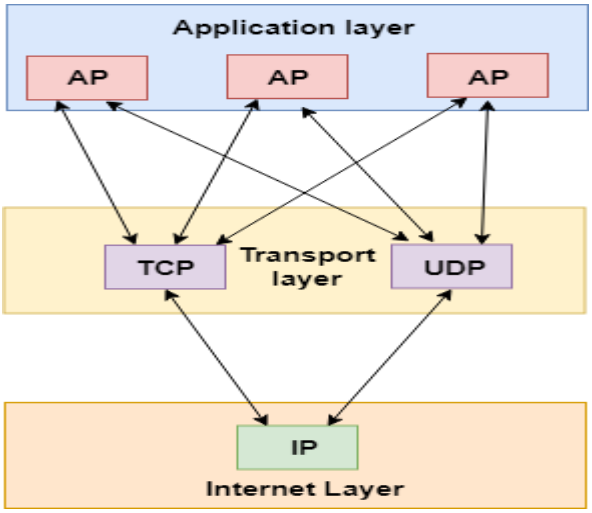
UNIT 4

1. INTRODUCTION / TRANSPORT-LAYER SERVICES

Transport Layer

- The transport layer is a 4th layer from the top.
- The main role of the transport layer is to provide the communication services directly to the application processes running on different hosts.
- The transport layer provides a logical communication between application processes running on different hosts. Although the application processes on different hosts are not physically connected, application processes use the logical communication provided by the transport layer to send the messages to each other.
- The transport layer protocols are implemented in the end systems but not in the network routers.
- A computer network provides more than one protocol to the network applications. For example, TCP and UDP are two transport layer protocols that provide a different set of services to the network layer.
- All transport layer protocols provide multiplexing/demultiplexing service. It also provides other services such as reliable data transfer, bandwidth guarantees, and delay guarantees.
- Each of the applications in the application layer has the ability to send a message by using TCP or UDP. The application communicates by using either of these two protocols. Both TCP and UDP will then communicate with the internet protocol in the internet layer. The applications can read and write to the transport layer. Therefore, we can say that communication is a two-way process.

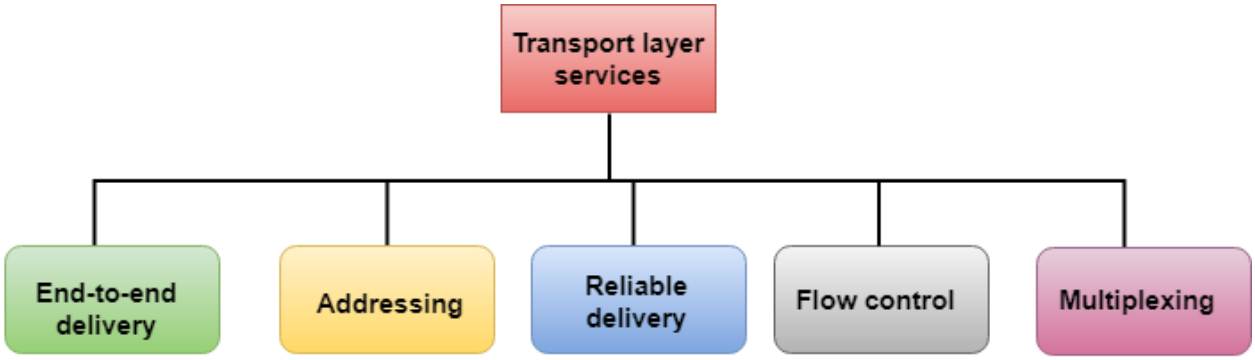


Transport Layer Services

The services provided by the transport layer are similar to those of the data link layer. The data link layer provides the services within a single network while the transport layer provides the services across an internetwork made up of many networks. The data link layer controls the physical layer while the transport layer controls all the lower layers.

The services provided by the transport layer protocols can be divided into five categories:

- End-to-end delivery
- Addressing
- Reliable delivery
- Flow control
- Multiplexing



End-to-end delivery:

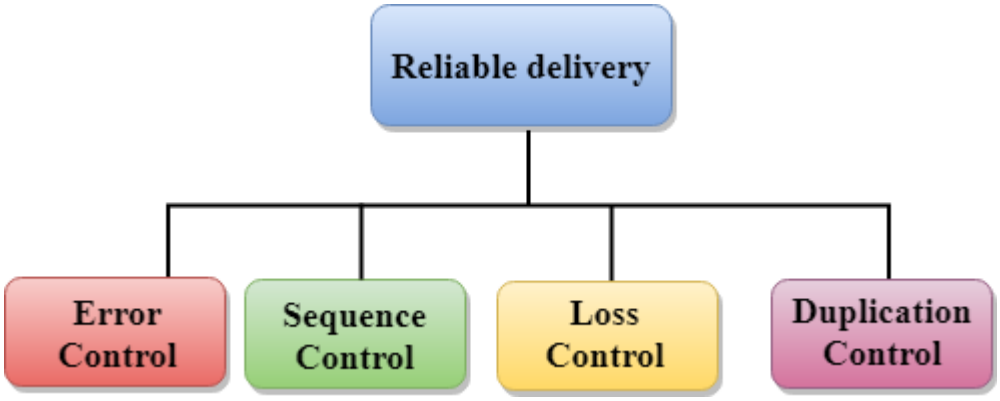
The transport layer transmits the entire message to the destination. Therefore, it ensures the end-to-end delivery of an entire message from a source to the destination.

Reliable delivery:

The transport layer provides reliability services by retransmitting the lost and damaged packets.

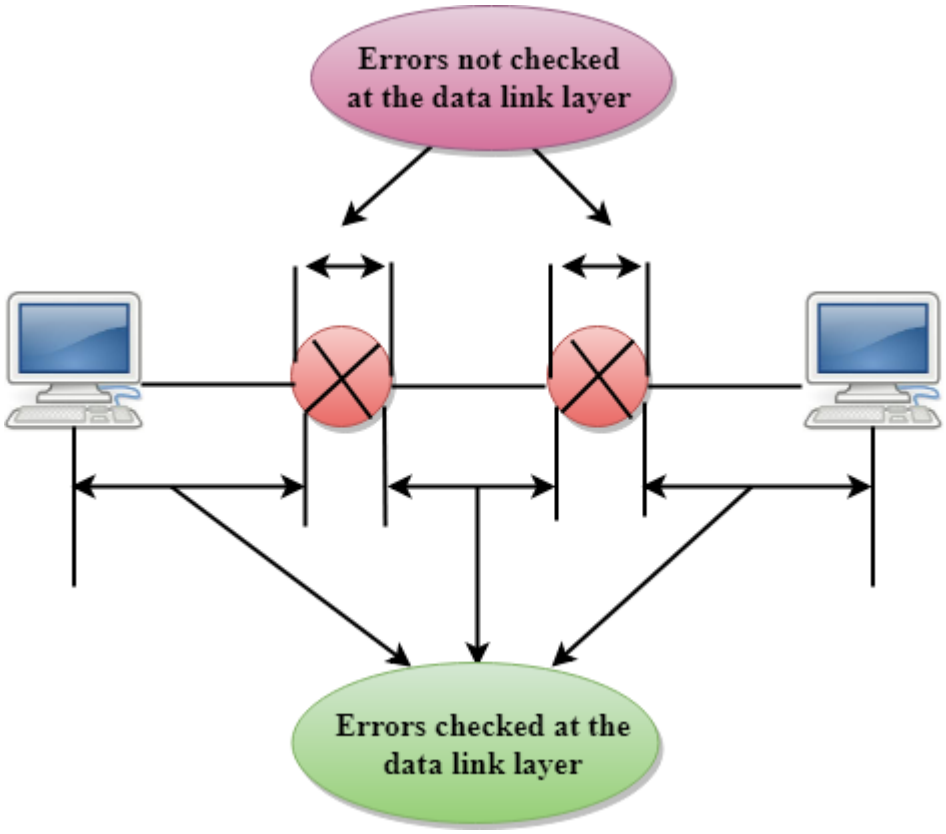
The reliable delivery has four aspects:

- Error control
- Sequence control
- Loss control
- Duplication control



Error Control

- The primary role of reliability is **Error Control**. In reality, no transmission will be 100 percent error-free delivery. Therefore, transport layer protocols are designed to provide error-free transmission.
- The data link layer also provides the error handling mechanism, but it ensures only node-to-node error-free delivery. However, node-to-node reliability does not ensure the end-to-end reliability.
- The data link layer checks for the error between each network. If an error is introduced inside one of the routers, then this error will not be caught by the data link layer. It only detects those errors that have been introduced between the beginning and end of the link. Therefore, the transport layer performs the checking for the errors end-to-end to ensure that the packet has arrived correctly.



Sequence Control

- The second aspect of the reliability is sequence control which is implemented at the transport layer.

- On the sending end, the transport layer is responsible for ensuring that the packets received from the upper layers can be used by the lower layers. On the receiving end, it ensures that the various segments of a transmission can be correctly reassembled.

Loss Control

- Loss Control is a third aspect of reliability. The transport layer ensures that all the fragments of a transmission arrive at the destination, not some of them. On the sending end, all the fragments of transmission are given sequence numbers by a transport layer. These sequence numbers allow the receiver?s transport layer to identify the missing segment.

Duplication Control

- Duplication Control is the fourth aspect of reliability. The transport layer guarantees that no duplicate data arrive at the destination. Sequence numbers are used to identify the lost packets; similarly, it allows the receiver to identify and discard duplicate segments.

Flow Control

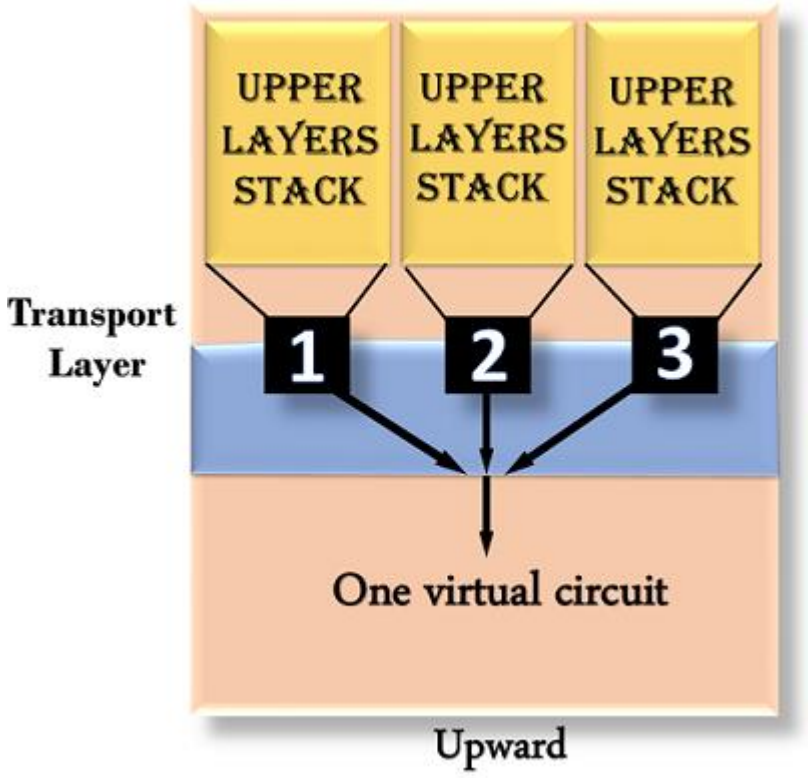
Flow control is used to prevent the sender from overwhelming the receiver. If the receiver is overloaded with too much data, then the receiver discards the packets and asking for the retransmission of packets. This increases network congestion and thus, reducing the system performance. The transport layer is responsible for flow control. It uses the sliding window protocol that makes the data transmission more efficient as well as it controls the flow of data so that the receiver does not become overwhelmed. Sliding window protocol is byte oriented rather than frame oriented.

Multiplexing

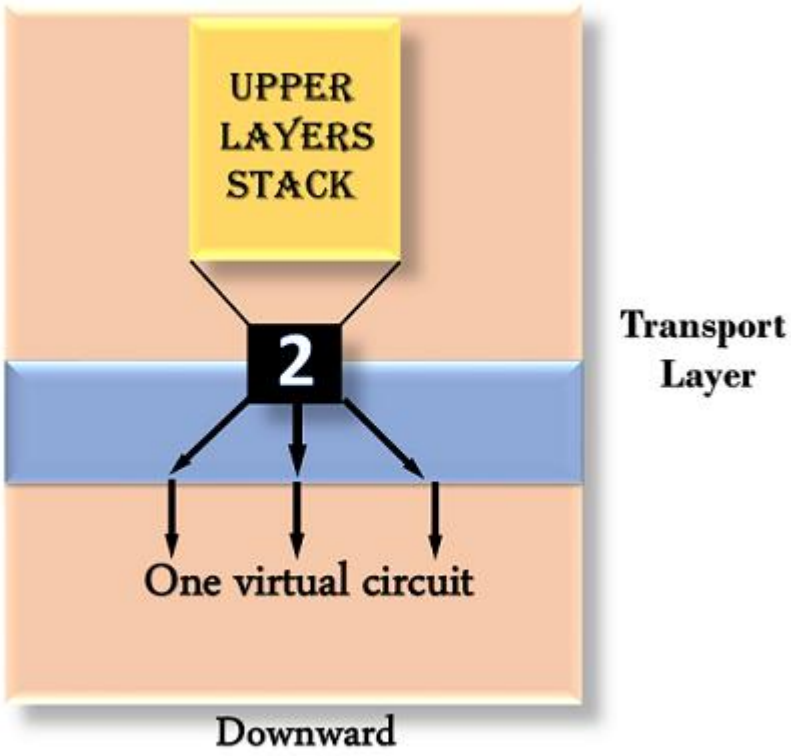
The transport layer uses the multiplexing to improve transmission efficiency.

Multiplexing can occur in two ways:

Upward multiplexing: Upward multiplexing means multiple transport layer connections use the same network connection. To make more cost-effective, the transport layer sends several transmissions bound for the same destination along the same path; this is achieved through upward multiplexing.

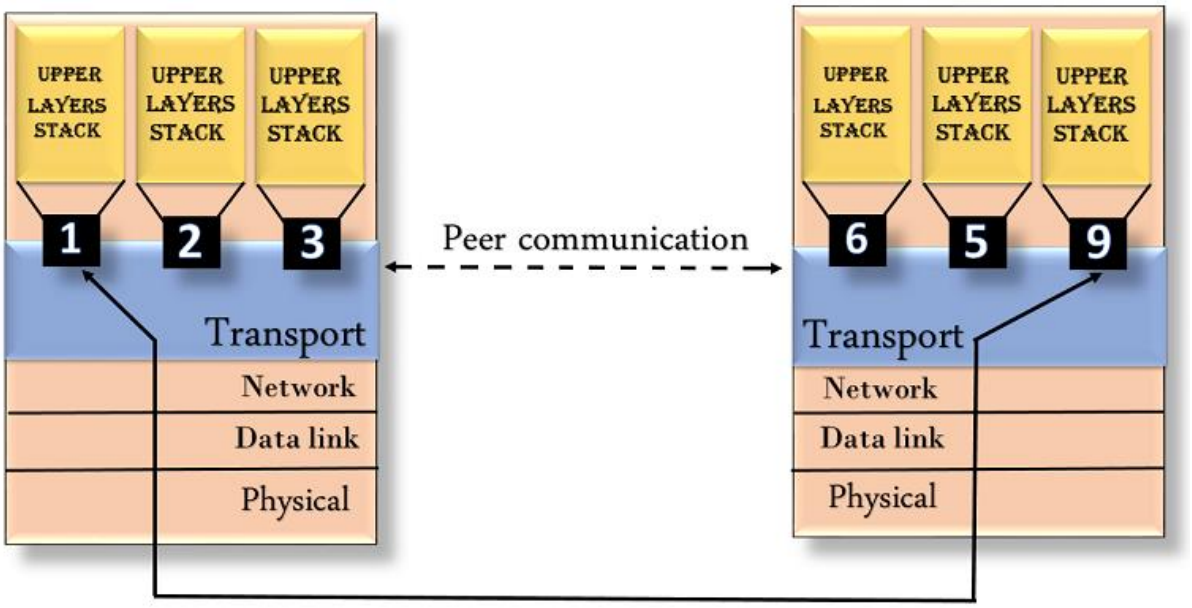


- **Downward multiplexing:** Downward multiplexing means one transport layer connection uses the multiple network connections. Downward multiplexing allows the transport layer to split a connection among several paths to improve the throughput. This type of multiplexing is used when networks have a low or slow capacity.



Addressing

- According to the layered model, the transport layer interacts with the functions of the session layer. Many protocols combine session, presentation, and application layer protocols into a single layer known as the application layer. In these cases, delivery to the session layer means the delivery to the application layer. Data generated by an application on one machine must be transmitted to the correct application on another machine. In this case, addressing is provided by the transport layer.
- The transport layer provides the user address which is specified as a station or port. The port variable represents a particular TS user of a specified station known as a Transport Service access point (TSAP). Each station has only one transport entity.
- The transport layer protocols need to know which upper-layer protocols are communicating.



2. CONNECTIONLESS AND CONNECTION-ORIENTED PROTOCOLS

A number of characteristics can be used to describe communications protocols. The most important is the distinction between *connection-oriented transport* services (COTS) and *connectionless transport* services (CLTS).

Connection-Oriented Protocols [TCP]

TCP is an example of a connection-oriented protocol. It requires a logical connection to be established between the two processes before data is exchanged. The connection must be maintained during the entire time that communication is taking place, then released afterwards. The process is much like a telephone call, where a virtual circuit is established--the caller must know the person's telephone number and the phone must be answered--before the message can be delivered.

TCP/IP is also a connection-oriented transport with orderly release. With orderly release, any data remaining in the buffer is sent before the connection is terminated. The release is accomplished in a three-way handshake between client and server processes. The connection-oriented protocols in the OSI protocol suite, on the other hand, do not support orderly release. Applications perform any handshake necessary for ensuring orderly release.

Examples of services that use connection-oriented transport services are telnet, rlogin, and ftp.

Connectionless Protocols [UDP]

Connectionless protocols, in contrast, allow data to be exchanged without setting up a link between processes. Each unit of data, with all the necessary information to route it to the intended destination, is transferred independent of other data packets and can travel over different paths to reach the final destination. Some data packets might be lost in transmission or might arrive out of sequence to other data packets.

