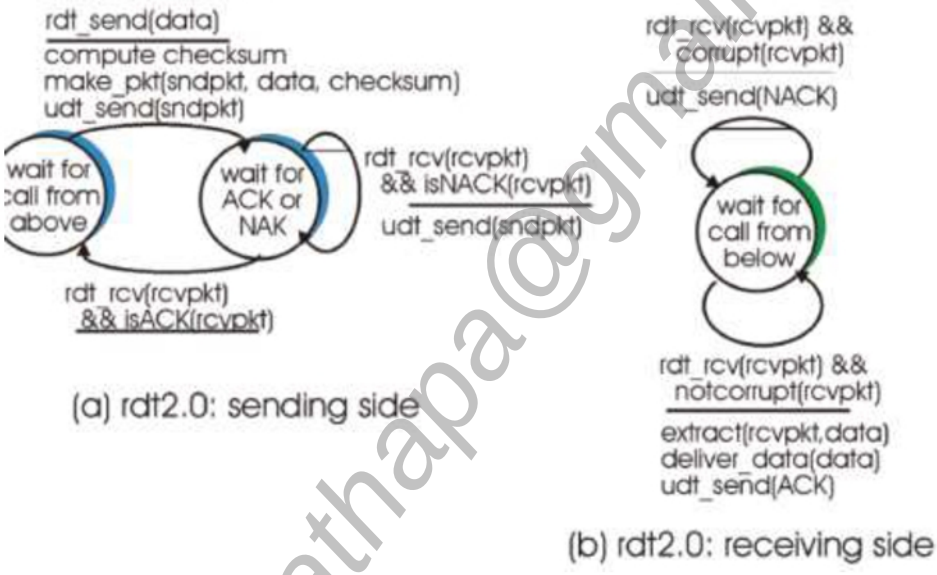
(a) rdt1.0: sending side (b) rdt1.0: receiving side

- The sending side of RDT simply accepts data from the upper-layer via the rdt_send(data)event, puts the data into a packet (via the action make_pkt(packet,data)) and sends the packet into the channel. In practice, the rdt_send(data)event would result from a procedure call (e.g., to rdt_send()) by the upper layer application.
- On the receiving side, RDT receives a packet from the underlying channel via the rdt_rcv(packet) event, removes the data from the packet (via the action extract(packet,data)) and passes the data up to the upper-layer. In practice, the rdt_rcv (packet)event would result from a procedure call (e.g., to rdt_rcv()) from the lower layer protocol.

- In this simple protocol, there is no difference between a unit of data and a packet. Also, all packet flow is from the sender to receiver - with a perfectly reliable channel there is no need for the receiver side to provide any feedback to the sender since nothing can go wrong.

ii. Reliable Data Transfer over a Channel with Bit Errors: rdt2.0/ Stop and wait protocol

- A more realistic model of the underlying channel is one in which bits in a packet may be corrupted or flipped.
- Such bit errors typically occur in the physical components of a network as a packet is transmitted, propagates, or is buffered.
- So this model uses checksum to detect bit error, now the question is how to recover from error.
- For this, this uses message-dictation protocol having positive acknowledgement ACK (i.e. ok i.e. O value) and negative acknowledgement NAK(i.e. please repeat that i.e. 1 value).
- The positive acknowledgement from receiver tells the sender that packets are received correctly. The negative acknowledgement from receiver tells the sender that packets are received with error so sender need to retransmit that packet.
- So, two additional protocol capabilities are required in ARQ protocols to handle the presence of bit errors: error detection and receiver feedback/acknowledgement.
- **In a computer network, reliable data transfer protocols based on such retransmission are known ARQ (Automatic Repeat request) protocols.**



- The send side of rdt2.0 has two states. In one state, the send-side protocol is waiting for data to be passed down from the upper layer. In the other state, the sender protocol is waiting for an ACK or a NAK packet from the receiver. If an ACK packet is received (the notation rdt_rcv(rcvpkt) && isACK(rcvpkt) above corresponds to this event), the sender knows the most recently transmitted packet has been received correctly and thus the protocol returns to the state of waiting for data from the upper layer. If a NAK is received, the protocol retransmits the last packet and waits for an ACK or NAK to be returned by the receiver in response to the retransmitted data packet.
- It is important to note that when the receiver is in the wait-for-ACK-or-NAK state, it cannot get more data from the upper layer; that will only happen after the sender receives an ACK and leaves this state.
- Thus, the sender will not send a new piece of data until it is sure that the receiver has correctly received the current packet. Because of this behavior, protocols such as rdt2.0 are known as **stop-and-wait protocols.**
- The receiver-side FSM for rdt2.0 still has a single state. On packet arrival, the receiver replies with either an ACK or a NAK, depending on whether or not the received packet is corrupted. In above, the notation rdt_rcv(rcvpkt) && corrupt (rcvpkt) corresponds to the event where a packet is received and is found to be in error.

