CHAPTER 5-TRANSPORT LAYER

5.1INTRODUCTION FUNCTION AND SERVICES

Transport layer offers peer-to-peer and end-to-end connection between two processes on remote hosts. Transport layer takes data from upper layer (i.e. Application layer) and then breaks it into smaller size segments, numbers each byte, and hands it over to the lower layer (Network Layer) for delivery.

Specific functions of the transport layer are as follows:

1. Service-point addressing(Port Address)

• Computers often run many programs at the same time. Due to this, source-to-destination delivery means delivery from a specific job (currently running program) on one computer to a specific job (currently running program) on the other system, not only one computer to the next.

• For this reason, the transport layer adds a specific type of address to its header, it is referred to as a service point address or port address.

• By this address each packet reaches the correct computer and also the transport layer gets the complete message to the correct process on that computer.

2. Segmentation and Reassembly

• In segmentation, a message is divided into transmittable segments; each segment containing a sequence number. This number enables this layer to reassemble the message.

• Upon arriving at its destination, the system message is reassembled correctly, identifying and replacing packets that were lost in transmission.

3. Connection Control

It can be either of two types:

i. Connectionless Transport Layer

ii. Connection Oriented Transport Layer

i) Connectionless Transport Layer

• This Transport Layer treats each packet as an individual and delivers it to the destination machine.

• In this type of transmission, the receiver does not send an acknowledgment to the sender about the receipt of a packet. This is a faster communication technique.

ii) Connection Oriented Transport Layer

• This Transport Layer creates a connection with the Transport Layer at the destination machine before transmitting the packets to the destination.

• To Create a connection following three steps are possible:

o Connection establishment

o Data transfer

o Connection termination

When all the data is transmitted, the connection is terminated. Connectionless Service is less reliable than connection Oriented Service.

4. Multiplexing and Demultiplexing

• Multiple packets from diverse applications are transmitted across a network and need very dedicated control mechanisms, which are found in the transport layer.

• The transport layer accepts packets from different processes. These packets are differentiated by their port numbers and pass them to the network layer after adding proper headers.

• In Demultiplexing, at the receiver's side to obtain the data coming from various processes. It receives the segments of data from the network layer and delivers it to the appropriate process running on the receiver's machine.

5. Flow control

• The transport layer is also responsible for the flow control mechanism between the adjacent layers of the TCP/IP model.

• By imposing flow control techniques data loss can be prevented from the cause of the sender and slow receiver.

For instance, it uses the method of sliding window protocol in this method receiver sends a window back to the sender to inform the size of the data it received.

6. Error Control

• Error Control is also performed end to end like the data link layer.

• In this layer to ensure that the entire message arrives at the receiving transport layer without any error(damage, loss or duplication). Error Correction is achieved through retransmission of the packet.

• The data has arrived or not and checks for the integrity of data, it uses the ACK and NACK services to inform the sender.

5.5. TRANSMISSION CONTROL PROTOCOL (TCP)

The transmission Control Protocol (TCP) is one of the most important protocols of the Internet Protocols suite. It is the most widely used protocol for data transmission in communication networks such as the internet.

Features

- TCP is a reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has a bright clue about whether the data packet has reached the destination or it needs to resend it.
- TCP ensures that the data reaches the intended destination in the same order it was sent.
- TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
- TCP provides error-checking and recovery mechanisms.
- TCP provides end-to-end communication.
- TCP provides flow control and quality of service.
- TCP provides a full duplex server, i.e. it can perform roles of both receiver and sender.

SERVICES

The various services provided by the TCP to the application layer are as follows:

1. Process-to-Process Communication –

   TCP provides process to process communication, i.e, the transfer of data takes place between individual processes executing on end systems. This is done using port numbers or port addresses. Port numbers are 16 bit long that help identify which process is sending or receiving data on a host.

2. Stream oriented –

   This means that the data is sent and received as a stream of bytes. However, the network layer that provides service for the TCP, sends packets of information not streams of bytes. Hence, TCP groups a number of bytes together into a *segment* and adds a header to each of these segments and then delivers these segments to the network layer. At the network layer, each of these segments are encapsulated in an IP packet for transmission. The TCP header has information that is required for control purposes.

3. Full duplex service –

   This means that the communication can take place in both directions at the same time.

4. Connection oriented service –

   Unlike UDP, TCP provides connection oriented service. It defines 3 different phases:

    - Connection establishment
    - Data transfer
    - Connection termination

5. Reliability –

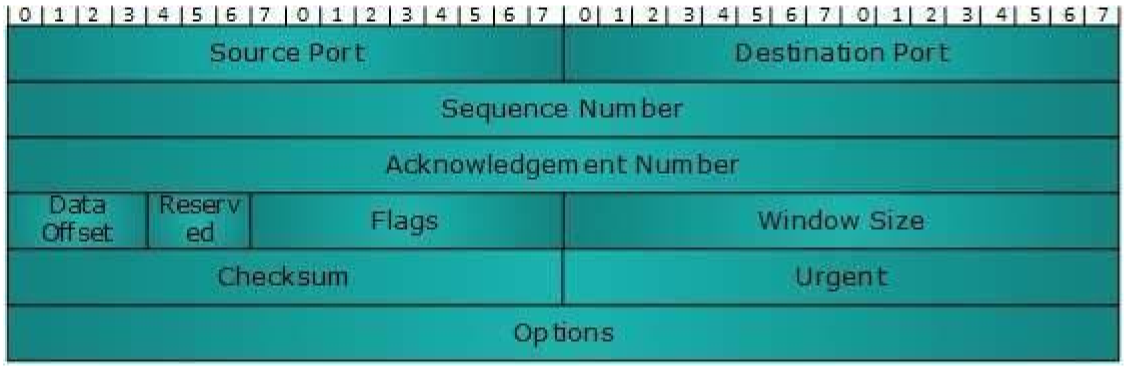
   TCP is reliable as it uses checksum for error detection, attempts to recover lost or corrupted packets by re-transmission, acknowledgement policy and timers. It uses features like byte number and sequence number and acknowledgement number so as to ensure reliability. Also, it uses congestion control mechanisms.

6. Multiplexing –

   TCP does multiplexing and demultiplexing at the sender and receiver ends respectively as a number of logical connections can be established between port numbers over a physical connection.

Header

The length of the TCP header is minimum 20 bytes long and maximum 60 bytes.



- Source Port (16-bits) - It identifies source port of the application process on the sending device.
- Destination Port (16-bits) - It identifies destination port of the application process on the receiving device.
- Sequence Number (32-bits) - Sequence number of data bytes of a segment in a session.

