UNIT-4

TRANSPORT LAYER

SYLLABUS

Transport Layer: The Transport Service: Services provided to the Upper Layers, Transport Service Primitives, Elements of Transport Protocols: Addressing, The Internet Transport protocols: UDP and TCP, SCTP protocols.

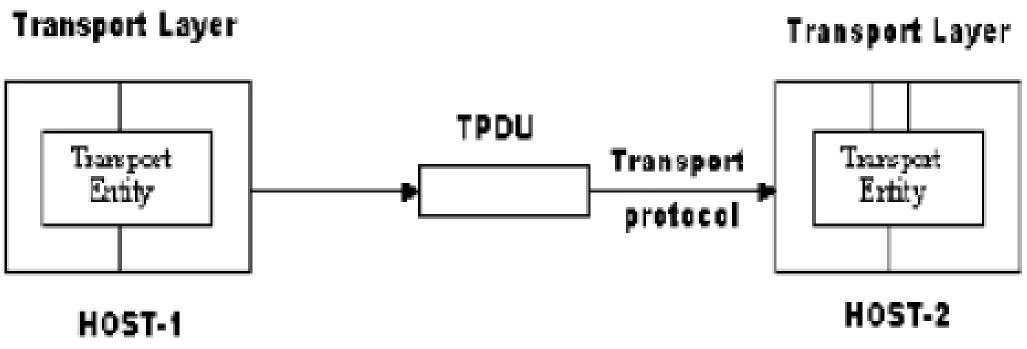
TRANSPORT LAYER

- The network layer provides end-to-end packet delivery using datagrams or virtual circuits.
- The transport layer builds on the network layer to provide data transport from a process on a source machine to a process on a destination machine with adesired level of reliability that is independent of the physical networks currently in use.

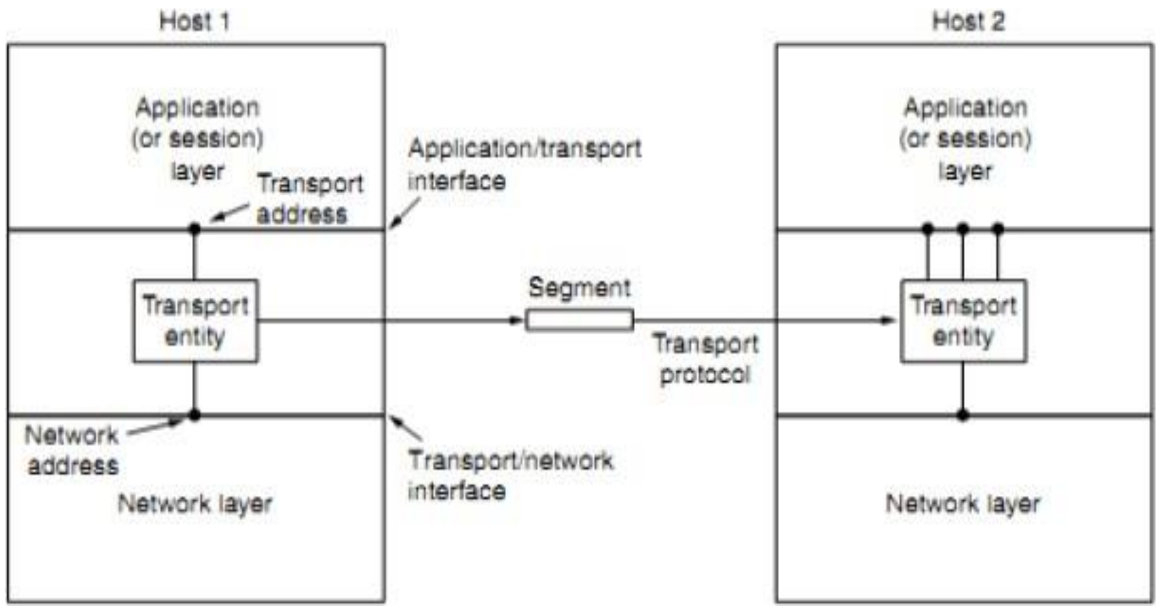
TRANSPORT LAYER SERVICES

Services Provided to the Upper Layers:

- The main goal of the transport layer is to provide efficient, reliable, and cost-effective data transmission service to its users, normally processes in theapplication layer.
- To achieve this, the transport layer makes use of the services provided by the network layer. The software and/or hardware within the transport layer that does the work is called the transport entity.

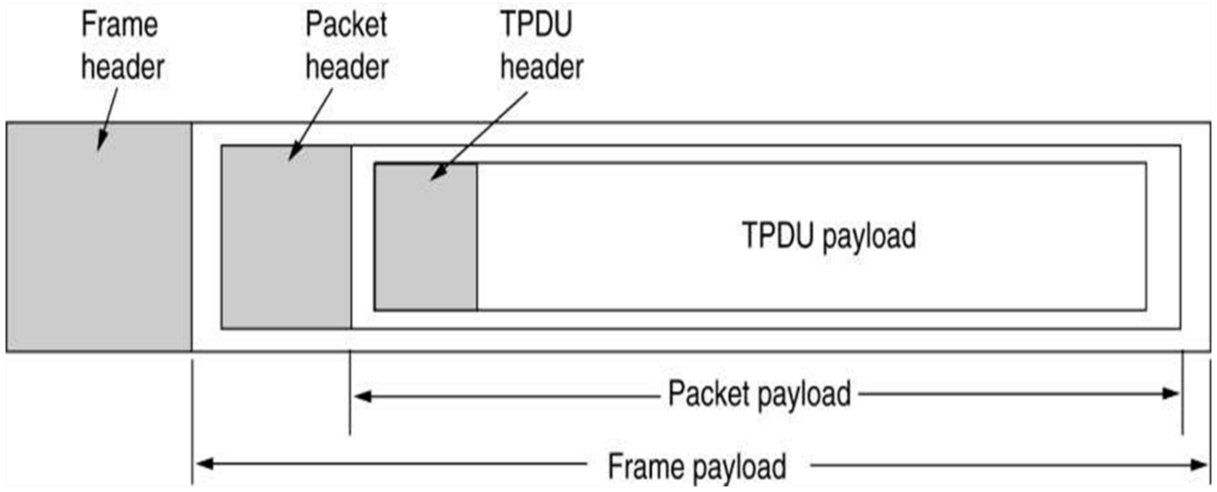


TPDU – Transport Protocol Data Unit



TPDU (Transport Protocol Data Unit):

- Transmissions of message between 2 transport entities are carried out by TPDU.
- The transport entity carries out the transport service primitives by blocking the caller and sending a packet the service.
- Encapsulated in the payload of this packet is a transport layer message for the server's transport entity.
- The task of the transport layer is to provide reliable, cost-effective data transport from the source machine to the destination machine, independent of physical network or networks currently in use.



Nesting of TPDUs, packets, and frames

The term segment for messages sent from transport entity to transport entity.

a) TCP, UDP and other Internet protocols use this term. Segments (exchanged by the transport layer) are contained in packets (exchanged by the networklayer).
b) These packets are contained in frames(exchanged by the data link layer).When a frame arrives, the data link layer processes the frame header and, if thedestination address matches for local delivery, passes the contents of the frame payload field up to the network entity.

c) The network entity similarly processes the packet header and then passes the contents of the packet payload up to the transport entity.

Transport Entity: The hardware and/or software which make use of services provided by the network layer, (within the transport layer) is called transportentity.

There are two types of network service

1. Connection-oriented
2. Connectionless

Similarly, there are also two types of transport service. The connection-oriented transport service is similar to the connection-oriented network service in manyways.

In both cases, connections have three phases:

1. Establishment
2. Data transfer
3. Release.

a) Addressing and flow control are also similar in both layers. Furthermore, the connectionless transport service is also very similar to the connectionlessnetwork service.

b) The bottom four layers can be seen as the transport service provider, whereas the upper layer(s) are the transport service user.

Major differences between Transport and Network layer

NETWORK LAYER	**TRANSPORT LAYER**
The code run on routers	The code runs entirely on the users machine
The packets may lose	The lost packets and damaged data can be detected
The network, service is used only by transport entity	In transport service the users can write their own transport entity

TRANSPORT SERVICE PRIMITIVES

To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface. Each transport service has its own interface.

a) The transport service is similar to the network service, but there are also some important differences.

b) The main difference is that the network service is intended to model the service offered by real networks. Real networks can lose packets, so thenetwork service is generally unreliable.

c) The (connection-oriented) transport service, in contrast, is reliable

As an example, consider two processes connected by pipes in UNIX. They assume the connection between them is perfect. They do not want to know about acknowledgements, lost packets, congestion, or anything like that. What they want is a 100 percent reliable connection. Process A puts data into one end of the pipe, and process B takes it out of the other.

A second difference between the network service and transport service is whom the services are intended for. The network service is used only by the transport entities. Consequently, the transport service must be convenient and easy to use.

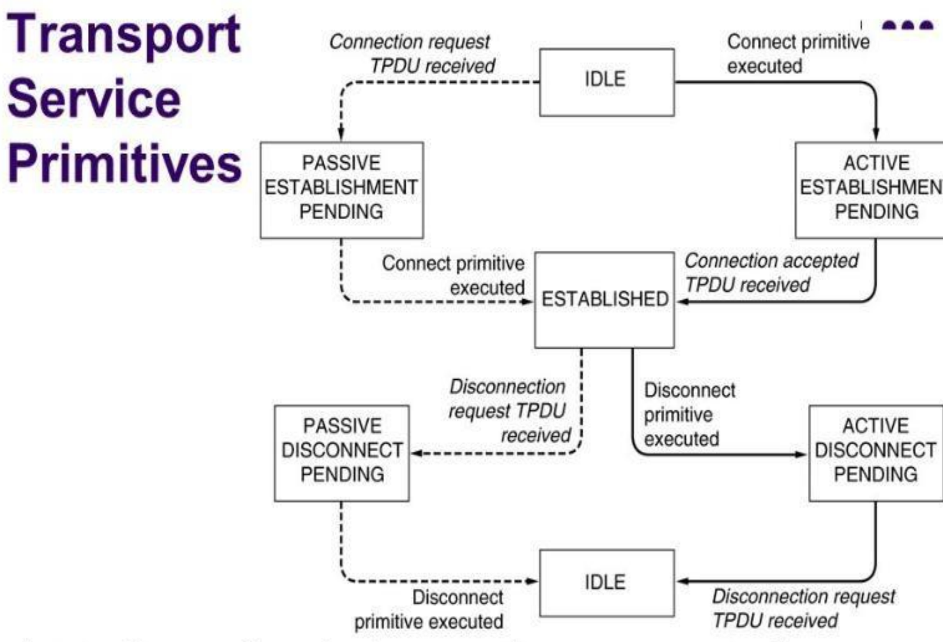
Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

The primitives for a simple transport service:

Eg: Consider an application with a server and a number of remote clients.

