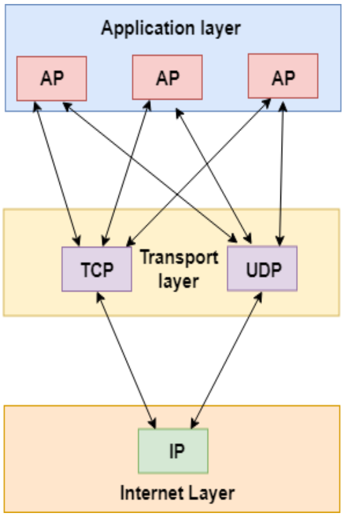
Transport Layer

- 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.

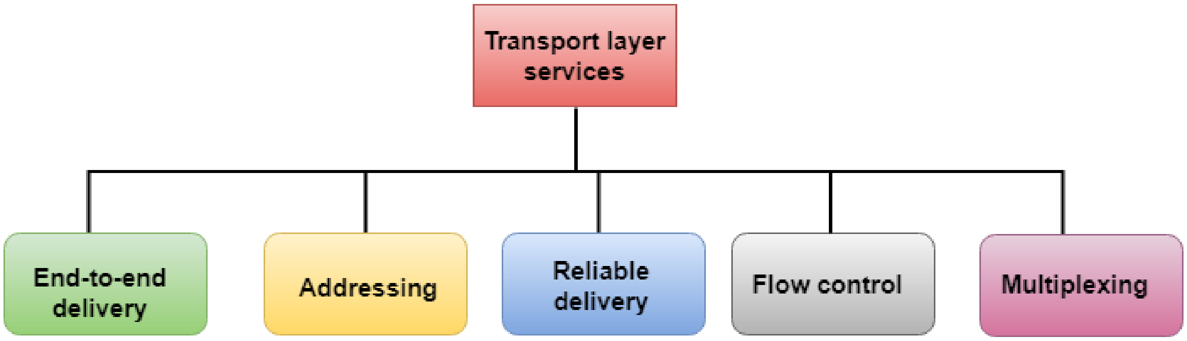


Services provided by the Transport Layer

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.

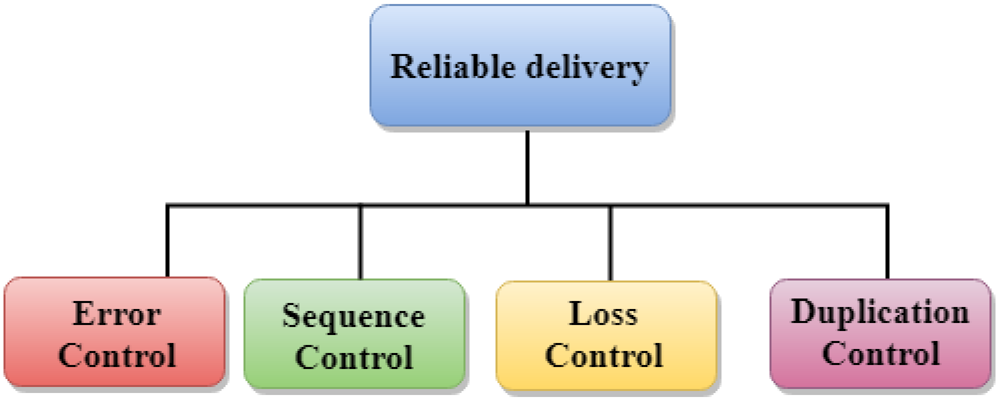
Reliable delivery:

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

The reliable delivery has four aspects:

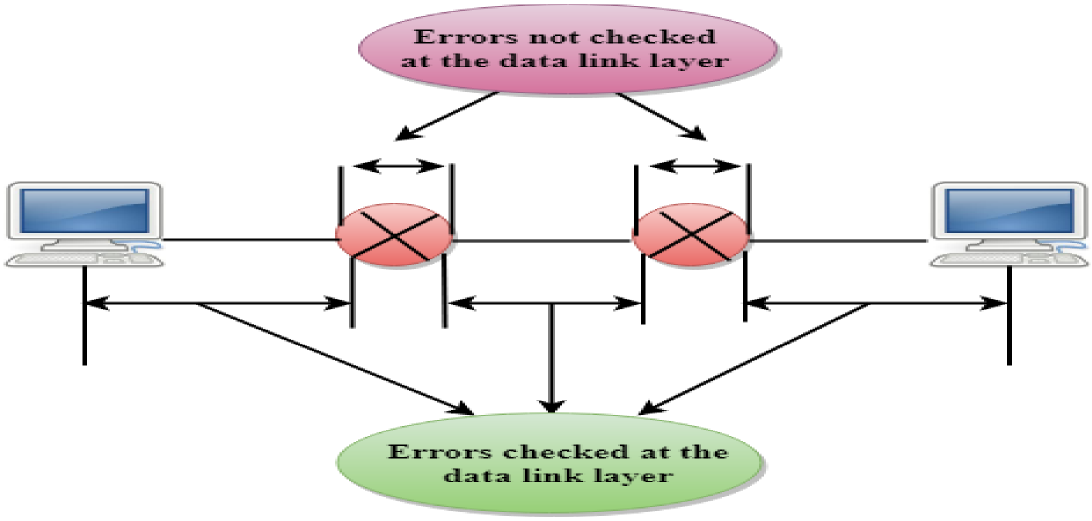
- Error control
- Sequence control

- 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.


Errors not checked at the data link layer
Errors checked at the data link layer

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

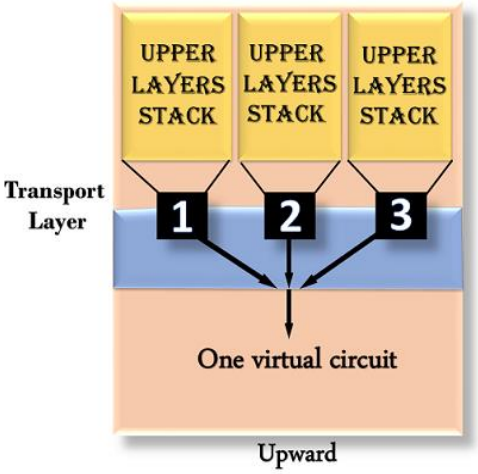
Flow control is used to prevent the sender from overwhelming the receiver. If the receiver is overloaded with too much data, then the receiver discards the packets and asking for the retransmission of packets. This increases network congestion and thus, reducing the system performance. The transport 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

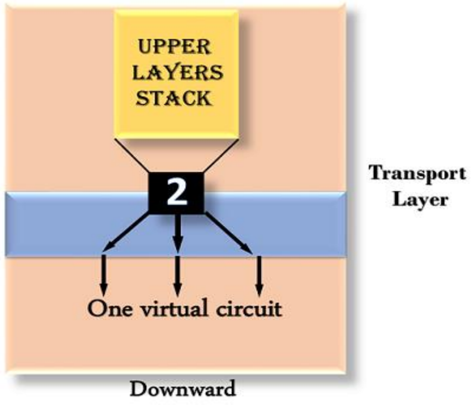
The transport layer uses the multiplexing to improve transmission efficiency.

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.