Problem with RDT 2.0

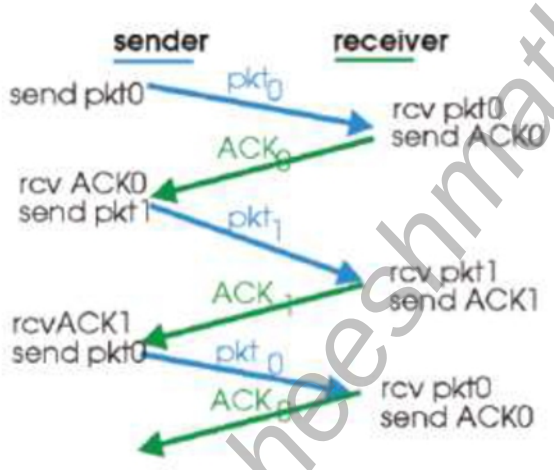
- Protocol rdt2.0 may look as if it works but unfortunately has a fatal flaw for the possibility that the ACK or NAK packet could be corrupted.
- The more difficult question is how the protocol should recover from errors in ACK or NAK packets. The difficulty here is that if an ACK or NAK is corrupted, the sender has no way of knowing whether or not the receiver has correctly received the last piece of transmitted data.
- **RDD 2.1** is able to handle the corrupt ACK/NAK

RDT 2.1

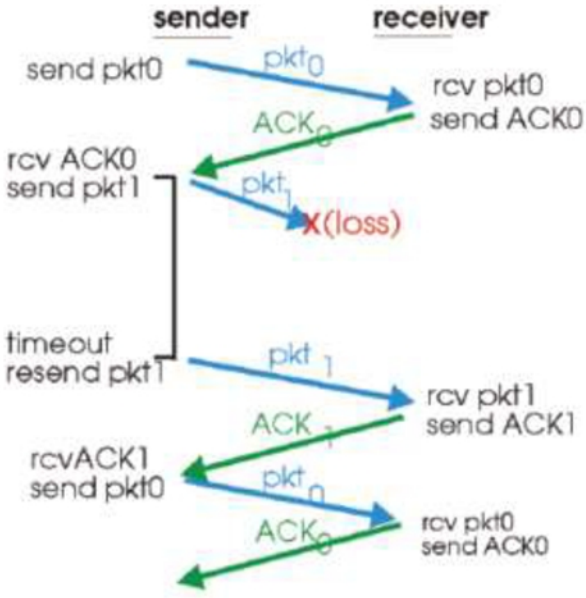
- One of the approach is for the sender to simply resend the current data packet when it receives a garbled ACK or NAK packet. This, however, might introduce **duplicate packets** into the sender-to-receiver channel, hence duplicate packet may arrive in the receiver side. The fundamental difficulty with duplicate packets is that the receiver **doesn't know** whether the ACK or NAK it last sent was received correctly at the sender. Thus, it cannot know a priori whether an arriving packet contains new data or is a retransmission!
- So for this, sender add sequence number 0 and 1 to alternatively to each sending packet
- The receiver then need only check this sequence number to determine whether or not the received packet is a retransmission.
- For this simple case of a stop-and-wait protocol, a 1-bit sequence number will suffice, since it will allow the receiver to know whether the sender is resending the previously transmitted packet (the sequence number of the received packet has the same sequence number as the most recently received packet) or a new packet (the sequence number changes)

iii. Reliable Data Transfer over a Channel with Bit Errors and loss: RDT 3.0/ Stop and wait protocol / 1 bit sliding window protocol