1. The server executes a “LISTEN” primitive by calling a library procedure that makes a System call to block the server until a client turns up.
2. When a client wants to talk to the server, it executes a “CONNECT” primitive, with “CONNECTION REQUEST” TPDU sent to the server.
3. When it arrives, the TE unblocks the server and sends a “CONNECTION ACCEPTED” TPDU back to the client.
4. When it arrives, the client is unblocked and the connection is established. Data can now be exchanged using “SEND” and “RECEIVE” primitives.
5. When a connection is no longer needed, it must be released to free up table space within the 2 transport entries, which is done with “DISCONNECT” primitive by sending “DISCONNECTION REQUEST” TPDU. This disconnection can b done either by asymmetric variant (connection is released, depending on other one) or by symmetric variant (connection is released, independent of other one).

STATE DIAGRAM

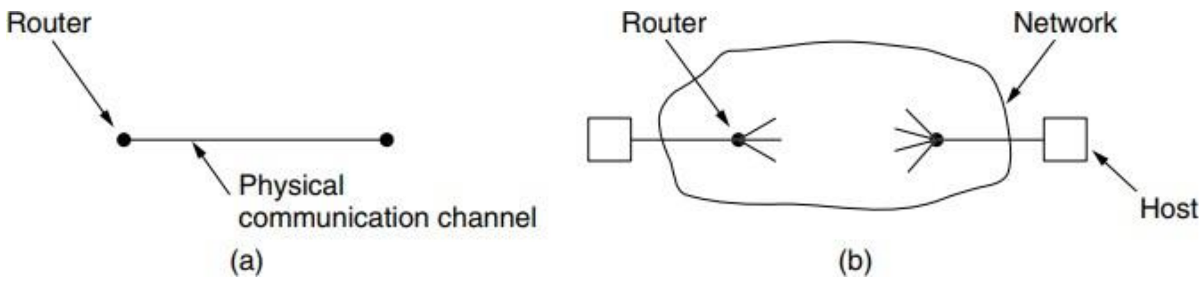

Transport Service Primitives
Connection request TPDU received
IDLE
Connect primitive executed
PASSIVE ESTABLISHMENT PENDING
ACTIVE ESTABLISHMENT PENDING
Connect primitive executed
ESTABLISHED
Connection accepted TPDU received
Disconnection request TPDU received
Disconnect primitive executed
PASSIVE DISCONNECT PENDING
ACTIVE DISCONNECT PENDING
IDLE
Disconnect primitive executed
Disconnection request TPDU received

A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Each transition is triggered by some event, either a primitive executed by the local transport user or an incoming packet. For simplicity, we assume here that each TPDU is separately acknowledged. We also assume that a symmetric disconnection model is used, with the client going first. Please note that this model is quite unsophisticated.

ELEMENTS OF TRANSPORT PROTOCOLS

- The transport service is implemented by a transport protocol used between the two transport entities.
- The transport protocols resemble the data link protocols. Both have to deal with error control, sequencing, and flow control, among other issues.The difference transport protocol and data link protocol depends upon the environment in which they are operated. These differences are due to major dissimilarities between the environments in which the two protocols operate, as shown in Figure.
- 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.



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

The elements of transport protocols are:

1. ADDRESSING

2. Connection Establishment.

3. Connection Release.

4. Error control and flow control

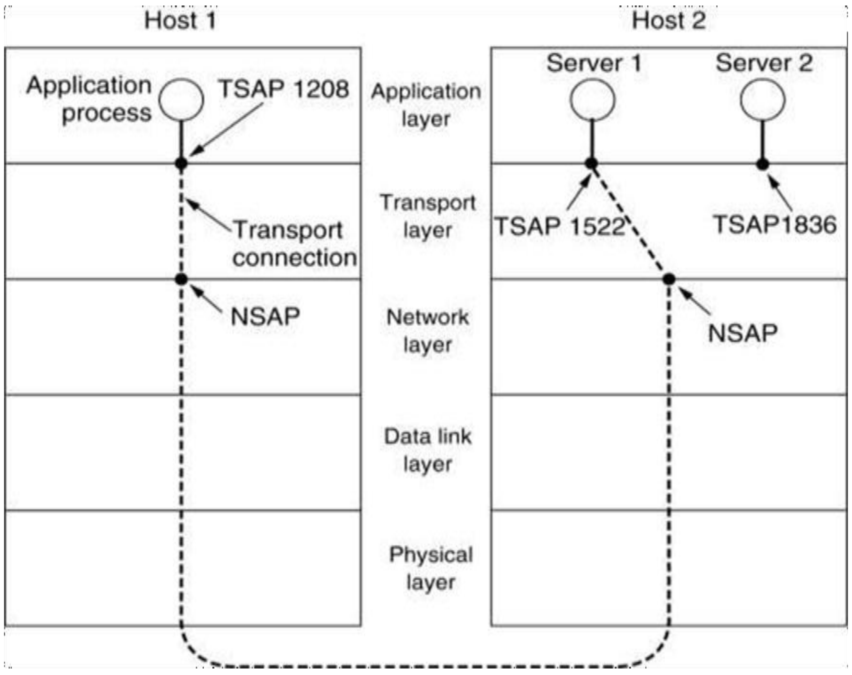
5. Multiplexing

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.



TSAP and NSAP network connections

The above diagram shows the relationship between the NSAP, TSAP and transport connection. Application processes, both clients and servers, can attach themselves to a TSAP to establish a connection to a remote TSAP.

- These connections run through NSAPs on each host, as shown. 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 end points that share that NSAP.
- 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:

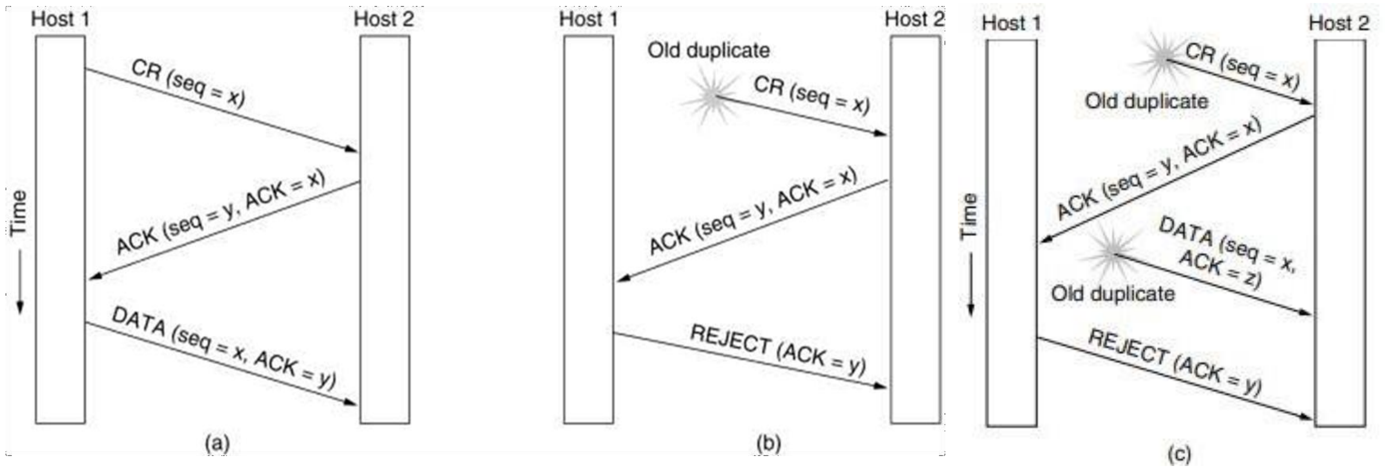
1. A mail server process attaches itself to TSAP 1522 on host 2 to wait for an incoming call. How a process attaches itself to a TSAP is outside the networking model and depends entirely on the local operating system. A call such as our LISTEN might be used, for example.
2. An application process on 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 devise 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 both sides to begin sending with the same sequencenumber.



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 appearing out 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 that it 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. Inthis 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 y tells 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 the protocol to fail and have a connection set up byaccident when no one wants it.

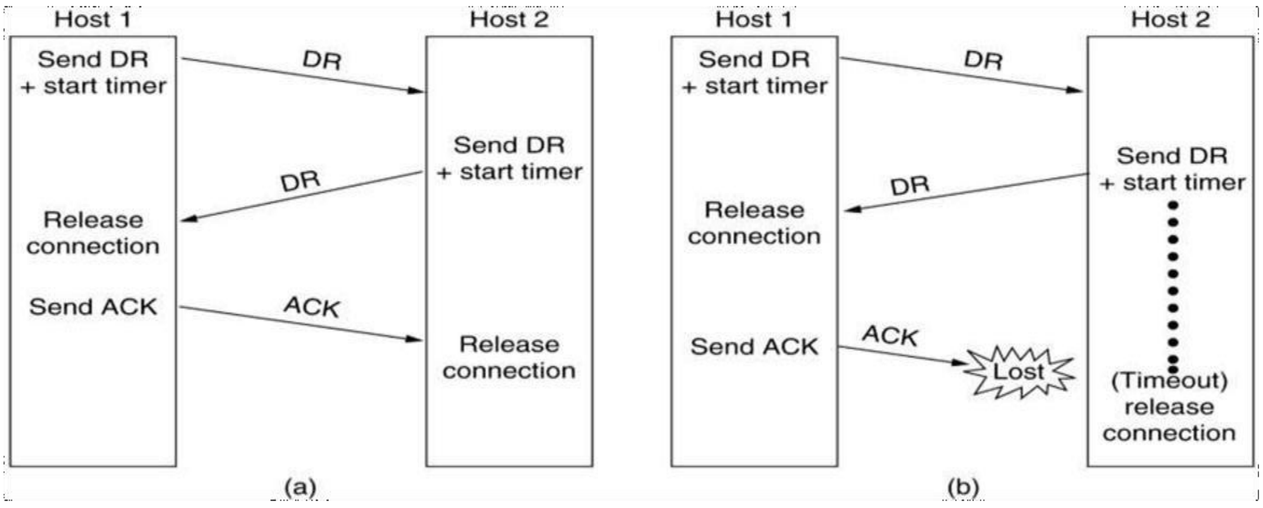
3. CONNECTION RELEASE:

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.