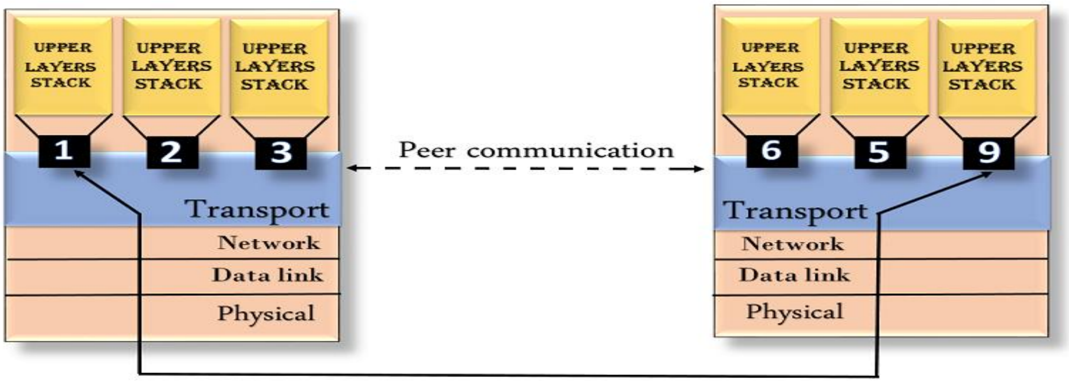


- **Downward multiplexing:** Downward multiplexing means one transport layer connection uses the multiple network connections. Downward multiplexing allows the transport layer to split a connection among several paths to improve the throughput. This type of multiplexing is used when networks have a low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



ELEMENTS OF TRANSPORT LAYER:

At the data link layer, two routers communicate directly via a physical channel, whether wired or wireless, whereas at the transport layer, this physical channel is replaced by the entire network. This difference has many important implications for the protocols.

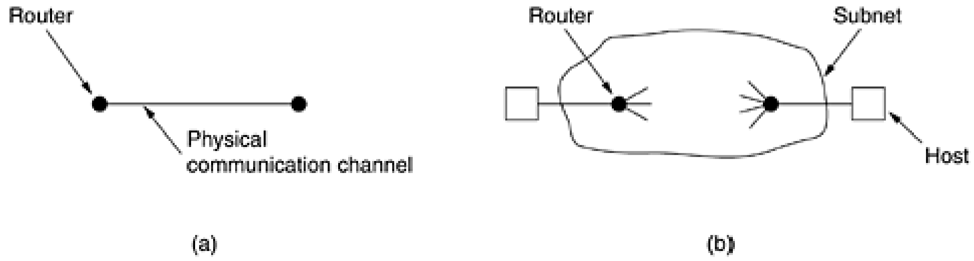


Figure (a) Environment of the data link layer. (b) Environment of the transport layer.

1. Addressing
2. Connection Establishment
3. Connection Release
4. Flow Control and Buffering
5. Multiplexing
6. Crash Recovery

1. ADDRESSING

When an application (e.g., a user) process wishes to set up a connection to a remote application process, it must specify which one to connect to. The method normally used is to define transport addresses to which processes can listen for connection requests. In the Internet, these endpoints are called **ports**.

There are two types of access points.

TSAP (Transport Service Access Point) to mean a specific endpoint in the transport layer.

The analogous endpoints in the network layer (i.e., network layer addresses) are not surprisingly called

NSAPs (Network Service Access Points). IP addresses are examples of NSAPs.

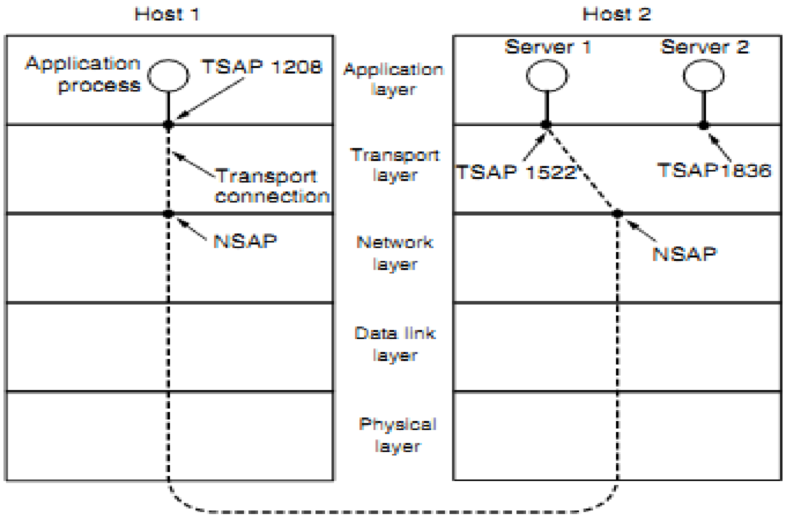


Fig 4.5: TSAP and NSAP network connections

Application processes, both clients and servers, can attach themselves to a local TSAP to establish a connection to a remote TSAP. These connections run through NSAPs on each host. The purpose of having TSAPs is that in some networks, each computer has a single NSAP, so some way is needed to distinguish multiple transport endpoints that share that NSAP.

A possible scenario for a transport connection is as follows:

1. A mail server process attaches itself to TSAP 1522 on host 2 to wait for an incoming call. How a process attaches itself to a TSAP is outside the networking model and depends entirely on the local operatingsystem. A call such as our LISTEN might be used, for example.
2. An application process on host 1 wants to send an email message, so it attaches itself to TSAP 1208 and issues a CONNECT request. The request specifies TSAP 1208 on host 1 as the source and TSAP 1522 on host 2 as the destination. This action ultimately results in a transport connection being established between the application process and the server.
3. The application process sends over the mail message.
4. The mail server responds to say that it will deliver the message.
5. The transport connection is released.

2.CONNECTION ESTABLISHMENT:

With packet lifetimes bounded, it is possible to device a fool proof way to establish connections safely.

Packet lifetime can be bounded to a known maximum using one of the following techniques:

- Restricted subnet design
- Putting a hop counter in each packet

- Time stamping in each packet

Using a 3-way hand shake, a connection can be established. This establishment protocol doesn’t require bothsides to begin sending with the same sequence number.

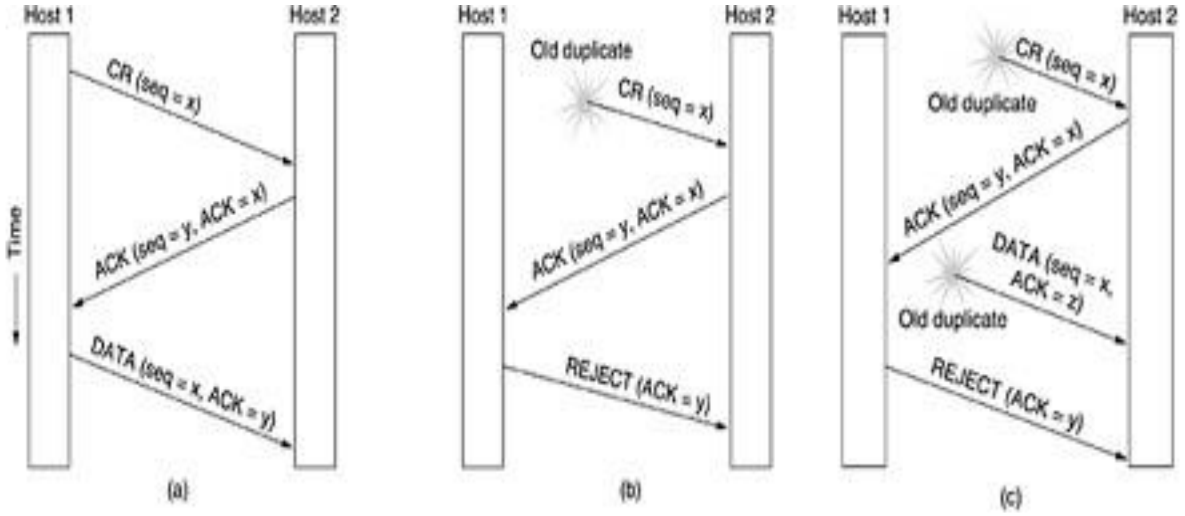


Fig 4.6: Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNEC TION REQUEST (a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearingout of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK .

- The **first technique** includes any method that prevents packets from looping, combined with some way of bounding delay including congestion over the longest possible path. It is difficult, given that internets may range from a single city to international in scope.

- The **second method** consists of having the hop count initialized to some appropriate value and decremented each time the packet is forwarded. The network protocol simply discards any packet whose hop counter becomes zero.
- The **third method** requires each packet to bear the time it was created, with the routers agreeing to discard any packet older than some agreed-upon time.

In **fig (A)** Tomlinson (1975) introduced the **three-way handshake**.

- This establishment protocol involves one peer checking with the other that the connection request is indeed current. Host 1 chooses a sequence number, x , and sends a CONNECTION

REQUEST segment containing it to host 2. Host 2replies with an ACK segment acknowledging x and announcing its own initial sequence number, y.

- Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data segment thatit sends

In **fig (B)** the first segment is a delayed duplicate CONNECTION REQUEST from an old connection.

- This segment arrives at host 2 without host 1's knowledge. Host 2 reacts to this segment by sending host1an ACK segment, in effect asking for verification that host 1 was indeed trying to set up a new connection.
- When host 1 rejects host 2's attempt to establish a connection, host 2 realizes that it was tricked by a delayed duplicate and abandons the connection. In this way, a delayed duplicate does no damage.

- The worst case is when both a delayed CONNECTION REQUEST and an ACK are floating around in the subnet.

In **fig (C)** previous example, host 2 gets a delayed CONNECTION REQUEST and replies to it.

- At this point, it is crucial to realize that host 2 has proposed using y as the initial sequence number for host 2 to host 1 traffic, knowing full well that no segments containing sequence number y or acknowledgements to y are still in existence.
- When the second delayed segment arrives at host 2, the fact that z has been acknowledged rather than ytells host 2 that this, too, is an old duplicate.
- The important thing to realize here is that there is no combination of old segments that can cause theprotocol to fail and have a connection set up by accident when no one wants it.

3.CONNECTION RELEASE:

A connection is released using either asymmetric or symmetric variant. But, the improved protocol for releasing a connection is a 3-way handshake protocol.

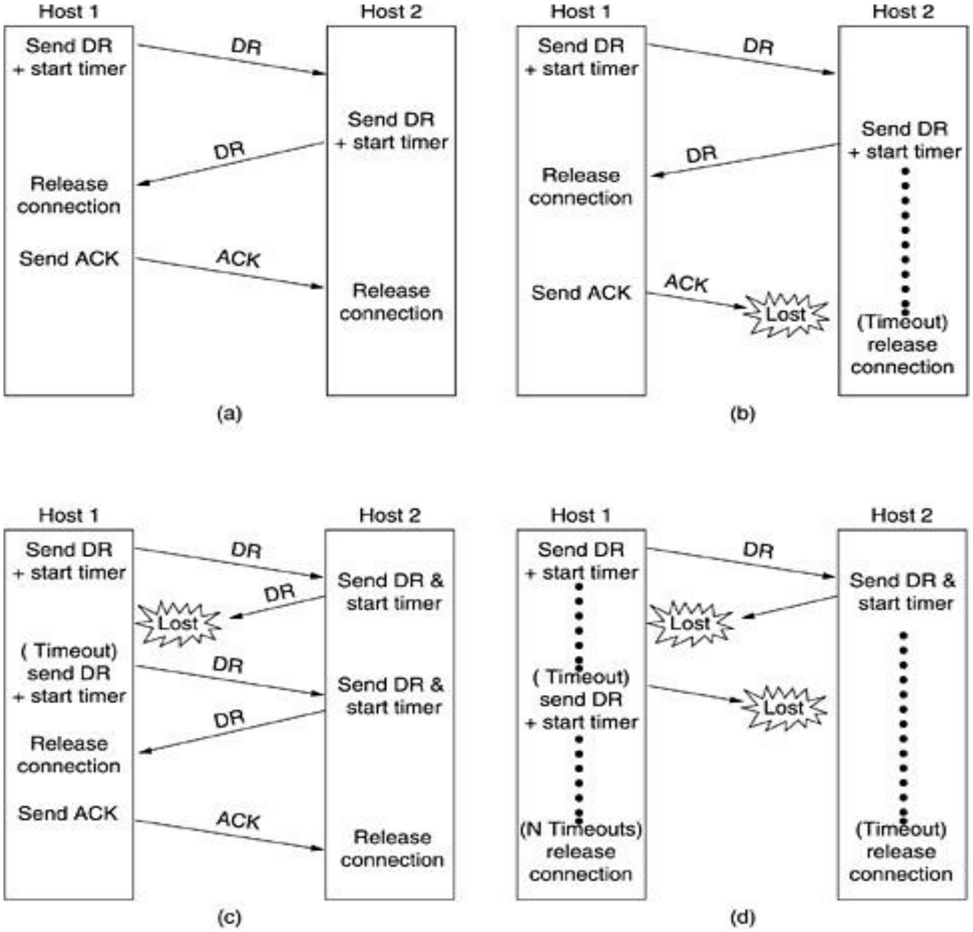
There are two styles of terminating a connection:

1) Asymmetric release and
2) Symmetric release.

Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken.

Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.

Fig-(a)	Fig-(b)	Fig-(c)	Fig-(d)
One of the user sends a DISCONNECTION REQUEST TPDU (Transport Protocol data unit) in order to initiate connection release. When it arrives, the recipient sends back a DR-TPDU, too, and starts a timer. When this DR arrives, the original sender sends back an ACK- TPDU and releases the connection. Finally, when the ACK-TPDU arrives, the receiver also releases the connection.	Initial process is done in the same way as in fig-(a). If the final ACK-TPDU is lost, the situation is saved by the timer. When the timer is expired, the connection is released.	If the second DR is lost, the user initiating the disconnection will not receive the expected response, and will timeout and starts all over again.	Same as in fig-(c) except that all repeatedattempts to retransmitthe DR is assumed to be failed due to lost TPDUs. After 'N' entries, the sender just gives up and releases the connection.



4.FLOW CONTROL AND BUFFERING:

Flow control is done by having a sliding window on each connection to keep a fast transmitter from over running a slow receiver. Buffering must be done by the sender, if the network service is unreliable. The sender buffers all the TPDUs sent to the receiver. The buffer size varies for different TPDUs.

They are:

a) Chained Fixed-size Buffers
b) Chained Variable-size Buffer
c) One large Circular Buffer per Connection

(a). Chained Fixed-size Buffers:

If most TPDUs are nearly the same size, the buffers are organized as a pool of identical size buffers,with one TPDU per buffer.

(b). Chained Variable-size Buffers:

This is an approach to the buffer-size problem. i.e., if there is wide variation in TPDU size, from a few characters typed at a terminal to thousands of characters from file transfers, some problems may occur:

- If the buffer size is chosen equal to the largest possible TPDU, space will be wasted whenever a shortTPDU arrives.
- If the buffer size is chosen less than the maximum TPDU size, multiple buffers will be needed for longTPDUs.

To overcome these problems, we employ variable-size buffers.

(c). One large Circular Buffer per Connection:

A single large circular buffer per connection is dedicated when all connections are heavily loaded.