- Underlying channel can corrupt bit as well as may loss packet (example, buffer overflow).
- This protocol addresses, how to detect packet loss and what to do when packet loss occurs.
- For this, it implements a time based retransmission mechanism using timer in all case of data packets loss, ack loss or ack was overdelayed.
- So, if a sender transmits a data packet and either that packet or receiver ack of that packet is lost then the sender simply retransmits the data packet after waiting certain time. The time can be as long as round trip delay between sender and receiver.
- The problem here is that the sender may retransmit the packet even though neither the data packet nor its ack have been lost if the the ack is not received within the time
- So this problem introduces duplicate data packet for receiver.
- For solving this problem, it uses alternating packet sequence number 0 and 1 so that if same sequence number packet arrives it is simply discarded by receiver.
- This protocol is also called as alternating bit protocol
- Figure below show RDT 3.0 implementation in different scenario.

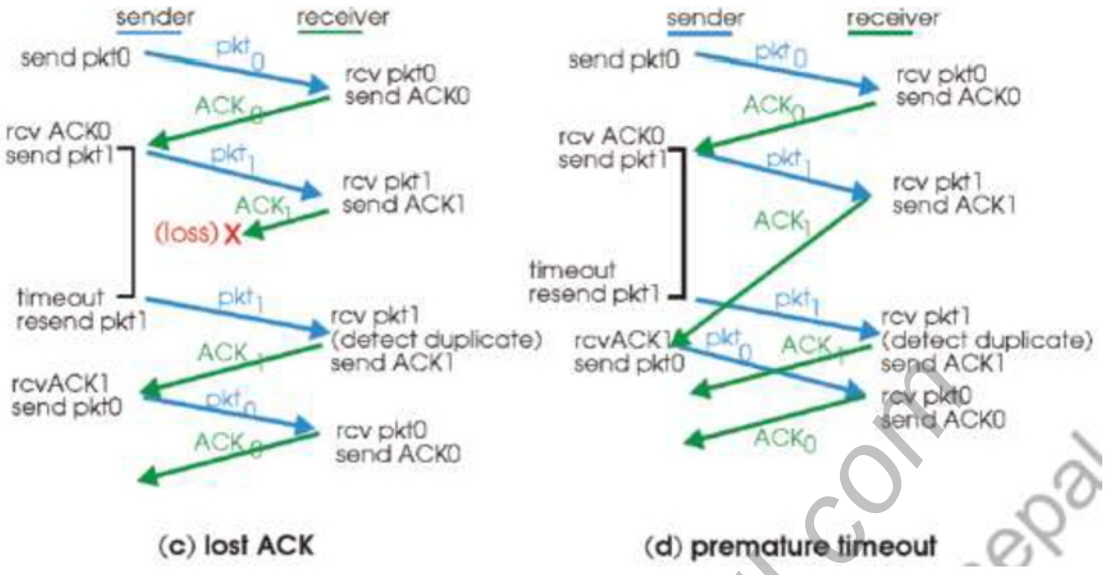


(a) operation with no loss



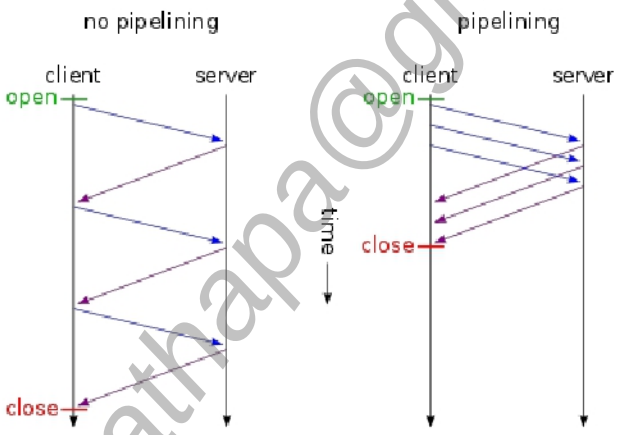
(b) lost packet

1.



Pipelined Reliable data transfer protocol: Based on Sliding window protocol

- Since RDT 3.0 is stop and wait protocol so it is less efficient in today's high speed network.
- Pipelining is the solution in which multiple packets are allowed to send without waiting for acknowledgement.