- Acknowledgement Number (32-bits) - When ACK flag is set, this number contains the next sequence number of the data byte expected and works as acknowledgement of the previous data received.
- Data Offset (4-bits) - This field implies both, the size of TCP header (32-bit words) and the offset of data in current packet in the whole TCP segment.
- Reserved (3-bits) - Reserved for future use and all are set zero by default.
- Flags (1-bit each)- Flags are used to indicate a particular state of connection or to provide some additional useful information like troubleshooting purposes or to handle a control of a particular connection. Most commonly used flags are "SYN", "ACK" and "FIN". Each flag corresponds to 1 bit information.
- Windows Size - This field is used for flow control between two stations and indicates the amount of buffer (in bytes) the receiver has allocated for a segment, i.e. how much data is the receiver expecting.
- Checksum - This field contains the checksum of Header, Data and Pseudo Headers.
- Urgent Pointer - It points to the urgent data byte if URG flag is set to 1.
- Options - It facilitates additional options which are not covered by the regular header. Option field is always described in 32-bit words.

5.3. USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is the simplest Transport Layer communication protocol available of the TCP/IP protocol suite. UDP is said to be an unreliable transport protocol but it uses IP services which provide the best effort delivery mechanism.

In UDP, the receiver does not generate an acknowledgement of the packet received and in turn, the sender does not wait for any acknowledgement of the packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

Requirement of UDP

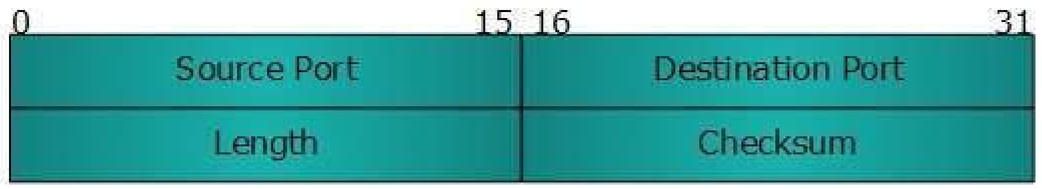
A question may arise, why do we need an unreliable protocol to transport the data? We deploy UDP where the acknowledgement packets share a significant amount of bandwidth along with the actual data. For example, in the case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amounts of bandwidth wastage. The best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost, the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features

- UDP is used when acknowledgement of data does not hold any significance.
- UDP is a good protocol for data flowing in one direction.
- UDP is simple and suitable for query based communications.
- UDP is not connection oriented.
- UDP does not provide a congestion control mechanism.
- UDP does not guarantee ordered delivery of data.
- UDP is stateless.
- UDP is a suitable protocol for streaming applications such as VoIP, multimedia streaming.

UDP Header

The UDP header is as simple as its function.



UDP header contains four main parameters:

- Source Port - This 16 bits information is used to identify the source port of the packet.
- Destination Port - This 16 bits information, is used identify application level service on destination machine.
- Length - Length field specifies the entire length of UDP packet (including header). It is a 16-bits field and minimum value is 8-byte, i.e. the size of the UDP header itself.
- Checksum - This field stores the checksum value generated by the sender before sending. IPv4 has this field as optional so when the checksum field does not contain any value it is made 0 and all its bits are set to zero.

UDP Operation

What UDP Does

UDP's only real task is to take data from higher-layer protocols and place it in UDP messages, which are then passed down to the Internet Protocol for transmission. The basic steps for transmission using UDP are:

1. Higher-Layer Data Transfer: An application sends a message to the UDP software.

2. UDP Message Encapsulation: The higher-layer message is encapsulated into the *Data* field of a UDP message. The headers of the UDP message are filled in, including the *Source Port* of the application that sent the data to UDP, and the *Destination Port* of the intended recipient. The checksum value may also be calculated.

3. Transfer Message To IP: The UDP message is passed to IP for transmission.

UDP application

Here are few applications where UDP is used to transmit data:

- Domain Name Services
- Simple Network Management Protocol
- Trivial File Transfer Protocol
- Routing Information Protocol

No.	TCP	UDP
1	This Connection oriented protocol	This is connection-less protocol
2	The TCP connection is byte stream	The UDP connection is a message stream
3	It does not support multicasting and broadcasting	It supports broadcasting
4	It provides error control and flow control	The error control and flow control is not provided
5	TCP supports full duplex transmission	UDP does not support full duplex transmission
6	It is reliable service of data transmission	This is an unreliable service of data transmission
7	The TCP packet is called as segment	The UDP packet is called as user datagram.

5.6 PRINCIPLE OF CONGESTION CONTROL

CONGESTION CONTROL

An important issue in a packet-switched network is congestion. Congestion in a network may occur if the load on the network-the number of packets sent to the

network-is greater than the capacity of the network-the number of packets a network can handle.
Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.

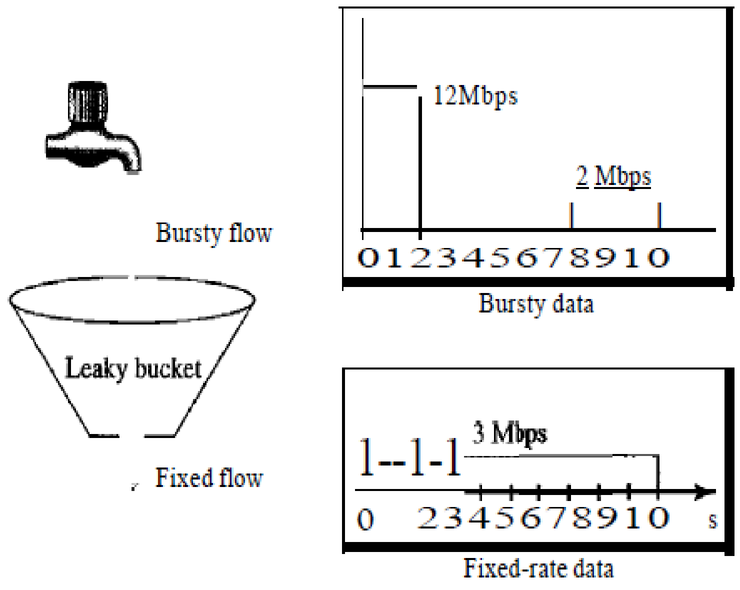
A state occurring in network layer when the message traffic is so heavy that it slows down network response
time.
Effects of Congestion
• As delay increases, performance decreases.
• If delay increases, retransmission occurs, making the situation worse.

