TRANSPORT LAYER

TCP and UDP

The transmission Control Protocol (TCP)

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

Features

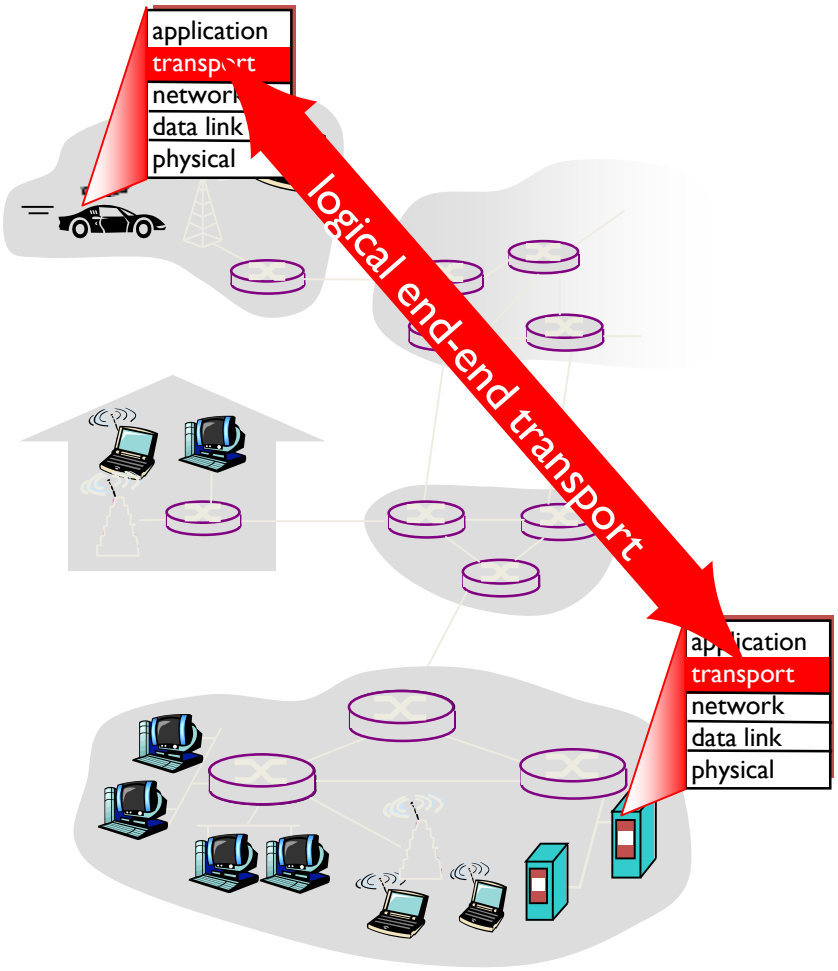
- TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.
- TCP ensures that the data reaches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

Transmission Control Protocol (TCP)

- Connection oriented
  - Explicit set-up and tear-down of TCP session
- Stream-of-bytes service
  - Sends and receives a stream of bytes, not messages
- Reliable, in-order delivery
  - Checksums to detect corrupted data
  - Acknowledgments & retransmissions for reliable delivery
  - Sequence numbers to detect losses and reorder data
- Flow control
  - Prevent overflow of the receiver's buffer space
- Congestion control
  - Adapt to network congestion for the greater good

Transport Services and Protocols

- Provide *logical communication* between app processes running on different hosts
- Transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - receive side: reassembles segments into messages, passes to app layer
- More than one transport protocol available to apps
  - Internet: TCP and UDP