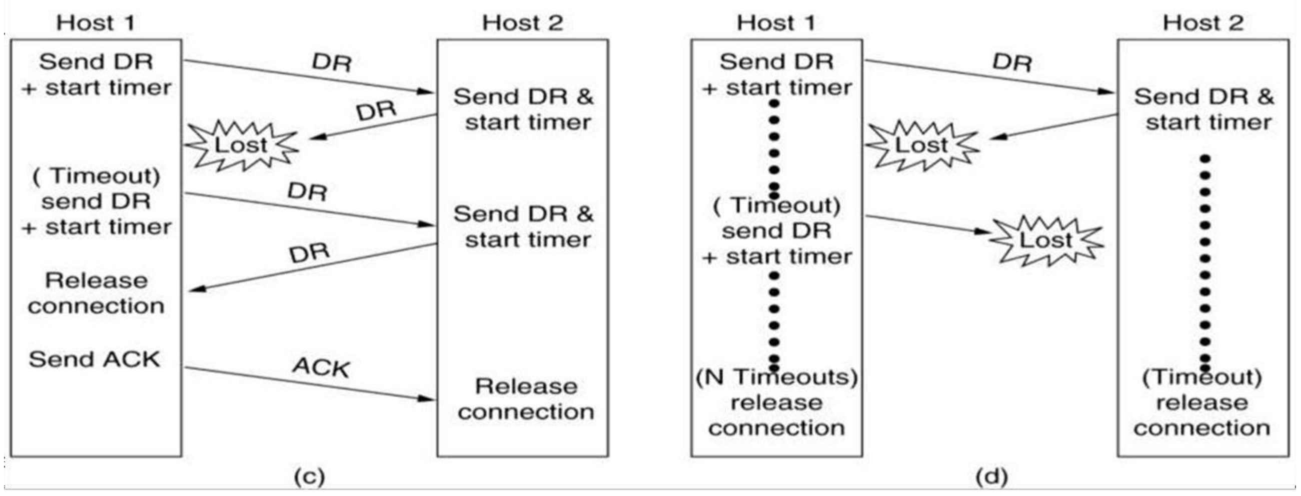


Fig-(a)	Fig-(b)	Fig-(c)	Fig-(d)
One of the user sends a DISCONNECTON REQUEST TPDU in order to initiate connection release. When it arrives, the recipient sends back a DR-TPDU, too, and starts a timer. When this DR arrives, the original sender sends back anACK-TPDU and releases theconnection. Finally, when the ACK-TPDUarrives, 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 repeated attempts to retransmit the 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 Buffers

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 charactersfrom file transfers, some problems may occur:

 If the buffer size is chosen equal to the largest possible TPDU, space will be wasted whenever a short TPDU arrives.

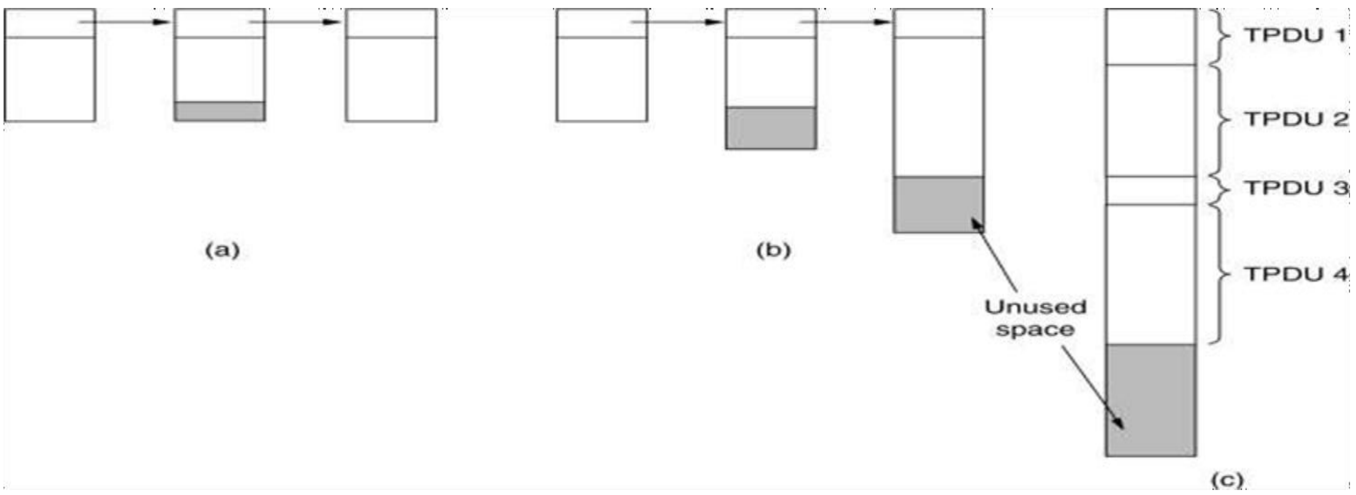
 If the buffer size is chosen less than the maximum TPDU size, multiple buffers will be needed for long TPDUs.

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.

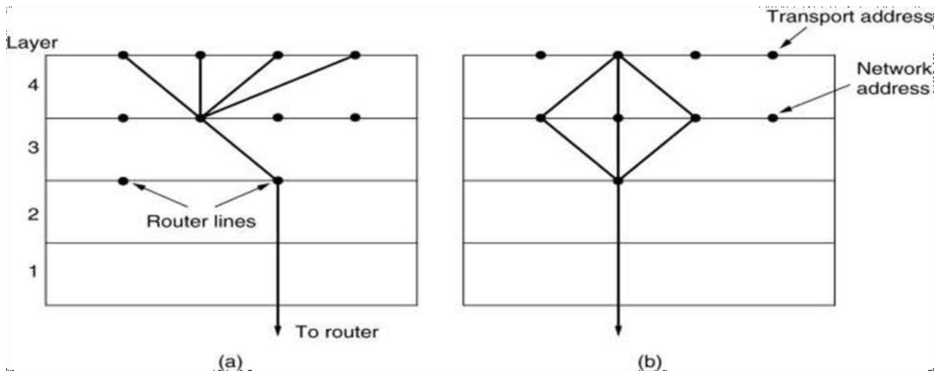
1. Source Buffering is used for low band width bursty traffic
2. Destination Buffering is used for high band width smooth traffic.
3. Dynamic Buffering is used if the traffic pattern changes randomly.



(a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per 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:



(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 service will be expensive.

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

TRANSPORT LAYER PROTOCOLS

USER DATAGRAM PROTOCOL

The Internet has two main protocols in the transport layer, a connectionless protocol and a connection-oriented one. The protocols complement each other. The connectionless protocol is UDP. It does almost nothing beyond sending packets between applications, letting applications build their own protocols on top as needed.

The connection-oriented protocol is TCP. It does almost everything. It makes connections and adds reliability with retransmissions, along with flow control and congestion control, all on behalf of the applications that use it. Since UDP is a transport layer protocol that typically runs in the operating system and protocols that use UDP typically run in user s pace, these uses might be considered applications.

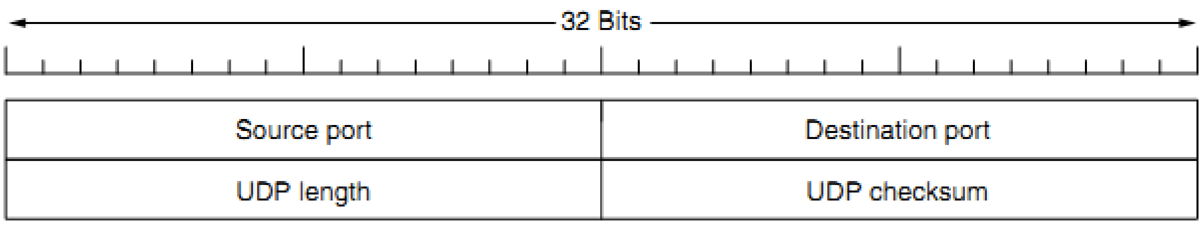
INTRODUCTION TO UDP

➢ The Internet protocol suite supports a connectionless transport protocol called UDP (User Datagram Protocol). UDP provides a way for applications to send encapsulated IP datagrams without having to establish a connection.

➢ UDP transmits segments consisting of an 8-byte header followed by the pay-load. The two ports serve to identify the end-points within the source and destination machines.

➢ When a UDP packet arrives, its payload is handed to the process attached to the destination port. This attachment occurs when the BIND primitive. Without the port fields, the transport layer would not know what to do with each incoming packet. With them, it delivers the embedded segment to the correct application.

The UDP header



Source port, destination port: Identifies the end points within the source and destination machines.
UDP length: Includes 8-byte header and the data
UDP checksum: Includes the UDP header, the UDP data padded out to an even number of bytes if need be. It is an optional field.
UDP is a connectionless, unreliable transport protocol with minimum overhead and perform limited error checking, hence it is suitable to send a small messages only.

Features and applications of UDP

- UDP is a connectionless, unreliable transport protocol with minimum overhead and perform limited error checking, hence it is suitable to send a small messagesonly.
- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. UDP is suitable for a process with internal flow and error control mechanisms.
- UDP is a suitable transport protocol for multicasting, it supports interactive multimedia traffic and uses RTP and RTCP as supporting protocols.UDP is used for management processes such as SNMP.
- UDP is used for Route Updating Protocols such as Routing Information Protocol (RIP).

TCP (TRANSMISSION CONTROL PROTOCOL)

It was specifically designed to provide a reliable end-to end byte stream over an unreliable network. It was designed to adapt dynamically to properties of the inter network and to be robust in the face of many kinds of failures.

Each machine supporting TCP has a TCP transport entity, which accepts user data streams from local processes, breaks them up into pieces not exceeding 64kbytes and sends each piece as a separate IP datagram.

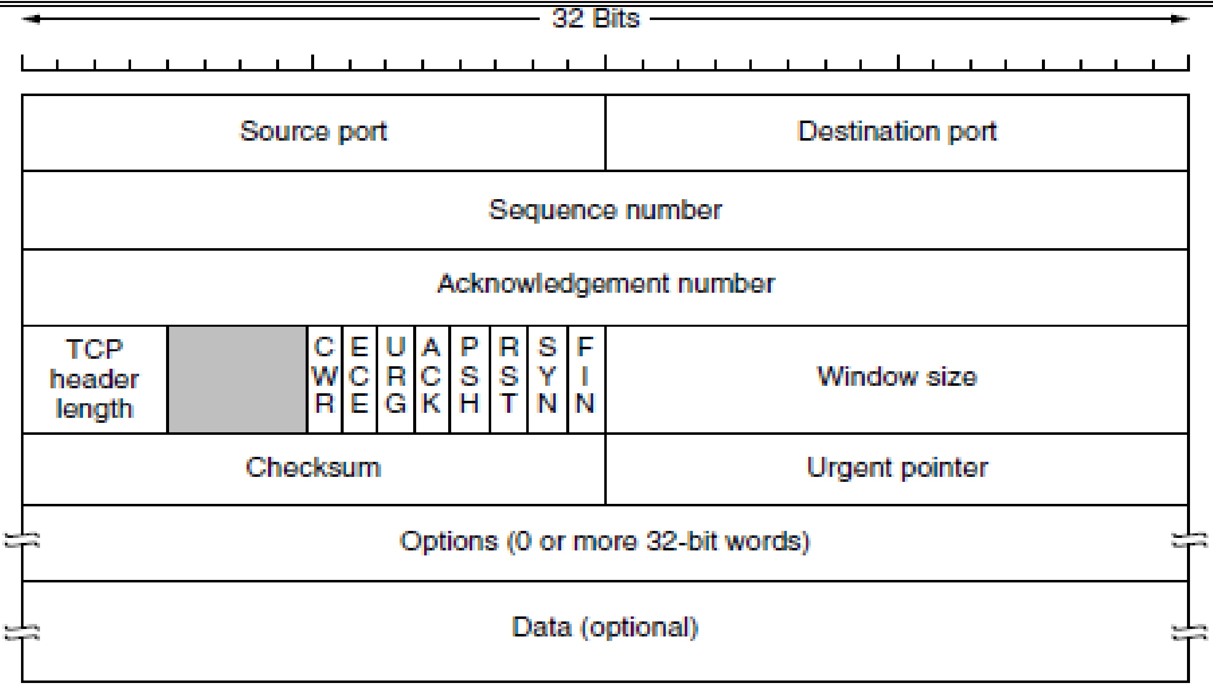
When these datagrams arrive at a machine, they are given to TCP entity, which reconstructs the original byte streams. It is up to TCP to time out and retransmits them as needed, also to reassemble datagrams into messages in proper sequence.