There are two traffic shaping algorithm:

1) **Leaky Bucket**

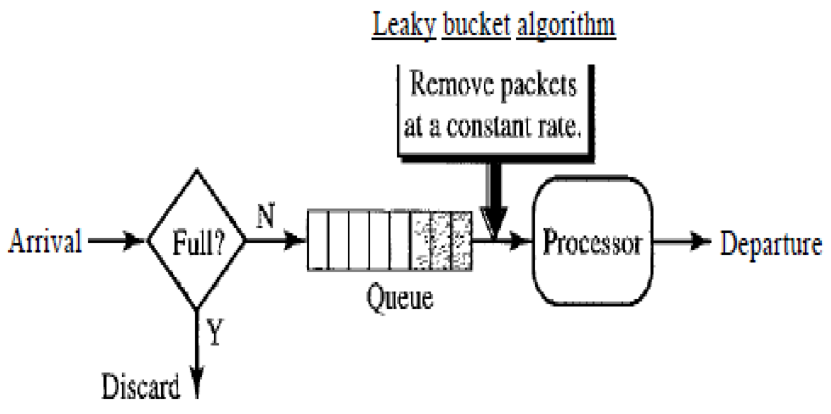
If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. ***The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic.*** Bursty chunks are stored in the bucket and sent out at an average rate.

Figure 24.19 *Leaky bucket*



In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In Figure 24.19 the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in lOs. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s. Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion. As an analogy, consider the freeway during rush hour (bursty traffic). If, instead, commuters could stagger their working hours, congestion on our freeways could be avoided.

Figure 24.20 *Leaky bucket implementation*



The following is an algorithm for variable-length packets:

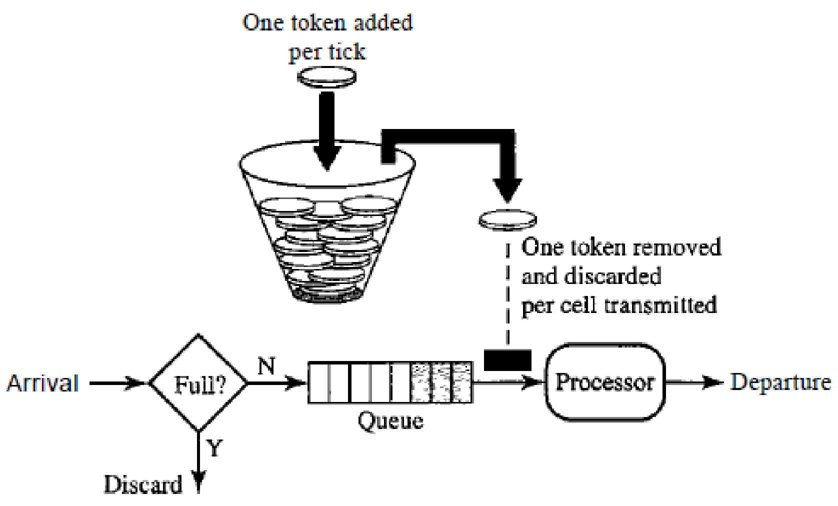
1. Initialize a counter to *n* at the tick of the clock.
2. If *n* is greater than the size of the packet, send the packet and decrement the counter by the packet size.
3.Repeat this step until *n* is smaller than the packet size.
4.Reset the counter and go to step 1.

2. Token Bucket

The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account.
On the other hand, the token bucket algorithm allows idle hosts to accumulate credit for the future in the form of tokens. For each tick of the clock, the system sends n tokens to the bucket. The system removes one token for every cell (or byte) of data sent.
For example, if *n* is 100 and the host is idle for 100 ticks, the bucket collects 10,000 tokens. Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1000 ticks with 10 cells per tick. In other words, the host

can send bursty data as long as the bucket is not empty. Figure 24.21 shows the idea. The token bucket can easily be implemented with a counter. The token is initialized to zero. ***Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.***

Figure 24.21 *Token bucket*



5.2. Elements of Transport Protocols: Addressing, Establishing and Releasing Connection, Flow Control & Buffering, Error Control, Multiplexing & Demultiplexing, Crash Recovery

5.2.1. 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.

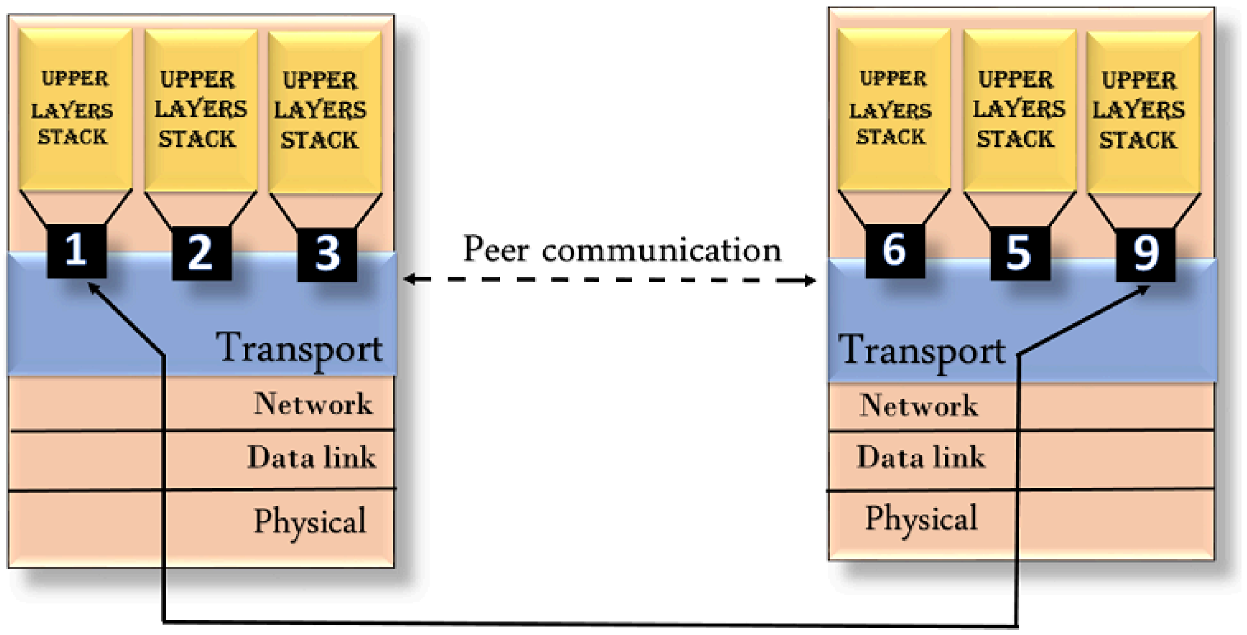
- The transport layer provides the user address which is specified as a station or port. The port variable represents a particular TS user of a specified station known as a Transport Service access point (TSAP).
- The transport layer protocols need to know which upper-layer protocols are communicating.