1. Source Buffering is used for low band width bursty traffic
2. Destination Buffering is used for high band width smooth traffic.
3. Dynamic Buffering is used if the traffic pattern changes randomly.

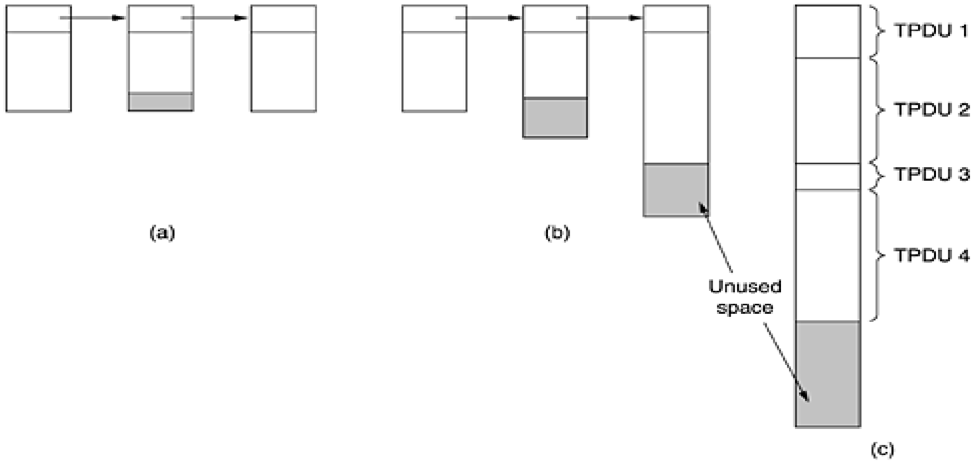


Figure 4.7. (a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular bufferper connection.

5.MULTIPLEXING:

In networks that use virtual circuits within the subnet, each open connection consumes some table space in the routers for the entire duration of the connection. If buffers are dedicated to the virtual circuit in each router as well, a user who left a terminal logged into a remote machine, there is need for multiplexing. There are 2 kinds of multiplexing:

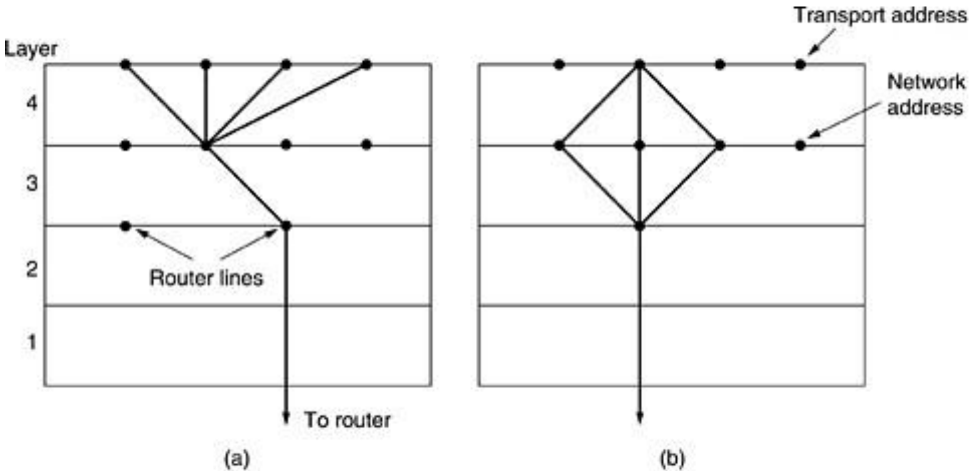


Figure 4.8. (a) Upward multiplexing. (b) Downward multiplexing

(a). UP-WARD MULTIPLEXING:

In the below figure, all the 4 distinct transport connections use the same network connection to the remote host. When connect time forms the major component of the carrier's bill, it is up to the transport layer to group port connections according to their destination and map each group onto the minimum number of port connections.

(b). DOWN-WARD MULTIPLEXING:

- If too many transport connections are mapped onto the one network connection, the performance will be poor.
- If too few transport connections are mapped onto one network connection, the servicewill be expensive.

The possible solution is to have the transport layer open multiple connections and distribute the traffic amongthem on round-robin basis, as indicated in the below figure: With 'k' network connections open, the effective band width is increased by a factor of 'k'.

TCP

TCP stands for **Transmission Control Protocol**. It is a transport layer protocol that facilitates the transmission of packets from source to destination. It is a connection-oriented protocol that means it establishes the connection prior to the communication that occurs between the computing devices in a network. This protocol is used with an IP protocol, so together, they are referred to as a TCP/IP.

The main functionality of the TCP is to take the data from the application layer. Then it divides the data into a several packets, provides numbering to these packets, and finally transmits these packets to the destination. The TCP, on the other side, will reassemble the packets and transmits them to the application layer. As we know that TCP is a connection-oriented protocol, so the connection will remain established until the communication is not completed between the sender and the receiver.

Features of TCP protocol

The following are the features of a TCP protocol:

- **Transport Layer Protocol**

TCP is a transport layer protocol as it is used in transmitting the data from the sender to the receiver.

- **Reliable**

TCP is a reliable protocol as it follows the flow and error control mechanism. It also supports the acknowledgment mechanism, which checks the state and sound arrival of the data. In the acknowledgment mechanism, the receiver sends either positive or negative acknowledgment to the sender so that the sender can get to know whether the data packet has been received or needs to resend.

- **Order of the data is maintained**

This protocol ensures that the data reaches the intended receiver in the same order in which it is sent. It orders and numbers each segment so that the TCP layer on the destination side can reassemble them based on their ordering.

- **Connection-oriented**

It is a connection-oriented service that means the data exchange occurs only after the connection establishment. When the data transfer is completed, then the connection will get terminated.

- **Full duplex**

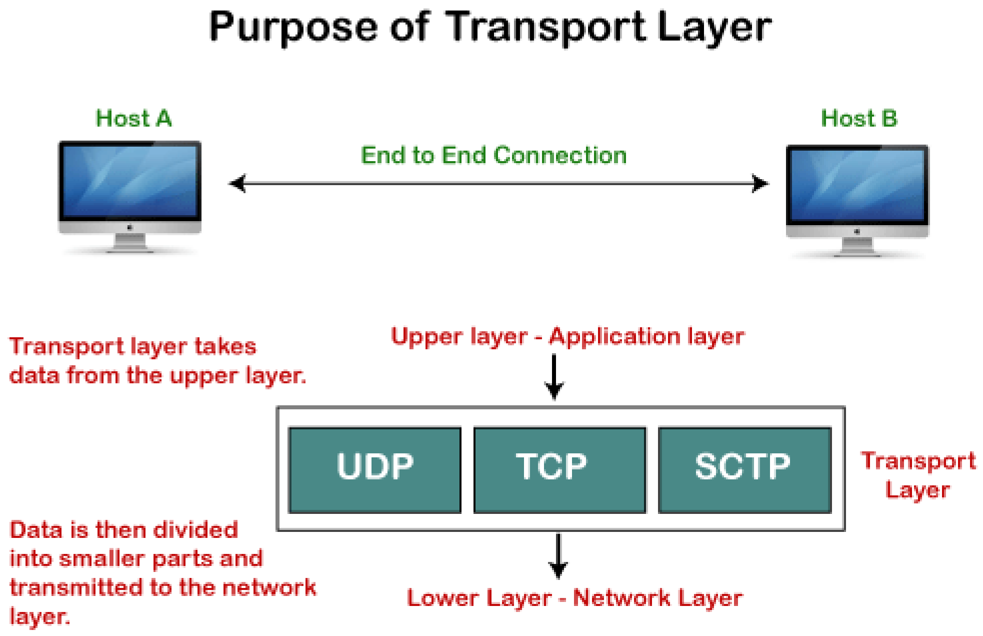
It is a full-duplex means that the data can transfer in both directions at the same time.