The TCP Protocol

- ➢ A key feature of TCP, and one which dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number.
- ➢ When the Internet began, the lines between routers were mostly 56-kbps leased lines, so a host blasting away at full speed took over 1 week to cyclethrough the sequence numbers.
- ➢ The basic protocol used by TCP entities is the **sliding window protocol**.
- ➢ When a sender transmits a segment, it also starts a timer.
- ➢ When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist, otherwise without data) bearing anacknowledgement number equal to the next sequence number it expects to receive.
- ➢ If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.

The TCP Segment Header

Every segment begins with a fixed-format, 20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.



Source Port, Destination Port : Identify local end points of the connections.
Sequence number: Specifies the sequence number of the segment.
Acknowledgement Number: Specifies the next byte expected.
TCP header length: Tells how many 32-bit words are contained in TCP header.
CWR and ECE: are used to signal congestion when ECN (Explicit Congestion Notification) is used, as specified in RFC 3168. *ECE* is set to signal an *ECN-Echo* to a TCP sender to tell it to slow down when the TCP receiver gets a congestion indication from the network. *CWR* is set to signal *Congestion Window Reduced* from the TCP sender to the TCP receiver so that it knows the sender has slowed down and can stop sending the *ECN-Echo*.
URG: It is set to 1 if URGENT pointer is in use, which indicates start of urgent data.
ACK: It is set to 1 to indicate that the acknowledgement number is valid.
PSH: Indicates pushed data
RST: It is used to reset a connection that has become confused due to reject an invalid segment or refuse an attempt to open a connection.
FIN: Used to release a connection.
SYN: Used to establish connections.

TCP Connection Establishment

To establish a connection, one side, say, the server, passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives, either specifying a specific source or nobody in particular.
The other side, say, the client, executes a CONNECT primitive, specifying the IP address and port to which it wants to connect, the maximum TCP segmentsize it is willing to accept, and optionally some user data (e.g., a password).

The CONNECT primitive sends a TCP segment with the *SYN* bit on and *ACK* bit off and waits for a response.

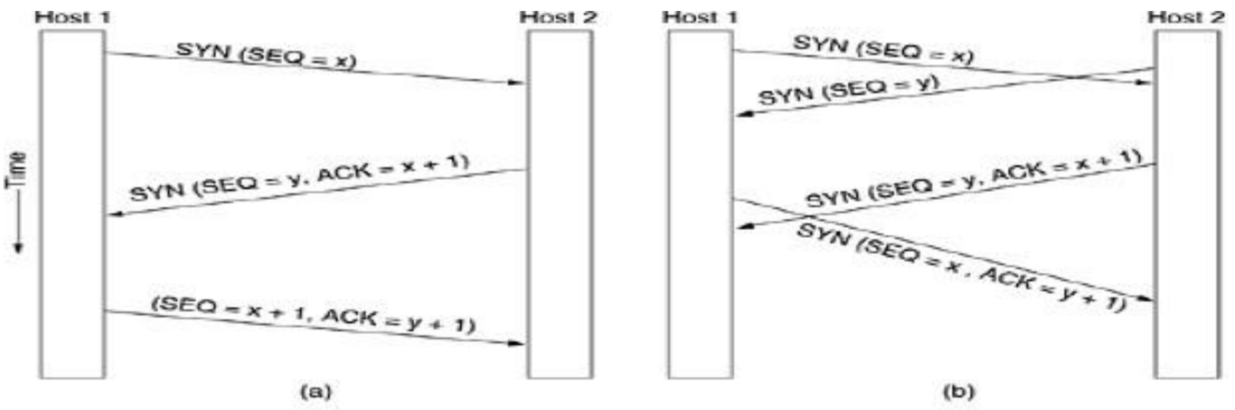

Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
Time
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x, ACK = y + 1)
(b)

Fig 4.12: a) TCP Connection establishment in the normal case b) Call Collision

TCP Connection Release

- Although TCP connections are full duplex, to understand how connections are released it is best to think of them as a pair of simplex connections.
- Each simplex connection is released independently of its sibling. To release a connection, either party can send a TCP segment with the *FIN* bit set, which means that it has no more data to transmit.
- When the *FIN* is acknowledged, that direction is shut down for new data. Data may continue to flow indefinitely in the other direction, however.
- When both directions have been shut down, the connection is released.
- Normally, four TCP segments are needed to release a connection, one *FIN* and one *ACK* for each direction. However, it is possible for the first *ACK* and thesecond *FIN* to be contained in the same segment, reducing the total count to three.

TCP Connection Management Modeling

- The steps required establishing and release connections can be represented in a finite state machine with the 11 states listed in below Figure.
- In each state, certain events are legal.
- When a legal event happens, some action may be taken. If some other event happens, an error is reported.

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

re 4.13. The states used in the TCP connection management finite state machine.

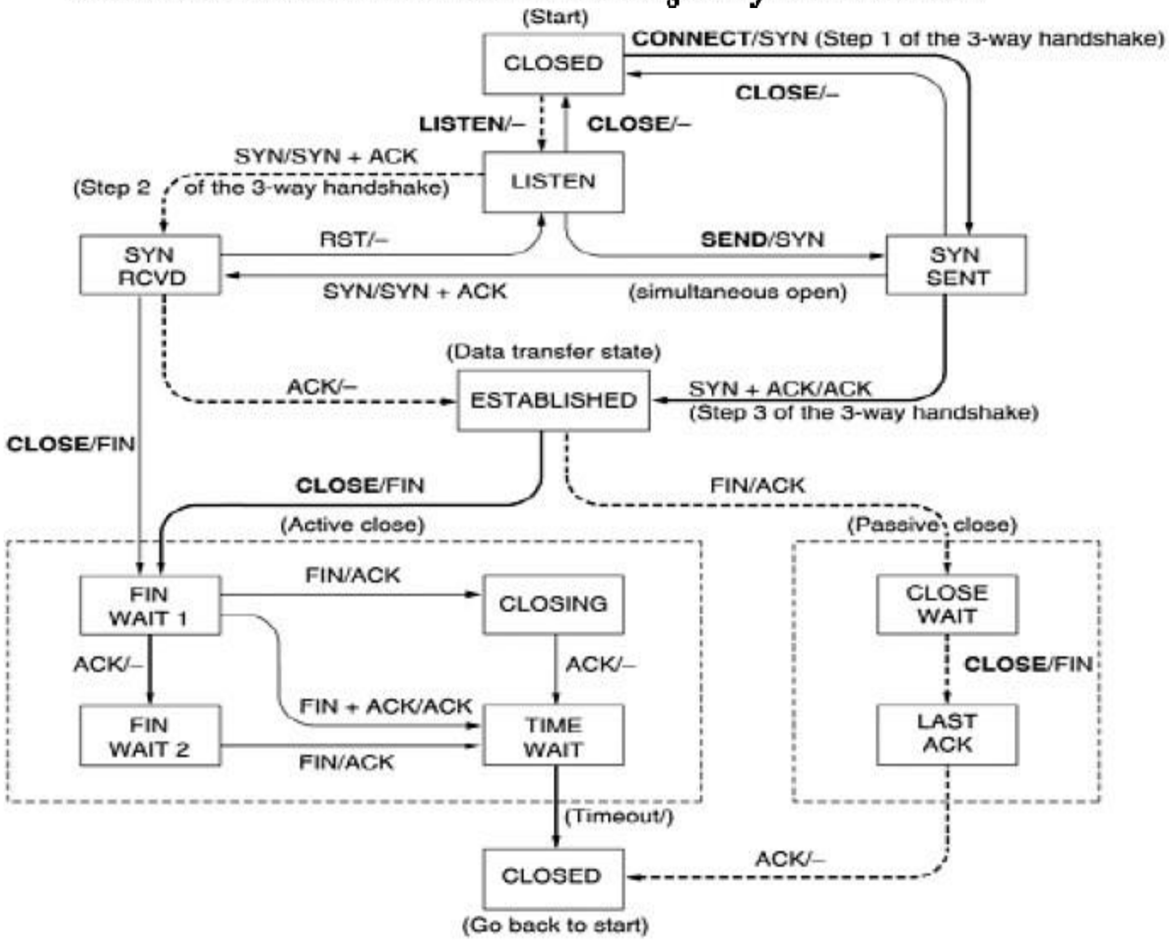


Figure 4.14 - TCP connection management finite state machine.

TCP Connection management from server's point of view:

1. The server does a **LISTEN** and settles down to see who turns up.
2. When a **SYN** comes in, the server acknowledges it and goes to the **SYNRCVD** state.
3. When the servers **SYN** is itself acknowledged the 3-way handshake is complete and server goes to the **ESTABLISHED** state. Data transfer can now occur.
4. When the client has had enough, it does a close, which causes a **FIN** to arrive at the server [dashed box marked passive close].
5. The server is then signaled.
6. When it too, does a **CLOSE**, a **FIN** is sent to the client.
7. When the client's acknowledgement shows up, the server releases the connection and deletes the connection record.

