Department Name: Computer Science

Course Name: M. Tech

Semester: 2nd

Paper Name: Advanced Computer Network (MTCS-201)

Topic: Transport Layer Protocols: UDP, TCP

TRANSPORT LAYER PROTOCOLS

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.

1. USER DATAGRAM PROTOCOL (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.

- The User Datagram Protocol (UDP) is a transport layer protocol defined for use with the IP network layer protocol. It is defined by RFC 768 written by John Postel. It provides a best-effort datagram service to an End System (IP host).

- 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.

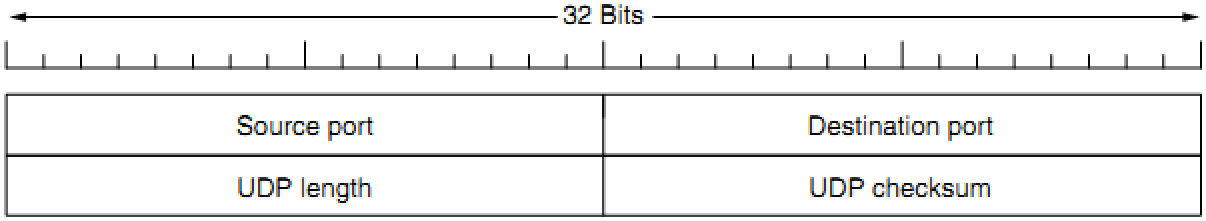


Fig : The UDP header

- The UDP header consists of four fields:- (i) Source Port (ii) Destination Port (iii) UDP length (iv) UDP checksum.
    - 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.
- The service provided by UDP is an unreliable service that provides no guarantees for delivery and no protection from duplication.
- UDP does not provide any services beyond multiplexing and demultiplexing. Datagrams may be delayed, lost, and reordered. In addition, use of the checksum field present in the UDP header is optional. Therefore, UDP is considered a “bare bones” transport layer protocol. UDP is often used for applications that tolerate packet loss but need low delay (e.g., real-time voice or video).

2. <u>TRANSMISSION CONTROL PROTOCOL (TCP)</u>

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 transmission control protocol operates in connection-oriented mode. Data transmissions between end systems require a connection setup step. Once the connection is established, TCP provides a stream abstraction that provides reliable, in-order delivery of data. To implement this type of stream data transfer, TCP uses reliability, flow control, and congestion control. TCP is widely used in the Internet, as reliable data transfers are imperative for many applications. The initial ideas were published by Cerf and Kahn.

➢ The different issues to be considered are:
  (i). The TCP Service Model
  (ii). The TCP Protocol
  (iii). The TCP Segment Header
  (iv). The Connection Management
  (v). TCP Transmission Policy
  (vi). TCP Congestion Control
  (vii). TCP Timer Management

(i) TCP Service Model

- TCP service is obtained by having both the sender and receiver create end points called **SOCKETS.**
- Each socket has a socket number(address)consisting of the IP address of the host, called a **“PORT”** (= TSAP).
- To obtain TCP service a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine.
- All TCP connections are full duplex and point to point i.e., multicasting or broadcasting is not supported.
- A TCP connection is a byte stream, not a message stream i.e., the data is delivered as chunks.