- **Stream-oriented**

TCP is a stream-oriented protocol as it allows the sender to send the data in the form of a stream of bytes and also allows the receiver to accept the data in the form of a stream of bytes. TCP creates an environment in which both the sender and receiver are connected by an imaginary tube known as a virtual circuit. This virtual circuit carries the stream of bytes across the internet.

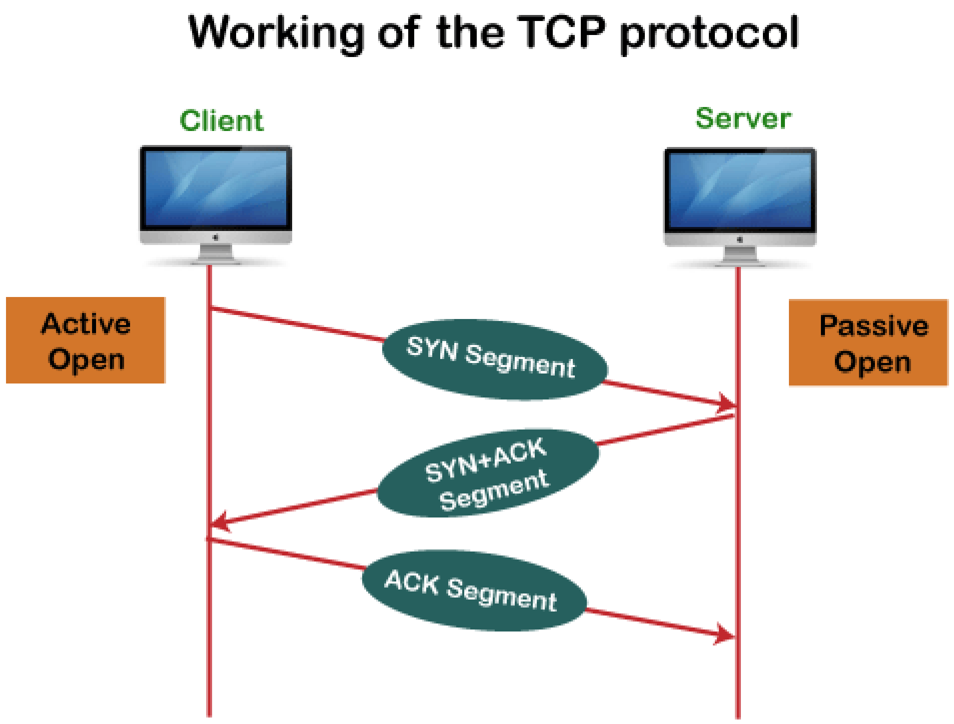
Need of Transport Control Protocol

In the layered architecture of a network model, the whole task is divided into smaller tasks. Each task is assigned to a particular layer that processes the task. In the TCP/IP model, five layers are application layer, transport layer, network layer, data link layer, and physical layer. The transport layer has a critical role in providing end-to-end communication to the directly application processes. It creates 65,000 ports so that the multiple applications can be accessed at the same time. It takes the data from the upper layer, and it divides the data into smaller packets and then transmits them to the network layer.


Purpose of Transport Layer
Host A
Host B
End to End Connection
Transport layer takes data from the upper layer.
Upper layer - Application layer
UDP
TCP
SCTP
Transport Layer
Data is then divided into smaller parts and transmitted to the network layer.
Lower Layer - Network Layer

Working of TCP

In TCP, the connection is established by using three-way handshaking. The client sends the segment with its sequence number. The server, in return, sends its segment with its own sequence number as well as the acknowledgement sequence, which is one more than the client sequence number. When the client receives the acknowledgment of its segment, then it sends the acknowledgment to the server. In this way, the connection is established between the client and the server.



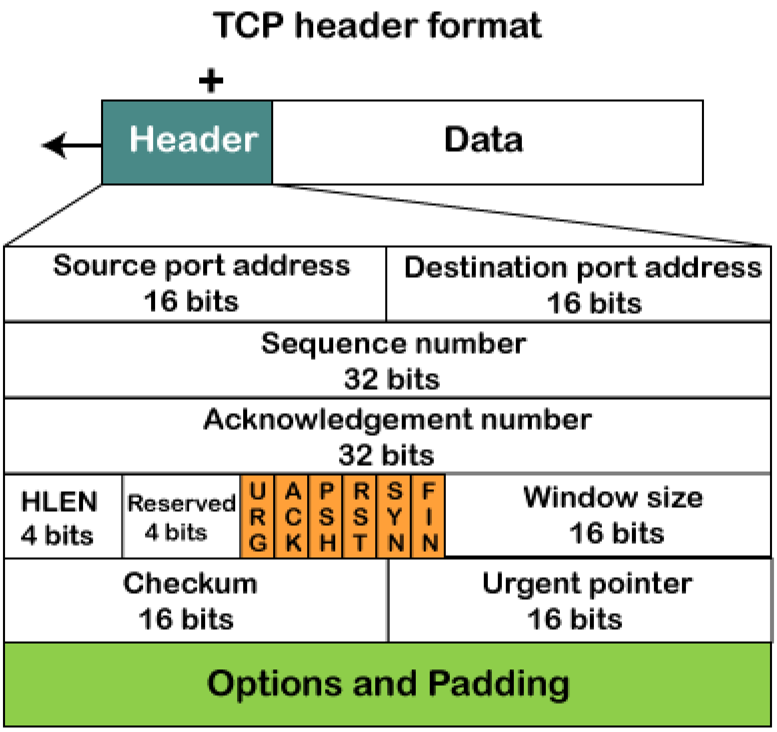
Advantages of TCP

- It provides a connection-oriented reliable service, which means that it guarantees the delivery of data packets. If the data packet is lost across the network, then the TCP will resend the lost packets.
- It provides a flow control mechanism using a sliding window protocol.
- It provides error detection by using checksum and error control by using Go Back or ARP protocol.
- It eliminates the congestion by using a network congestion avoidance algorithm that includes various schemes such as additive increase/multiplicative decrease (AIMD), slow start, and congestion window.

Disadvantage of TCP

It increases a large amount of overhead as each segment gets its own TCP header, so fragmentation by the router increases the overhead.

TCP Header format



- **Source port:** It defines the port of the application, which is sending the data. So, this field contains the source port address, which is 16 bits.
- **Destination port:** It defines the port of the application on the receiving side. So, this field contains the destination port address, which is 16 bits.
- **Sequence number:** This field contains the sequence number of data bytes in a particular session.
- **Acknowledgment number:** When the ACK flag is set, then this contains the next sequence number of the data byte and works as an acknowledgment for the previous data received. For example, if the receiver receives the segment number 'x', then it responds 'x+1' as an acknowledgment number.
- **HLEN:** It specifies the length of the header indicated by the 4-byte words in the header. The size of the header lies between 20 and 60 bytes. Therefore, the value of this field would lie between 5 and 15.
- **Reserved:** It is a 4-bit field reserved for future use, and by default, all are set to zero.
- **Flags**

  There are six control bits or flags:

  1. **URG:** It represents an urgent pointer. If it is set, then the data is processed urgently.