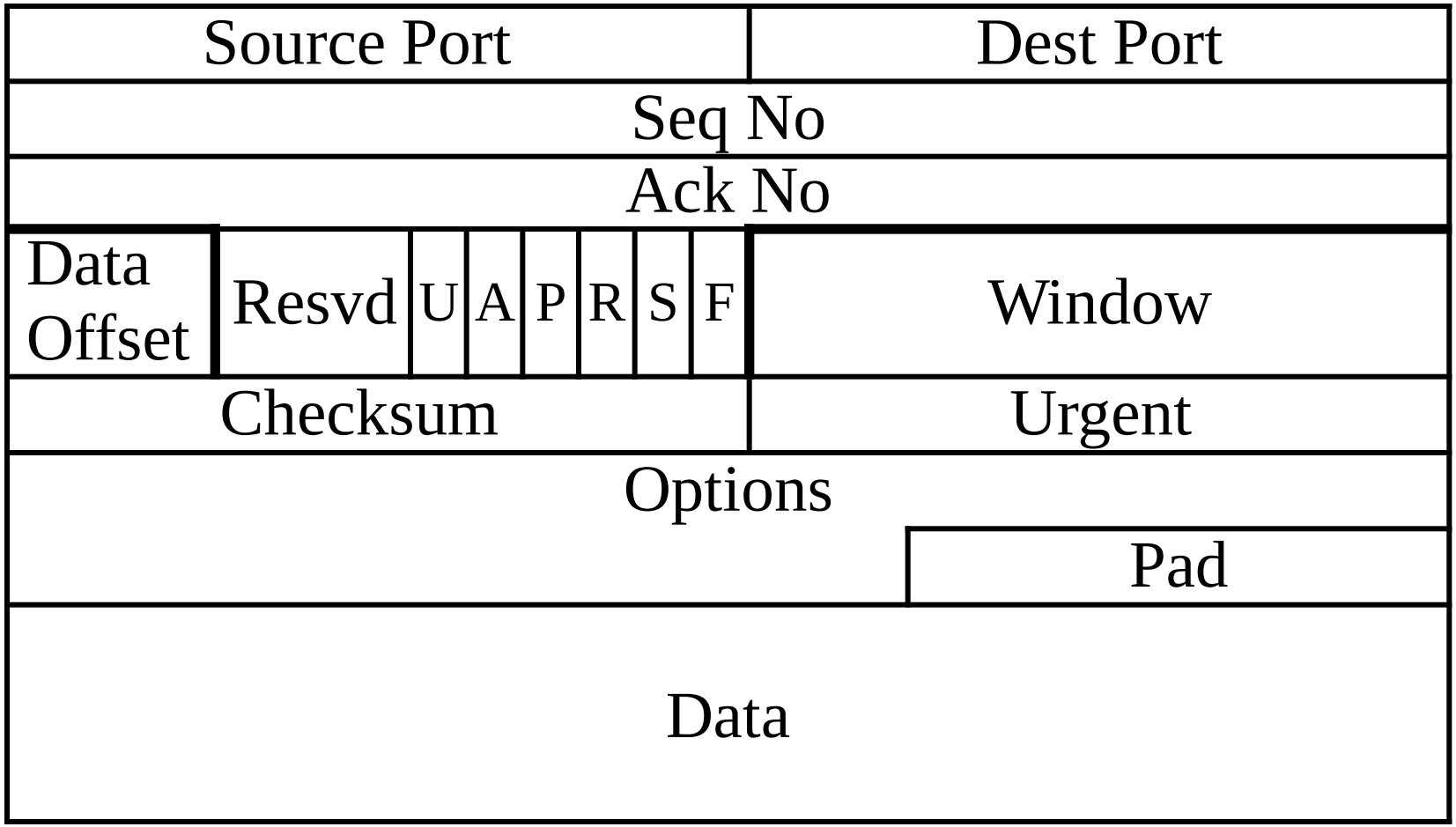


Transport vs. Network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

Header

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



- **Source Port (16-bits)** - It identifies source port of the application process on the sending device.
- **Destination Port (16-bits)** - It identifies destination port of the application process on the receiving device.
- **Sequence Number (32-bits)** - Sequence number of data bytes of a segment in a session.
- **Acknowledgement Number (32-bits)** - When ACK flag is set, this number contains the next sequence number of the data byte expected and works as acknowledgement of the previous data received.
- **Data Offset (4-bits)** - This field implies both, the size of TCP header (32-bit words) and the offset of data in current packet in the whole TCP segment.