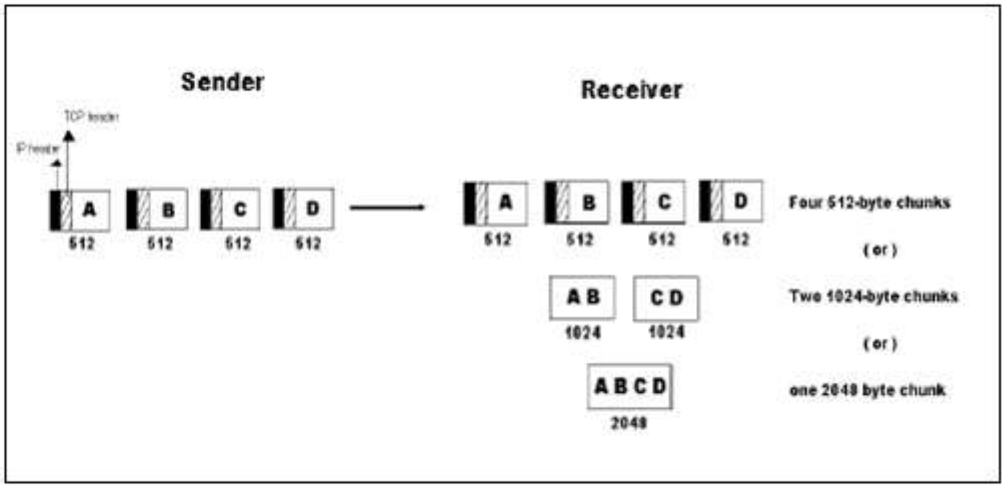
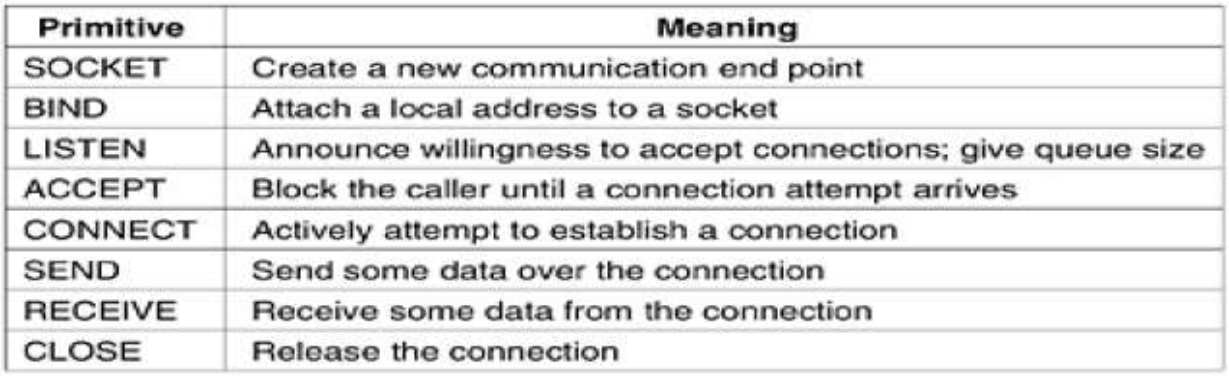


Fig: *4 * 512 bytes of data is to be transmitted.*

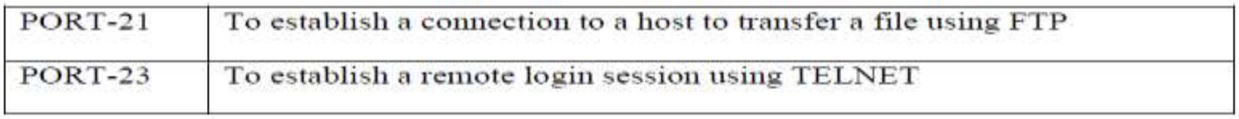
Sockets:

A socket may be used for multiple connections at the same time. In other words, 2 or more connections may terminate at same socket. Connections are identified by socket identifiers at same socket. Connections are identified by socket identifiers at both ends. Some of the sockets are listed below:



Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Ports: Port numbers below 256 are called Well- known ports and are reserved for standard services.
Eg:



PORT-21	To establish a connection to a host to transfer a file using FTP
PORT-23	To establish a remote login session using TELNET

(ii) 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 cycle through 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 an acknowledgement 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.

(iii) 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.

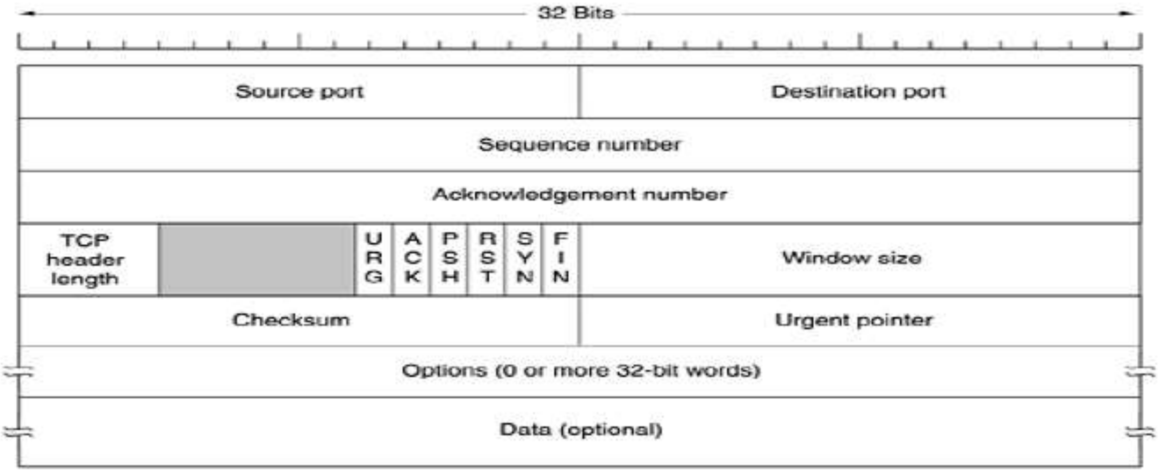


Fig: TCP Header

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

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.