2. **ACK:** If the ACK is set to 0, then it means that the data packet does not contain an acknowledgment.
3. **PSH:** If this field is set, then it requests the receiving device to push the data to the receiving application without buffering it.
4. **RST:** If it is set, then it requests to restart a connection.
5. **SYN:** It is used to establish a connection between the hosts.
6. **FIN:** It is used to release a connection, and no further data exchange will happen.

- **Windowsize**
  It is a 16-bit field. It contains the size of data that the receiver can accept. This field is used for the flow control between the sender and receiver and also determines the amount of buffer allocated by the receiver for a segment. The value of this field is determined by the receiver.
- **Checksum**
  It is a 16-bit field. This field is optional in UDP, but in the case of TCP/IP, this field is mandatory.
- **Urgentpointer**
  It is a pointer that points to the urgent data byte if the URG flag is set to 1. It defines a value that will be added to the sequence number to get the sequence number of the last urgent byte.
- **Options**
  It provides additional options. The optional field is represented in 32-bits. If this field contains the data less than 32-bit, then padding is required to obtain the remaining bits.

Congestion in Network

Congestion refers to a network state where-

The message traffic becomes so heavy that it slows down the network response time.

- Congestion is an important issue that can arise in **Packet Switched Network**.
- Congestion leads to the loss of packets in transit.
- So, it is necessary to control the congestion in network.
- It is not possible to completely avoid the congestion.

Congestion Control-

- Congestion control refers to techniques and mechanisms that can-
- Either prevent congestion before it happens
- Or remove congestion after it has happened

TCP reacts to congestion by reducing the sender window size.

The size of the sender window is determined by the following two factors-

1. Receiver window size
2. Congestion window size

1. Receiver Window Size-

- Sender should not send data greater than receiver window size.
- Otherwise, it leads to dropping the TCP segments which causes **TCP Retransmission**.
- So, sender should always send data less than or equal to receiver window size.
- Receiver dictates its window size to the sender through **TCP Header**.

2. Congestion Window-

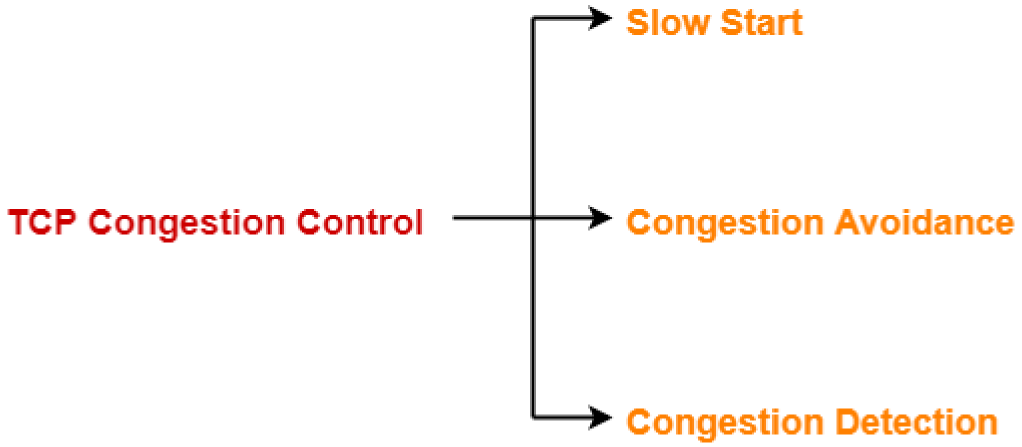
- Sender should not send data greater than congestion window size.
- Otherwise, it leads to dropping the TCP segments which causes TCP Retransmission.
- So, sender should always send data less than or equal to congestion window size.
- Different variants of TCP use different approaches to calculate the size of congestion window.
- Congestion window is known only to the sender and is not sent over the links.

So, always-

Sender window size = Minimum (Receiver window size, Congestion window size)

TCP Congestion Policy-

TCP's general policy for handling congestion consists of following three phases-



Congestion policy in TCP –

1. Slow Start Phase: starts slowly increment is exponential to threshold (Thresholds are **defined values that determine if a statistic is above, below, or within a normal range on your network**.)
2. Congestion Avoidance Phase: After reaching the threshold increment is by 1
3. Congestion Detection Phase: Sender goes back to Slow start phase or Congestion avoidance phase.

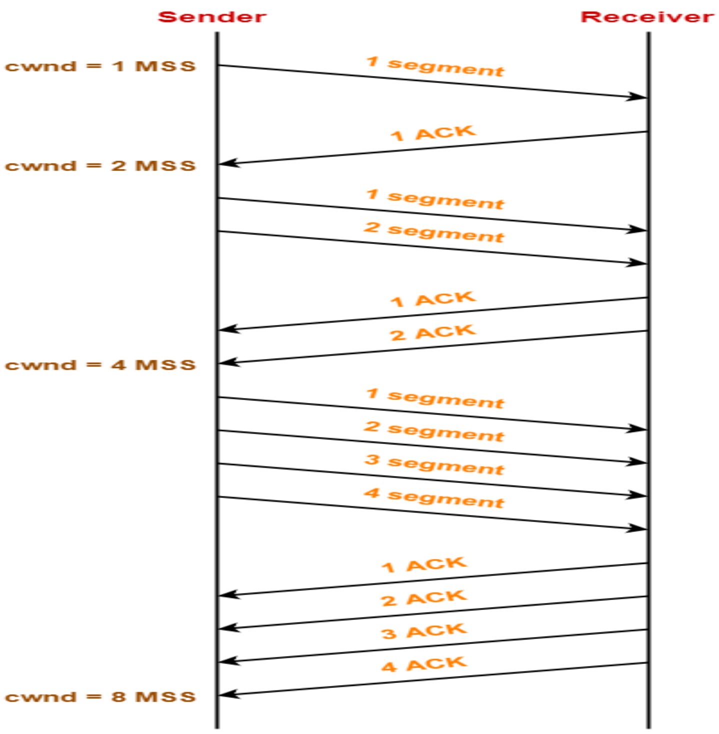
1. Slow Start Phase : exponential increment – In this phase after every RTT(**Round-trip Time**) the congestion window size increments exponentially.

Initially cwnd = 1

After 1 RTT, cwnd = 2^(1) = 2

2 RTT, cwnd = 2^(2) = 4

3 RTT, cwnd = 2^(3) = 8



(cwnd = congestion window size)

- After 1 round trip time, congestion window size = $(2)^1$ = 2 MSS
- After 2 round trip time, congestion window size = $(2)^2$ = 4 MSS
- After 3 round trip time, congestion window size = $(2)^3$ = 8 MSS and so on.

his phase continues until the congestion window size reaches the slow start threshold.