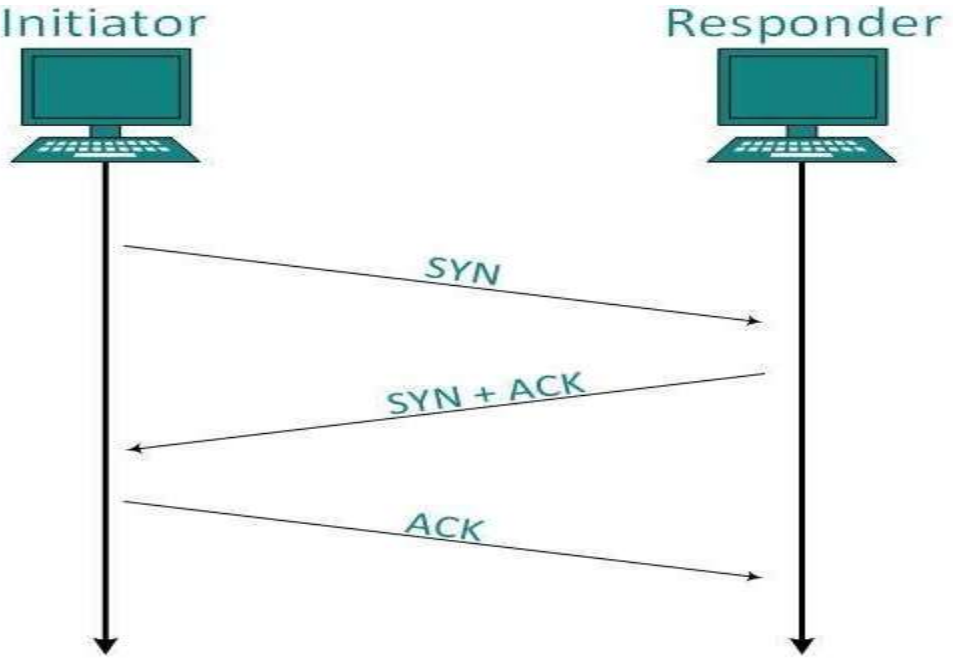
- **Reserved (3-bits)** - Reserved for future use and all are set zero by default.
- **Flags (1-bit each)**
  - **NS** - The nonce sum flag is still an experimental flag used to help protect against accidental malicious concealment of packets from the sender.
  - **CWR** - When a host receives packet with ECE bit set, it sets **Congestion Windows Reduced** to acknowledge that ECE received.
  - **ECE** -The first, ECN-Echo (**ECE**) is used to echo back the congestion indication (i.e. signal the sender to reduce the amount of information it sends). The second, Congestion Window Reduced (**CWR**), to acknowledge that the congestion-indication echoing was received.
  - **URG** The urgent flag is used to notify the receiver to process the urgent packets before processing all other packets. The receiver will be notified when all known urgent data has been received
  - **ACK** - The acknowledgment flag is used to acknowledge the successful receipt of a packet. As we can see from the diagram above, the receiver sends an ACK as well as a SYN in the second step of the three way handshake process to tell the sender that it received its initial packet..
  - **PSH** - The push flag is somewhat similar to the URG flag When set, it is a request to the receiving station to PUSH data (as soon as it comes) to the receiving application without buffering it.

- **RST** - Reset flag has the following features:
  - It is used to refuse an incoming connection.
  - It is used to reject a segment.
  - It is used to restart a connection.
- **SYN** - The synchronization **flag** is used as a first step in establishing a three way handshake between two hosts. Only the first packet from both the sender and receiver should have this **flag** set.
- **FIN** - The finished flag means there is no more data from the sender. Therefore, it is used in the last packet sent from the sender

- **Windows Size** - This field is used for flow control between two stations and indicates the amount of buffer (in bytes) the receiver has allocated for a segment, i.e. how much data is the receiver expecting.
- **Checksum** - This field contains the checksum of Header, Data and Pseudo Headers.
- **Urgent Pointer** - It points to the urgent data byte if URG flag is set to 1.
- **Options** - It facilitates additional options which are not covered by the regular header. Option field is always described in 32-bit words. If this field contains data less than 32-bit, padding is used to cover the remaining bits to reach 32-bit boundary.