(iv) TCP Connection Establishment/Management

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 segment size 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.

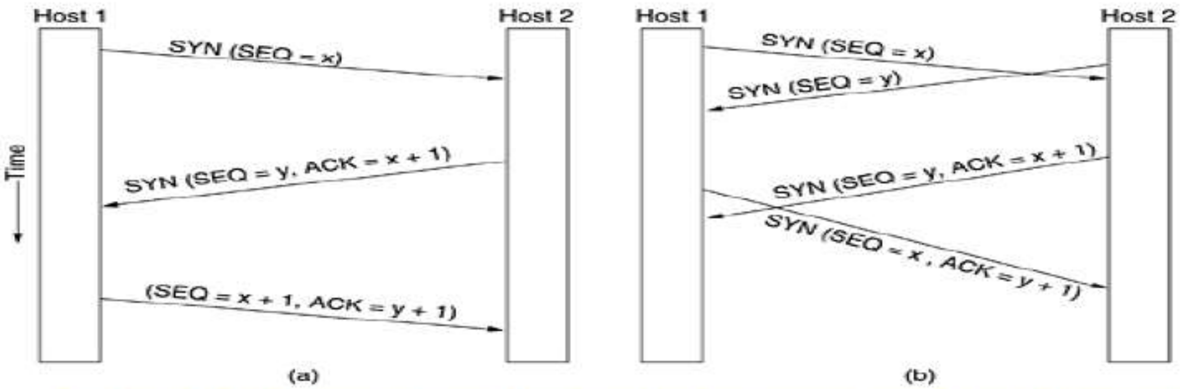

Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
Time
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x, ACK = y + 1)
(b)

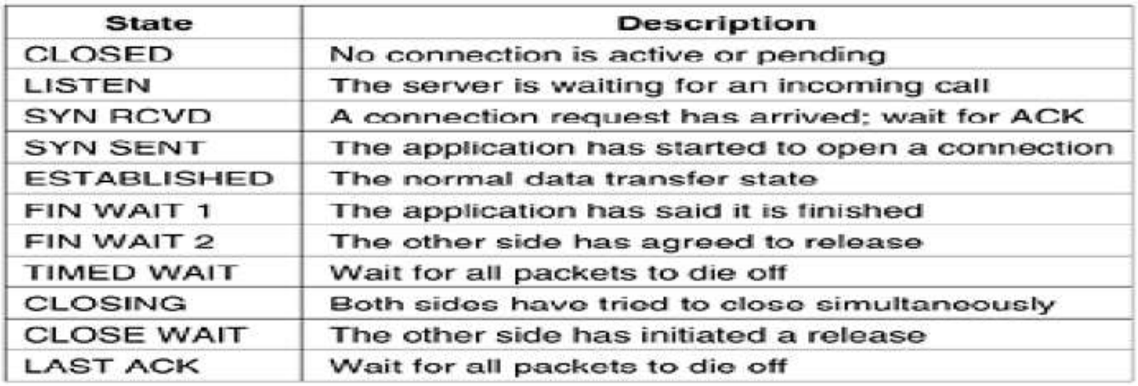
Fig : a) TCP Connection establishment in the normal case b) Call Collision

(v) 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 the second *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 Fig. below. 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