TCP Transmission Policy

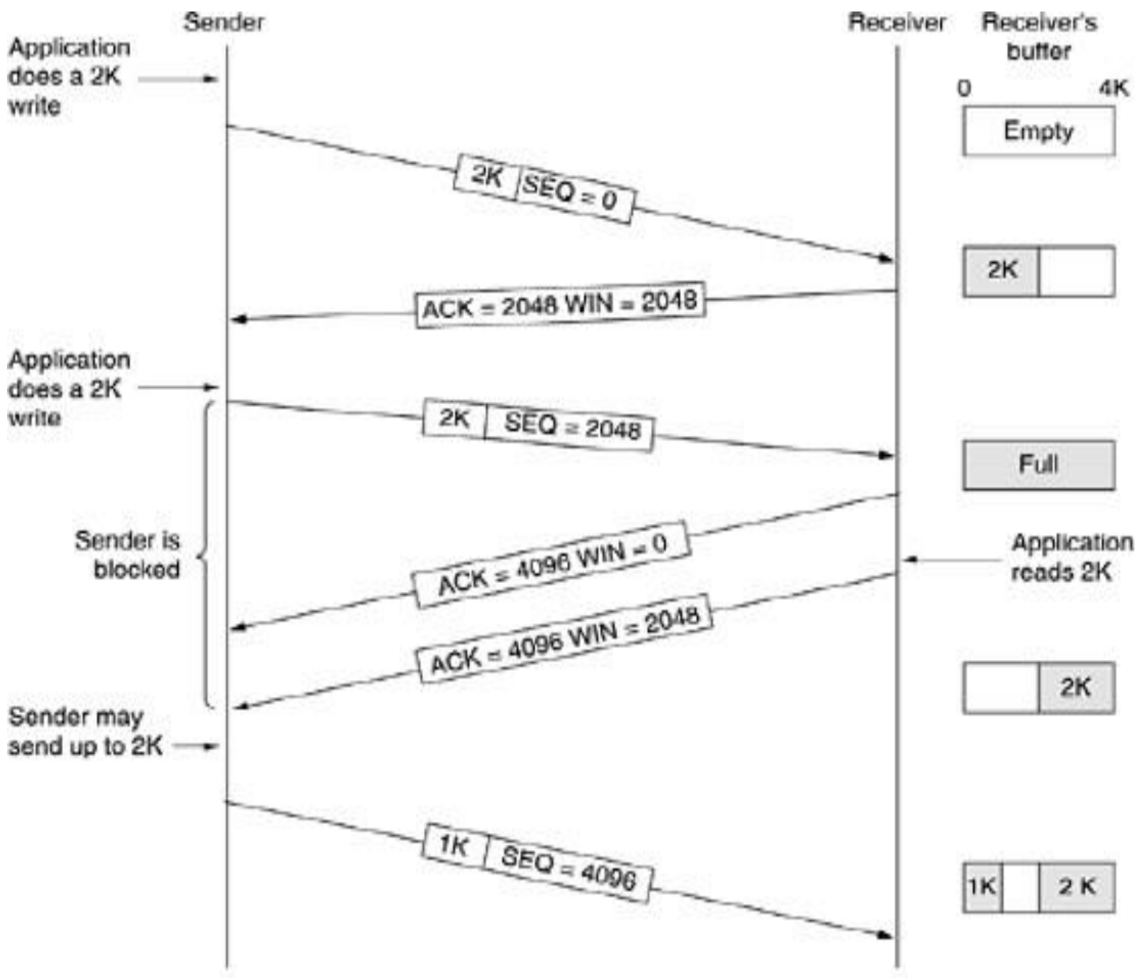


Figure 4.15 - Window management in TCP.

In the above example, the receiver has 4096-byte buffer.
2. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
3. Now the receiver will advertise a window of 2048 as it has only 2048 of buffer space, now.
4. Now the sender transmits another 2048 bytes which are acknowledged, but the advertised window is'0'.
5. The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a layerwindow.

TCP SLIDING WINDOW

- Window management in TCP decouples the issues of acknowledgement of the correct receipt of segments and receiver buffer allocation.
- For example, suppose the receiver has a 4096-byte buffer, as shown in below figure. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment. However, since it now has only 2048 bytes of buffer space (until the application removes some data from the buffer), it will advertise a window of 2048 starting at the next byte expected.

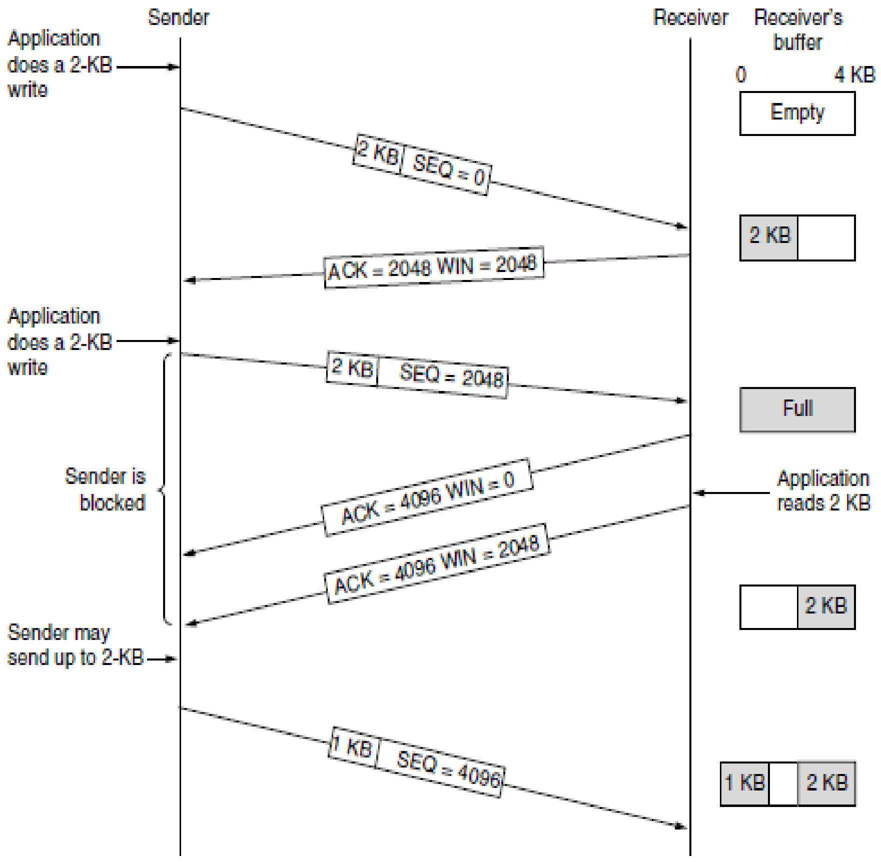


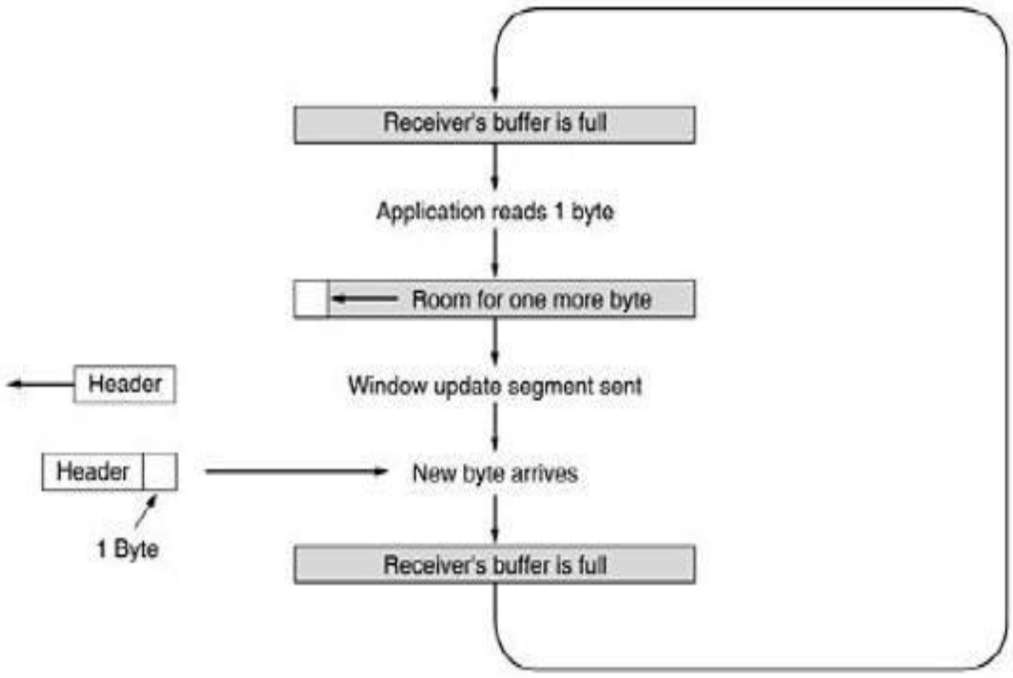
Figure 6-40. Window management in TCP.

- Now the sender transmits another 2048 bytes, which are acknowledged, but the advertised window is of size 0.
- The sender must stop until the application. process on the receiving host has removed some data from the buffer, at which time TCP can advertise a larger window and more data can be sent.
- When the window is 0, the sender may not normally send segments, with two exceptions.
- First, urgent data may be sent, for example, to allow the user to kill the process running on the remote machine.

- Second, the sender may send a 1-byte segment to force the receiver to reannounce the next byte expected and the window size. This packet is called a **window probe**.
- The TCP standard explicitly provides this option to prevent deadlock if a window update ever gets lost. Senders are not required to transmit data as soon as they come in from the application. Neither are receivers required to send acknowledgements as soon as possible.
- One approach that many TCP implementations use to optimize this situation is called **delayed acknowledgements**. The idea is to delay acknowledgements and window updates for up to 500 msec in the hope of acquiring some data on which to hitch a free ride.

SILLY WINDOW SYNDROME:

This is one of the problems that ruin the TCP performance, which occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads data 1 byte at a time.



- Initially the TCP buffer on the receiving side is full and the sender knows this(win=0)
- Then the interactive application reads 1 character from TCP stream.
- Now, the receiving TCP sends a window update to the sender saying that it is all right to send 1 byte.
- The sender obligates and sends 1 byte.
- The buffer is now full, and so the receiver acknowledges the 1 byte segment but sets window to zero.

Nagle's Algorithm:

- Delayed acknowledgements reduce the load placed on the network by the receiver, a sender that sends multiple short packets (e.g., 41-byte packets containing 1 byte of data) is still operating inefficiently. A way to reduce this usage is known as **Nagle's algorithm** (Nagle, 1984).
- Nagle suggested when data come into the sender in small pieces, just send the first piece and buffer all the rest until the first piece is acknowledged.
- Then send all the buffered data in one TCP segment and start buffering again until the next segment is acknowledged. That is, only one short packet can be outstanding at any time. If many pieces of data are sent by the application in one round-trip time,
- Nagle's algorithm will put the many pieces in one segment, greatly reducing the bandwidth used.

- The algorithm additionally says that a new segment should be sent if enough data have trickled in to fill a maximum segment. Nagle's algorithm is widely used by TCP implementations, but there are times when it is better to disable it.
- Particularly, in interactive games that are run over the Internet, the players typically want a rapid stream of short update packets. Gathering the updates to send them in bursts makes the game respond erratically, which makes for unhappy users.
- A more subtle problem is that Nagle's algorithm can sometimes interact with delayed acknowledgements to cause a temporary deadlock: the receiver waits for data on which to piggyback an acknowledgement, and the sender waits on the acknowledgement to send more data.
- This interaction can delay the downloads of Web pages. Because of these problems, Nagle's algorithm can be disabled (which is called the *TCP NODELAY* option).