UDP is a connectionless protocol. It is known as a datagram protocol because it is analogous to sending a letter where you don't acknowledge receipt.

Examples of applications that use connectionless transport services are broadcasting and tftp. Early implementations of NFS used UDP, whereas newer implementations prefer to use TCP.

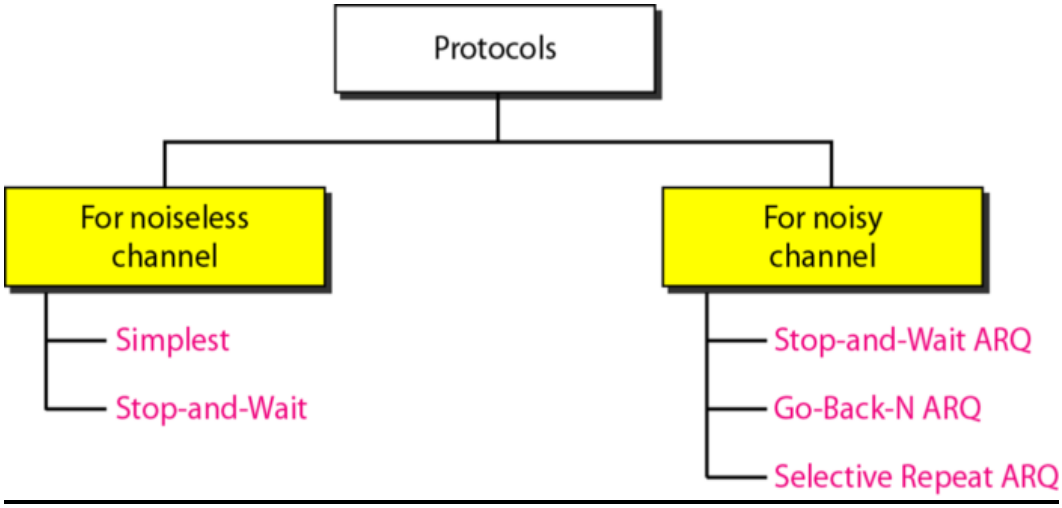
Differences b/w TCP & UDP

Basis for Comparison	TCP	UDP
Definition	TCP establishes a virtual circuit before transmitting the data.	UDP transmits the data directly to the destination computer without verifying whether the receiver is ready to receive or not.
Connection Type	It is a Connection-Oriented protocol	It is a Connectionless protocol

Speed	slow	high
Reliability	It is a reliable protocol.	It is an unreliable protocol.
Header size	20 bytes	8 bytes
acknowledgement	It waits for the acknowledgement of data and has the ability to resend the lost packets.	It neither takes the acknowledgement, nor it retransmits the damaged frame.

3. Transport layer protocols

- Simple Protocol
- Stop and Wait Protocol
- Go Back N Protocol
- Selective Repeat Protocol
- Bidirectional Protocols: Piggybacking



Noiseless Channels

- Noiseless Channel means that there will not any kind of disturbance in the path when data is carried forward from sender to receiver.
- Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted.

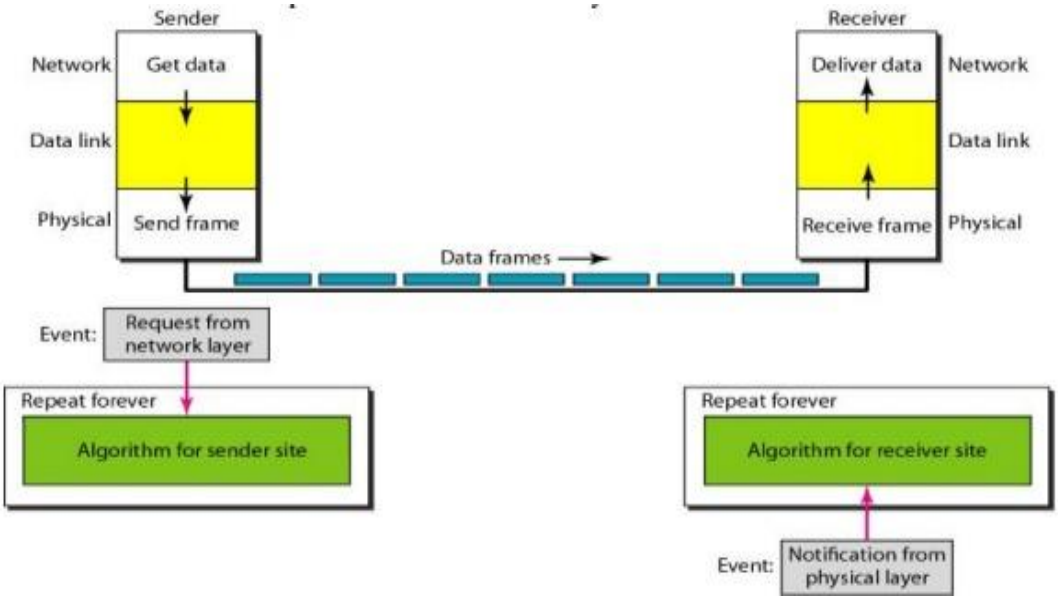
Two protocols for noiseless channels:

1. **Simplest Protocol**
2. **Stop-and-Wait Protocol**

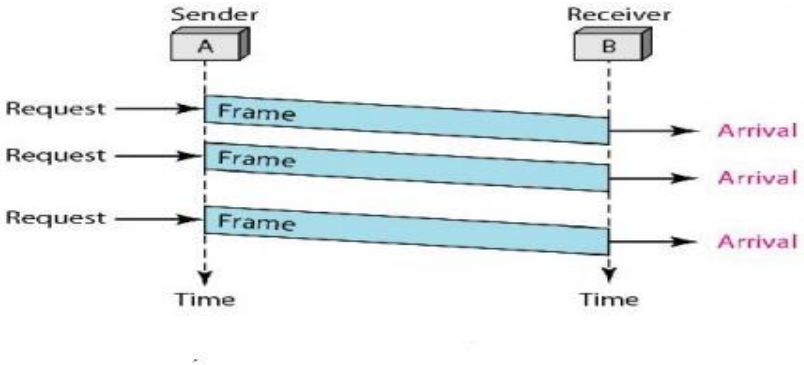
1. Simplest Protocol

- It has no flow or error control. It is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver. The data link layer of the receiver immediately removes the header from the frame and hands the data packet to its network layer, which can also accept the packet immediately.

The design of the simplest protocol with no flow or error control



Flow Diagram for Simplest Protocol

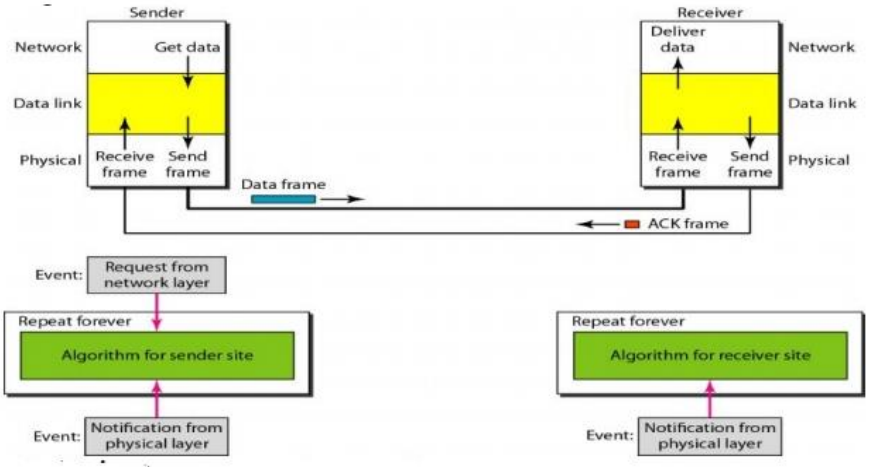


2. Stop-and-Wait Protocol

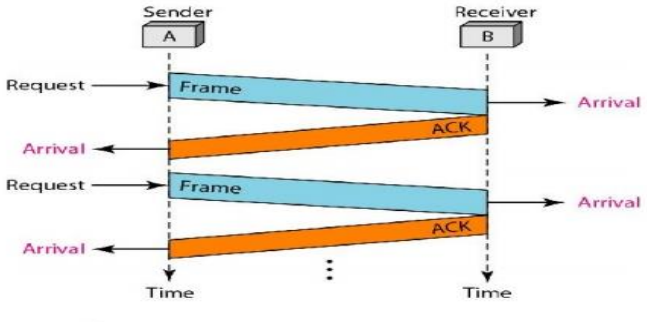
- If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use.

- In Stop-and-Wait Protocol the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.

Design of Stop-and-Wait Protocol



Flow Diagram for Stop-and-Wait Protocol



Noisy Channels

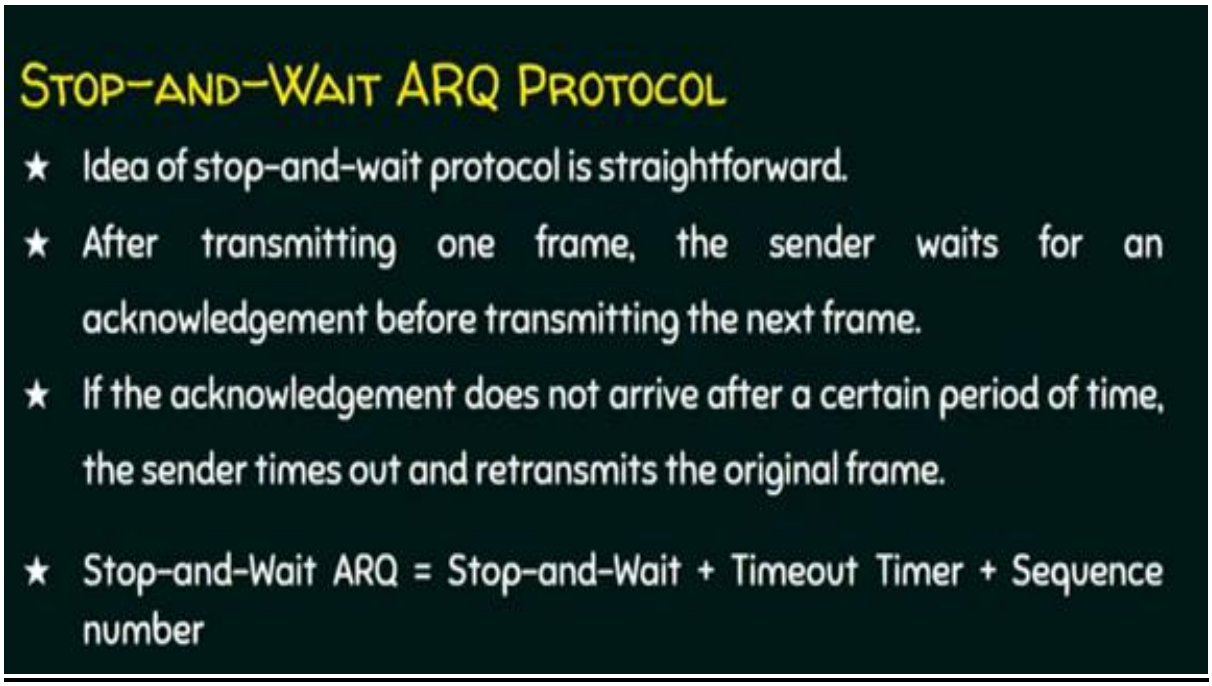
Noisy Channel means that there will be lot of disturbance in the path when data is carried forward from sender to receiver.

Protocols for noisy channels:

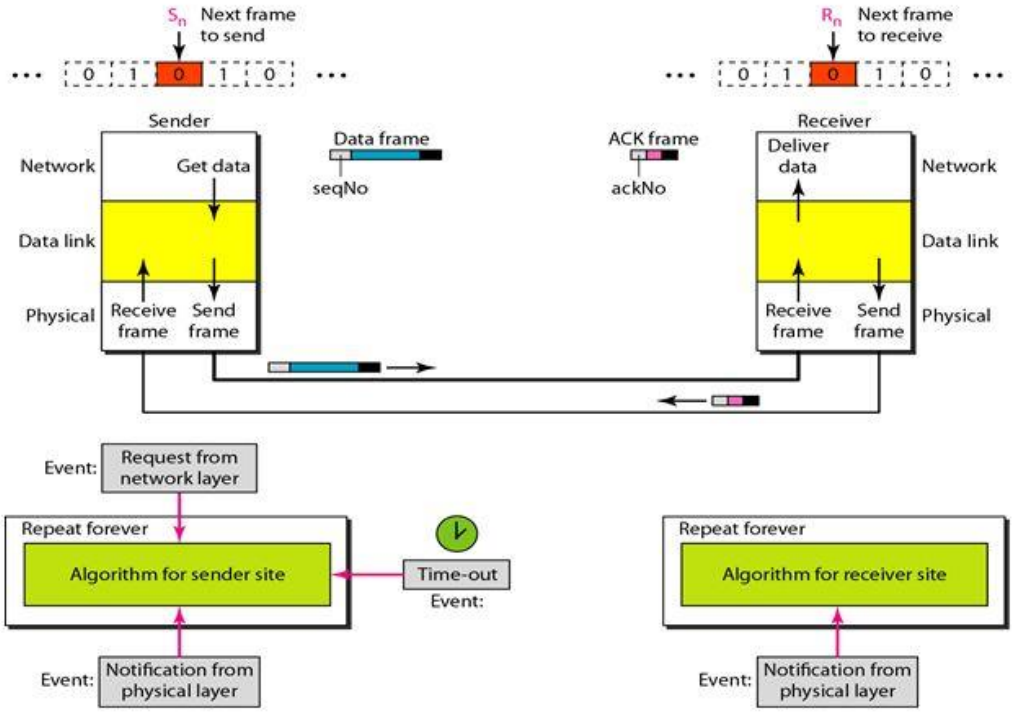
1. **Stop-and-Wait ARQ**
2. **Go-Back-N ARQ**
3. **Selective-Repeat ARQ**

Protocols → **Sliding Window**

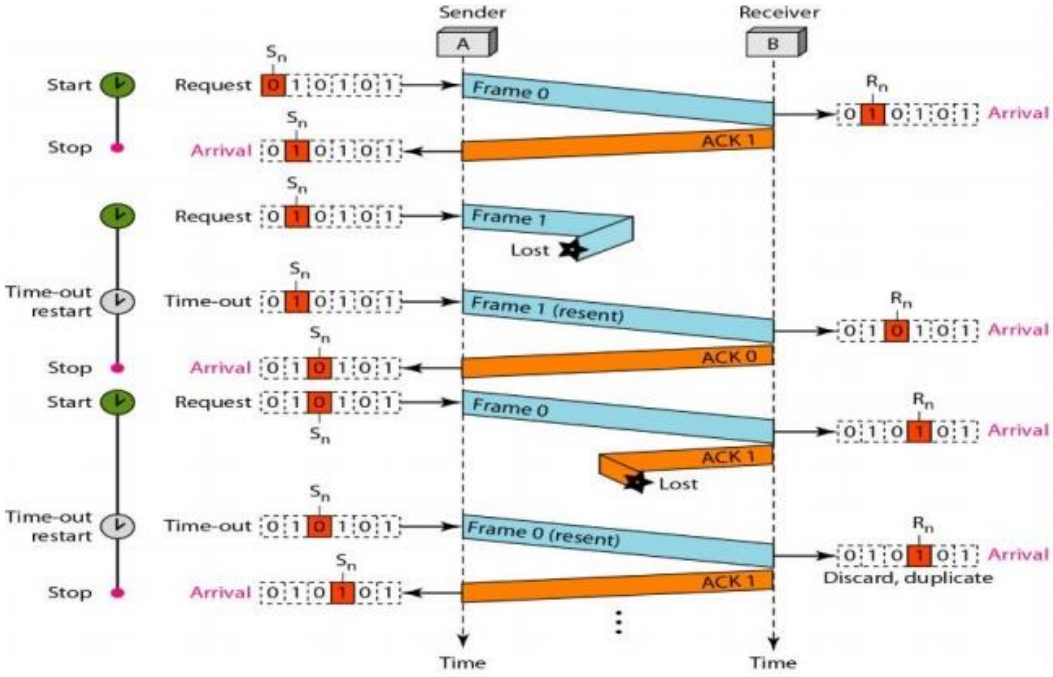
Stop and wait ARQ (Automatic repeat request)



Design of stop and wait arq



Flow Diagram Of Stop And Wait Arq



Go-Back-N ARQ

Before understanding the working of Go-Back-N ARQ, we first look at the sliding window protocol. As we know that the sliding window protocol is different from the stop-and-wait protocol. In the stop-and-wait protocol, the sender can send only one frame at a time and cannot send the next frame without receiving the acknowledgment of the previously sent frame, whereas, in the case of sliding window protocol, the multiple frames can be sent at a time. The variations of sliding window protocol are Go-Back-N ARQ and Selective Repeat ARQ. Let's understand 'what is Go-Back-N ARQ'.

What is Go-Back-N ARQ?

ln Go-Back-N ARQ, **N** is the sender's window size. Suppose we say that Go-Back-3, which means that the three frames can be sent at a time before expecting the acknowledgment from the receiver.

It uses the principle of protocol pipelining in which the multiple frames can be sent before receiving the acknowledgment of the first frame. If we have five frames and the concept is Go-Back-3, which means that the three frames can be sent, i.e., frame no 1, frame no 2, frame no 3 can be sent before expecting the acknowledgment of frame no 1.

In Go-Back-N ARQ, the frames are numbered sequentially as Go-Back-N ARQ sends the multiple frames at a time that requires the numbering approach to distinguish the frame from another frame, and these numbers are known as the sequential numbers.

The number of frames that can be sent at a time totally depends on the size of the sender's window. So, we can say that 'N' is the number of frames that can be sent at a time before receiving the acknowledgment from the receiver.

If the acknowledgment of a frame is not received within an agreed-upon time period, then all the frames available in the current window will be retransmitted. Suppose we have sent the frame no 5, but we didn't receive the acknowledgment of frame no 5, and the current window is holding three frames, then these three frames will be retransmitted.

The sequence number of the outbound frames depends upon the size of the sender's window. Suppose the sender's window size is 2, and we have ten frames to send, then the sequence numbers will not be 1,2,3,4,5,6,7,8,9,10. Let's understand through an example.