- For pipelining RDT protocol,
  - i. The range of sequence number must be increased
  - ii. The sender and receiver-sides of the protocols may have to buffer more than one packet. Minimally, the sender will have to buffer packets that have been transmitted, but not yet acknowledged. Buffering of correctly received packets may also be needed at the receiver.
- Two basic approaches toward building reliable pipelined protocol
  1. Go back N ARQ
  2. Selective Repeat ARQ

Go back N(GBN) ARQ

- In this protocol sender is allowed to transmit multiple packets without waiting for an acknowledgement, but is constrained to have no more than some max number N of unacknowledged packets in the pipeline.
- It is based on **sliding window protocol i.e.** the number of unacknowledged packet is determined by window size using the sliding window flow control technique.

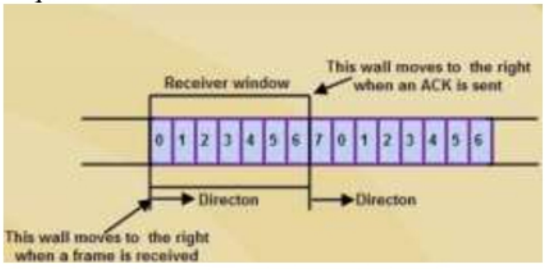


Fig: Sliding window of size= 7

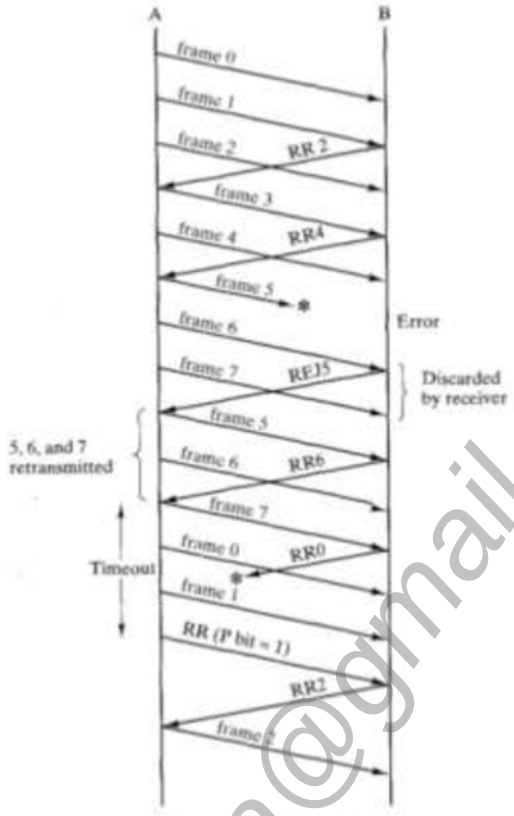


Fig: GBN operation

- Above figure shows the operation of GBN for window size=7 frames, so sender is allowed to send at most 7 frames without any acknowledgement. Also each frames are assigned a sequence number from 0 to 7.
- When sender/transmitter wants to send packet, the sender first checks to see if the window is full, i.e., whether there are N outstanding, unacknowledged frames. If the window is not full, a frame is created and sent, and window boundary variable are appropriately updated
- Here, an acknowledgement for frame with sequence number N will be taken to be a **cumulative acknowledgement**, indicating that all frame with a sequence number up to and including N have been correctly received at the receiver.
- While no error, if a frame with sequence number N is received correctly and is in order, the receiver sends positive acknowledge (RR: Receive Ready) for that frame with next frame sequence number.
- But if the destination station detects an error in a packet
  1. It discards that frame and the entire future incoming frame until the frame in error is correctly received.
  2. It sends negative acknowledgement (Rej: Reject) for that particular frame.
- Thus when the source station, receive a negative acknowledgement, **it must retransmit the frame in error plus all the succeeding frame that were transmitted before.**
- Also, sometime there is a situation that sender has transmitted a frame and waiting for an acknowledgement that might be lost on the way. In this case, the receiver neither send RR nor REJ. Since the sender sets a timer for every frame it sends. If it doesn't receive any acknowledgement from the receiver before the timer expires, it sends the RR frame that includes a bit called P bit which is set to 1. When receiver receive RR frame with P bit set to 1, it acknowledges the sender by RR acknowledge with sequence number i of the frame that it expects. Now upon receiving RR i , the source resends frame starting at i.