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

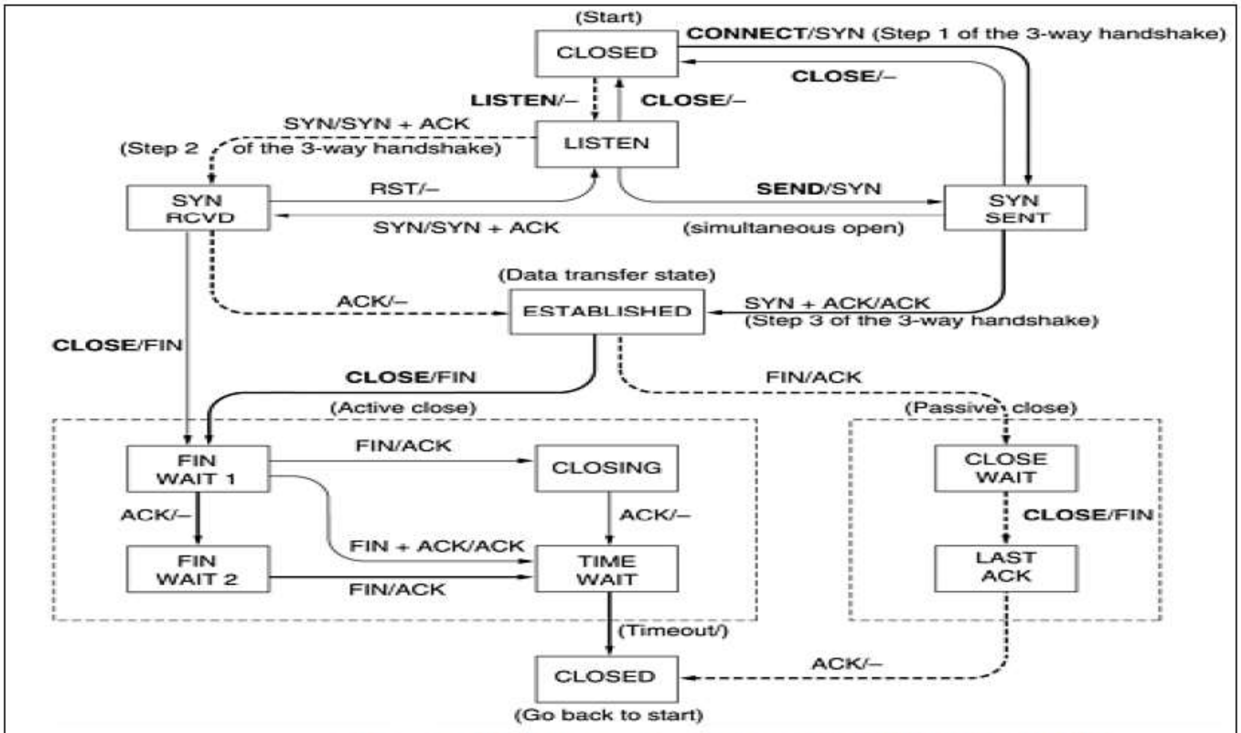


Figure - TCP connection management finite state machine.

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.

(vii). TCP Transmission Policy

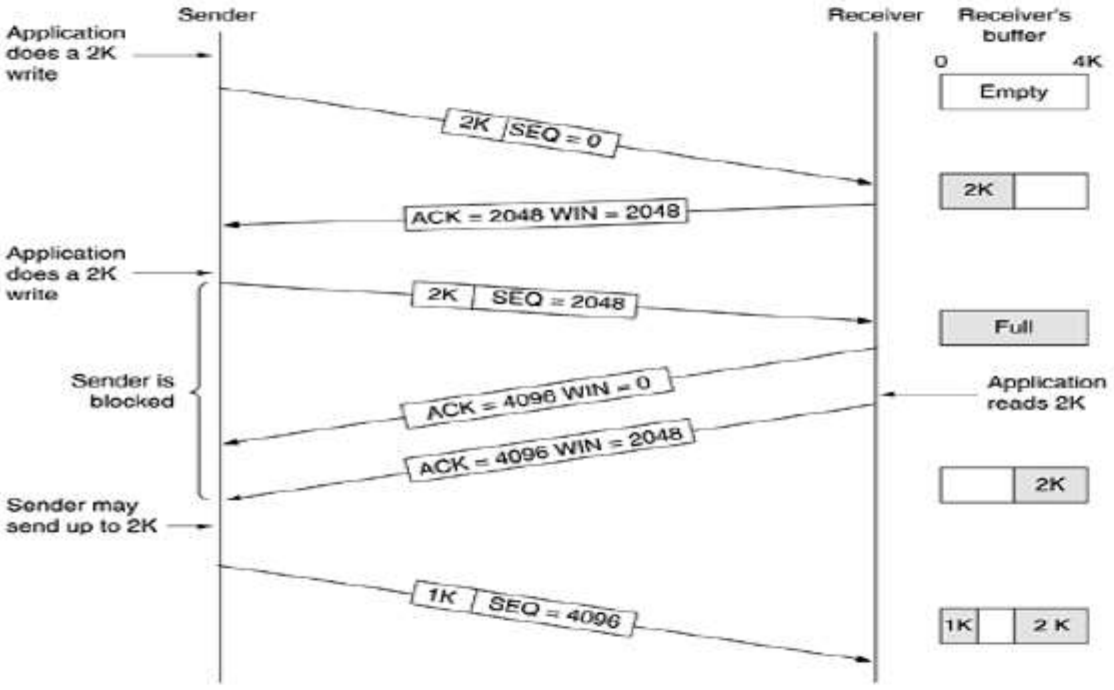


Figure - Window management in TCP.