N is the sender's window size.

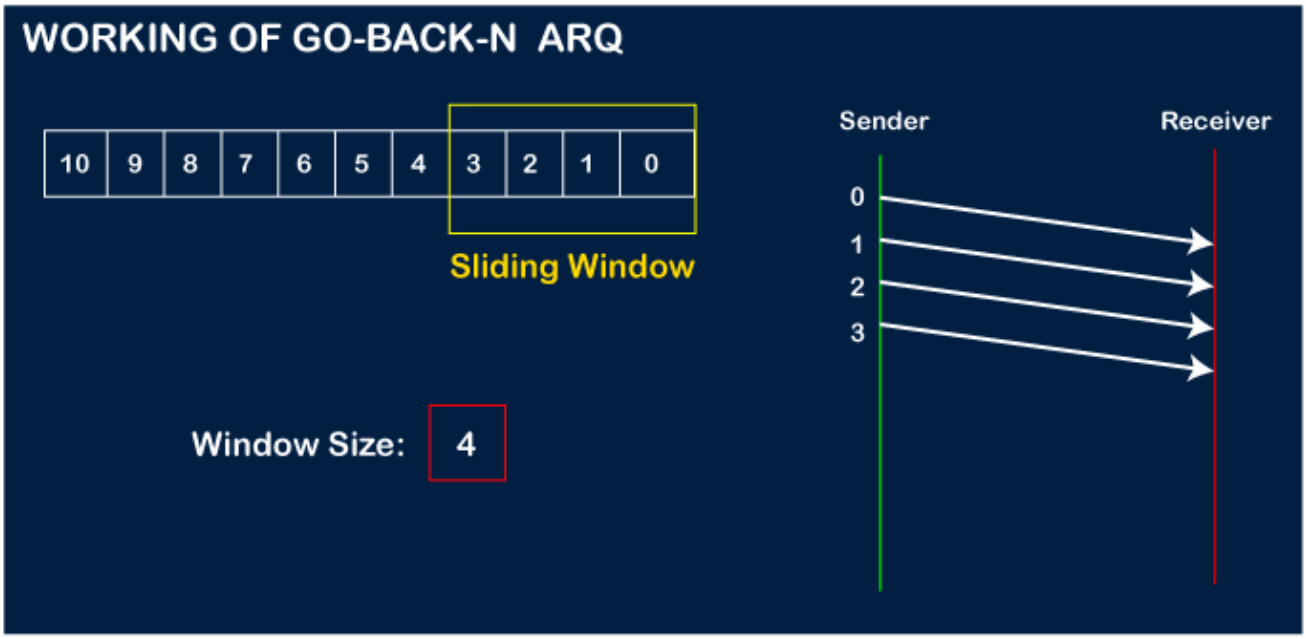
If the size of the sender's window is 4 then the sequence number will be 0,1,2,3,0,1,2,3,0,1,2, and so on.

The number of bits in the sequence number is 2 to generate the binary sequence 00,01,10,11.

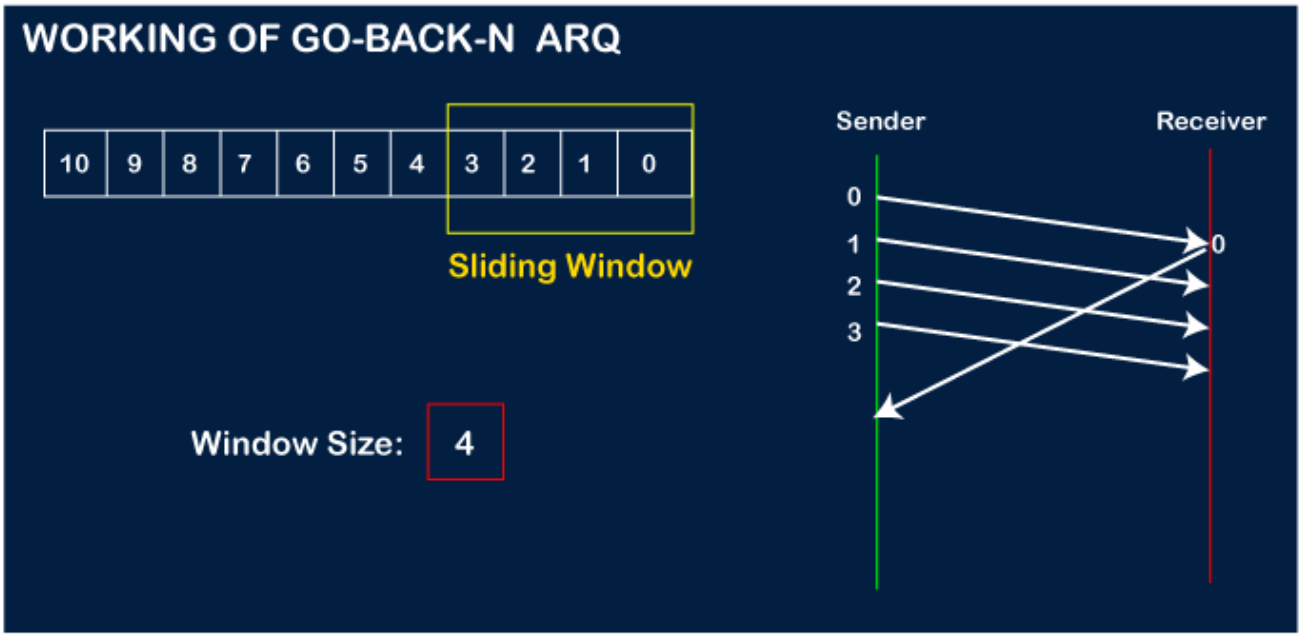
Working of Go-Back-N ARQ

Suppose there are a sender and a receiver, and let's assume that there are 11 frames to be sent. These frames are represented as 0,1,2,3,4,5,6,7,8,9,10, and these are the sequence numbers of the frames. Mainly, the sequence number is decided by the sender's window size. But, for the better understanding, we took the running sequence numbers, i.e., 0,1,2,3,4,5,6,7,8,9,10. Let's consider the window size as 4, which means that the four frames can be sent at a time before expecting the acknowledgment of the first frame.

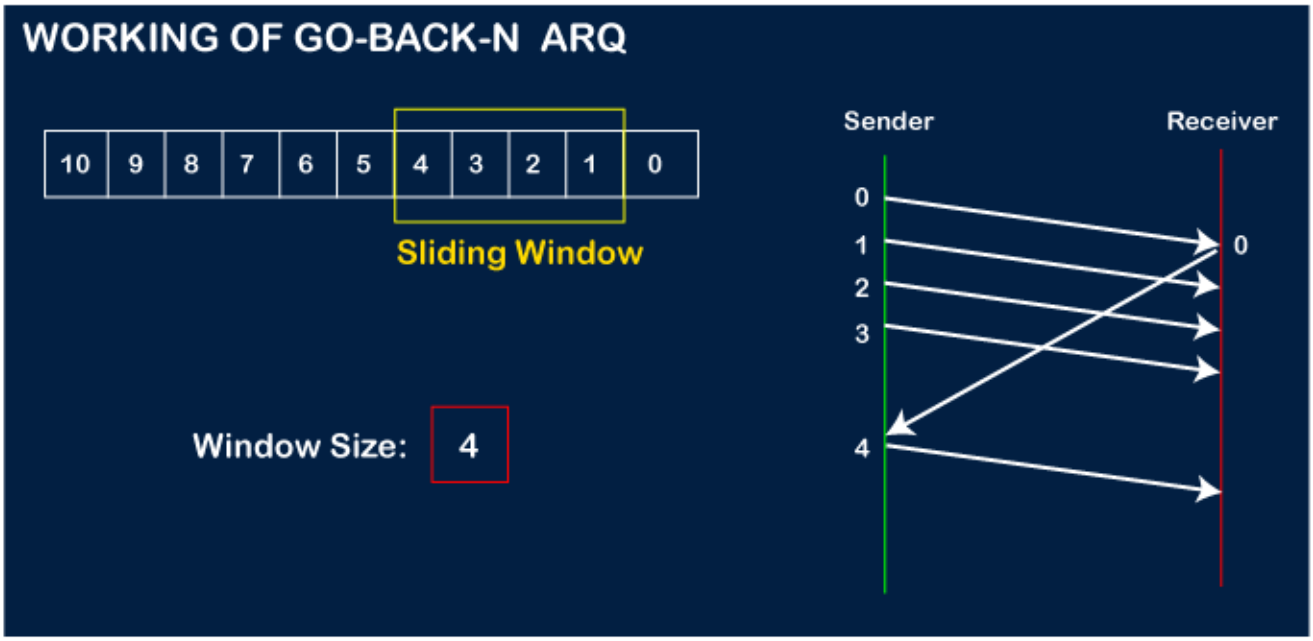
Step 1: Firstly, the sender will send the first four frames to the receiver, i.e., 0,1,2,3, and now the sender is expected to receive the acknowledgment of the 0^{th} frame.



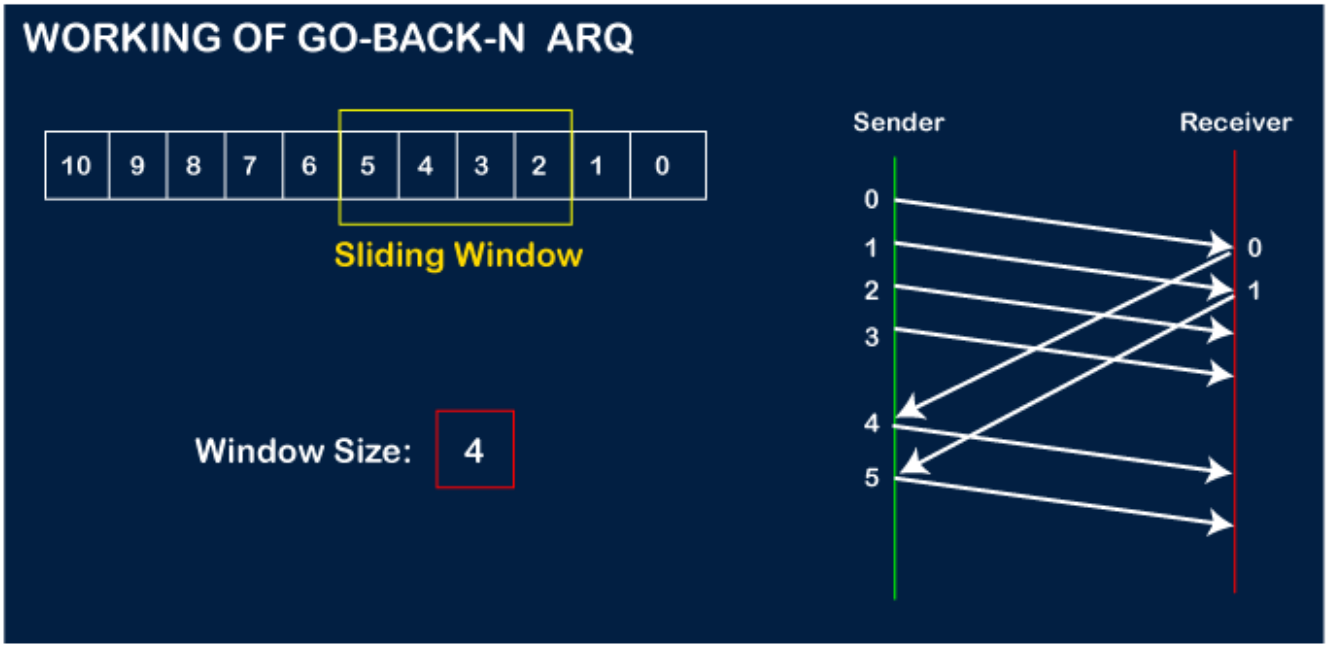
Let's assume that the receiver has sent the acknowledgment for the 0 frame, and the receiver has successfully received it.



The sender will then send the next frame, i.e., 4, and the window slides containing four frames (1,2,3,4).

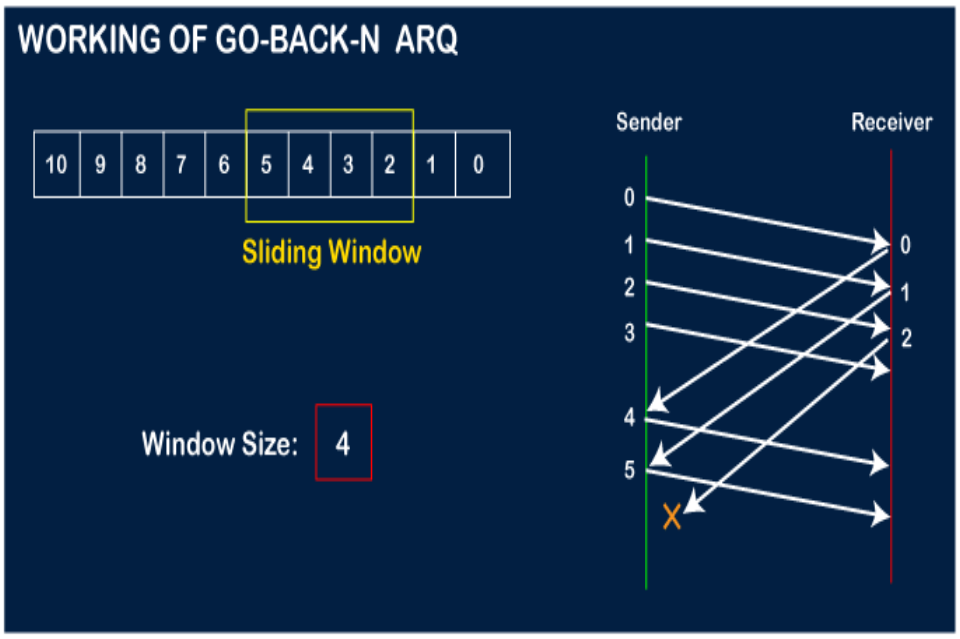


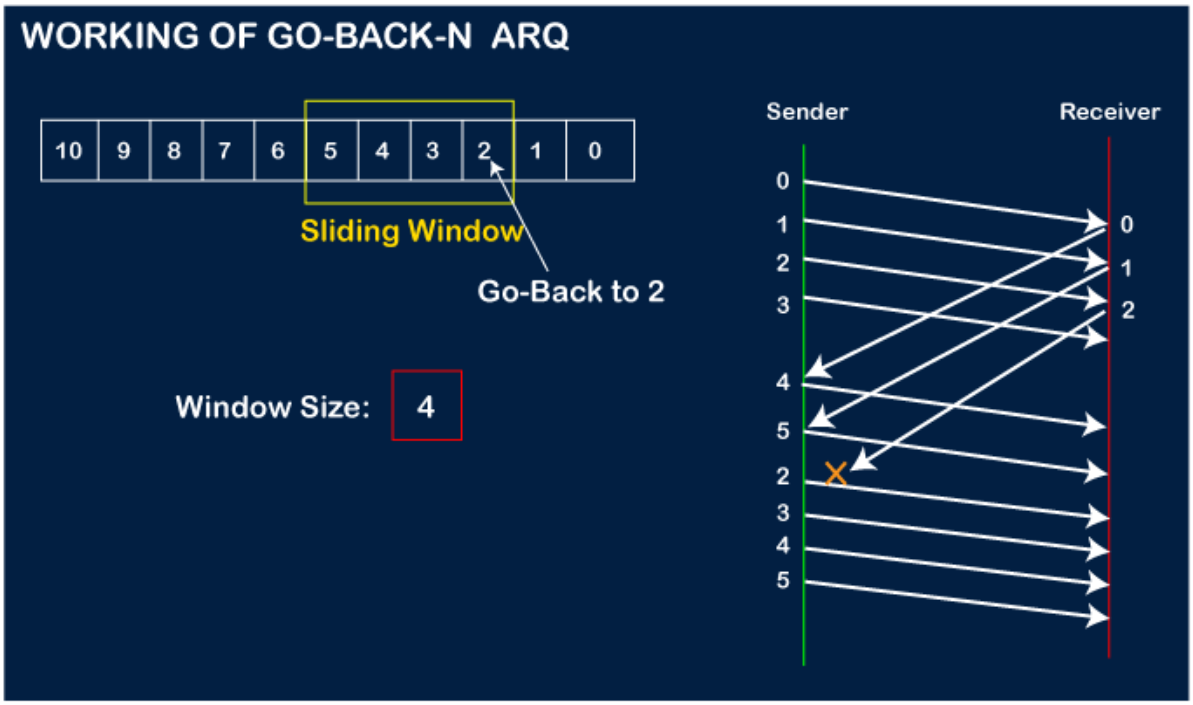
The receiver will then send the acknowledgment for the frame no 1. After receiving the acknowledgment, the sender will send the next frame, i.e., frame no 5, and the window will slide having four frames (2,3,4,5).



Now, let's assume that the receiver is not acknowledging the frame no 2, either the frame is lost, or the acknowledgment is lost. Instead of sending the frame no 6, the sender Go-Back

to 2, which is the first frame of the current window, retransmits all the frames in the current window, i.e., 2,3,4,5.





Important points related to Go-Back-N ARQ:

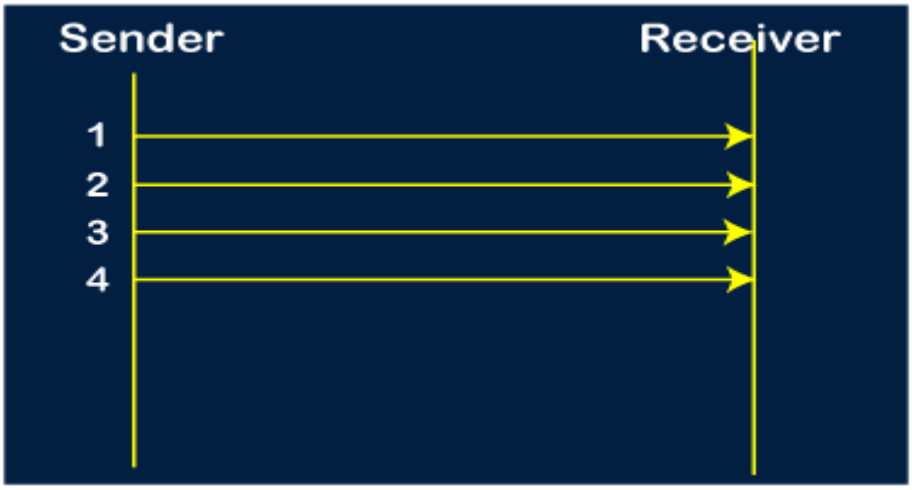
- In Go-Back-N, N determines the sender's window size, and the size of the receiver's window is always 1.
- It does not consider the corrupted frames and simply discards them.
- It does not accept the frames which are out of order and discards them.
- If the sender does not receive the acknowledgment, it leads to the retransmission of all the current window frames.