PORT ADDRESS

A port number **is a 16-bit address used to identify any client-server program uniquely**.

- **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.
- Well-known ports. The well known ports are those from 0 - 1,023. ...
- Registered ports. The registered ports are those from 1,024 - 49,151. ...

- Dynamic and/or private ports. The dynamic and/or private ports are those from 49,152 - 65,535.



5.2.2. ESTABLISHING AND RELEASING CONNECTION IN TRANSPORT LAYER

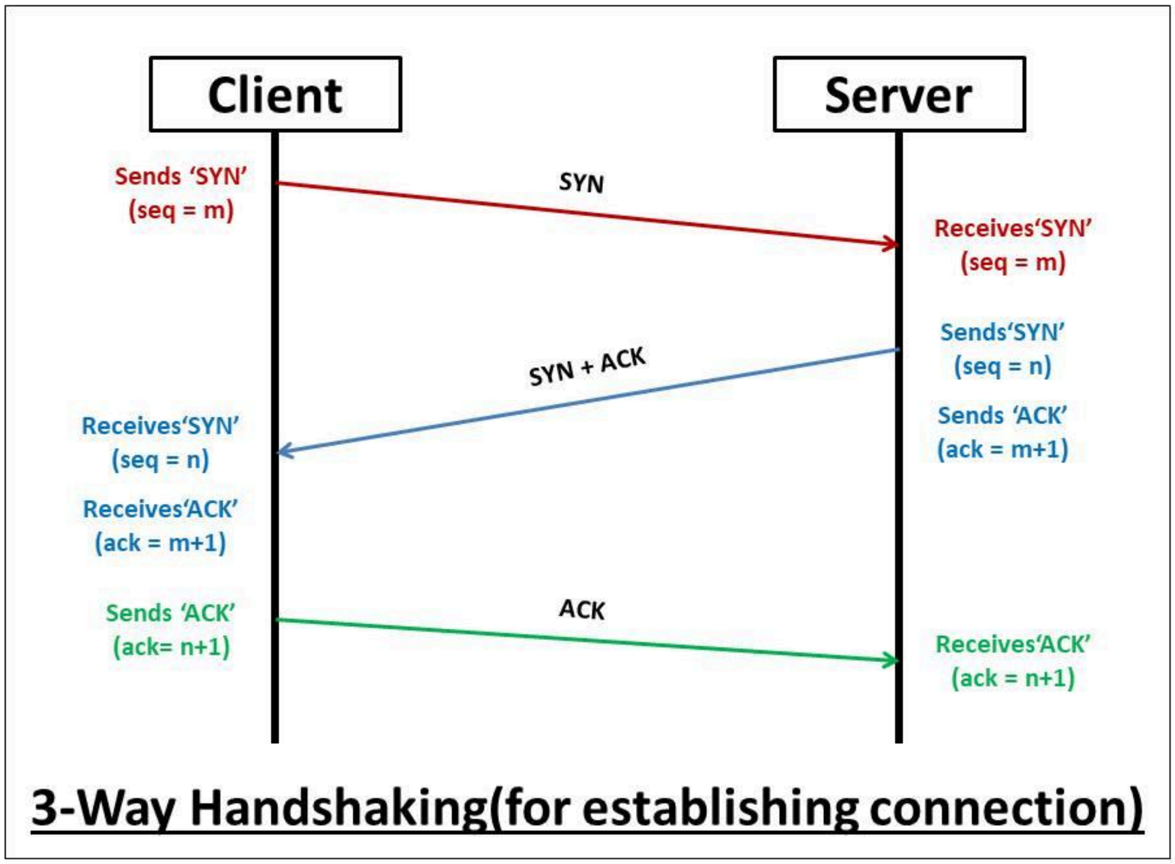
TCP provides us with a secure and reliable connection link between two devices. And, this is possible only due to the 3-way handshake process that takes place in the TCP during establishing and closing connections between two devices. As the name suggests, there are three steps for both establishing and closing the connection.

The 3-Way Handshake process is the defined set of steps that takes place in the TCP for creating a secure and reliable communication link and also closing it.

Actually, TCP uses the 3-way handshake process to establish a connection between two devices before transmitting the data. After the establishment of the connection, the data transfer takes place between the devices. After which the connection needs to be terminated, which is also done by using the 3-way handshake process. The secure and reliable connection is established to reserve the CPU, buffer, and bandwidth of the devices to communicate properly. Thus, it is a must to free these resources by terminating the connection after data transmission. Hence, the TCP 3-way handshake process can be used to establish and terminate connections in the network in a secure way.

1. SYN Flag: SYN stands for synchronization. It can be described as a request for establishing a connection. If SYN is 1, it means that the device wants to establish a secure connection, else not.
2. ACK Flag: ACK stands for acknowledgement. It can be described as the response of SYN. If ACK is 1, the device has received the SYN message and acknowledges it, else not.
3. FIN Flag: FIN stands for Finished. After the data transmission has been completed, devices have to terminate the connection using the FIN flag. If FIN is 1, the device wants to terminate the connection, else not.

Below is the pictorial representation of the connection establishment using the 3-way handshake process.