1. 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 layer window.

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.

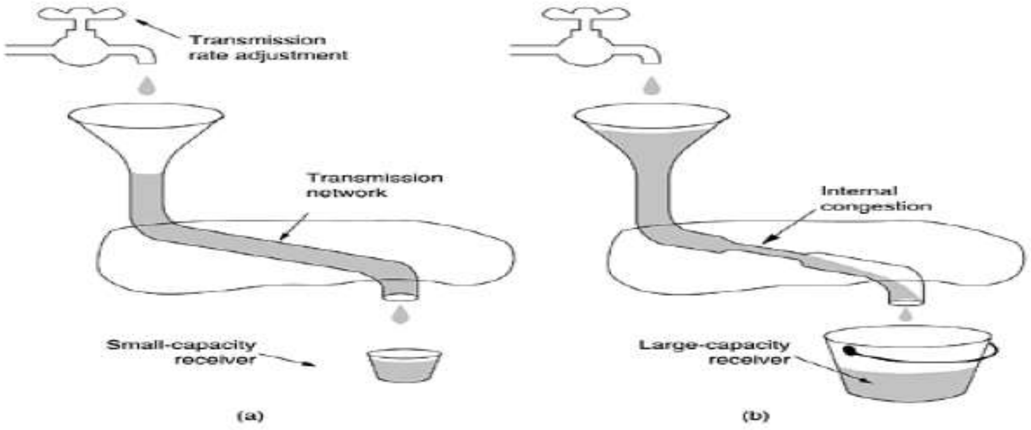


Figure (a) A fast network feeding a low-capacity receiver.
(b) A slow network feeding a high-capacity receiver.

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 will be 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 and some 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:

- RTT variance Estimation.
- Exponential RTO back-off Re-transmission Timer Management
- Karn's Algorithm
- Slow Start
- Dynamic window sizing on congestion
- Fast Retransmit Window Management
- Fast Recovery

TCP TIMER MANAGEMENT:

TCP uses 3 kinds of timers:

1. Retransmission timer
2. Persistence timer
3. Keep-Alive timer.

1. **Retransmission timer:** When a segment is sent, a timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted and the timer is started again. The algorithm that constantly adjusts the time-out interval, based on continuous measurements of n/w performance was proposed by JACOBSON and works as follows:

    - for each connection, TCP maintains a variable RTT, that is the best current estimate of the round trip time to the destination in question.

    - When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long.

    - If the acknowledgement gets back before the timer expires, TCP measures how long the measurements took say M.

    - It then updates RTT according to the formula

$$\mathbf{RTT} = \alpha\, \mathbf{RTT} + (\mathbf{1}\text{-}\alpha)\, \mathbf{M}$$

Where α = a smoothing factor that determines how much weight is given to the old value. Typically, α ***=7/8***

Retransmission timeout is calculated as

$$\mathbf{D} = \alpha\, \mathbf{D} + (\mathbf{1}\text{-}\alpha)\, |\mathbf{RTT\text{-}M}|$$

Where D = another smoothed variable, Mean RTT = expected acknowledgement value

M = observed acknowledgement value

$$\mathbf{Timeout = RTT+(4*D)}$$

2. **Persistence timer**:

It is designed to prevent the following deadlock:

- The receiver sends an acknowledgement with a window size of ‘0’ telling the sender to wait later, the receiver updates the window, but the packet with the update is lost now both the sender and receiver are waiting for each other to do something
- when the persistence timer goes off, the sender transmits a probe to the receiver the response to the probe gives the window size.
- if it is still zero, the persistence timer is set again and the cycle repeats.
- if it is non zero, data can now be sent

3. Keep-Alive timer: When a connection has been idle for a long time, this timer may go off to cause one side to check if other side is still there. If it fails to respond, the connection is terminated.

3. ANOTHER, LESS COMMONLY USED TRANSPORT LAYER PROTOCOL FOLLOWS.

Stream Control Transmission Protocol (SCTP): This transport layer protocol combines some aspects of UDP and TCP. SCTP provides reliability similar to TCP but maintains a separation between data transmissions (called “chunks”) similar to datagrams in UDP. Further, SCTP supports multiple parallel streams of chunks (for both data and control information).