Let's understand the Go-Back-N ARQ through an example.

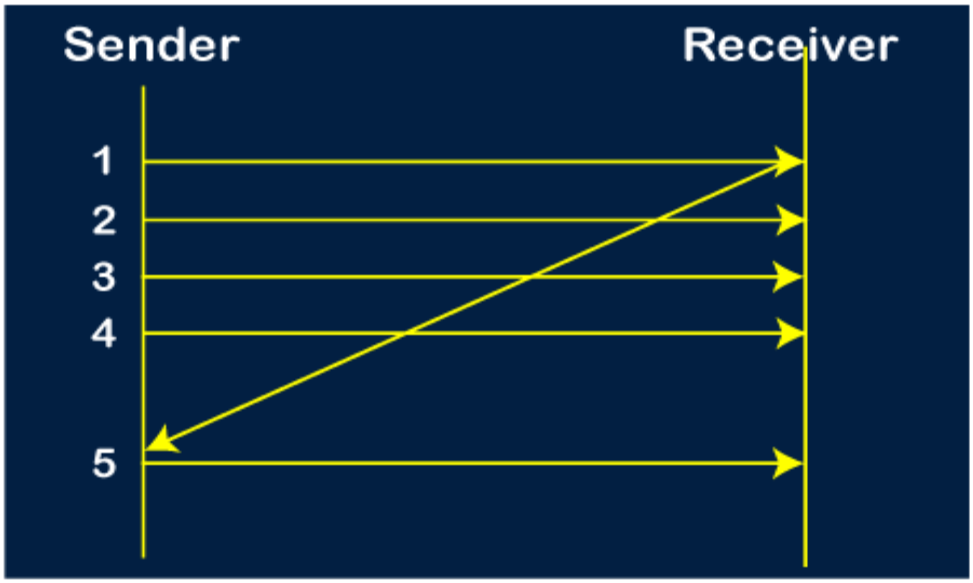
Example 1: In GB4, if every 6^{th} packet being transmitted is lost and if we have to spend 10 packets then how many transmissions are required?

Solution: Here, GB4 means that N is equal to 4. The size of the sender's window is 4.

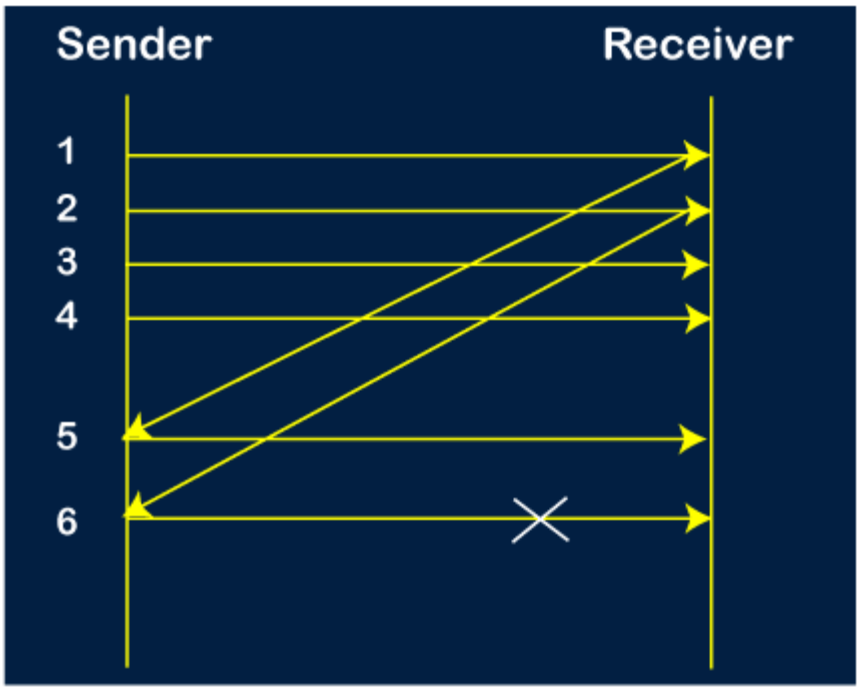
Step 1: As the window size is 4, so four packets are transferred at a time, i.e., packet no 1, packet no 2, packet no 3, and packet no 4.



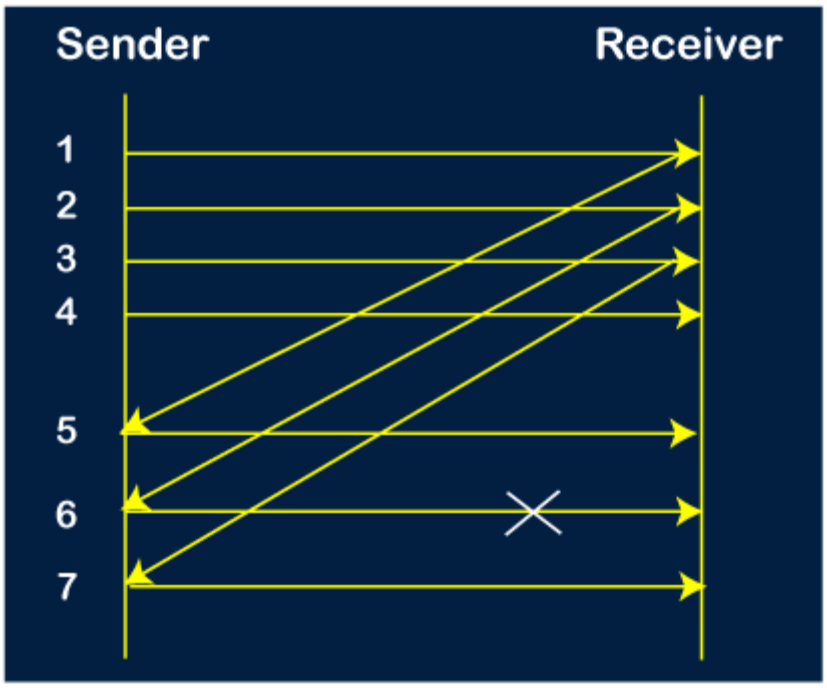
Step 2: Once the transfer of window size is completed, the sender receives the acknowledgment of the first frame, i.e., packet no1. As the acknowledgment receives, the sender sends the next packet, i.e., packet no 5. In this case, the window slides having four packets, i.e., 2,3,4,5 and excluded the packet 1 as the acknowledgment of the packet 1 has been received successfully.



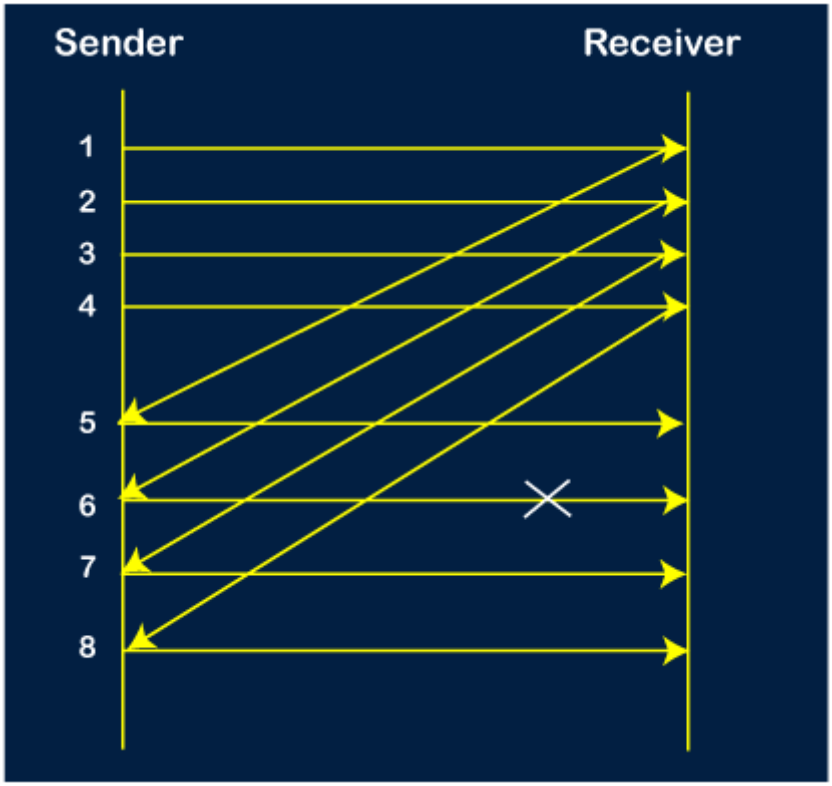
Step 3: Now, the sender receives the acknowledgment of packet 2. After receiving the acknowledgment for packet 2, the sender sends the next packet, i.e., packet no 6. As mentioned in the question that every 6th is being lost, so this 6th packet is lost, but the sender does not know that the 6th packet has been lost.



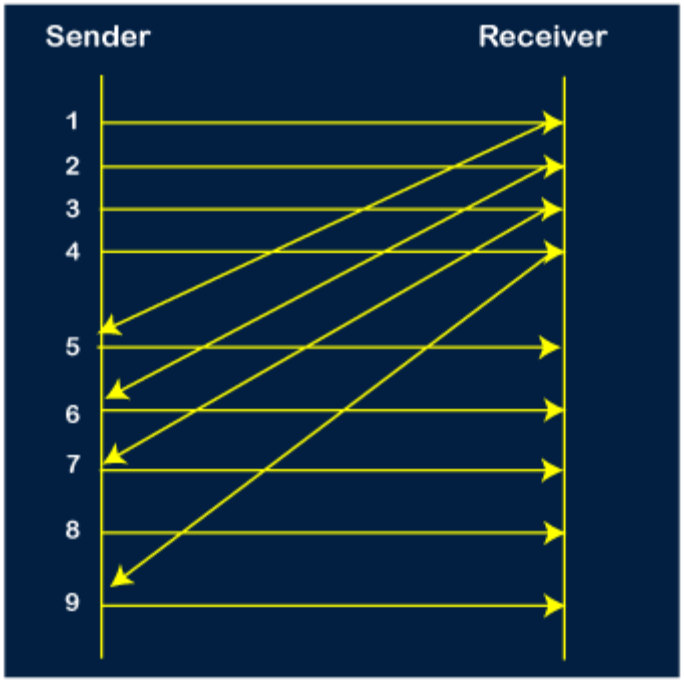
Step 4: The sender receives the acknowledgment for the packet no 3. After receiving the acknowledgment of 3rd packet, the sender sends the next packet, i.e., 7th packet. The window will slide having four packets, i.e., 4, 5, 6, 7.



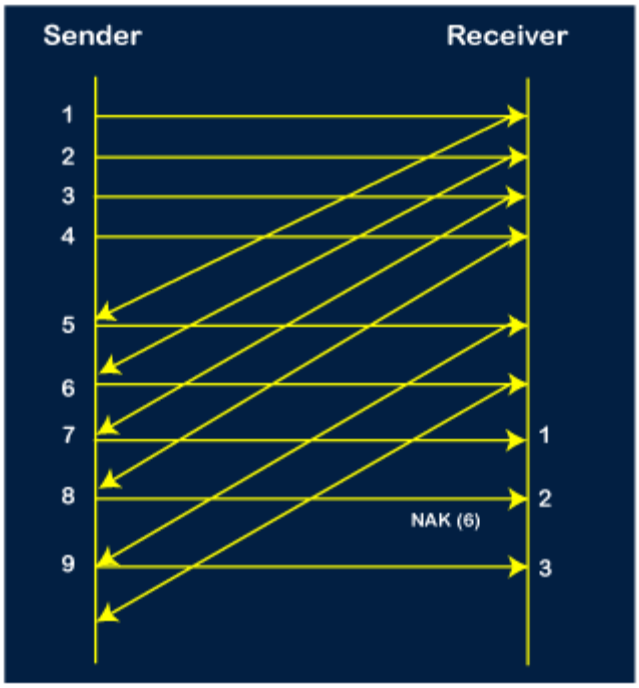
Step 5: When the packet 7 has been sent, then the sender receives the acknowledgment for the packet no 4. When the sender has received the acknowledgment, then the sender sends the next packet, i.e., the 8th packet. The window will slide having four packets, i.e., 5, 6, 7, 8.



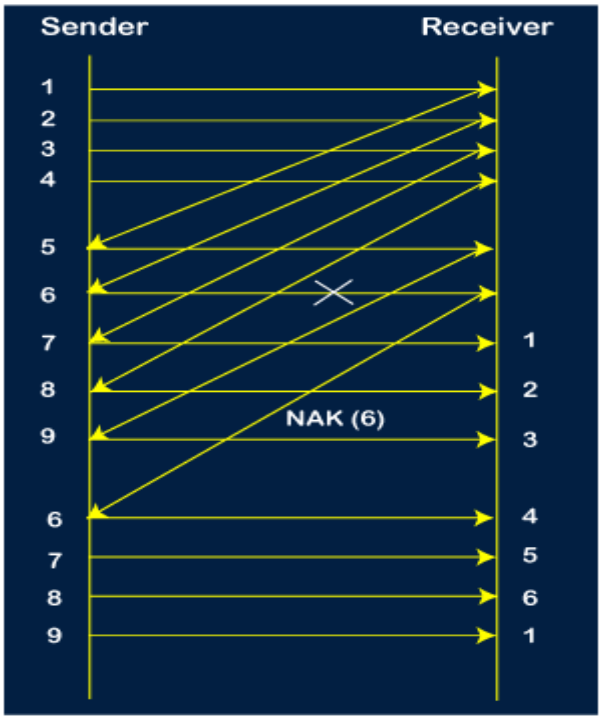
Step 6: When the packet 8 is sent, then the sender receives the acknowledgment of packet 5. On receiving the acknowledgment of packet 5, the sender sends the next packet, i.e., 9th packet. The window will slide having four packets, i.e., 6, 7, 8, 9.



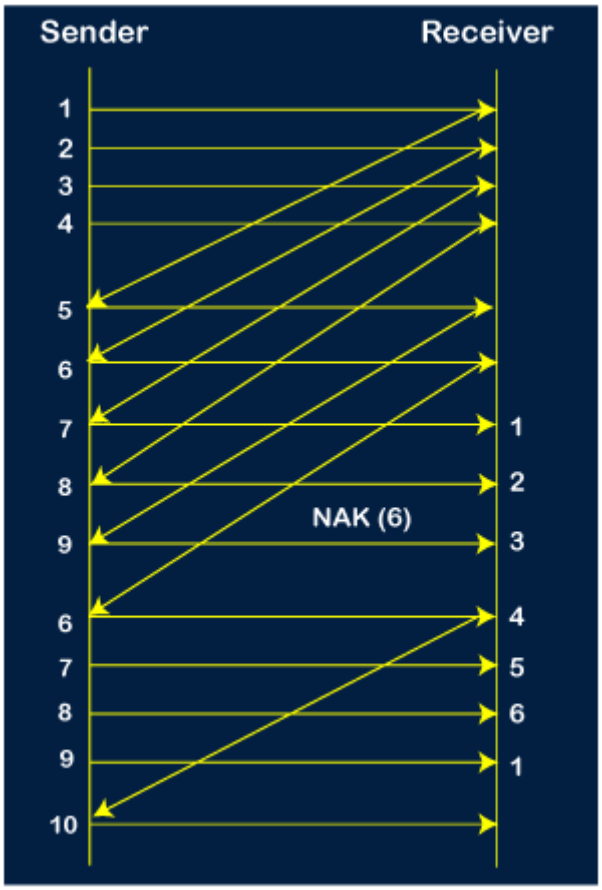
Step 7: The current window is holding four packets, i.e., 6, 7, 8, 9, where the 6th packet is the first packet in the window. As we know, the 6th packet has been lost, so the sender receives the negative acknowledgment NAK(6). As we know that every 6th packet is being lost, so the counter will be restarted from 1. So, the counter values 1, 2, 3 are given to the 7th packet, 8th packet, 9th packet respectively.



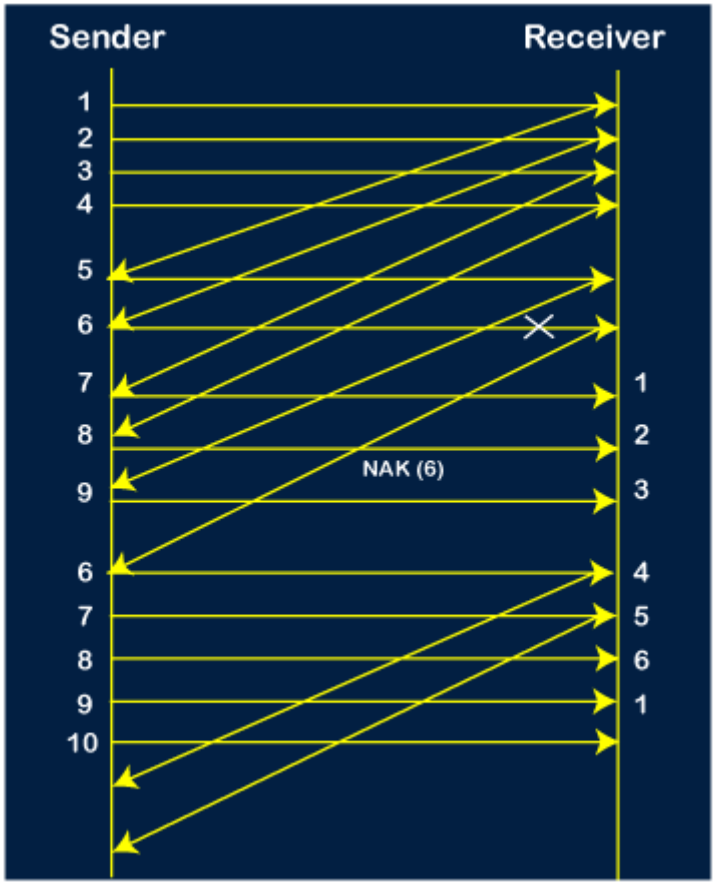
Step 8: As it is Go-BACK, so it retransmits all the packets of the current window. It will resend 6, 7, 8, 9. The counter values of 6, 7, 8, 9 are 4, 5, 6, 1, respectively. In this case, the 8th packet is lost as it has a 6-counter value, so the counter variable will again be restarted from 1.