Connection Management

- TCP communication works in Server/Client model. The client initiates the connection and the server either accepts or rejects it. Three-way handshaking is used for connection management.



- **Step 1 (SYN) :** In the first step, client wants to establish a connection with server, so it sends a segment with SYN(Synchronize Sequence Number) which informs server that client is likely to start communication and with what sequence number it starts segments with
- **Step 2 (SYN + ACK):** Server responds to the client request with SYN-ACK signal bits set. Acknowledgement(ACK) signifies the response of segment it received and SYN signifies with what sequence number it is likely to start the segments with
- **Step 3 (ACK) :** In the final part client acknowledges the response of server and they both establish a reliable connection with which they will start the actual data transfer
- The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.

Bandwidth Management

- TCP uses the concept of window size to accommodate the need of Bandwidth management. Window size tells the sender at the remote end, the number of data byte segments the receiver at this end can receive. TCP uses slow start phase by using window size 1 and increases the window size exponentially after each successful communication.
- For example, the client uses windows size 2 and sends 2 bytes of data. When the acknowledgement of this segment received the windows size is doubled to 4 and next sent the segment sent will be 4 data bytes long. When the acknowledgement of 4-byte data segment is received, the client sets windows size to 8 and so on.
- If an acknowledgement is missed, i.e. data lost in transit network or it received NACK, then the window size is reduced to half and starts again.

Error Control &and Flow Control

- TCP uses sequence numbers to synchronize itself with the remote host. All data segments are sent and received with sequence numbers. The Sender knows which last data segment was received by the Receiver when it gets ACK. The Receiver knows about the last segment sent by the Sender by referring to the sequence number of recently received packet.
- If the sequence number of a segment recently received does not match with the sequence number the receiver was expecting, then it is discarded and NACK is sent back. If two segments arrive with the same sequence number, the TCP timestamp value is compared to make a decision.

Congestion Control

- When large amount of data is fed to system which is not capable of handling it, congestion occurs. TCP controls congestion by means of Window mechanism. TCP sets a window size telling the other end how much data segment to send. TCP may use three algorithms for congestion control:
- Additive increase, Multiplicative Decrease
- Slow Start
- Timeout React

Timer Management

- TCP uses different types of timer to control and management various tasks:
- Keep-alive timer:
- This timer is used to check the integrity and validity of a connection.
- When keep-alive time expires, the host sends a probe to check if the connection still exists.
- Retransmission timer:
- This timer maintains stateful session of data sent.
- If the acknowledgement of sent data does not receive within the Retransmission time, the data segment is sent again.

- Persist timer:
- TCP session can be paused by either host by sending Window Size 0.
- To resume the session a host needs to send Window Size with some larger value.
- If this segment never reaches the other end, both ends may wait for each other for infinite time.
- When the Persist timer expires, the host re-sends its window size to let the other end know.
- Persist Timer helps avoid deadlocks in communication.

- Timed-Wait:
- After releasing a connection, either of the hosts waits for a Timed-Wait time to terminate the connection completely.
- This is in order to make sure that the other end has received the acknowledgement of its connection termination request.
- Timed-out can be a maximum of 240 seconds (4 minutes).

Error Control in TCP

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

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

Checksum – Every segment contains a checksum field which is used to find corrupted segment. If the segment is corrupted, then that segment is discarded by the destination TCP and is considered as lost

Acknowledgement – TCP has another mechanism called acknowledgement to affirm that the data segments have been delivered.

Retransmission – When a segment is missing, delayed to deliver to receiver, corrupted when it is checked by receiver then that segment is retransmitted again. Segments are retransmitted only during two events: when the sender receives three duplicate acknowledgements (ACK) or when a retransmission timer expires.