Selective Repeat ARQ

- GBN suffer from performance problems. When the window size is large, many frame can be in the pipeline. A single frame error can thus cause GBN to retransmit a large number of frame, many of which may be unnecessary.
- But, with Selective Repeat ARQ technique only that frame from the source station to destination station again retransmits which has received negative acknowledgement.
- So, it is more efficient than go back N ARQ because it minimizes the amount of retransmission.

- On the other hand, the receiver must maintain a buffer large enough to save post-REJ frames until the frame in error is retransmitted, and it must contain logic for reinserting that frame in the proper sequence.
- The transmitter, too, requires more complex logic to be able to send a frame out of sequence. Because of such complications, select-reject ARQ is much less used than go-back- N ARQ.

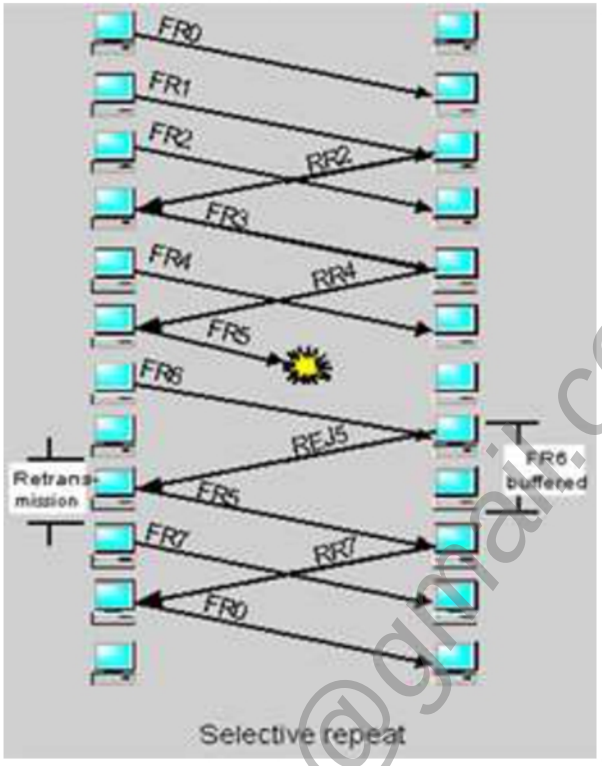


Fig: Selective Repeat Operation (Window size=7, Frame sequence number=0 to 7)

- In the above figure, when the receiver receive frame 5 with error, then it sends negative acknowledgement REJ5 to the sender. The receiver then buffers all the incoming frames (here, frame6) until frame frame5 is received. The sender then resends the out of order frame frame5. Once the receiver receives frame5, then it sends frame5 and buffered frame frame6 in sequence to upper application layer. Also it sends positive acknowledgement RR7 indicating it has successfully received all the frame up to the sequence number 6 and is ready to receive frame frame7.

Flow control

- Sometime sender may systematically want to transmits faster than the receiver can accept frame. Such situation arises when, sender is running on fast computer and the receiver is running on a slow computer.
- Even if there is no transmission error, at a certain point the receiver will simply not be able to handle the frames as they arrive and will start to lose some of them.
- The solution to such problem is flow control.
- Flow control is a technique for assuring that the transmitting entity doesn't over load the receiving entity with data.
- TCP maintain flow control by having the sender maintain a variable called receiver window which gives the sender and idea of how much free buffer is available at the receiver.
- Techniques for flow control
  1. Stop and wait flow control
  2. Sliding window flow control