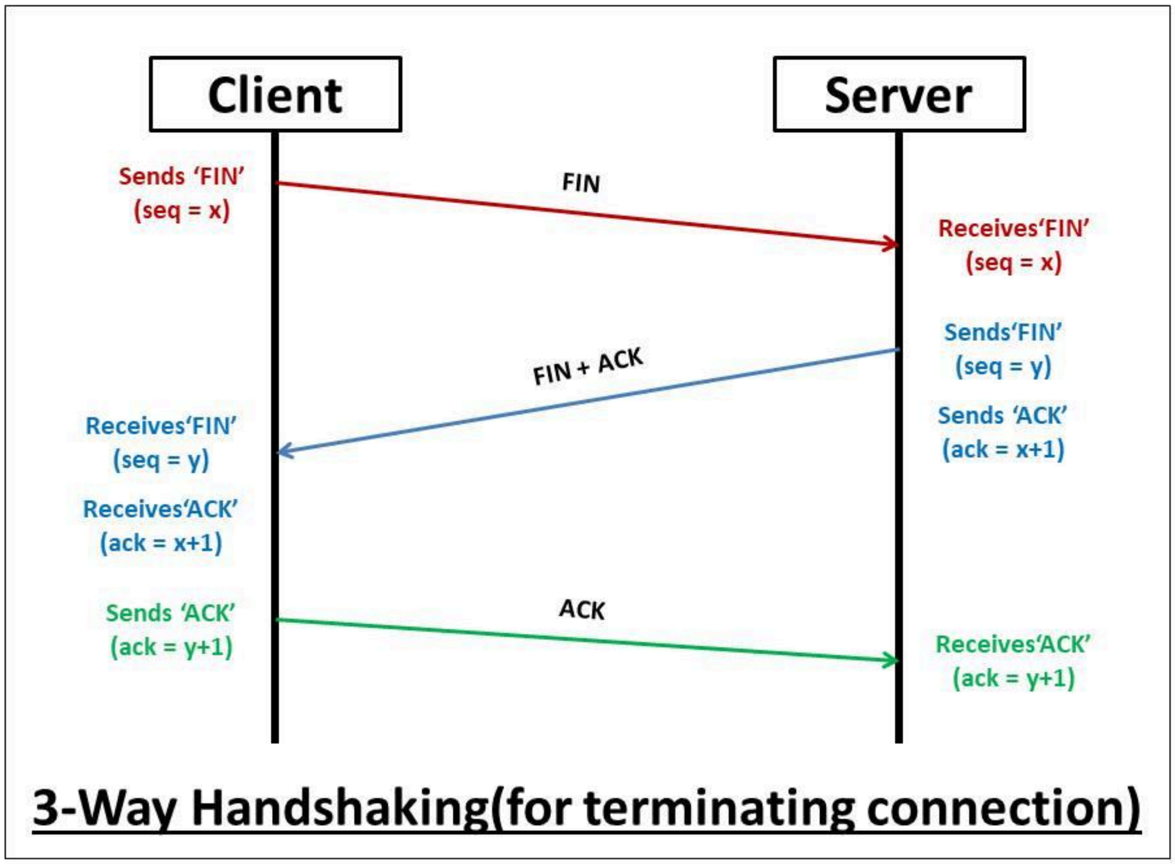
Following are the three steps involved in establishing the connection using the 3-way handshake process in TCP:

1. The client sends the SYN to the server: When the client wants to connect to the server. It sets the 'SYN' flag as 1 and sends the message to the server. The message also has some additional information like the sequence number(any random 32 bits number), the ACK is set here to 0, the window size, and the maximum segment size.
2. The server replies with the SYN and the ACK to the client: After receiving the client's synchronization request, the server sends an acknowledgement to the client by setting the ACK flag to '1'. The acknowledgement number of

the ACK is one more than the received sequence number.

3. The client sends the ACK to the server: After receiving the SYN from the server, the client sets the ACK flag to '1' and sends it with an acknowledgement number 1 greater than the server's SYN sequence number to the client. Here, the SYN flag is kept '0'.
4. After completion of this step, the connection is now established from the server to the client-side also. After the connection is established, the minimum of the sender's and receiver's maximum segment size is taken under consideration for data transmission.

Below is the pictorial representation of the connection termination using the 3-way handshake process.



Following are the three steps involved in terminating the connection using the 3-way handshake process in TCP:

1. The client sends the FIN to the server: When the client wants to terminate the connection. It sets the FIN flag as '1' and sends the message to the server with a random sequence number. Here, the ACK is set to 0.
2. The server replies with the FIN and the ACK to the client: After receiving the client's termination request, the server sends an acknowledgement to the client by setting the ACK flag to '1'. The acknowledgement number of the

ACK is one more than the received sequence number. For Example, if the client has sent the FIN with sequence number = 1000, then the server will send the ACK with acknowledgement number = 1001. Also, the server sets the FIN flag to '1' and sends it to the client, if the server also wants to terminate the connection. The sequence number used here for the FIN will be different from the client's FIN. After completion of this step, the connection is terminated from the client to the server-side.

3. The client sends the ACK to the server: After receiving the FIN from the server, the client sets the ACK flag to '1' and sends it with an acknowledgement number 1 greater than the server's FIN sequence number to the client. Here, the FIN flag is kept '0'. After completion of this step, the connection is now terminated from the server to the client-side also.

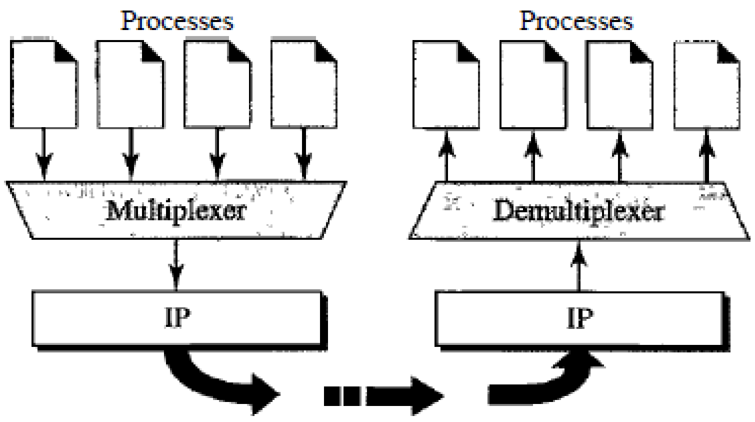
5.2.4.MULTIPLEXING AND DEMULTIPLEXING

Multiplexing

- 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.

Demultiplexing

- At the receiver site, the relationship is one-to-many and requires demultiplexing.
- The transport layer receives datagrams from the network layer.
- After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.



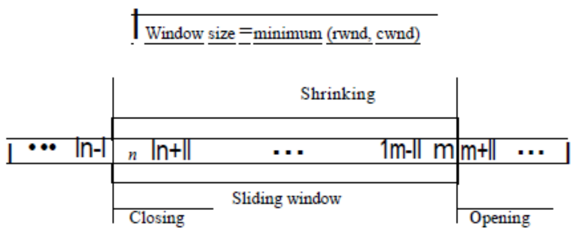
5.2.3.FLOW CONTROL AND BUFFERING

- TCP uses a sliding window to handle flow control.
- The sliding window protocol used by TCP, however, is something between the Go-Back-N and Selective Repeat sliding window.