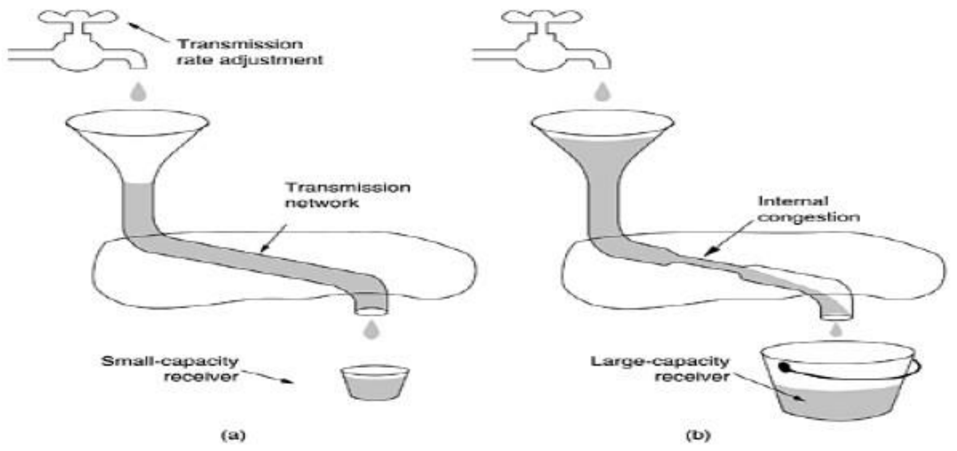
Clark's Solution:

- Clark's solution is to prevent the receiver from sending a window update for 1 byte. Instead, it is forced to wait until it has a decent amount of space available and advertise that instead.
- The receiver should not send a window update until it can handle the maximum segment size it advertised when the connection was established or until its buffer is half empty, whichever is smaller.
- The sender can also help by not sending tiny segments. Instead, it should wait until it can send a full segment, or at least one containing half of the receiver's buffer size.
- *Nagle's algorithm and Clark's solution to the silly window syndrome are complementary.*
- Nagle was trying to solve the problem caused by the sending application delivering data to TCP a byte at a time.
- Clark was trying to solve the problem of the receiving application sucking the data up from TCP a byte at a time. *Both solutions are valid and can work together.*
- The goal is for the sender not to send small segments and the receiver not to ask for them. The receiving TCP can go further in improving performance than just doing window updates in large units.
- Like the sending TCP, it can also buffer data, so it can block a READ request from the application until it has a large chunk of data for it. Doing so reduces the number of calls to TCP (and the overhead). It also increases the response time, but for noninteractive applications like file transfer, efficiency may be more important than response time to individual requests.
- Another issue that the receiver must handle is that segments may arrive out of order. The receiver will buffer the data until it can be passed up to the application in order. Actually, nothing bad would happen if out-of-order segments were discarded, since they would eventually be retransmitted by the sender, but it would be wasteful.
- Acknowledgements can be sent only when all the data up to the byte acknowledged have been received. This is called a **cumulative acknowledgement**. If the receiver gets segments 0, 1, 2, 4, 5, 6, and 7, it can acknowledge everything up to and including the last byte in segment 2. When the sender times out, it then retransmits segment 3. As the receiver has buffered segments 4 through 7, upon receipt of segment 3 it can acknowledge all bytes up to the end of segment 7.

TCP CONGESTION CONTROL

TCP does to try to prevent the congestion from occurring in the first place in the following way:

When a connection is established, a suitable window size is chosen and the receiver specifies a window based on its buffer size. If the sender sticks to this window size, problems will not occur due to buffer overflow at the receiving end. But they may still occur due to internal congestion within the network. Let's see this problem occurs.



a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver.

In fig (a): We see a thick pipe leading to a small- capacity receiver. As long as the sender does not send more water than the bucket can contain, no water willbe lost.

In fig (b): The limiting factor is not the bucket capacity, but the internal carrying capacity of the n/w. if too much water comes in too fast, it will backup andsome will be lost.

- When a connection is established, the sender initializes the congestion window to the size of the max segment in use our connection.
- It then sends one max segment .if this max segment is acknowledged before the timer goes off, it adds one segment s worth of bytes to the congestion window to make it two maximum size segments and sends 2 segments.
- As each of these segments is acknowledged, the congestion window is increased by one max segment size.
- When the congestion window is 'n' segments, if all 'n' are acknowledged on time, the congestion window is increased by the byte count corresponding to'n' segments.
- The congestion window keeps growing exponentially until either a time out occurs or the receiver's window is reached.
- The internet congestion control algorithm uses a third parameter, the **"threshold"** in addition to receiver and congestion windows.

Different congestion control algorithms used by TCP are:

- **Slow Start**
- **Fast Recovery**

TCP uses a congestion window and a congestion policy that avoid congestion. Previously, we assumed that only the receiver can dictate the sender's window size. We ignored another entity here, the network. If the network cannot deliver the data as fast as it is created by the sender, it must tell the sender to slow down. In other words, in addition to the receiver, the network is a second entity that determines the size of the sender's window.

Congestion policy in TCP

Slow Start Phase: starts slowly increment is exponential to threshold.
Congestion Avoidance Phase: After reaching the threshold increment is by 1.
Congestion Detection Phase: Sender goes back to Slow start phase or Congestion avoidance phase.

<u>Slow Start Phase:</u>

Exponential increment – In this phase after every RTT 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

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

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.

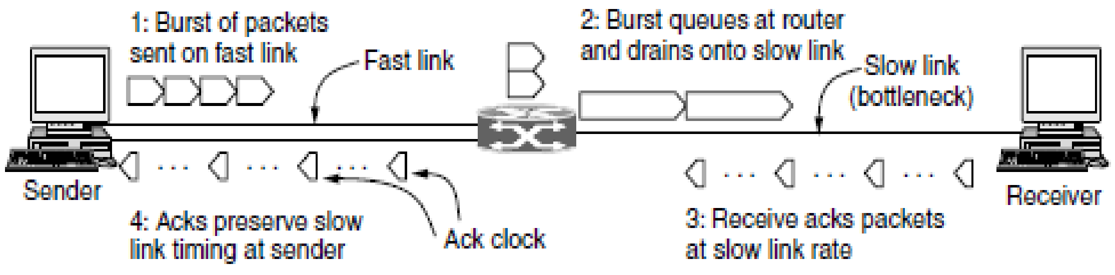

1: Burst of packets sent on fast link
Fast link
2: Burst queues at router and drains onto slow link
Slow link (bottleneck)
Sender
Receiver
4: Acks preserve slow link timing at sender
Ack clock
3: Receive acks packets at slow link rate

Figure 6-43. A burst of packets from a sender and the returning ack clock.

- **Case 1 : Retransmission due to Timeout** – In this case congestion possibility is high.

  (a) ssthresh(slow start threshold) is reduced to half of the current window size.

  (b) set cwnd(congestion window) = 1

  (c) start with slow start phase again.

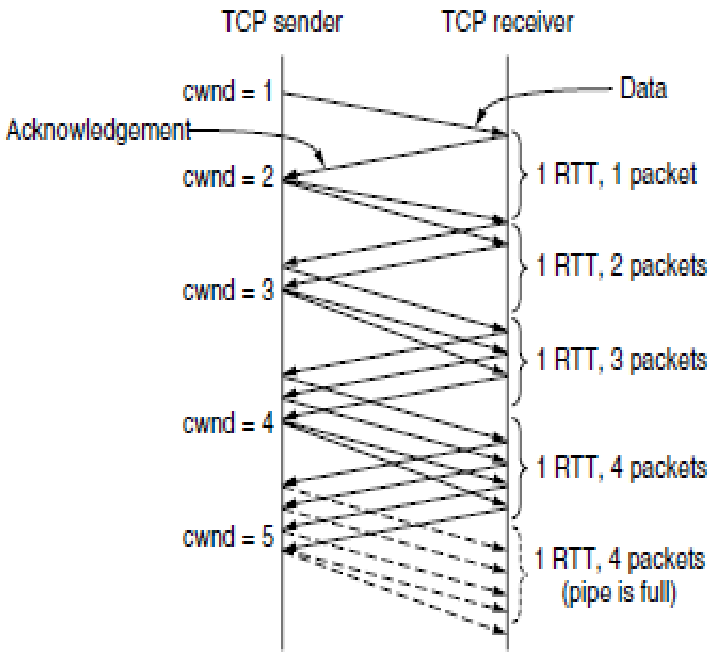


Fig: Additive increase from an initial congestion window of one segment.

- **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

TCP somewhat arbitrarily assumes that three duplicate acknowledgements imply that a packet has been lost. The identity of the lost packet can be inferred from the acknowledgement number as well. It is the very next packet in sequence. This packet can then be retransmitted right away, before the retransmission timeout fires.

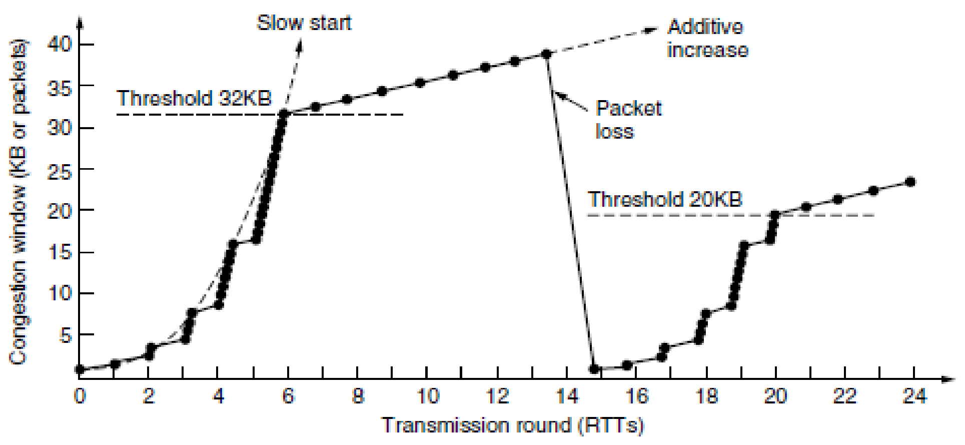


Figure 6-46. Slow start followed by additive increase in TCP Tahoe.

- This heuristic is called **fast retransmission**. This version of TCP is called TCP Tahoe.

- After it fires, the slow start threshold is still set to half the current congestion window, just as with a timeout. Slow start can be restarted by setting the congestion window to one packet.
- With this window size, a new packet will be sent after the one round-trip time that it takes to acknowledge the retransmitted packet along with all data that had been sent before the loss was detected.
- Example: The maximum segment size here is 1 KB. Initially, the congestion window was 64 KB, but a timeout occurred, so the threshold is set to 32 KB and the congestion window to 1 KB for transmission 0. The congestion window grows exponentially until it hits the threshold (32 KB). The window is increased every time a new acknowledgement arrives rather than continuously, which leads to the discrete staircase pattern. After the threshold is passed, the window grows linearly. It is increased by one segment every RTT. The transmissions in round 13 are unlucky (they should have known), and one of them is lost in the network. This is detected when three duplicate acknowledgements arrive. At that time, the lost packet is retransmitted, the threshold is set to half the current window (by now 40 KB, so half is 20 KB), and slow start is initiated all over again. Restarting with a congestion window of one packet takes one round-trip time for all of the previously transmitted data to leave the network and be acknowledged, including the retransmitted packet. The congestion window grows with slow start as it did previously, until it reaches the new threshold of 20 KB. At that time, the growth becomes linear again. It will continue in this fashion until another packet loss is detected via duplicate acknowledgements or a timeout (or the receiver's window becomes the limit).