Step 9: After the retransmission, the sender receives the acknowledgment of packet 6. On receiving the acknowledgment of packet 6, the sender sends the 10th packet. Now, the current window is holding four packets, i.e., 7, 8, 9, 10.

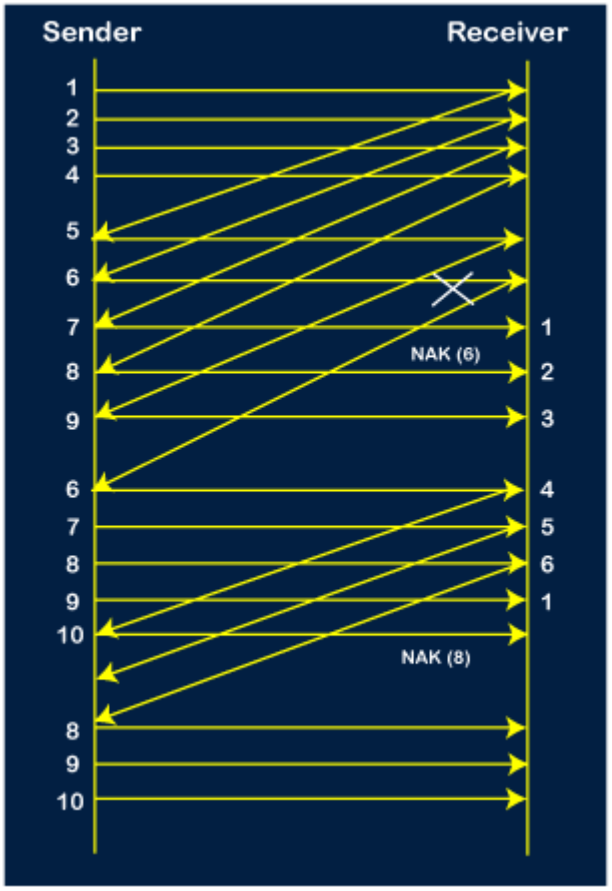


Step 10: When the 10th packet is sent, the sender receives the acknowledgment of packet 7. Now the current window is holding three packets, 8, 9 and 10. The counter values of 8, 9, 10 are 6, 1, 2.



Step 11: As the 8th packet has 6 counter value which means that 8th packet has been lost, and the sender receives NAK (8).

Step 12: Since the sender has received the negative acknowledgment for the 8th packet, it resends all the packets of the current window, i.e., 8, 9, 10.



Step 13: The counter values of 8, 9, 10 are 3, 4, 5, respectively, so their acknowledgments have been received successfully.

We conclude from the above figure that total 17 transmissions are required.

Selective-Repeat ARQ

What is Selective Repeat ARQ?

The selective repeat ARQ is one of the Sliding Window Protocol strategies that is used where reliable in-order delivery of the data packets is required. The selective repeat ARQ is used for noisy channels or links and it manages the flow and error control between the sender and the receiver.

In the selective repeat ARQ, we only resend the data frames that are damaged or lost. On the other hand, the correct frames are received at the receiver's send and are buffered for future

usage. As we only resend the selected damaged frames so we name this technique the **Selective Repeat ARQ** technique.

In the Selective Repeat ARQ protocol, ARQ stands for Automatic Repeat Request. ARQ is an error-control strategy that ensures that a sequence of information is delivered in order and without any error or duplications despite transmission errors and losses.

In selective repeat ARQ, the sender sets a timer for each frame so whenever the timer is over and the sender has not received any acknowledgment for the frame, then the sender knows that the particular frame is either lost or damaged. So, the sender sends back the lost or damaged frame once the timer is out.

Now, as we can see that the sender needs to wait for the timer to expire before retransmission. So, we use negative acknowledgment or NACK. As the receiver receives frames from the sender, it keeps track of the sequence number of the frame and buffers it into the memory, and sends an ACK from the frame. The ack or acknowledgment lets the sender know that the particular frame is correctly received by the receiver.

Now, if any frame is lost or damaged then the receiver sends a NACK to the sender. When the sender gets NACK then it retransmits the lost frame only. The receiver does not discard all the subsequent frames following a lost frame, it just sends back A NACK and stores the rest of the frames in its memory buffer. In this way, the sender does not need to wait for the timer to expire. Hence, the performance of the selective repeat ARQ increases.

The ACK and the NACK have the sequence number of the frame that helps the sender to identify the lost frame.

As the receiver may receive the frames in a different order, the receiver has the capability of sorting the frames present in the memory buffer using the sequence numbers. On the other hand, the sender must be capable enough to search for the lost frame for which the NACK has been received. So searching at the sender's end and sorting at the receiver's are two minor drawbacks of the selective repeat ARQ.

Note: The sliding window protocol is a data link layer protocol that is useful in the sequential and reliable delivery of the data frames. Using the sliding window protocol, the sender can send multiple frames at a time. Other two sliding window protocol strategies are:

- 1-bit or Stop & Wait ARQ, and
- Go-Back-N ARQ.

Let us learn the working of the selective repeat ARQ using an example in the next section.

Working of Selective Repeat Protocol

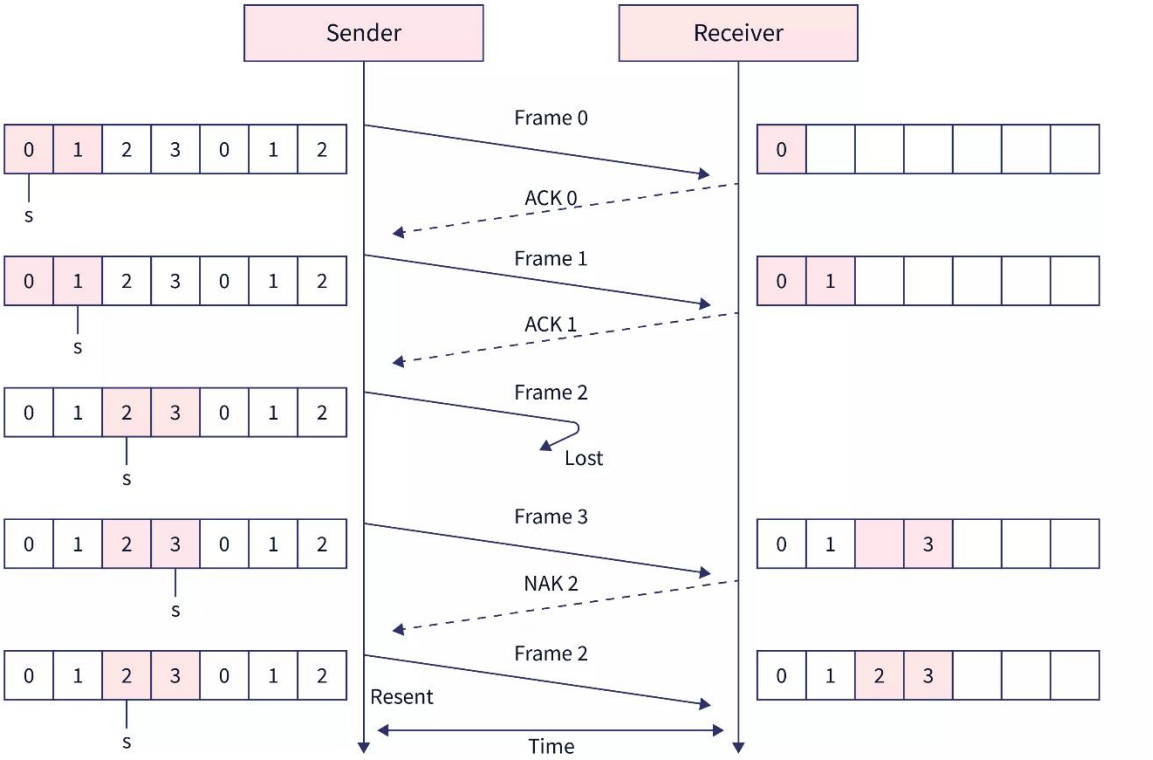
Before learning about the working of the selective repeat ARQ, we should be familiar with the window size of sender and receiver as the selective repeat ARQ is a type of sliding window protocol only.

In the selective repeat ARQ, both the sender and the receiver have windows of the same size. The window on the sender's side covers the sequence of data packets that are sent (or to be sent). On the other hand, the window on the receiver's side covers the sequence of data packets that are received (or to be received).

The size of the sender's window is **2^(m-1)**, where m is the number of bits used in the header of the packet to express the packet's sequence number. The window size of the receiver is the same as that of the sender i.e. 2^(m-1). The sender's window size is represented using **Ws** and the receiver's window size is represented using **Wr**.

The overall working of the selective repeat ARQ is simple. Initially, the sender sends several frames according to the window size. The receiver on the other end receives the frames and sends the ACK for correct frames and NACK for lost or damaged frames. The sender re-transmits the frames for which the NACK was sent by the receiver. After receiving all the frames, the receiver sorts the frame according to the sequence number for further usage.

Now, let us take an example to visualize the working of selective repeat ARQ or how the data packet is transmitted using the selective repeat ARQ protocol. The image below shows the transmission of frames. Let us suppose that the window size of the sender and the receiver is 2.



Difference between the Go-Back-N ARQ and Selective Repeat ARQ?

Go-Back-N ARQ	Selective Repeat ARQ
If a frame is corrupted or lost in it,all subsequent frames have to be sent again.	In this, only the frame is sent again, which is corrupted or lost.
If it has a high error rate,it wastes a lot of bandwidth.	There is a loss of low bandwidth.
It is less complex.	It is more complex because it has to do sorting and searching as well. And it also requires more storage.
It does not require sorting.	In this, sorting is done to get the frames in the correct order.
It does not require searching.	The search operation is performed in it.
It is used more.	It is used less because it is more complex.

ADVANTAGE OF USING GO BACK N AND SELECTIVE REPEAT ARQ:

a. The advantage of using the Go-Back-N protocol is that we can have a larger send window size. We can send more packets before waiting for their acknowledgment. The disadvantage of using this protocol is that the receive window size is only 1. The receiver cannot accept and store the out of-order received packets; they will be discarded. Discarding of the out-of order packets means resending these packets by the sender, resulting in congestion of the network and reducing the capacity of the pipe. So the advantage seen by a larger send window may disappear by filling the network with resent packets.

b. The advantage of using the Selective-Repeat protocol is that the receive window can be much larger than 1. This allows the receive window to store the out-of-order packets and avoids resending them to congest the network. The disadvantage of this protocol is that the send window size is half of the Go-Back-N, which means that we can send fewer packets before waiting for the acknowledgment.

We can conclude that if the bandwidth-delay product of the network is large, the reliability is good, and the delay is low, we should choose the Go-Back-N protocol to use more of the network capacity. On the other hand, if the bandwidth-delay product is small, or the network is not very reliable, or the network creates long delays, we need to use Selective-Repeat.

Bidirectional Protocols: Piggybacking

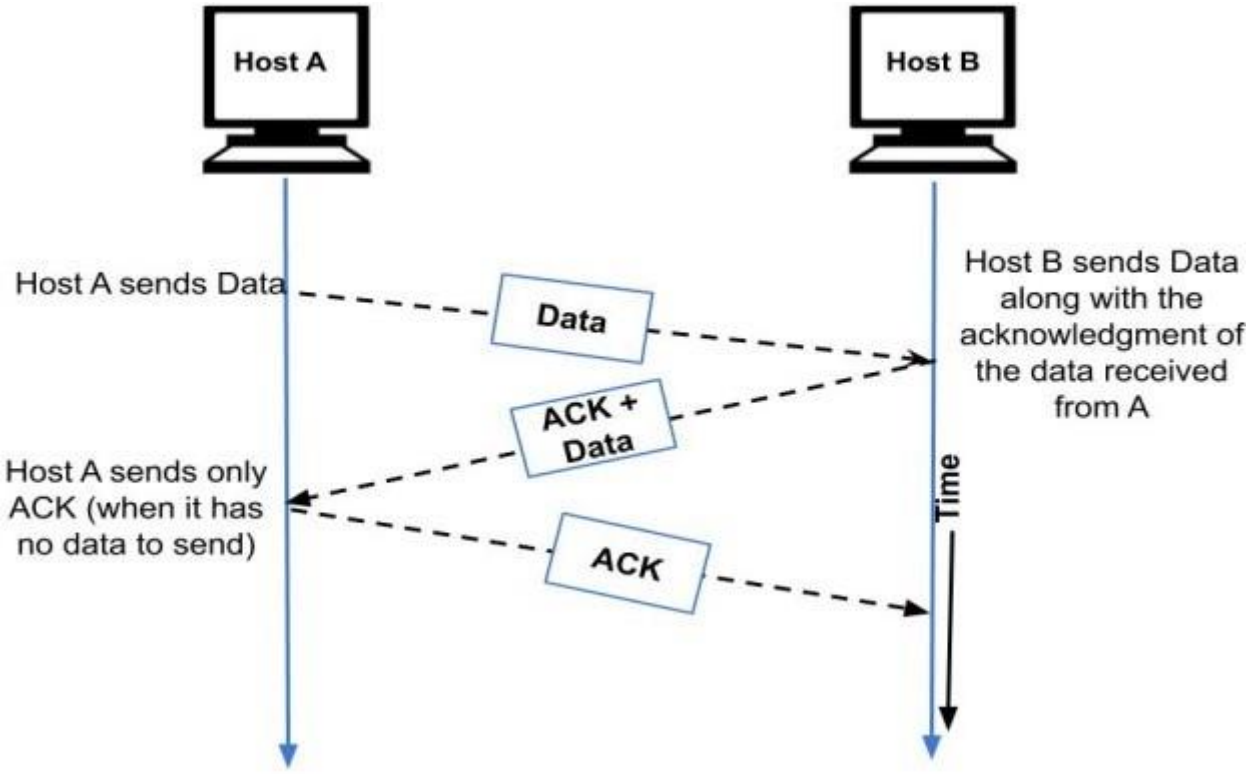
What is Piggybacking?

Mostly the transmission of data is bidirectional i.e. full-duplex transmission. The data flows in both directions, so the control information i.e. acknowledgment, etc. also needs to be flowed in both the directions. This can be done by making two simplex connections. The first simplex connection for the sender has a separate channel, each for sending the data and receiving the acknowledgment. Similarly, for the receiver, it has a separate channel for sending the data and receiving the acknowledgment. But, due to this, the traffic would increase and half of the transmission would consist of acknowledgments. So, the bandwidth of the channel would be wasted. An improved solution to this problem is **Piggybacking.** So, let's get started and know more about it.

Piggybacking

Piggybacking is a method of **attaching acknowledgment to the outgoing data packet** . The concept of piggybacking is explained as follows:

Consider a two-way transmission between host **A** and host **B** . When host A sends a data frame to B, then B does not send the acknowledgment of the frame sent immediately. The acknowledgment is delayed until the next data frame of host B is available for transmission. The delayed acknowledgment is then attached to the outgoing data frame of B. This process of delaying acknowledgment so that it can be attached to the outgoing frame is called **piggybacking** .



Now, as we are communicating between the host A and host B, three conditions can arise:

1. When the host has both data and the acknowledgment to send, then it will attach the data along with the acknowledgment. In the above diagram, the host B will attach the data frame along with the acknowledgment of the last frame received from host A.
2. When the host does not have any data to send then it will send only the acknowledgment. In the above diagram, when host A does not have any data frame to send. So, it will only send the acknowledgment of the last frame received.
3. When the host has only data to send then it will send the data along with the acknowledgment of the last frame received. The duplicate acknowledgment will be discarded by the receiver and the data would be accepted.

Advantages of Piggybacking

1. The available channel bandwidth is used efficiently.

Disadvantages of Piggybacking

1. As there is delayed transmission of acknowledgment so if the acknowledgment is not received within the fixed time then the sender has to retransmit the data.
2. There is additional complexity for implementing this method.

Transport-Layer Protocols

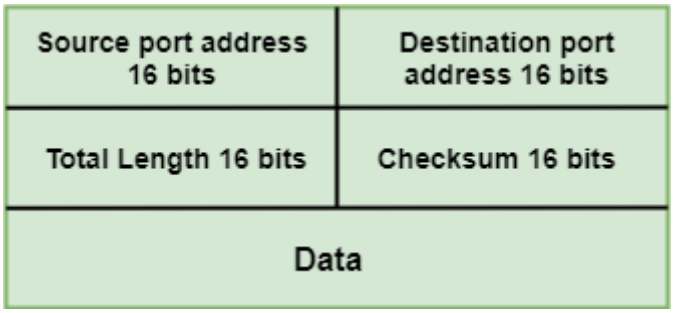
- USER DATAGRAM PROTOCOL
- TRANSMISSION CONTROL PROTOCOL

UDP

- UDP stands for **User Datagram Protocol**.
- UDP is a simple protocol and it provides non sequenced transport functionality.
- UDP is a connectionless protocol.
- This type of protocol is used when reliability and security are less important than speed and size.
- UDP is an end-to-end transport level protocol that adds transport-level addresses, checksum error control, and length information to the data from the upper layer.
- The packet produced by the UDP protocol is known as a user datagram.

User Datagram Format

The user datagram has a 16-byte header which is shown below:



Where,

- **Source port address:** It defines the address of the application process that has delivered a message. The source port address is of 16 bits address.
- **Destination port address:** It defines the address of the application process that will receive the message. The destination port address is of a 16-bit address.
- **Total length:** It defines the total length of the user datagram in bytes. It is a 16-bit field.
- **Checksum:** The checksum is a 16-bit field which is used in error detection.

Well-Known Ports for UDP

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters

53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
III	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Operation / Services of UDP

Given below are different operations of UDP:

1.Connectionless Services

The User datagram protocol offers Connectionless Services which simply means that each user datagram that is sent by the UDP is an independent datagram. In different datagrams, there is no relationship, even if they are coming from the same source process and also going to the same destination program.

User datagrams are not numbered, there is no connection establishment and no connection termination.

Each datagram mainly travels through different paths.

2.Flow Control and Error Control

User datagram is a very simple and unreliable transport protocol. It does not provide any flow control mechanism and hence there is no window mechanism. Due to which the receiver may overflow with the incoming messages.