NOTE: GO BACK N AND SELECTIVE REPEAT STUDY FROM DATA LINK LAYER

- The sliding window protocol in TCP looks like

Figure 23.22 *Sliding window*

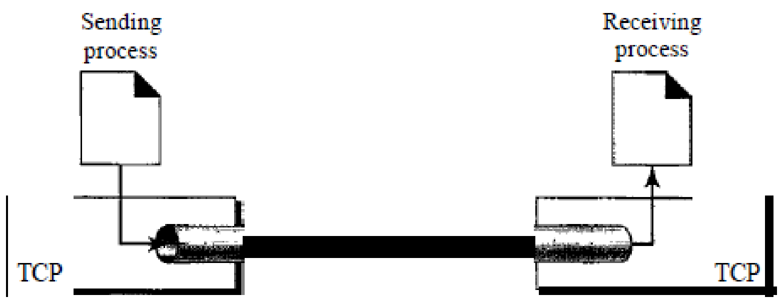

Window size = minimum (rwnd, cwnd)
Shrinking
... n-1 n n+1 ... m-1 m m+1 ...
Sliding window
Closing
Opening

- A sliding window is used to make transmission more efficient to control the flow of data so that the destination does not become overwhelmed with data.
- TCP sliding windows are byte-oriented.
- Some points about TCP sliding windows:
- o The size of the window is the lesser of rwnd and cwnd.
- o The source does not have to send a full window's worth of data.
- o The window can be opened or closed by the receiver, but should not be shrunk.
- o The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
- o The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut down.

Stream Delivery Service: Buffer

- TCP, on the other hand, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes.
- TCP creates an environment in which the two processes seem to be connected by an imaginary "tube“ that carries their data across the Internet.

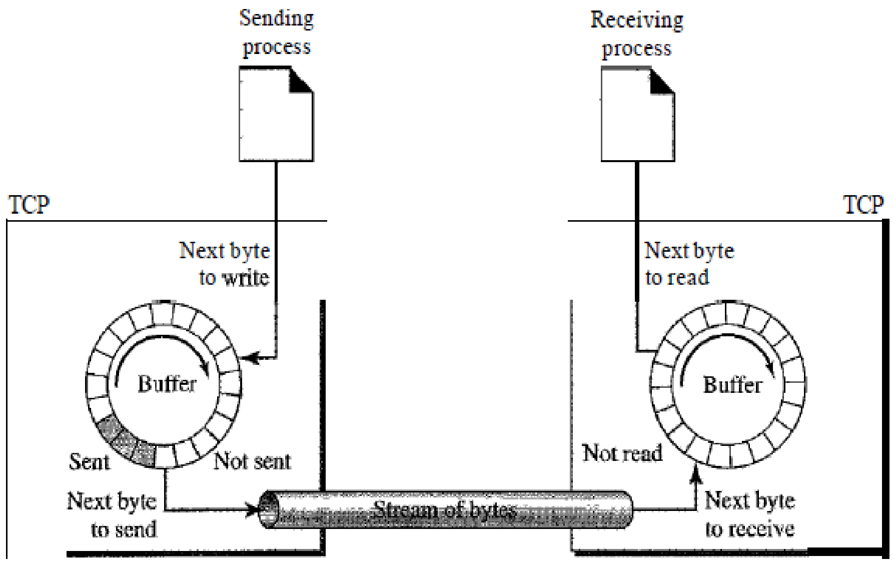
Figure 23.13 *Stream delivery*



Sending and Receiving Buffers

- Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage.
- There are two buffers, the sending buffer and the receiving buffer, one for each direction

Figure 23.14 *Sending and receiving buffers*



5.2.5. CRASH RECOVERY

TCP is a very reliable protocol. It provides a sequence number to each byte sent in the segment. It provides the feedback mechanism i.e. when a host receives a packet, it is bound to ACK that packet having the next sequence number expected (if it is not the last segment).

When a TCP Server crashes mid-way communication and re-starts its process it sends a TPDU(Transport Protocol Data Unit) broadcast to all its hosts. The hosts can then send the last data segment which was never unacknowledged and carry onwards.

5.4. RELIABLE DATA TRANSFER

Transport Layer Protocols are the central layer of layered architectures, which provides the logical communication between application processes. These processes use logical communication to transfer data from transport layer to network layer and this transfer of data should be reliable and secure. **The data is transferred in the form of packets but the problem occurs in reliable transfer of data.**

The problem of transferring the data occurs not only at the transport layer, but also at the application layer as well as in the link layer.

This problem occurs when a reliable service runs on an unreliable service, For example, TCP (Transmission Control Protocol) is a reliable data transfer protocol that is implemented on top of an unreliable layer, i.e., Internet Protocol (IP) is an end to end network layer protocol.

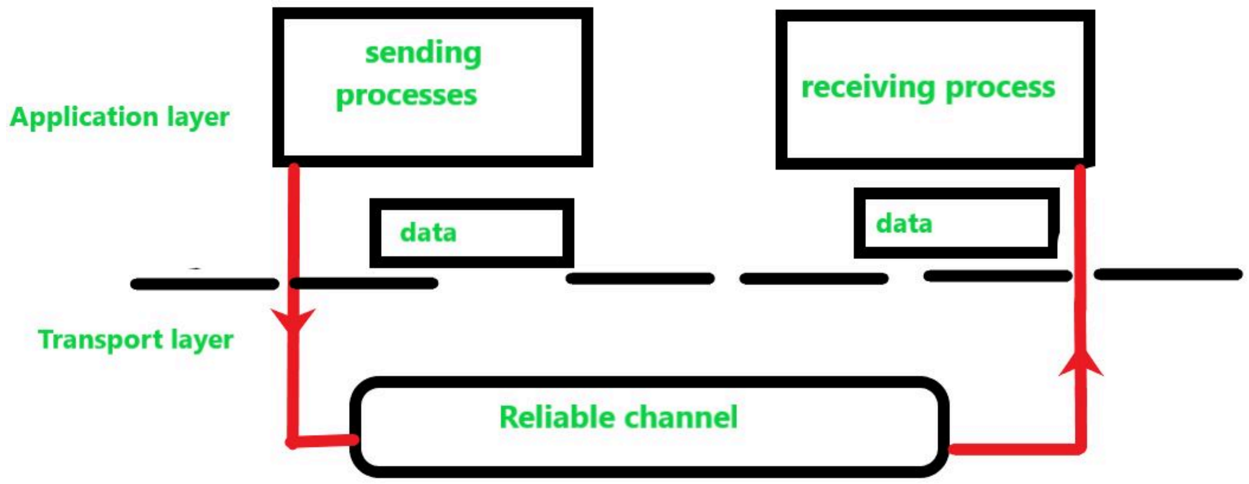


Figure: Study of Reliable Data Transfer

In this model, we have designed the sender and receiver sides of a protocol over a reliable channel.