Fast recovery:

- It is a temporary mode that aims to maintain the ack clock running with a congestion window that is the new threshold, or half the value of the congestion window at the time of the fast retransmission.
- To do this, duplicate acknowledgements are counted (including the three that triggered fast retransmission) until the number of packets in the network has fallen to the new threshold. This takes about half a round-trip time.
- From then on, a new packet can be sent for each duplicate acknowledgement that is received. One round-trip time after the fast retransmission, the lost packet will have been acknowledged. At that time, the stream of duplicate acknowledgements will cease and fast recovery mode will be exited.
- The congestion window will be set to the new slow start threshold and grows by linear increase. The upshot of this heuristic is that TCP avoids slow start, except when the connection is first started and when a timeout occurs.
- The latter can still happen when more than one packet is lost and fast retransmission does not recover adequately.
- Instead of repeated slow starts, the congestion window of a running connection follows a **sawtooth** pattern of additive increase (by one segment every RTT) and multiplicative decrease (by half in one RTT).
- This is exactly the AIMD rule that we sought to implement rule that we sought to implement. This sawtooth behavior is produced by TCP Reno, named after the 4.3BSD Reno release in 1990 in which it was included. TCP Reno is essentially TCP Tahoe plus fast recovery.
- After an initial slow start, the congestion window climbs linearly until a packet loss is detected by duplicate acknowledgements.
- The lost packet is retransmitted and fast recovery is used to keep the ack clock running until the retransmission is acknowledged. At that time, the congestion window is resumed from the new slow start threshold, rather than from 1.
- This behavior continues indefinitely, and the connection spends most of the time with its congestion window close to the optimum value of the bandwidth-delay product.

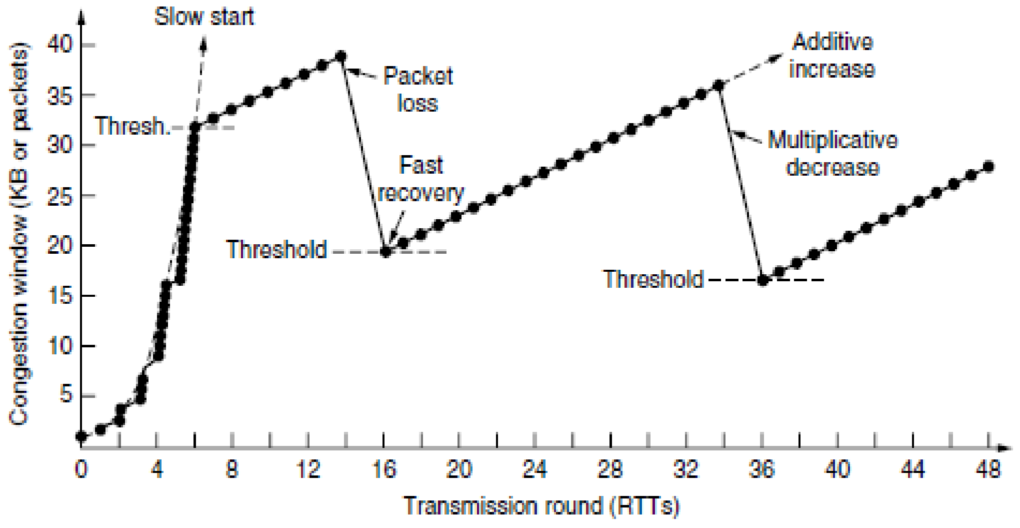


Figure 6-47. Fast recovery and the sawtooth pattern of TCP Reno.

- Complexity of TCP comes from inferring from a stream of duplicate acknowledgements which packets have arrived and which packets have been lost. The cumulative acknowledgement number does not provide this information.
- A simple fix is the use of **SACK** (**Selective ACKnowledgements**), which lists up to three ranges of bytes that have been received. With this information, the sender can more directly decide what packets to retransmit and track the packets in flight to implement the congestion window.
- When the sender and receiver set up a connection, they each send the *SACK permitted* TCP option to signal that they understand selective acknowledgements. Once SACK is enabled for a connection, it works as shown below figure.
- A receiver uses the TCP *Acknowledgement number* field in the normal manner, as a cumulative acknowledgement of the highest in-order byte that has been received.
- When it receives packet 3 out of order (because packet 2 was lost), it sends a *SACK option* for the received data along with the (duplicate) cumulative acknowledgement for packet 1.
- The *SACK option* gives the byte ranges that have been received above the number given by the cumulative acknowledgement. The first range is the packet that triggered the duplicate acknowledgement.
- The next ranges, if present, are older blocks. Up to three ranges are commonly used. By the time packet 6 is received, two SACK byte ranges are used to indicate that packet 6 and packets 3 to 4 have been received, in addition to all packets up to packet 1. From the information in each *SACK option* that it receives, the sender can decide which packets to retransmit.
- ECN (Explicit Congestion Notification) in addition to packet loss as a congestion signal. ECN is an IP layer mechanism to notify hosts of congestion that the TCP receiver can receive congestion signals from IP.
- The use of ECN is enabled for a TCP connection when both the sender and receiver indicate that they are capable of using ECN by setting the *ECE* and *CWR* bits during connection establishment.
- If ECN is used, each packet that carries a TCP segment is flagged in the IP header to show that it can carry an ECN signal. Routers that support ECN will set a congestion signal on packets that can carry ECN flags when congestion is approaching, instead of dropping those packets after congestion has occurred.

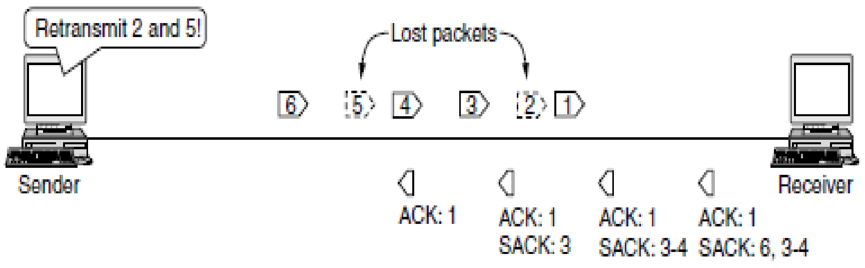


Figure 6-48. Selective acknowledgements.

- The TCP receiver is informed if any packet that arrives carries an ECN congestion signal.
- The receiver then uses the *ECE* (ECN Echo) flag to signal the TCP sender that its packets have experienced congestion. T
- he sender tells the receiver that it has heard the signal by using the *CWR* (Congestion Window Reduced) flag. The TCP sender reacts to these congestion notifications in exactly the same way as it does to packet loss that is detected via duplicate acknowledgements.

Differences between TCP and UDP

The following table highlights the major differences between TCP and UDP.

Key/parameter	TCP	UDP
Definition	It is a communications protocol, using which the data is transmitted between systems over the network.In this, the data is transmitted in the form of packets. It includes error-checking, guarantees the delivery and preserves the order of the data packets.	It is same as the TCP protocol except this doesn't guarantee the error-checking and data recovery. If you use this protocol, the data will be sent continuously, irrespective of the issues in the receiving end.
Design/Type of connection	It is a connection-oriented protocol, which means that the connection needs to be established before the data is transmitted over the network.	It is a connectionless protocol, which means that it sends the data without checking whether the system is ready to receive or not.
Reliability	TCP is more reliable as it provides error checking support and also guarantees delivery of data to the destination router.	UDP, on the other hand, provides only basic error checking support using checksum. So, the delivery of data to the destination cannot be guaranteed in UDP as in case of TCP.
Speed	TCP is slower than UDP as it performs error checking, flow control, and provides assurance for the delivery of	UDP is faster than TCP as it does not guarantee the delivery of data packets.

Data transmission	In TCP, the data is transmitted in a particular sequence which means that packets arrive in-order at the receiver.	There is no sequencing of data in UDP in order to implement ordering it has to be managed by the application layer.
Performance	TCP is slower and less efficient in performance as compared to UDP. Also TCP is heavy-weight as compared to UDP.	UDP is faster and more efficient than TCP.
Retransmission	Retransmission of data packets is possible in TCP in case packet get lost or need to resend.	Retransmission of packets is not possible in UDP.
Sequencing	The Transmission Control Protocol has a function that allows data to be sequenced (TCP). This implies that packets arrive at the recipient in the sequence they were sent.	In UDP, there is no data sequencing. The application layer must control the order if it is necessary.
Acknowledgment	TCP uses the three-way-handshake concept. In this concept, if the sender receives the ACK, then the sender will send the data. TCP also has the ability to resend the lost data.	UDP does not wait for any acknowledgment; it just sends the data.
Flow control mechanism	It follows the flow control mechanism in which too many packets cannot be sent to the receiver at the same time.	This protocol follows no such mechanism.
Error checking	TCP performs error checking by using a checksum. When the data is corrected, then the data is retransmitted to the receiver.	It does not perform any error checking, and also does not resend the lost data packets.
Header size	TCP uses a variable-length (20-60) bytes header.	UDP has a fixed-length header of 8 bytes.
Handshake	Handshakes such as SYN, ACK, and SYNACK are used.	It's a connectionless protocol, which means it doesn't require a handshake.
Broadcasting	Broadcasting is not supported by TCP.	Broadcasting is supported by UDP.
Examples	HTTP, HTTPs, FTP, SMTP, and Telnet use TCP, military services, web browsing, and e-mail..	DNS, DHCP, TFTP, SNMP, RIP, and VoIP use UDP, game streaming, video and music streaming.

STREAM CONTROL TRANSMISSION PROTOCOL

- SCTP stands for **Stream Control Transmission Protocol**.
- It is a connection- oriented protocol in computer networks which provides a full-duplex association i.e., transmitting multiple streams of data between two end points at the same time that have established a connection in network.
- It is sometimes referred to as next generation TCP or TCPng, SCTP makes it easier to support telephonic conversation on Internet. A telephonic conversation requires transmitting of voice along with other data at the same time on both ends, SCTP protocol makes it easier to establish reliable connection.
- SCTP is also intended to make it easier to establish connection over wireless network and managing transmission of multimedia data. SCTP is a standard protocol (RFC 2960) and is developed by Internet Engineering Task Force (IETF).

Characteristics of SCTP :

1. **Unicast with Multiple properties –**
   It is a point-to-point protocol which can use different paths to reach end host.
2. **Reliable Transmission –**
   It uses SACK and checksums to detect damaged, corrupted, discarded, duplicate and reordered data. It is similar to TCP but SCTP is more efficient when it comes to reordering of data.