No error control mechanism is provided by UDP except checksum. Due to which the sender does not know if any message is has been lost or duplicated.

As there is a lack of flow control and error control it means that the process that uses the UDP should provide these mechanisms.

3.Encapsulation and decapsulation

In order to send the message from one process to another, the user datagram protocol encapsulates and de capsulates the message in the form of an IP datagram.

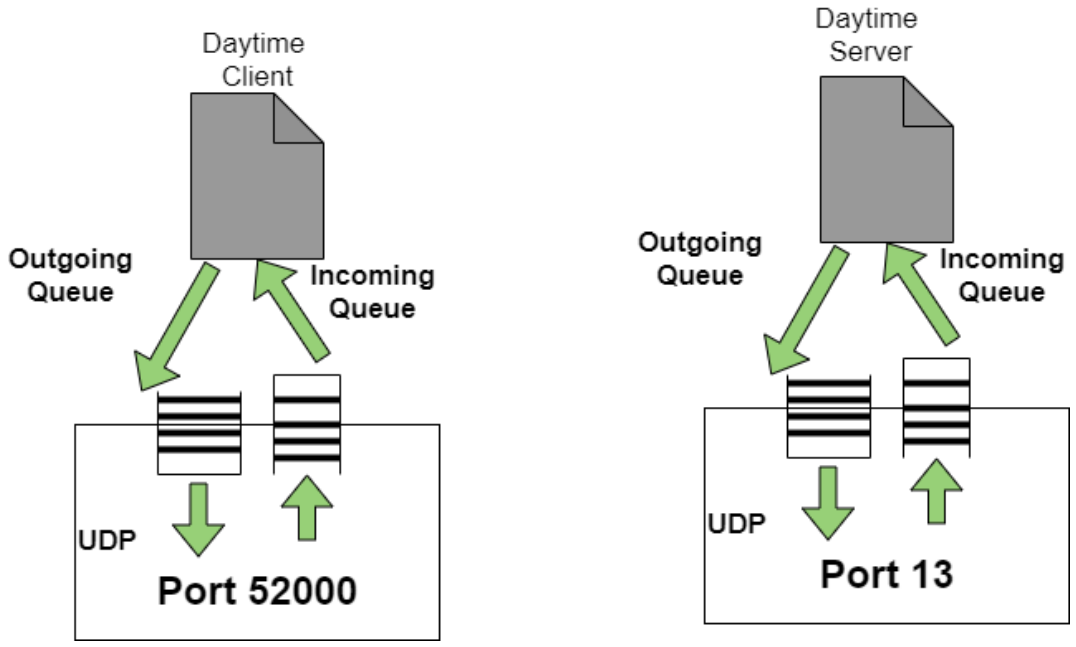
Applications of UDP

Given below are some applications of the User datagram protocol:

- **UDP** is used by those applications that require one response for one request.
- It is used by **broadcasting and multicasting applications.**
- Management processes such as **SNMP** make use of **UDP**.
- Route updating protocols like **Routing Information Protocol(RIP)** make use of User Datagram Protocol.
- The process that has an error and flows control mechanism makes use of UDP. One Application for the same is **Trivial File Transfer Protocol(TFTP).**

Queuing concept in User Datagram Protocol

In User datagram protocol, generally, queues are associated with ports:



The incoming queue is mainly used for receiving the messages, while the outgoing queue is mainly used for sending the messages. Queue mainly functions at the time when the process is in a running state when the process terminates, then the queue gets destroyed at the same time.

There are several processes that want to use the services provided by UDP.

User datagram protocol mainly multiplexes and demultiplexes the processes in order to run multiple processes on a single host.

Advantages of UDP

Given below are some advantages of UDP:

1. With UDP, broadcast and multicast transmission is possible.
2. UDP uses the bandwidth efficiently, as there is a small packet overhead.
3. As there is no need for connection establishment, hence UDP is very fast.
4. There is no buffering and numbering of packets.
5. There is no need for handshaking.
6. There is no congestion control so it is used for real-time applications.

Disadvantages of UDP

Now its time to take a look at UDP:

1.There is a lack of guaranteed delivery.

2.There is no flow control.

3.There is no congestion control mechanism.

TRANSMISSION CONTROL PROTOCOL

TCP

- TCP stands for Transmission Control Protocol.
- It provides full transport layer services to applications.

- It is a connection-oriented protocol means the connection established between both the ends of the transmission. For creating the connection, TCP generates a virtual circuit between sender and receiver for the duration of a transmission.

TCP Services

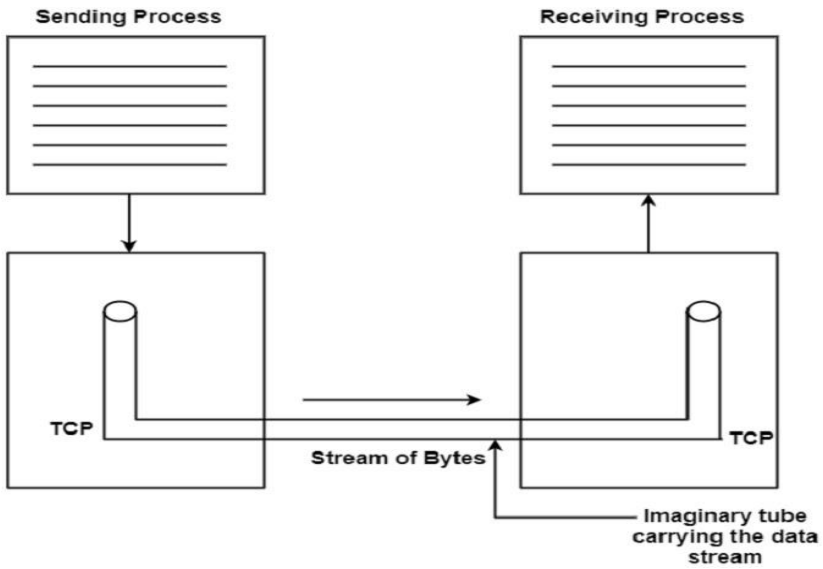
Following are some of the services offered by the Transmission Control Protocol (TCP) to the processes at the application layer:

- Stream Delivery Service.
- Sending and Receiving Buffers.
- Bytes and Segments.
- Full Duplex Service
- Connection Oriented Service.
- Reliable Service.

Stream Delivery Service

TCP is a stream-oriented protocol. It enables the sending process to deliver data as a stream of bytes and the receiving process to acquire data as a stream of bytes.

TCP creates a working environment so that the sending and receiving procedures are connected by an imaginary "tube", as shown in the figure below:



Sending and Receiving Buffers

The sending and receiving processes cannot produce and receive data at the same speed. Hence, TCP needs a buffer for storage.

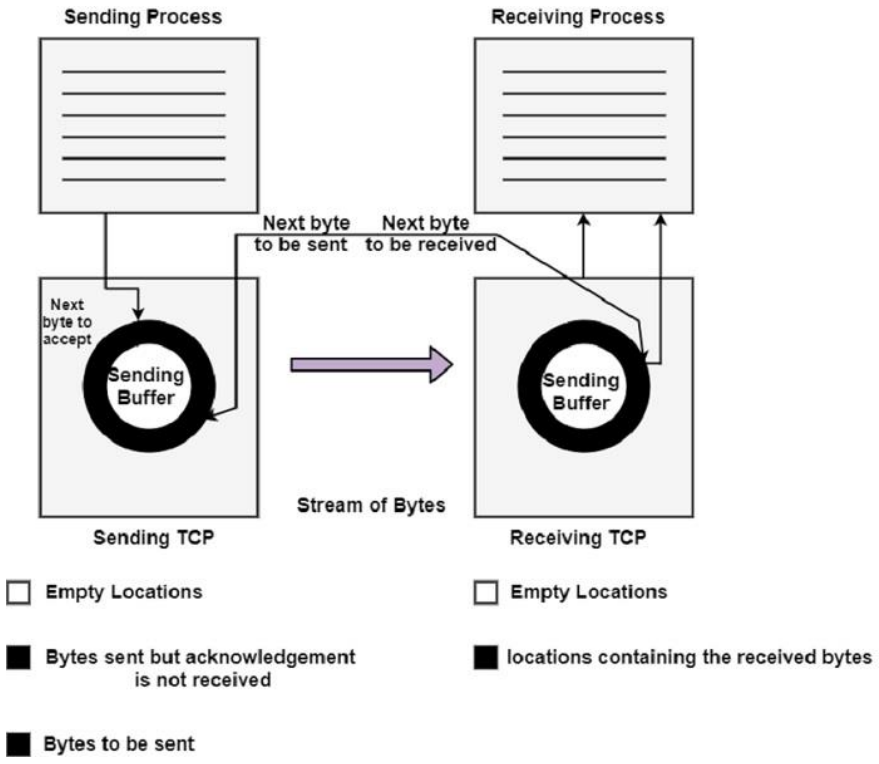
There are two methods of buffers used in each dissection, which are as follows:

- Sending Buffer
- Receiving Buffer

A buffer can be implemented by using a circular array of 1-byte location, as shown in the figure below. The figure shows the movement of data in one direction on the sending side.

The buffer has three types of locations, which are as follows:

- Empty Locations.
- Locations that contain the bytes which have been sent, but not acknowledged. These bytes are kept in the buffer till an acknowledgment is received.
- The location that contains the bytes which are to be sent by the sending TCP



In practice, the TCP may send only a port of data due to the slowness of the receiving process or congestion in the network.

The buffer at the receiver is divided into two parts as mentioned below:

- The part was containing empty locations.
- The part was containing the received bytes, which the sending process can consume.

Bytes and Segments

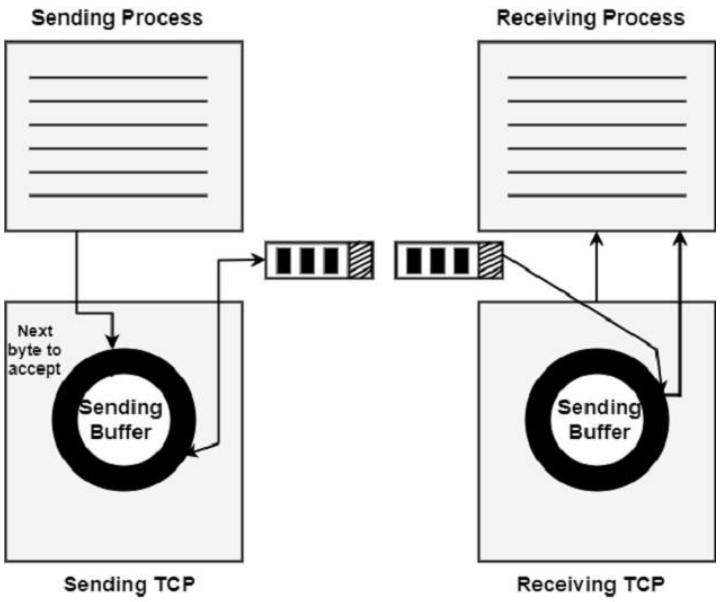
Buffering is used to handle the difference between the speed of data transmission and data consumption. But only buffering is not enough.

We need one more step before sending the data on the Internet Protocol (IP) layer as a TCP service provider. It needs to send data in the form of packets and not as a stream of bytes.

At the transport layer, TCP groups several bytes into a packet and this is called a segment. A header is added to each segment to exercise control.

The segment is encapsulated in an IP diagram and then transmitted. The entire operation is transparent to the receiving process. The segment may be deceived out of order, lost or corrupted when it receives the receiving end.

The figure given below shows how the segments are created from the bytes in the buffers:



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

Full-Duplex Service

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

Connection-Oriented Service

We are already aware that the TCP is a connection-oriented protocol. When a process wants to communicate (send and receive) with another process (process -2), the sequence of operations is as follows:

- TCP of process-1 informs TCP of process-2 and gets its approval.
- TCP of process-1 tells TCP of process-2 exchange data in both directions.
- After completing the data exchange, when buffers on both sides are empty, the two TCPs destroy their buffers

The type of connection in TCP is not physical, but it is virtual. The TCP segment encapsulated in an IP datagram can be sent out of order. These segments can get lost or corrupted and may have to be resend. Each segment may take a different path to reach the destination.

Reliable Service

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

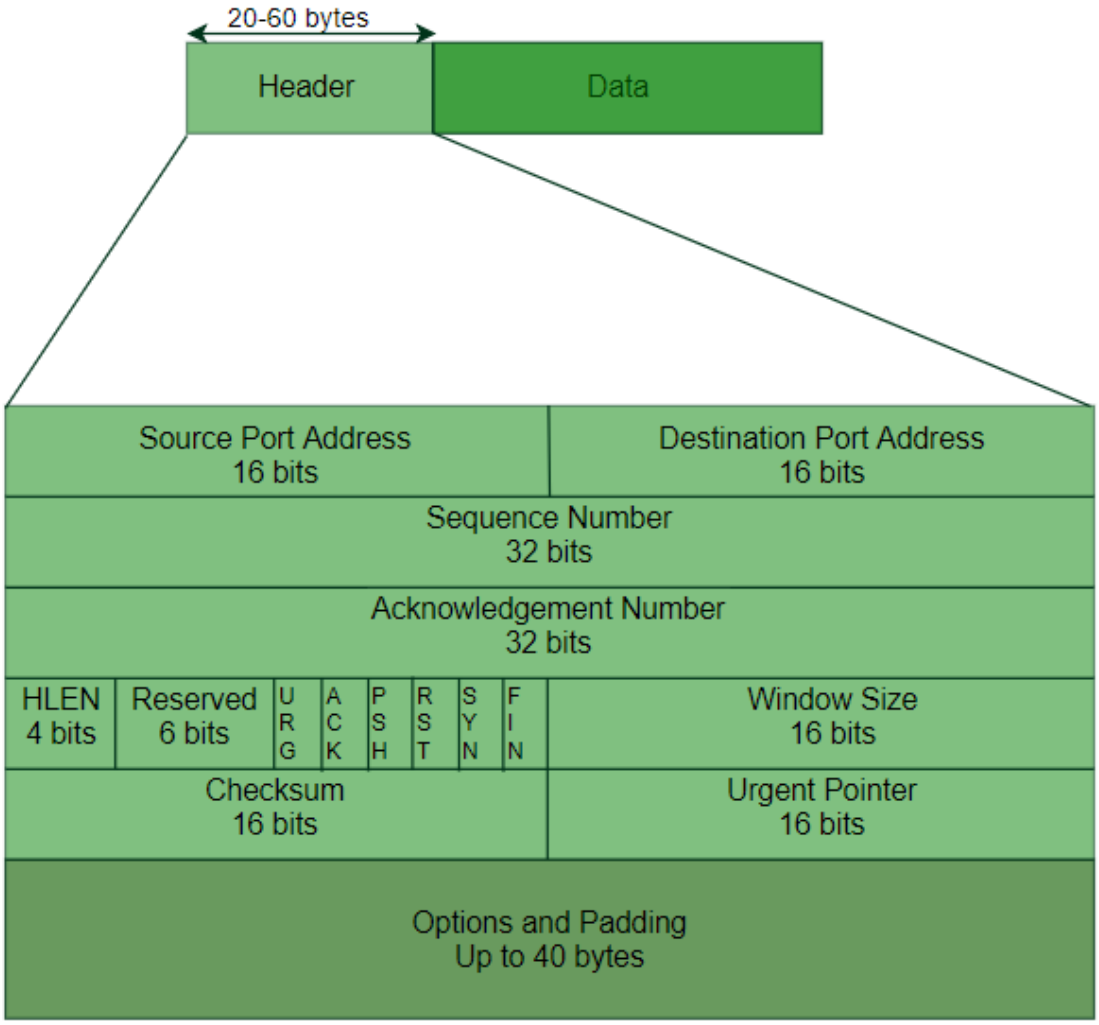
Features Of TCP protocol

- **Stream data transfer:** TCP protocol transfers the data in the form of contiguous stream of bytes. TCP group the bytes in the form of TCP segments and then passed it to the IP layer for transmission to the destination. TCP itself segments the data and forward to the IP.
- **Reliability:** TCP assigns a sequence number to each byte transmitted and expects a positive acknowledgement from the receiving TCP. If ACK is not received within a timeout interval, then the data is retransmitted to the destination. The receiving TCP uses the sequence number to reassemble the segments if they arrive out of order or to eliminate the duplicate segments.

- **Flow Control:** When receiving TCP sends an acknowledgement back to the sender indicating the number the bytes it can receive without overflowing its internal buffer. The number of bytes is sent in ACK in the form of the highest sequence number that it can receive without any problem. This mechanism is also referred to as a window mechanism.
- **Multiplexing:** Multiplexing is a process of accepting the data from different applications and forwarding to the different applications on different computers. At the receiving end, the data is forwarded to the correct application. This process is known as demultiplexing. TCP transmits the packet to the correct application by using the logical channels known as ports.
- **Logical Connections:** The combination of sockets, sequence numbers, and window sizes, is called a logical connection. Each connection is identified by the pair of sockets used by sending and receiving processes.
- **Full Duplex:** TCP provides Full Duplex service, i.e., the data flow in both the directions at the same time. To achieve Full Duplex service, each TCP should have sending and receiving buffers so that the segments can flow in both the directions. TCP is a connection-oriented protocol. Suppose the process A wants to send and receive the data from process B. The following steps occur:
    - Establish a connection between two TCPs.
    - Data is exchanged in both the directions.
    - The Connection is terminated.

TCP Segment

A TCP segment consists of data bytes to be sent and a header that is added to the data by TCP as shown:



The header of a TCP segment can range from 20-60 bytes. 40 bytes are for options. If there are no options, a header is 20 bytes else it can be of upmost 60 bytes.
Header fields:

- **Source Port Address –**
  A 16-bit field that holds the port address of the application that is sending the data segment.

- **Destination Port Address –**
  A 16-bit field that holds the port address of the application in the host that is receiving the data segment.

- **Sequence Number –**
  A 32-bit field that holds the sequence number, i.e, the byte number of the first byte that is sent in that particular segment. It is used to reassemble the message at the receiving end of the segments that are received out of order.

- **Acknowledgement Number –**
  A 32-bit field that holds the acknowledgement number, i.e, the byte number that the receiver expects to receive next. It is an acknowledgement for the previous bytes being received successfully.