Threshold

= Maximum number of TCP segments that receiver window can accommodate / 2

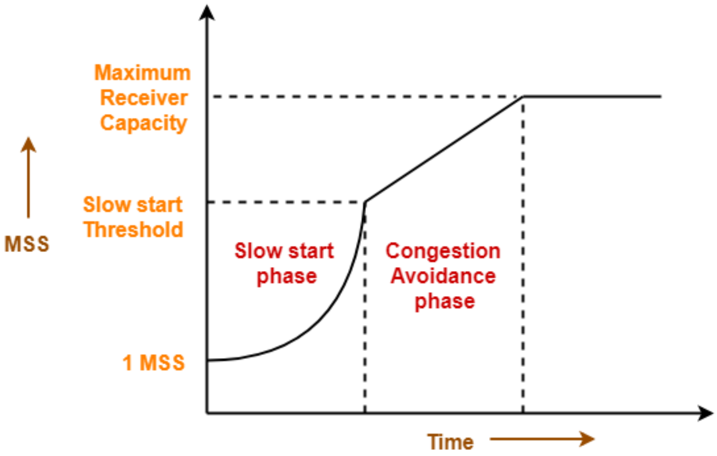
= (Receiver window size / Maximum Segment Size) / 2

2.Congestion Avoidance Phase : additive increment – This phase starts after the threshold value also denoted as *ssthresh*. The size of *cwnd*(congestion window) increases additive. After each RTT cwnd = cwnd + 1.
Initially cwnd = i

After 1 RTT, cwnd = i+1

2 RTT, cwnd = i+2

3 RTT, cwnd = i+3

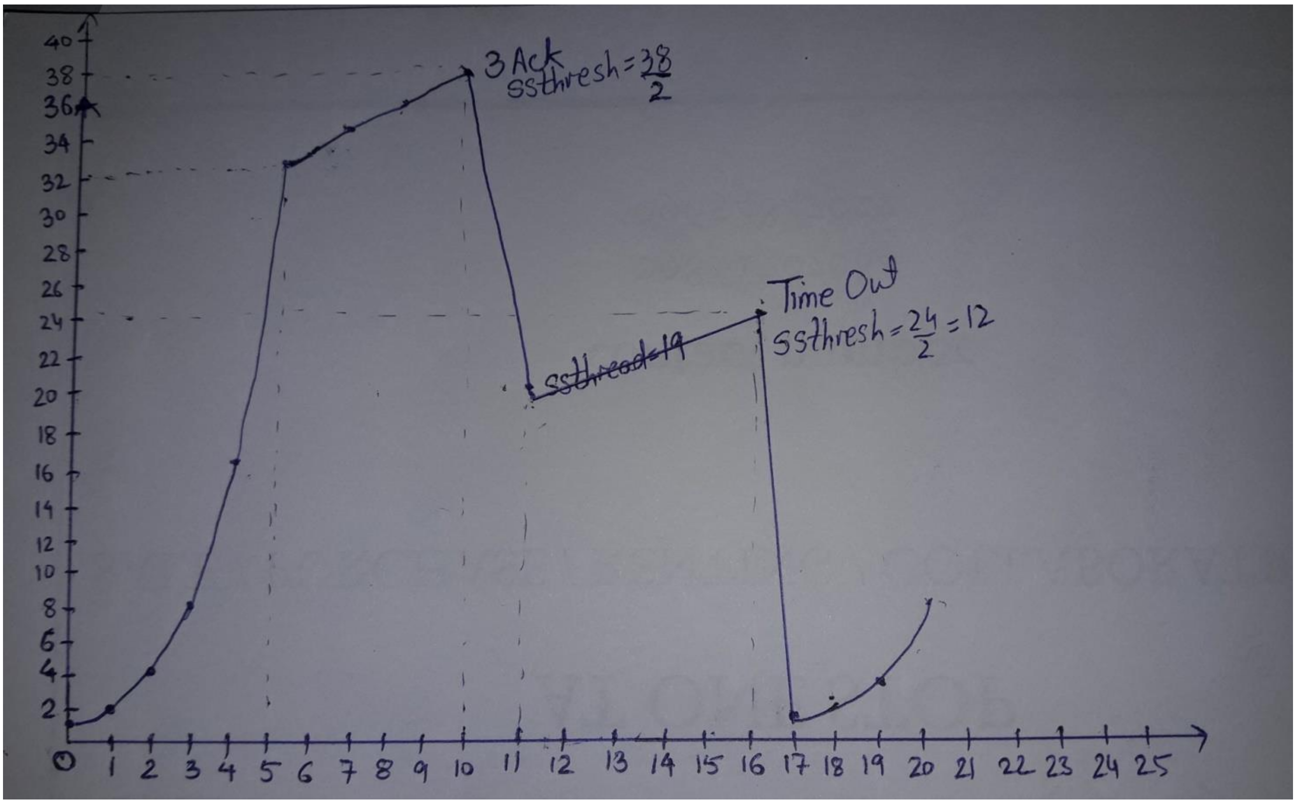


3.Congestion Detection Phase : multiplicative decrement – If congestion occurs, the congestion window size is decreased. The only way a sender can guess that congestion has occurred is the need to retransmit a segment. Retransmission is needed to recover a missing packet that is assumed to have been dropped by a router due to congestion. Retransmission can occur in one of two cases: when the RTO timer times out or when three duplicate ACKs are received.

- **Case 1 : Retransmission due to Timeout –** In this case congestion possibility is high.
  (a) ssthresh is reduced to half of the current window size.
  (b) set cwnd = 1
  (c) start with slow start phase again.

- **Case 2 : Retransmission due to 3 Acknowledgement Duplicates –** In this case congestion possibility is less.
  (a) ssthresh value reduces to half of the current window size.
  (b) set cwnd= ssthresh
  (c) start with congestion avoidance phase

  Example – Assume a TCP protocol experiencing the behavior of slow start. At 5th transmission round with a threshold (ssthresh) value of 32 goes into congestion avoidance phase and continues till 10th transmission. At 10th transmission round, 3 duplicate ACKs are received by the receiver and enter into additive increase mode. Timeout occurs at 16th transmission round. Plot the transmission round (time) vs congestion window size of TCP segments.



UDP Protocol

In computer networking, the UDP stands for User Datagram Protocol. The David P. Reed developed the UDP protocol in 1980. It is defined in RFC 768, and it is a part of the TCP/IP protocol, so it is a standard protocol over the internet. The UDP protocol allows the computer applications to send the messages in the form of datagrams from one machine to another machine over the Internet Protocol (IP) network. The UDP is an alternative communication protocol to the TCP protocol (transmission control protocol). Like TCP, UDP provides a set of rules that governs how the data should be exchanged over the internet. The UDP works by encapsulating the data into the packet and providing its own header information to the packet. Then, this UDP packet is encapsulated to the IP packet and sent off to its destination. Both the TCP and UDP protocols send the data over the internet protocol network, so it is also known as TCP/IP and UDP/IP.

UDP also provides a different port number to distinguish different user requests and also provides the checksum capability to verify whether the complete data has arrived or not; the IP layer does not provide these two services.

Features of UDP protocol

The following are the features of the UDP protocol:

- **Transport layer protocol**

UDP is the simplest transport layer communication protocol. It contains a minimum amount of communication mechanisms. It is considered an unreliable protocol, and it is based on best-effort delivery services. UDP provides no acknowledgment mechanism, which means that the receiver does not send the acknowledgment for the received packet, and the sender also does not wait for the acknowledgment for the packet that it has sent.

- **Connectionless**

The UDP is a connectionless protocol as it does not create a virtual path to transfer the data. It does not use the virtual path, so packets are sent in different paths between the sender and the receiver, which leads to the loss of packets or received out of order.

- **Ordered delivery of data is not guaranteed.**

In the case of UDP, the datagrams are sent in some order will be received in the same order is not guaranteed as the datagrams are not numbered.

- **Ports**

The UDP protocol uses different port numbers so that the data can be sent to the correct destination. The port numbers are defined between 0 and 1023.

- **Faster transmission**

UDP enables faster transmission as it is a connectionless protocol, i.e., no virtual path is required to transfer the data. But there is a chance that the individual packet is lost, which affects the transmission quality. On the other hand, if the packet is lost in TCP connection, that packet will be resent, so it guarantees the delivery of the data packets.