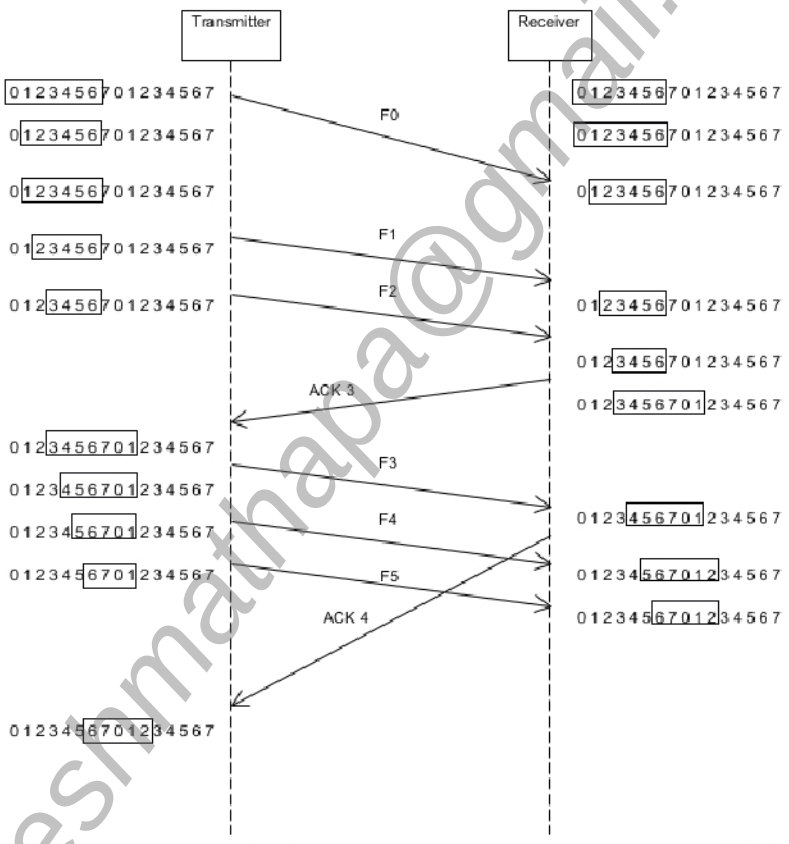
1. Stop and Wait Flow Control

- The simplest form of flow control technique is stop and wait technique.
- Here Whenever a source entity transmits a frame and is received by the destination, the destination indicates its willingness to accept another frame by sending back an acknowledgement to the frame just received.
- The source must wait until it receives acknowledgement before sending next frame.

- The destination can thus stop the flow of data simply by withholding acknowledgement.
- This technique is inefficient in today's high speed network.

2. **Sliding window flow control**

- This technique allows us to send multiple frames without waiting for acknowledgement.
- Let us consider two systems A and B connected through full duplex link.
- Station B allocates buffer space for W frame. Thus B can accept W frames and A is allowed to send W frames without waiting for any acknowledgement.
- To keep track of which frames have been acknowledged, each is labelled with sequence number.
- B acknowledges a frame by sending an acknowledgement that includes the sequence number of next frame.
- This technique can also be used to acknowledge multiple frames.
- This acknowledgement also indicates that B is ready to receive next W frame, beginning with the number specified.
- Station A maintains a list of sequence number that is allowed to send, and station B maintains a list of sequence number that is prepared to receive.
- Each list can be thought as a window of frame. This operation is referred to as sliding-window flow control.



- In the above figure, three-bit sequence number is used so that the frames are sequentially number from 0 to 7. There are 7 frames in window. Each time the frame is sent, the window size shrinks and each time the acknowledgement is received the window size grows.
- Initially A and B have window indication A can transmit 7 frames.
- After transmitting three frames f0, f1 and f2 without acknowledgement, A shrink its window size to four frame.
- When these frames numbered f0, f1 and f2 are received, B then transmit an acknowledgement ack3 indicating it is ready to receive frame number 3 i.e. prepared to receive 7 frames beginning from frame numbered 3.
- When A receives acknowledgement ack3, then now A is permit to transmit next 7 frames beginning from frame3.
- Now B transmits frame f3, f4 and f5.
- B returns ack4 which acknowledges frame f3, which allows transmission of frame f4 through the next instance of f2. But frame f4 and f5 are already transmitted therefore A may only open its window to permit sending 5 frames beginning from frame f6.

Principle of congestion control

- Congestion in network may occur if the load on the network i.e. no of the packet sent to the network, is greater than the capacity of the network i.e. the number of packet a network can handle.
- Congestion may lead to lost packet or long queueing in the router buffer.
- Congestion control refers to the mechanism and technique to control the congestion and keep the load below the capacity.
- Two broad ways toward congestion control are
  1. End to end congestion control
  2. Network assisted congestion control.

1. End to end congestion control
   - In an end-end approach towards congestion control, the network layer provides no explicit support to the transport layer for congestion control purposes.
   - Even the presence of congestion in the network must be inferred by the end systems based only on observed network behavior (e.g., packet loss and delay).
   - TCP must necessarily take this end-end approach towards congestion control, since the IP layer provides no feedback to the end systems regarding network congestion.
   - TCP segment loss (as indicated by a timeout) is taken as an indication of network congestion and TCP decreases its window size accordingly.