- **Header Length (HLEN) –**
  This is a 4-bit field that indicates the length of the TCP header by a number of 4-byte words in the header, i.e if the header is 20 bytes(min length of TCP header), then this field will hold 5 (because 5 x 4 = 20) and the maximum length: 60 bytes, then it'll hold the value 15(because 15 x 4 = 60). Hence, the value of this field is always between 5 and 15.

- **Control flags –**
  These are 6 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc. Their function is:
    - URG: Urgent pointer is valid
    - ACK: Acknowledgement number is valid(used in case of cumulative acknowledgement)
    - PSH: Request for push
    - RST: Reset the connection
    - SYN: Synchronize sequence numbers
    - FIN: Terminate the connection
- **Window size –**
  This field tells the window size of the sending TCP in bytes.

- **Checksum –**
  This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP.

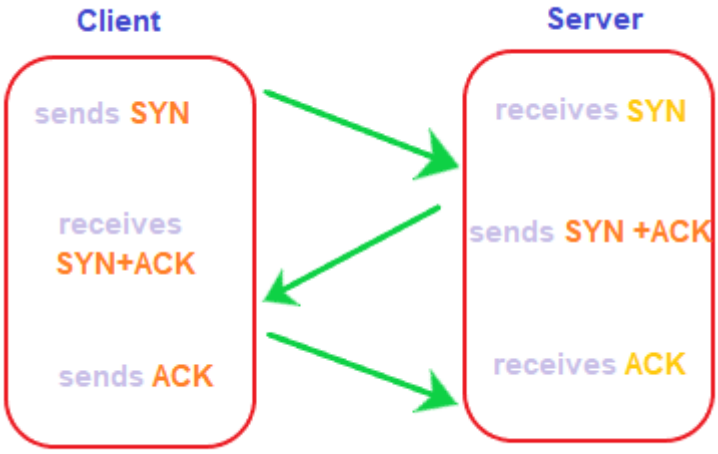
- **Urgent pointer –**
  This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.

TCP Connection (A 3-way handshake)

Handshake refers to the process to establish connection between the client and server. Handshake is simply defined as the process to establish a communication link. To transmit a packet, TCP needs a three way handshake before it starts sending data. The reliable communication in TCP is termed as **PAR** (Positive Acknowledgement Re-transmission). When a sender sends the data to the receiver, it requires a positive acknowledgement from the receiver confirming the arrival of data. If the acknowledgement has not reached the sender, it needs to resend that data. The positive acknowledgement from the receiver establishes a successful connection.

Here, the server is the server and client is the receiver. The above diagram shows 3 steps for successful connection. A 3-way handshake is commonly known as SYN-SYN-ACK and requires both the client and server response to exchange the data. SYN means **synchronize Sequence Number** and ACK means **acknowledgment**. Each step is a type of handshake between the sender and the receiver.

The diagram of a successful TCP connection showing the three handshakes is shown below:



The three handshakes are discussed in the below steps:

Step 1: SYN

SYN is a segment sent by the client to the server. It acts as a **connection request** between the client and server. It informs the server that the client wants to establish a connection. Synchronizing sequence numbers also helps synchronize sequence numbers sent between any two devices, where the same SYN segment asks for the sequence number with the connection request.

Step 2: SYN-ACK

It is an SYN-ACK segment or an SYN + ACK segment sent by the server. The ACK segment informs the client that the server has received the connection request and it is ready to build the connection. The SYN segment informs the sequence number with which the server is ready to start with the segments.

Step 3: ACK

ACK (Acknowledgment) is the last step before establishing a successful TCP connection between the client and server. The ACK segment is sent by the client as the response of the received ACK and SN from the server. It results in the establishment of a reliable data connection.

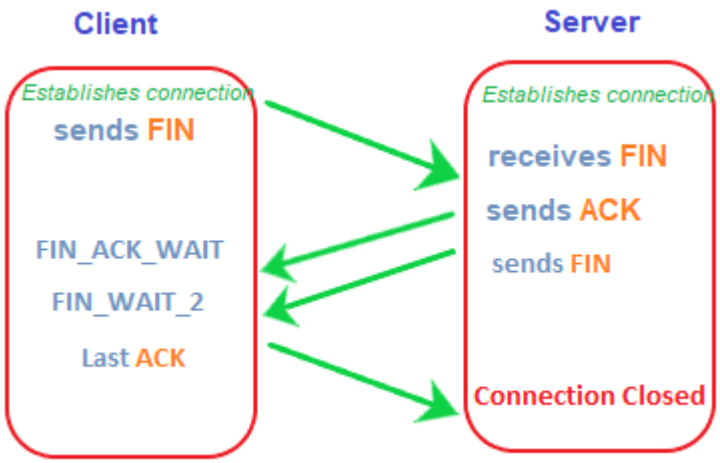
After these three steps, the client and server are ready for the data communication process. TCP connection and termination are full-duplex, which means that the data can travel in both the directions simultaneously.

TCP Termination (A 4-way handshake)

Any device establishes a connection before proceeding with the termination. TCP requires 3-way handshake to establish a connection between the client and server before sending the data. Similarly, to terminate or stop the data transmission, it requires a 4-way handshake. The segments required for TCP termination are similar to the segments to build a TCP connection (ACK and SYN) except the FIN segment. The FIN segment specifies a termination request sent by one device to the other.

The client is the data transmitter and the server is a receiver in a data transmission process between the sender and receiver. Consider the below TCP termination diagram that shows the exchange of segments between the client and server.

The diagram of a successful TCP termination showing the four handshakes is shown below:



Let's discuss the TCP termination process with the help of six steps that includes the sent requests and the waiting states. The steps are as follows:

Step 1: FIN

FIN refers to the **termination request** sent by the client to the server. The first FIN termination request is sent by the client to the server. It depicts the start of the termination process between the client and server.

Step 2: FIN_ACK_WAIT

The client waits for the ACK of the FIN termination request from the server. It is a **waiting state** for the client.

Step 3: ACK

The server sends the ACK (Acknowledgement) segment when it receives the FIN termination request. It depicts that the server is ready to close and terminate the connection.

Step 4: FIN _WAIT_2

The client waits for the FIN segment from the server. It is a type of approved signal sent by the server that shows that the server is ready to terminate the connection.

Step 5: FIN

The FIN segment is now sent by the server to the client. It is a confirmation signal that the server sends to the client. It depicts the successful approval for the termination.

Step 6: ACK

The client now sends the ACK (Acknowledgement) segment to the server that it has received the FIN signal, which is a signal from the server to terminate the connection. As soon as the server receives the ACK segment, it terminates the connection.

Tcp State Transition Diagram

- A TCP connection goes through a series of states during its lifetime. Figure 8.28 shows the state transition diagram. Each state transition is indicated by an arrow, and the associated label indicates associated events and actions.
- Connection establishment begins in the CLOSED state and proceeds to the ESTABLISHED state. Connection termination goes from the ESTABLISHED state to the CLOSED state. The normal transitions for a client are indicated by thick solid lines, and the normal transitions for a server are denoted by dashed lines.
- Thus when a client does an active open, it goes from the CLOSED state, to SYN_SENT, and then to ESTABLISHED. The server carrying out a passive open goes from the CLOSED state, to LISTEN, SYN_RCVD, and then to ESTABLISHED.
- The client normally initiates the termination of the connection by sending a FIN. The associated state trajectory goes from the ESTABLISHED state, to FIN_WAIT_1 while it waits for an ACK, to FIN_WAIT_2 while it waits for the other side's FIN, and then to TIME WAIT after it sends the final ACK.
- When the TIME_WAIT 2MSL period expires, the connection is closed and the transmission control block that stores all the TCP connection variables is deleted. Note that the state transition diagram does not show all error conditions that may arise, especially in relation to the TIME_WAIT state.
- The server normally goes from the ESTABLISHED state to the CLOSE_WAIT state after it receives a FIN, to the LAST_ACK when it sends its FIN, and finally to CLOSE when it receives the final ACK.

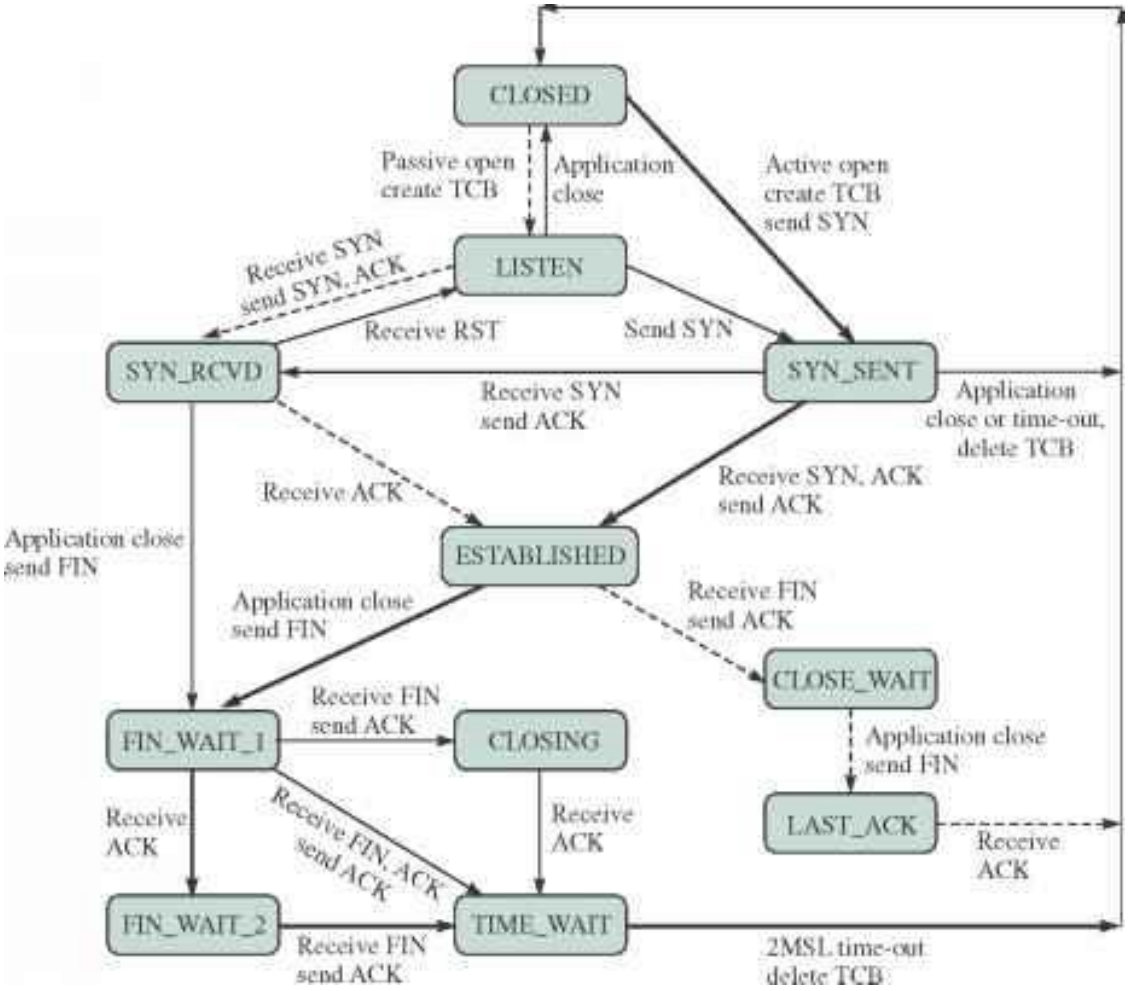


FIGURE 8.28 TCP state transition diagram

Note: The thick solid line is the normal state trajectory for a client; the dashed line is the normal state trajectory for a server transmission control block that stores all the TCP connection variables is deleted. Note that the state transition diagram does not show all error conditions that may arise, especially in relation to the TIME_WAIT state. The server normally goes from the ESTABLISHED state to the CLOSE_WAIT state after it receives a FIN, to the LASTACK when it sends its FIN, and finally to CLOSE when it receives the final ACK

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

Windows in TCP

TCP keeps track the amount of data allowed to be sent at a given point; this is dictated by the host receiving the data, the host sending the data and the conditions of the network.

There are three TCP windows used in a TCP connection:

- Receive Window (RWIN)
- Send Window (SWIN)
- Congestion Window (CWIN)

Each window serves an important purpose for the flow of data between the TCP sender and TCP receiver. Here they are referred to as sender and receiver, in practice they are usually referred to as client and server. The reason for this is that data can flow in either direction client to server or server to client.

- The RWIN dictates how much data a TCP receiver is willing to accept before sending an acknowledgement (ACK) back to the TCP sender. The receiver will advertise its RWIN to the sender and thus the sender knows it can't send more than an RWINs worth of data before receiving and ACK from receiver.

- The SWIN dictates how much data a TCP sender will be allowed send before it must receive an ACK from the TCP receiver.

- The CWIN is a variable that changes dynamically according to the conditions of the network. If data is lost or delivered out-of-order the CWIN is typically reduced.

- The RWIN and SWIN are configurable values on the host, while the CWIN is dynamic and can't be configured.

Flow Control and Congestion Control in TCP

1. Overview

Two popular traffic controlling methods in TCP: flow and congestion control. First, we'll present in detail how flow and congestion control works in TCP.

Finally, **we'll talk about the core differences between them.**

2. Flow Control in TCP

Flow control deals with the amount of data sent to the receiver side without receiving any acknowledgment. **It makes sure that the receiver will not be overwhelmed with data.** It's a kind of speed synchronization process between the sender and the receiver. **The** data link layer **in the** OSI model **is responsible for facilitating flow control.**

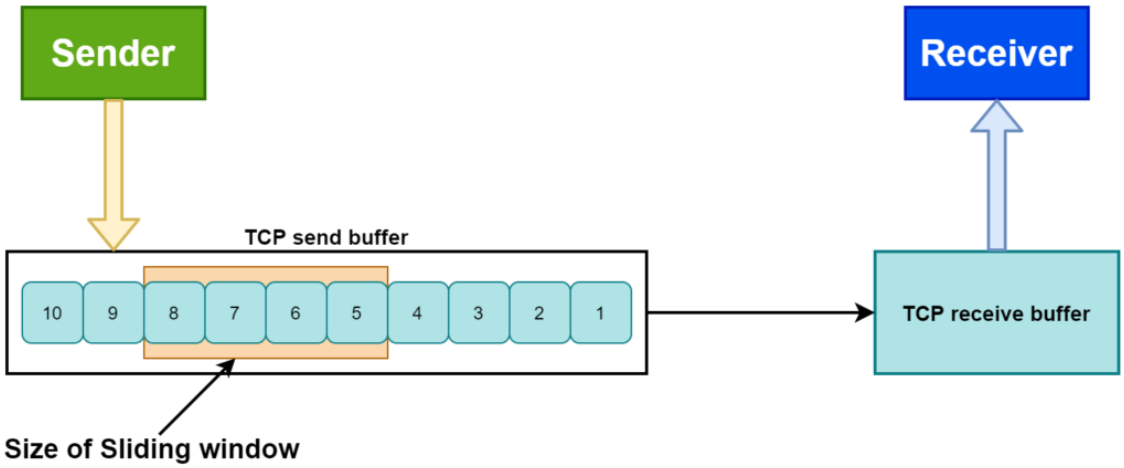
Let’s take a practical example to understand flow control. Jack is attending a training session. Let’s assume he’s slow in grasping concepts taught by the teacher. On the other hand, the teacher is teaching very fast without taking any acknowledgment from the students.

After some time, every word that comes out from his teacher is overflowing over Jack’s head. Hence, he doesn’t understand anything. Here, the teacher should be having information about how many concepts a student can handle at a time.

After some time, Jack requested the teacher to slow down the pace as he was overwhelmed with the data. The teacher decided to teach some of the concepts first and then wait for acknowledgment from the students before proceeding to the following concepts.

Similar to the example, **the flow control mechanism tells the sender the maximum speed at which the data can be sent to the receiver device.** The sender adjusts the speed as per the receiver’s capacity to reduce the frame loss from the receiver side.

Flow control in TCP ensures that it’ll not send more data to the receiver in the case when the receiver buffer is already completely filled. The receiver buffer indicates the capacity of the receiver. The receiver won’t be able to process the data when the receiver buffer is full:



3. Introduction to Sliding Window Protocol

One of the popular flow control mechanisms in TCP is the sliding window protocol. It's a byte-oriented process of variable size.

In this method, when we establish a connection between the sender and the receiver, the receiver sends the receiver window to the sender. The receiver window is the size that is currently available in the receiver's buffer.

Now **from the available receiver window, TCP calculates how much data can be sent further without acknowledgment.** Although, if the receiver window size is zero, TCP halts the data transmission until it becomes a non-zero value.

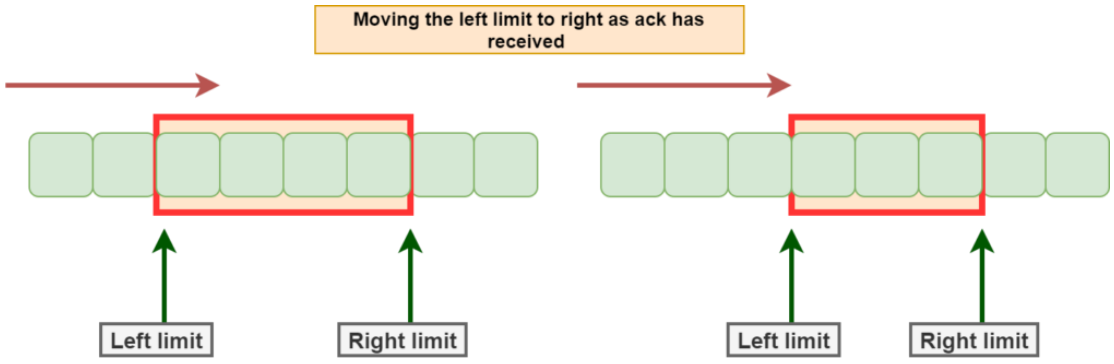
The receiver window size is the part of the frame of TCP segments. **The length of the window size is 16 bits, which means that the maximum size of the window is 65,535 bytes.**

The receiver decides the size of the window. The receiver sends the currently available receiver window size with every acknowledgment message.