In the reliable transfer of data the layer receives the data from the above layer, breaks the message in the form of segment and puts the header on each segment and transfers.

Below layer below receives the segments and removes the header from each segment and makes it a packet by adding to the header.

The data which is transferred from the above has no transferred data bits corrupted or lost, and all are delivered in the same sequence in which they were sent to the below layer; this is a reliable data transfer protocol.

This service model is offered by TCP to the Internet applications that invoke this transfer of data.

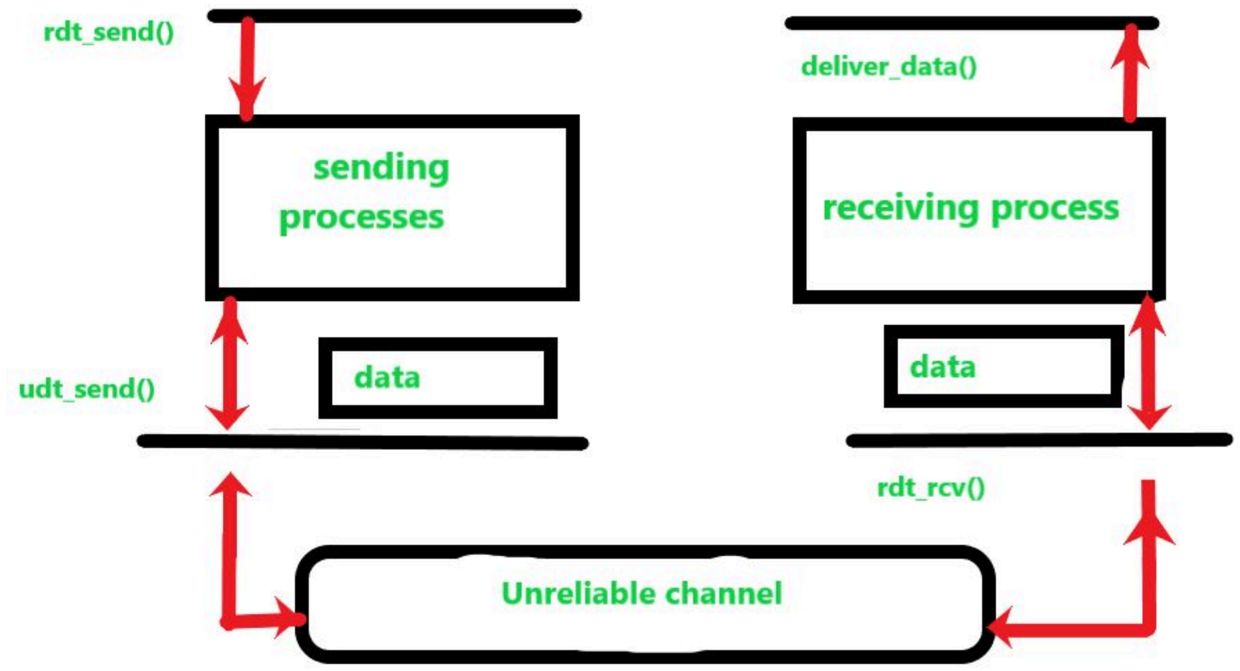


Figure: Study of Unreliable Data Transfer

Similarly in an unreliable channel we have designed the sending and receiving side.

The sending side of the protocol is called from the above layer to **rdt_send()** then it will pass the data that is to be delivered to the application layer at the receiving side (here

rdt-send() is a function for sending data where rdt stands for reliable data transfer protocol and _send() is used for the sending side).

On the receiving side, **rdt_rcv()** (rdt_rcv() is a function for receiving data where -rcv() is used for the receiving side), and will be called when a packet arrives from the receiving side of the unreliable channel.

When the rdt protocol wants to deliver data to the application layer, it will do so by calling deliver_data() (where deliver_data() is a function for delivering data to the upper layer).

In reliable data transfer protocol, we only consider the case of unidirectional data transfer, that is transfer of data from the sending side to receiving side(i.e. only in one direction). In case of bidirectional (full duplex or transfer of data on both the sides) data transfer is conceptually more difficult.

Although we only consider unidirectional data transfer, it is important to note that the sending and receiving sides of our protocol will need to transmit packets in both directions, as shown in above figure.

In order to exchange packets containing the data that is needed to be transferred the both (sending and receiving) sides of rdt also need to exchange control packets in both direction (i.e., back and forth), both the sides of rdt send packets to the other side by a call to **udt_send()** (udt_send() is a function used for sending data to other side where udt stands for unreliable data transfer protocol).

Reliable Data Transfer Protocols: Pipelined (Selective Repeat)

Stop-and-Wait is called Stop-and-Wait since the sender has to wait until the acknowledgement for the one outstanding packet arrives or until the timeout for the one outstanding packet expires.

In some cases, this means that the **sender can be idle for a large fraction of the time.**

If the time it takes for a packet to reach the receiver is much longer than the time it takes for the sender to transmit a packet, then the sender will spend most of its time idle waiting for the acknowledgment.

If the sender needs to send several packets, then the time until the last of the packets is sent might be quite long due to this waiting.

Pipelined reliable data transfer protocols address this problem.

In a pipelined reliable data transfer protocol, **the sender can start sending a second data packet before the sender receives the acknowledgment** for the first data packet.

Thus, if the sender needs to send several packets, then the time until the last of the packets is sent will be shorter with a pipelined protocol.

Thus, **a pipelined protocol can have better performance than the Stop-and-Wait protocol.**

Errors can occur when a pipelined protocol is used, just as they can occur when the stop-and-wait protocol is used.

The same two cases exist:

1) a lost or corrupted data packet or

2) a lost or corrupted acknowledgement packet.

There are two approaches to this problem.

In one approach, called Go-Back-N, the sender retransmits all the data packets it had sent since the lost or corrupted data packet.

In the second approach, called Selective Repeat, the sender just retransmits the lost or corrupted data packet. Selective Repeat causes fewer packets to be retransmitted.

QUESTION: Describe the working of go back n and selective repeat .