2. Network assisted congestion control
   - With network-assisted congestion control, network-layer components (i.e., routers) provide explicit feedback to the sender regarding the congestion state in the network.
   - This feedback may be as simple as a single bit indicating congestion at a link.
   - Choke packet is one of the **technique to congestion control** in which a router observing congestion sends warning directly to the source node directly. The intermediate nodes through which the packet has travelled are not warned.
   - When a router in the Internet is overwhelmed by datagrams, it may discard some of them; but it informs the source. host, using a **source quench** ICMP message. warning message goes directly to the source station; the intermediate routers, and does not take any action. Figure shows the idea of a choke packet.

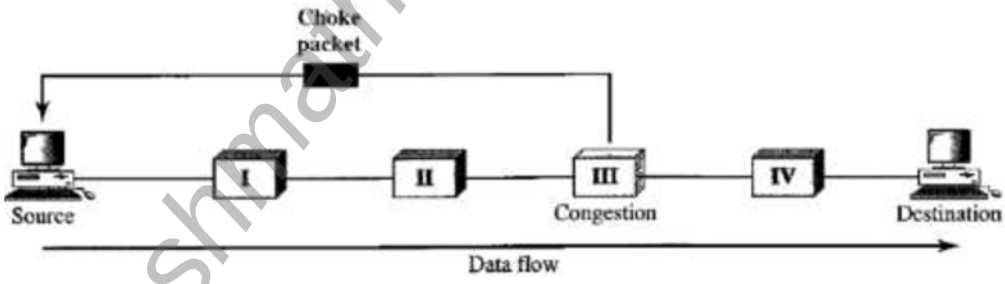


Fig: Choke Packet

Overview of TCP congestion control

- The TCP congestion control mechanism has each side of the connection keep track of variables like **the congestion window(CongWin), ReceiveWindow(RcvWin) and threshold in addition to variables like RcvWin**. The congestion window, determines much traffic a host can send into a connection.
- Specifically, the amount of unacknowledged data that a host can have within a TCP connection may not exceed the minimum of CongWin and RcvWin, i.e.,

  LastByteSent - LastByteAcked <= min{CongWin, RcvWin}.
- The **threshold**, is a variable that effects how CongWin grows as described below.
- let us assume that the TCP receive buffer is so large that the receive window constraint can be ignored. In this case, the amount of unacknowledged data that can be send is solely determined by CongWin. Further let's assume that a sender has a very **large amount of data** to send to a receiver.

- Once a TCP connection is established between the two end systems, TCP chunks the message in size of MSS (Maximum segment size), encapsulates each chunk within a TCP segment, and passes the segments to the network layer for transmission across the network.
- So, initially, the congestion window is equal to one MSS. TCP sends the first segment into the network and waits for an acknowledgement. If this segment is acknowledged **before its timer times out**, the sender increases the congestion window by **one MSS** and sends out **two maximum-size segments**. If these segments are acknowledged before their timeouts, the sender increases the congestion window by one MSS for each **of the acknowledged segments, giving a congestion window of four MSS, and sends out four** maximum-sized segments. This procedure continues as long as the congestion window is below the threshold and the acknowledgements arrive before their corresponding timeouts. This phase of the algorithm is called **slow start** because it begins with a small congestion window equal to one MSS.
- The **slow start** phase ends when the window size exceed the value of threshold. Once the congestion window is larger than the current value of threshold, the congestion window grows **linearly i.e CongWind in increased by 1,** rather than exponentially. This phase of the algorithm is called **congestion avoidance.**
- The congestion avoidance phase continues as long as the acknowledgements arrive before their corresponding timeouts. But the window size, and hence the rate at which the TCP sender can send, cannot increase forever. Eventually, the TCP rate will be such that one of the links along the path becomes saturated, and which point loss (and a resulting timeout at the sender) will occur. When a timeout occurs, the value of **threshold** is set to half the value of the current congestion window, and the congestion window is reset to **one MSS**. The sender then again grows the congestion window exponentially fast using the slow start procedure until the congestion window hits the threshold.

In summary:

- When the congestion window is below the threshold, the congestion window grows exponentially.
- When the congestion window is above the threshold, the congestion window grows linearly.
- Whenever there is a timeout, the threshold is set to one half of the current congestion window and the congestion window is then set to one.