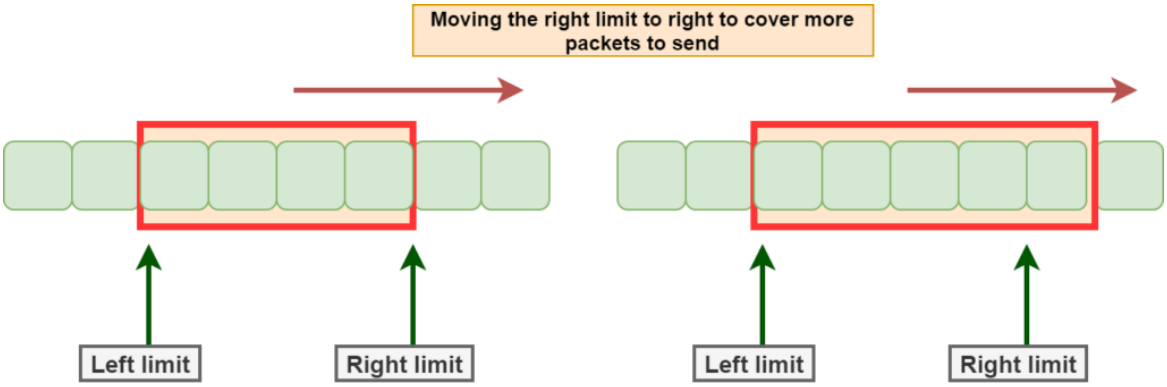
3.1. Processes in Sliding Window Protocol

There are three processes in the sliding window protocol: opening window, closing window, shrinking window.

The opening window process allows more data to be in the buffer that can be sent:



The closing window process indicates some sent data bytes have been acknowledged, and the sender needs not to worry about them anymore:

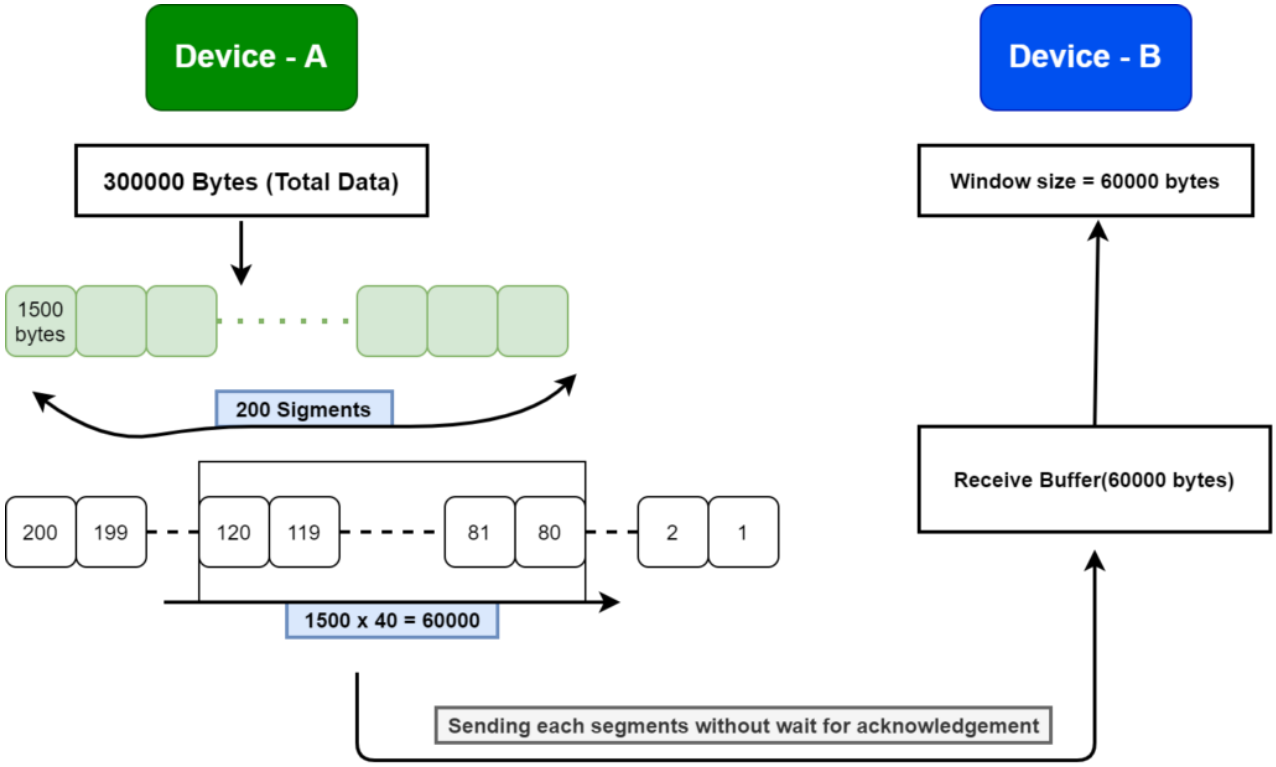


The shrinking windows process denotes the decrement in the window size. It means some bytes are eligible to transmit. Although because of the decrement in the window size, the sender won't transfer those bytes. The receiver can open or close the window but shrinking the window is not recommended.

3.2. Example

Suppose there are two devices: Device-A and Device-B. **Device-A wants to send 300000 bytes of data to Device-B.** Firstly, TCP breaks the data into small frames of size 1500 bytes each. **Hence, there are 200 packets in total.** In the beginning, let's assume Device-B notifies the available receiver window to Device-A, which is 60000 bytes. It means Device-A can send 60000 bytes to Device-B without receiving the acknowledgment from Device-B.

Now Device-A can send up to 40 packets (1500*40 = 60000) without waiting for an acknowledgment. Therefore, **Device-A has to wait for five acknowledgment messages to complete the whole transmission.** The number of acknowledgment messages needed is calculated and by considering the receiver buffer's capacity of Device-B:



Here, TCP is responsible for arranging the data packets and making the window size 60000 bytes. Furthermore, TCP helps in transmitting the packets which fall in the window range from Device-A to Device-B. After sending the packets, it waits for the acknowledgment. If the acknowledgment has been received, it slides the window to the next 60000 bytes and sends the packets.

We know TCP stops the transmission when the receiver sends the zero-size window. However, there is a possibility that the receiver has sent an acknowledgment but was missed by the sender.

In that situation, the receiver waits for the next packet. But on the other hand, the sender is waiting for non-zero window size. Hence, **this situation would result in a deadlock.** To handle this case, TCP starts a persist timer when it receives a zero-window size. It also sends packets periodically to the receiver.

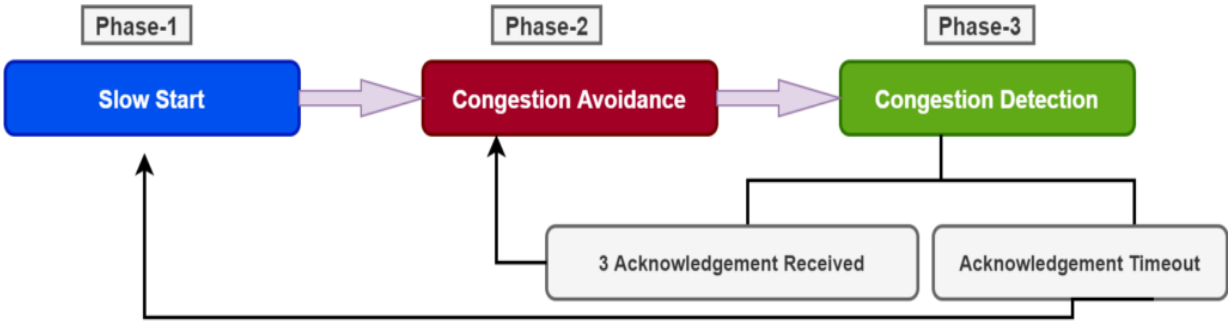
Congestion Control In TCP

Congestion control is a mechanism that limits the flow of packets at each node of the network. Thus, it prevents any node from being overloaded with excessive

packets. Congestion occurs when the rate of flow of packets towards any node is more than its handling capacity.

When congestion occurs, it slows down the network response time. As a result, the performance of the network decreases. Congestion occurs because routers and switches have a queue buffer, holding the packets for processing. After the holding process completes, they pass the packet to the next node, resulting in congestion in the network.

There are three phases that TCP uses for congestion control: slow start, congestion avoidance, and congestion detection:



4.1. Slow Start

In the first phase of congestion control, **the sender sends the packet and gradually increases the number of packets until it reaches a threshold.** We know from the flow control discussion, the window size is decided by the receiver.

In the same way, there is a window size in the congestion control mechanism. It increases gradually. In the flow control, the window size (RWND) is sent with the TCP header. Although in congestion control, the sender node or the end device stores it.

The sender starts sending packets with window size (CWND) of 1 MSS (Max Segment Size). MSS denotes the maximum amount of data that the sender can send in a single segment. Initially, the sender sends only one segment and waits for the acknowledgment for segment 1. After that, the sender increases the window size (CWND) by 1 MSS each time.

The slow start process can't continue for an indefinite time. Therefore, we **generally use a threshold known as the slow start threshold (SSTHRESH) in order to terminate the slow start process.** When the window size (CWND)

reaches the slow-start threshold, the slow start process stops, and the next phase of congestion control starts. **In most implementations, the value of the slow start threshold is 65,535 bytes.**

4.2. Congestion Avoidance

The next phase of congestion control is congestion avoidance. In the slow start phase, window size (CWND) increases exponentially. Now, if we continue increasing window size exponentially, there'll be a time when it'll cause congestion in the network.

To avoid that, we use a congestion avoidance algorithm. The congestion avoidance algorithm defines how to calculate window size (CWND):

After each acknowledgment, **TCP ensures the linear increment in window size (CWND), slower than the slow start phase.**

4.3. Congestion Detection

The third phase is congestion detection. Now in the congestion avoidance phase, we've decreased the rate of increment in window size (CWND) in order to reduce the probability of congestion in the network. If there is still congestion in the network, we apply a congestion detection mechanism.

There're two conditions when TCP detects congestion. Firstly, when there is no acknowledgment received for a packet sent by the sender within an estimated time. The second condition occurs when the receiver gets three duplicate acknowledgments.

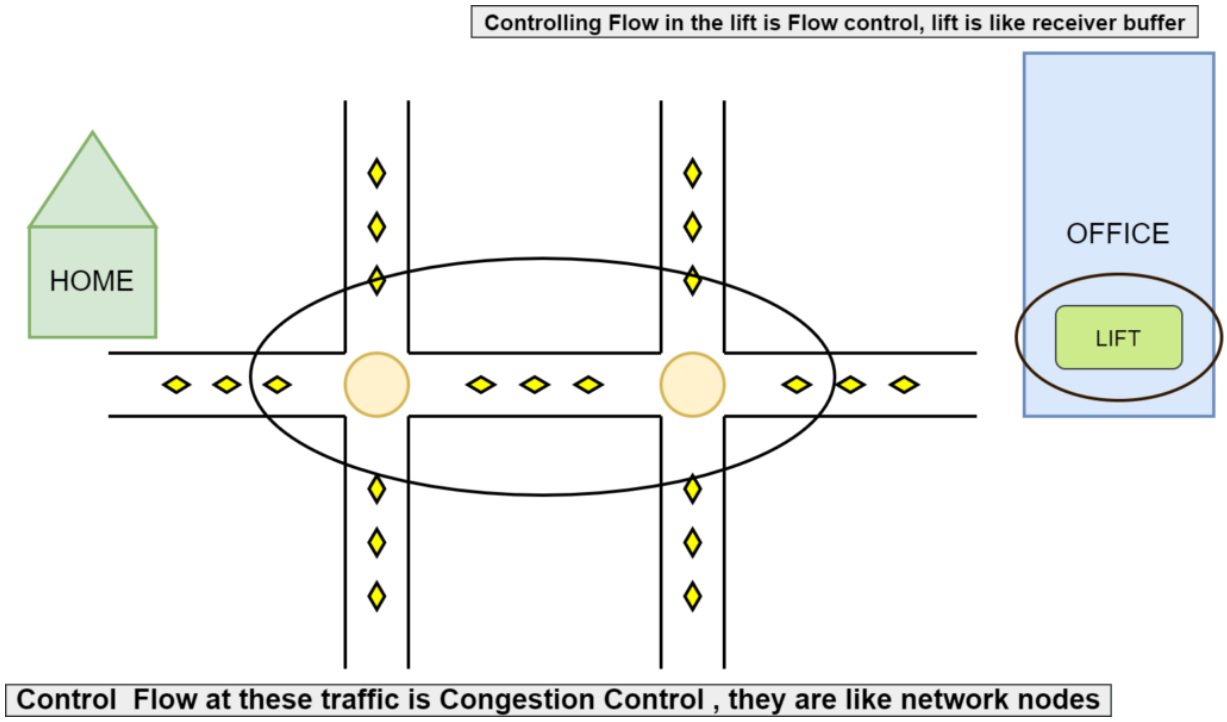
In the case of timeout, we start a new slow start phase. To handle the second situation, we begin a new congestion avoidance phase.

5. A Practical Example

To summarize the flow and congestion control in TCP, let's take a practical example.

Jack wakes up in the morning and gets ready for the office. On the way, he has to cross multiple road junctions, and there are traffic signals on each junction. In the office, there is a lift which drops him on his office floor.

In this example, all the traffic signals try to control the congestion from one junction to another. It works the same as maintaining the flow of segments from one node to another node in a network. Finally, when Jack reaches the office building, the flow control starts. The security guard controls access of the people for the lift as per the capacity of the lift. It's like the receiver buffer concept inflow control:



6. Differences

Core differences between flow and congestion control in TCP:

Flow Control	Congestion Control
Controls the traffic from sender to receiver.	Controls the network traffic.
Transport layer and data link layer is responsible for handling.	Transport layer and network layer is responsible for facilitating.
Makes sure that the receiver is not overwhelmed with data.	Prevents network congestion.
Only sender can send data.	Data is sent to network by the transport layer only.
Sender adjusts the speed as per the receiver's capacity in order to reduce traffic.	The transport layer slows down the transfer of data in order to reduce traffic.

Error control in TCP

Error control in TCP is mainly done through the use of **three simple techniques**:

1. **Checksum –** Every segment contains a checksum field which is used to find corrupted segments. If the segment is corrupted, then that segment is discarded by the destination TCP and is considered lost.
2. **Acknowledgement –** TCP has another mechanism called acknowledgement to affirm that the data segments have been delivered. Control segments that contain no data but have sequence numbers will be acknowledged as well but ACK segments are not acknowledged.
3. **Retransmission –** When a segment is missing, delayed to deliver to a receiver, corrupted when it is checked by the receiver then that segment is retransmitted again. Segments are retransmitted only during two events: when the sender receives three duplicate acknowledgements (ACK) or when a retransmission timer expires.
   - **Retransmission after RTO:** TCP always preserves one retransmission time-out (RTO) timer for all sent but not acknowledged segments. When the timer runs out of time, the earliest segment is retransmitted. Here no timer is set for acknowledgement. In TCP, the RTO value is dynamic in nature and it is updated using the round trip time (RTT) of segments. RTT is the time duration needed for a segment to reach the receiver and an acknowledgement to be received by the sender.

- **Retransmission after Three duplicate ACK segments:** RTO method works well when the value of RTO is small. If it is large, more time is needed to get confirmation about whether a segment has been delivered or not. Sometimes one segment is lost and the receiver receives so many out-of-order segments that they cannot be saved. In order to solve this situation, three duplicate acknowledgement method is used and missing segment is retransmitted immediately instead of retransmitting already delivered segment. This is a fast retransmission because it makes it possible to quickly retransmit lost segments instead of waiting for timer to end.

TCP Timers

TCP uses several timers to ensure that excessive delays are not encountered during communications. Several of these timers are elegant, handling problems that are not immediately obvious at first analysis. Each of the timers used by TCP is examined in the following sections, which reveal its role in ensuring data is properly sent from one connection to another.

TCP implementation uses four timers –

- **Retransmission Timer –** To retransmit lost segments, TCP uses retransmission timeout (RTO). When TCP sends a segment the timer starts and stops when the acknowledgment is received. If the timer expires timeout occurs and the segment is retransmitted. RTO (retransmission timeout is for 1 RTT) to calculate retransmission timeout we first need to calculate the RTT(round trip time).
  RTT three types –
  - **Measured RTT(RTTm) –** The measured round-trip time for a segment is the time required for the segment to reach the destination and be acknowledged, although the acknowledgement may include other segments.
  - **Smoothed RTT(RTTs)** – It is the weighted average of RTTm. RTTm is likely to change and its fluctuation is so high that a single measurement cannot be used to calculate RTO.

```
Initially -> No value

After the first measurement -> RTTs=RTTm
```

```
After each measurement -> RTTs= (1-t)*RTTs + t*RTTm

Note: t=1/8 (default if not given)
```

- **Deviated RTT(RTTd) –** Most implementations do not use RTTs alone so RTT deviated is also calculated to find out RTO.

```
Initially -> No value

After the first measurement -> RTTd=RTTm/2

After each measurement -> RTTd= (1-k)*RTTd + k*(RTTm-RTTs)

Note: k=1/4 (default if not given)
```

- **Persistent Timer –** To deal with a zero-window-size deadlock situation, TCP uses a persistence timer. When the sending TCP receives an acknowledgment with a window size of zero, it starts a persistence timer. When the persistence timer goes off, the sending TCP sends a special segment called a probe. This segment contains only 1 byte of new data. It has a sequence number, but its sequence number is never acknowledged; it is even ignored in calculating the sequence number for the rest of the data. The probe causes the receiving TCP to resend the acknowledgment which was lost.

- **Keep Alive Timer –** A keepalive timer is used to prevent a long idle connection between two TCPs. If a client opens a TCP connection to a server transfers some data and becomes silent the client will crash. In this case, the connection remains open forever. So a keepalive timer is used. Each time the server hears from a client, it resets this timer. The time-out is usually 2 hours. If the server does not hear from the client after 2 hours, it sends a probe segment. If there is no response after 10 probes, each of which is 75 s apart, it assumes that the client is down and terminates the connection.

- **Time Wait Timer –** This timer is used during tcp connection termination. The timer starts after sending the last Ack for 2nd FIN and closing the connection.
  After a TCP connection is closed, it is possible for datagrams that are still making their way through the network to attempt to access the closed port. The quiet timer is intended to prevent the just-closed port from reopening again quickly and receiving these last datagrams.
  The **quiet timer** is usually set to twice the maximum segment lifetime (the same value as the Time-To-Live field in an IP header), ensuring that all segments still heading for the port have been discarded.