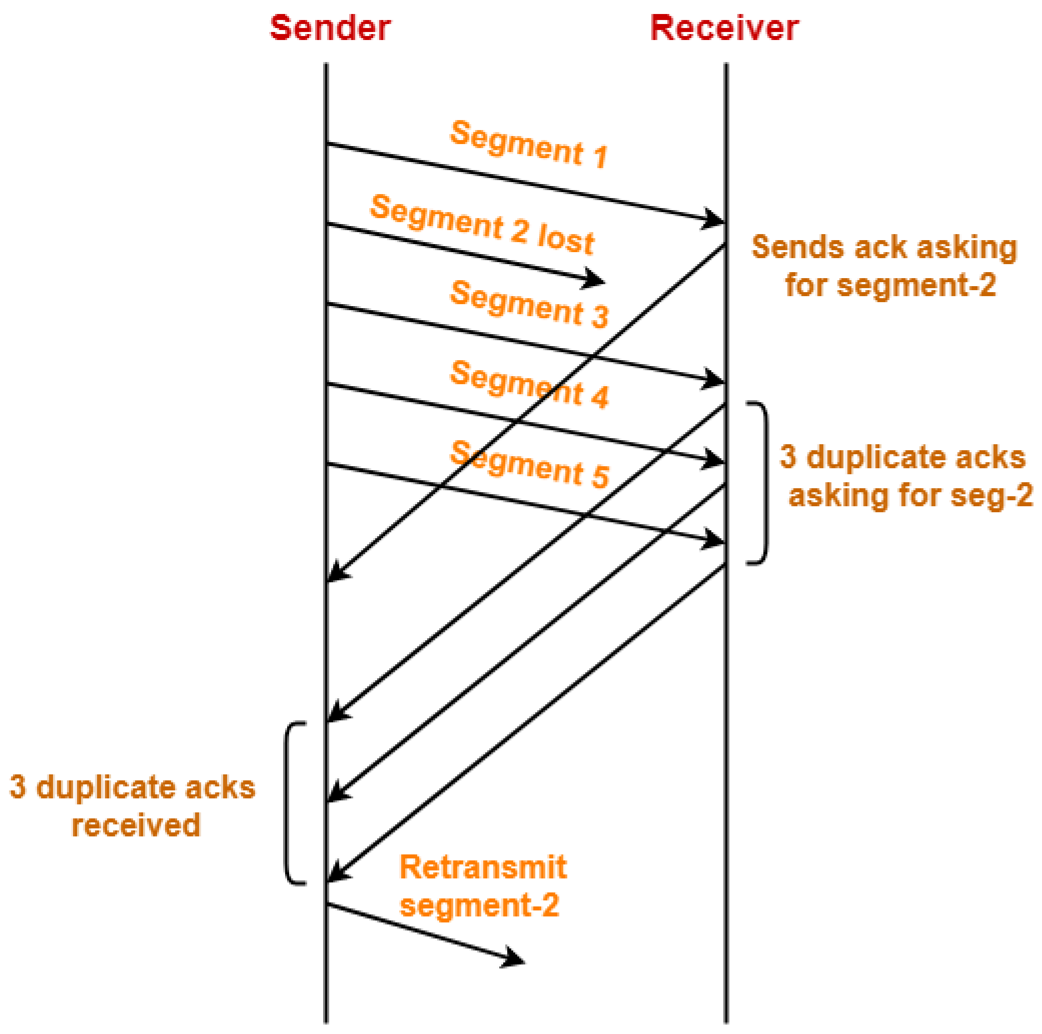
ERROR CONTROL IN TRANSPORT LAYER

TCP protocol has methods for finding out corrupted segments, missing segments, out-of-order segments and duplicated segments.

Error control in TCP is mainly done through 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 as 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 has sequence number will be acknowledged as well but ACK segments are not acknowledged.
3. **Retransmission –** When a segment is missing, delayed to deliver to receiver, corrupted when it is checked by 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.
    1. **Retransmission after RTO :** TCP always preserves one retransmission time-out (RTO) timer for all sent but not acknowledged segments.
    2. When the timer runs out of time, the earliest segment is retransmitted. Here no timer is set for acknowledgement.
    3. **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 the timer to end.


Sender
Receiver
Segment 1
Segment 2 lost
Segment 3
Segment 4
Segment 5
Sends ack asking for segment-2
3 duplicate acks asking for seg-2
3 duplicate acks received
Retransmit segment-2

Retransmission after receiving 3 duplicate acks

REMOTE PROCEDURE CALL(RPC)

Remote Procedure Call (RPC) is a protocol that **one program can use to request a service from a program located in another computer** on a network without having to understand the network's details.

RPC is used to call other processes on remote systems like a local system. A procedure call is also sometimes known as a ***function call*** **or a** ***subroutine call.***

RPC uses the client-server model.

The requesting program is a client, and the service-providing program is the server.

The interface definition language (IDL) -- the specification language used to describe a software component's application programming interface (API) -- is commonly used in Remote Procedure Call software.

In this case, **IDL provides a bridge between the machines at either end of the link that may be using different operating systems (OSes) and computer languages.**

Note:

A stub in distributed computing is a piece of code that converts parameters passed between client and server during a remote procedure call (RPC). The main idea of an

RPC is to allow a local computer (client) to remotely call procedures on a different computer (server).

How does RPC work?

During an RPC, the following steps take place:

1. The client calls the client stub. The call is a local procedure call with parameters pushed onto the stack in the normal way.
2. The client stub packs the procedure parameters into a message and makes a system call to send the message. The packing of the procedure parameters is called ***marshalling*****.**
3. The client's local OS sends the message from the client machine to the remote server machine.
4. The server OS passes the incoming packets to the server stub.
5. The server stub unpacks the parameters -- called *unmarshalling* -- from the message.
6. When the server procedure is finished, it returns to the server stub, which marshals the return values into a message. The server stub then hands the message to the transport layer.
7. The transport layer sends the resulting message back to the client transport layer, which hands the message back to the client stub.

8. The client stub unmarshalls the return parameters, and execution returns to the caller.

Advantages of Remote Procedure Call

The advantages of Remote Procedure Call include the following:

- helps clients communicate with servers via the traditional use of procedure calls in high-level languages;
- can be used in a distributed environment, as well as the local environment;
- hides the internal message-passing mechanism from the user;
- requires only minimal effort to rewrite and redevelop the code;
- provides abstraction, i.e., the message-passing nature of network communication is hidden from the user; and
- omits many of the protocol layers to improve performance.