- **Acknowledgment mechanism**

The UDP does have any acknowledgment mechanism, i.e., there is no handshaking between the UDP sender and UDP receiver. If the message is sent in TCP, then the receiver acknowledges that I am ready, then the sender sends the data. In the case of TCP, the handshaking occurs between the sender and the receiver, whereas in UDP, there is no handshaking between the sender and the receiver.

- **Segments are handled independently.**

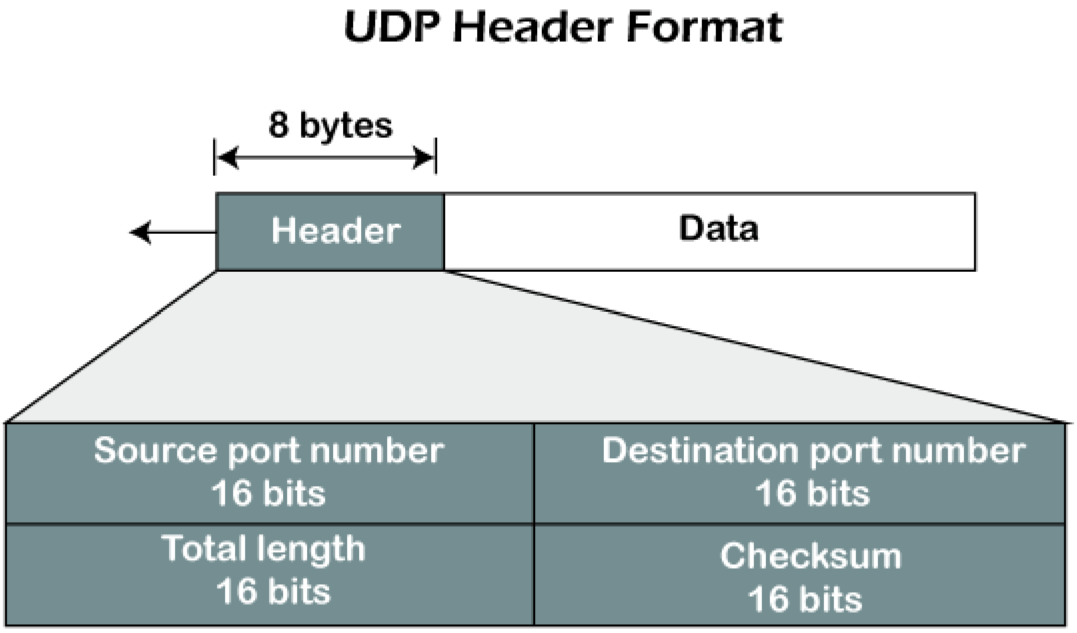
Each UDP segment is handled individually of others as each segment takes different path to reach the destination. The UDP segments can be lost or delivered out of order to reach the destination as there is no connection setup between the sender and the receiver.

- **Stateless:** It is a stateless protocol that means that the sender does not get the acknowledgement for the packet which has been sent.

Why do we require the UDP protocol?

As we know that the UDP is an unreliable protocol, but we still require a UDP protocol in some cases. The UDP is deployed where the packets require a large amount of bandwidth along with the actual data. For example, in video streaming, acknowledging thousands of packets is troublesome and wastes a lot of bandwidth. In the case of video streaming, the loss of some packets couldn't create a problem, and it can also be ignored.

UDP Header Format



In UDP, the header size is 8 bytes, and the packet size is upto 65,535 bytes. But this packet size is not possible as the data needs to be encapsulated in the IP datagram, and an IP packet, the header size can be 20 bytes; therefore, the maximum of UDP would be 65,535 minus 20. The size of the data that the UDP packet can carry would be 65,535 minus 28 as 8 bytes for the header of the UDP packet and 20 bytes for IP header.

The UDP header contains four fields:

- **Source port number:** It is 16-bit information that identifies which port is going t send the packet.
- **Destination port number:** It identifies which port is going to accept the information. It is 16-bit information which is used to identify application-level service on the destination machine.
- **Length:** It is 16-bit field that specifies the entire length of the UDP packet that includes the header also. The minimum value would be 8-byte as the size of the header is 8 bytes.
- **Checksum:** It is a 16-bits field, and it is an optional field. This checksum field checks whether the information is accurate or not as there is the possibility that the information can be corrupted while transmission. It is an optional field, which means that it depends upon the application, whether it wants to write the checksum or not. If it does not want to write the

checksum, then all the 16 bits are zero; otherwise, it writes the checksum. In UDP, the checksum field is applied to the entire packet, i.e., header as well as data part whereas, in IP, the checksum field is applied to only the header field.

Limitations

- It provides an unreliable connection delivery service. It does not provide any services of IP except that it provides process-to-process communication.
- The UDP message can be lost, delayed, duplicated, or can be out of order.
- It does not provide a reliable transport delivery service. It does not provide any acknowledgment or flow control mechanism. However, it does provide error control to some extent.

Advantages

- It produces a minimal number of overheads.

REMOTE PROCEDURE CALL

- In a certain sense, sending a message to a remote host and getting a reply back is like making a function call in a programming language. This is to arrange request-reply interactions on networks to be cast in the form of procedure calls.
- For example, just imagine a procedure named *get IP address* (*host name*) that works by sending a UDP packet to a DNS server and waiting or the reply, timing out and trying again if one is not forthcoming quickly enough. In this way, all the details of networking can be hidden from the programmer.

- RPC is used to call remote programs using the procedural call. When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2.

- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the application programmer. This technique is known as **RPC** (**Remote Procedure Call**) and has become the basis for many networking applications.

  Traditionally, the calling procedure is known as the **client** and the called procedure is known as the server.
- In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure, called the **client stub**,that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure called the **server stub**. These procedures hide the fact that the procedure call from the client to the server is not local.

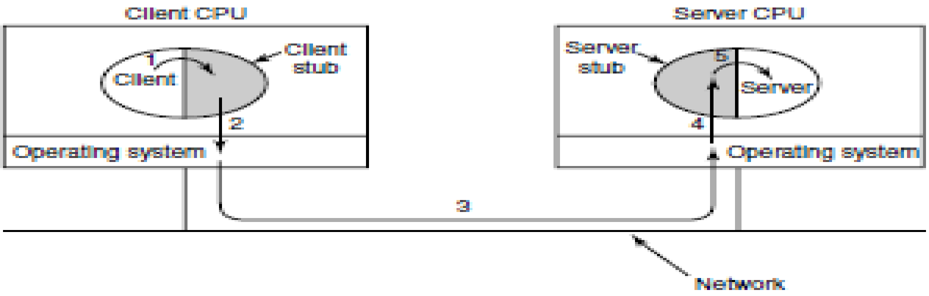


Fig 4.10: Steps in making a RPC

Step 1 is the client calling the client stub. This call is a local procedure call, with the parameters pushed ontothe stack in the normal way.
Step 2 is the client stub packing the parameters into a message and making a system call to send the message.Packing the parameters is called **marshaling**.
Step 3 is the operating system sending the message from the client machine to the server machine.
Step 4 is the operating system passing the incoming packet to the server stub.
Step 5 is the server stub calling the server procedure with the **unmarshaled** parameters. The reply traces thesame path in the other direction.

The key item to note here is that the client procedure, written by the user, just makes a normal (i.e., local) procedure call to the client stub, which has the same name as the server procedure. Since the client procedure and client stub are in the same address space, the parameters are passed in the usual way.

Similarly, the server procedure is called by a procedure in its address space with the parameters it expects. To the server procedure, nothing is unusual. In this way, instead of I/O being done on sockets, network communication is done by faking a normal procedure call. With RPC, passing pointers is impossible because the client and server are in different address spaces.