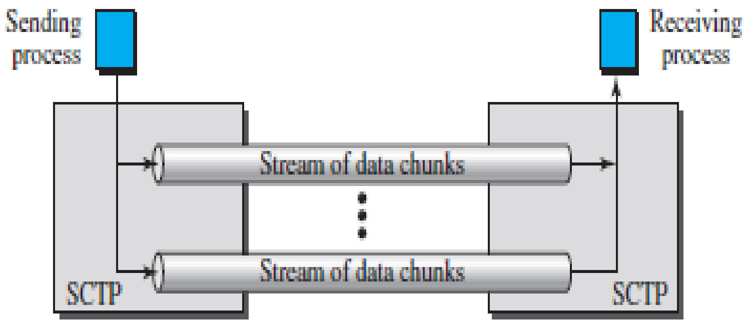


Fig: Multiple-stream concept

3. **Message oriented –**
   Each message can be framed and we can keep order of datastream and tabs on structure. For this, In TCP, we need a different layer for abstraction.
4. **Multi-homing –**
   It can establish multiple connection paths between two end points and does not need to rely on IP layer for resilience.

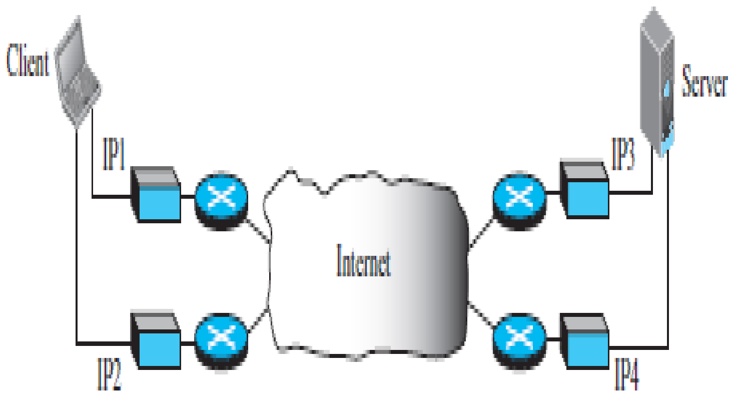


Fig: Multihoming Concept

5. **Security** –
   Another characteristic of SCTP that is security. In SCTP, resource allocation for association establishment only takes place following cookie exchange identification verification for the client (INIT ACK). Man-in-the-middle and denial-of-service attacks are less likely as a result. Furthermore, SCTP doesn't allow for half-open connections, making it more resistant to network floods and masquerade attacks.

Advantages of SCTP :

1. It is a full- duplex connection i.e. users can send and receive data simultaneously.
2. It allows half- closed connections.
3. The message's boundaries are maintained and application doesn't have to split messages.
4. It has properties of both TCP and UDP protocol.
5. It doesn't rely on IP layer for resilience of paths.

Disadvantages of SCTP :

1. One of key challenges is that it requires changes in transport stack on node.
2. Applications need to be modified to use SCTP instead of TCP/UDP.
3. Applications need to be modified to handle multiple simultaneous streams.

Features of SCTP

There are various features of SCTP, which are as follows −

- **Transmission Sequence Number**
  The unit of data in TCP is a byte. Data transfer in TCP is controlled by numbering bytes by using a sequence number. On the other hand, the unit of data in SCTP is a DATA chunk that may or may not have a one-to-one relationship with the message coming from the process because of fragmentation.

- **Stream Identifier**
  In TCP, there is only one stream in each connection. In SCTP, there may be several streams in each association. Each stream in SCTP needs to be identified by using a stream identifier (SI). Each data chunk must carry the SI in its header so that when it arrives at the destination, it can be properly placed in its stream. The 51 is a 16-bit number starting from O.

- **Stream Sequence Number**
  When a data chunk arrives at the destination SCTP, it is delivered to the appropriate stream and in the proper order. This means that, in addition to an SI, SCTP defines each data chunk in each stream with a stream sequence number (SSN).

- **Packets**
  In TCP, a segment carries data and control information. Data is carried as a collection of bytes; control information is defined by six control flags in the header.

  The design of SCTP is totally different: data is carried as data chunks; control information is carried as control chunks.

Figure 24.40 *Comparison between a TCP segment and an SCTP packet*

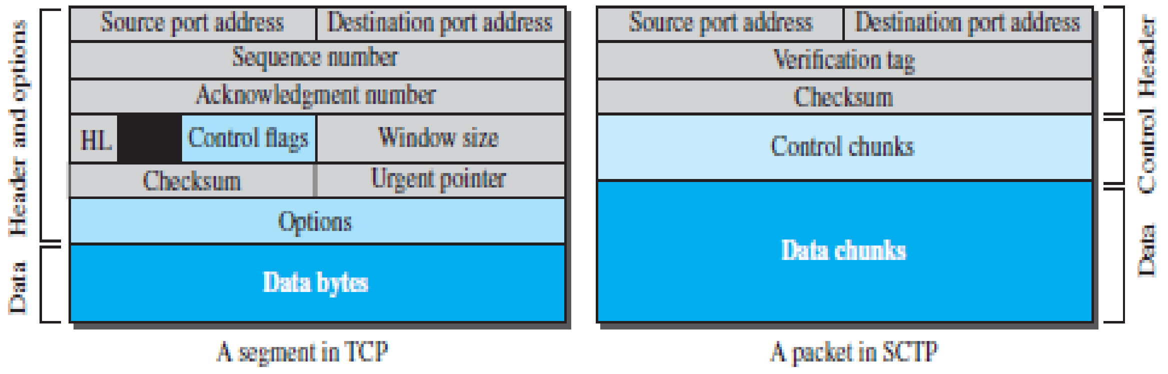


Figure 24.41 *Packets, data chunks, and streams*

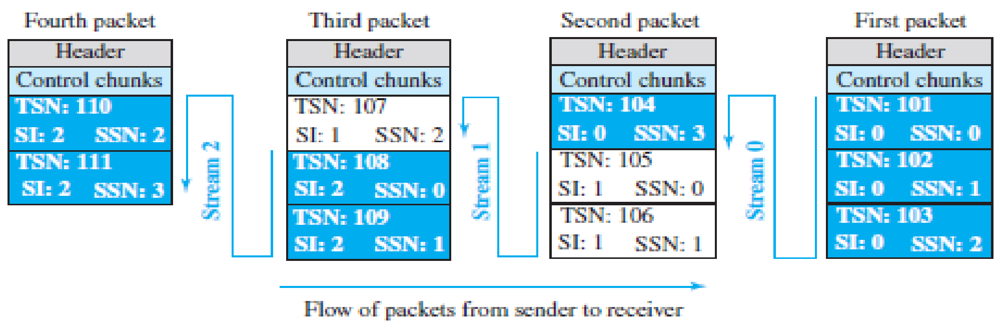


Figure 24.42 *SCTP packet format*

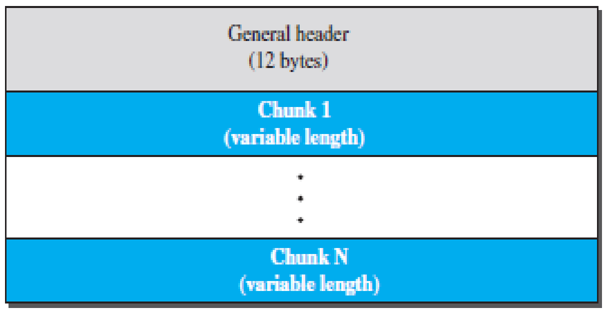
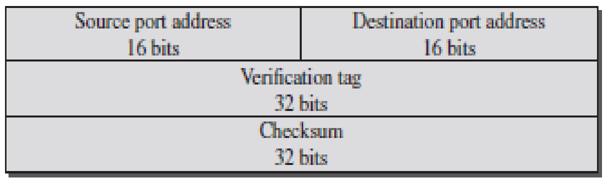


Figure 24.43 *General header*



Flow Control

Like TCP, SCTP implements flow control to avoid overwhelming the receiver.

Error Control

Like TCP, SCTP implements error control to provide reliability. TSN numbers and acknowledgement numbers are used for error control.

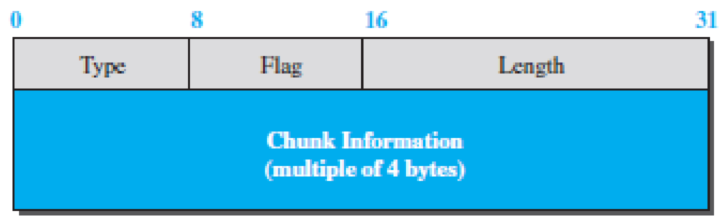
Congestion Control

Like TCP, SCTP implements congestion control to determine how many data chunks can be injected into the network.

Chunks

- Control information or user data are carried in chunks. Chunks have a common layout, as shown in below Figure.
- The first three fields are common to all chunks; the information field depends on the type of chunk. The type field can define up to 256 types of chunks.
- Only a few have been defined so far; the rest are reserved for future use. The flag field defines special flags that a particular chunk may need.
- Each bit has a different meaning depending on the type of chunk. The length field defines the total size of the chunk, in bytes, including the type, flag, and length fields.
- Since the size of the information section is dependent on the type of chunk, we need to define the chunk boundaries.
- If a chunk carries no information, the value of the length field is 4 (4 bytes). Note that the length of the padding, if any, is not included in the calculation of the length field.
- This helps the receiver find out how many useful bytes a chunk carries. If the value is not a multiple of 4, the receiver knows there is padding.

Figure 24.44 *Common layout of a chunk*



Types of Chunks:

Table 24.3 *Chunks*

Type	*Chunk*	*Description*
0	DATA	User data
1	INIT	Sets up an association
2	INIT ACK	Acknowledges INIT chunk
3	SACK	Selective acknowledgment
4	HEARTBEAT	Probes the peer for liveliness
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk
6	ABORT	Aborts an association
7	SHUTDOWN	Terminates an association
8	SHUTDOWN ACK	Acknowledges SHUTDOWN chunk
9	ERROR	Reports errors without shutting down
10	COOKIE ECHO	Third packet in association establishment
11	COOKIE ACK	Acknowledges COOKIE ECHO chunk
14	SHUTDOWN COMPLETE	Third packet in association termination
192	FORWARD TSN	For adjusting cumulating TSN

Difference between SCTP and TCP

Stream Control Transmission Protocol (SCTP) : SCTP is connection- oriented protocol in computer networks which provides full-duplex association i.e., transmitting multiple streams of data between two end points at the same time that have established connection in network.

Transmission Control Protocol (TCP) : TCP is connection oriented reliable protocol which supports guaranteed data transmission. TCP provides reliable data transmission from the connection establishment itself.

Difference between SCTP and TCP :

Parameter	SCTP	TCP
Multistreaming	SCTP supports multistreaming.	TCP doesn't supports multistreaming.
Selective ACKs	In SCTP, there are selective ACKs.	In TCP, the selective ACKs are optional.
Multihoming	Multihoming is supported by SCTP.	TCP doesn't supports Multihoming.
Data Transfer	There is more reliable data transfer in SCTP.	There is less reliable data transfer in TCP.
Security	SCTP has more secure data transfer.	TCP data transfer is less secure.
Partial Data Transfer	There is partial data transfer in SCTP.	There is no partial data transfer in TCP.
Unordered data delivery	There is unordered data delivery in SCTP.	TCP does not have unordered data delivery.