Retransmission after RTO : TCP always preserve one retransmission time-out (RTO) timer for all sent but not acknowledged segments. When the timer runs out of time, the earliest segment is retransmitted. Here no timer is set for acknowledgement. In TCP, RTO value is dynamic in nature and it is updated using round trip time (RTT) of segments.

RTT(round trip time) is the time duration needed for a segment to reach receiver and an acknowledgement to be received to the sender.

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

User Datagram Protocol (UDP)

- **User Datagram Protocol (UDP)** is a Transport Layer protocol. UDP is a part of Internet Protocol suite, referred as UDP/IP suite. Unlike TCP, it is **unreliable and connectionless protocol.** So, there is no need to establish connection prior to data transfer.
- Though Transmission Control Protocol (TCP) is the dominant transport layer protocol used with most of Internet services; provides assured delivery, reliability and much more but all these services cost us with additional overhead and latency. Here, UDP comes into picture. For the realtime services like computer gaming, voice or video communication, live conferences; we need UDP. Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets. There is no error checking in UDP, so it also save bandwidth. User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.

- The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.
- In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

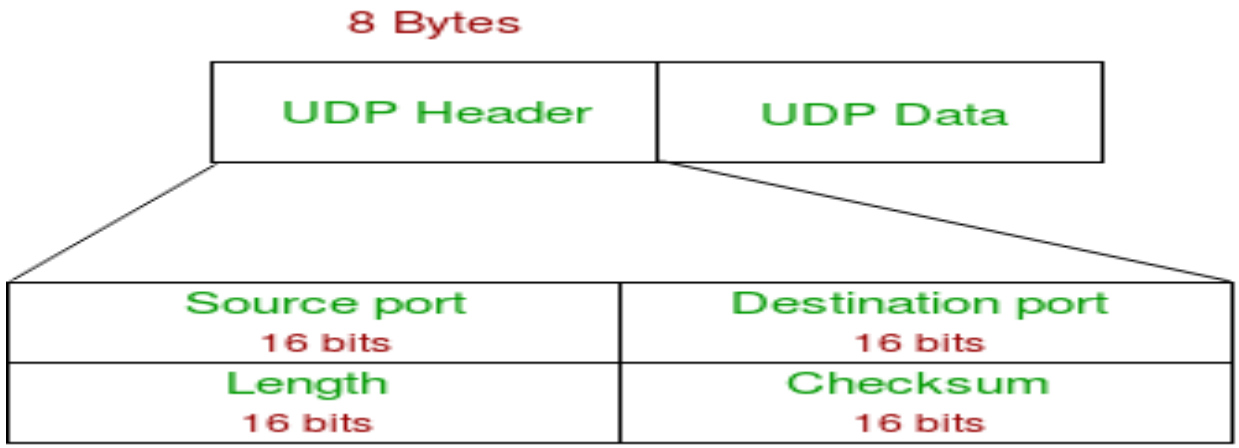
Features

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

UDP Header

- UDP header is **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes. First 8 Bytes contains all necessary header information and remaining part consist of data. UDP port number fields are each 16 bits long, therefore range for port numbers defined from 0 to 65535; port number 0 is reserved. Port numbers help to distinguish different user requests or process.
-

- **Source Port :** Source Port is 2 Byte long field used to identify port number of source.
- **Destination Port :** It is 2 Byte long field, used to identify the port of destined packet.
- **Length :** Length is the length of UDP including header and the data. It is 16-bits field.
- **Checksum :** Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, pseudo header of information from the IP header and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.



Applications of UDP:

- Used for simple request response communication when size of data is less and hence there is lesser concern about flow and error control.
- It is suitable protocol for multicasting as UDP supports packet switching.
- UDP is used for some routing update protocols like RIP(Routing Information Protocol).
- Normally used for real time applications which can not tolerate uneven delays between sections of a received message.
- Following implementations uses UDP as a transport layer protocol:
  - NTP (Network Time Protocol)
  - DNS (Domain Name Service)
  - BOOTP, DHCP.
  - NNP (Network News Protocol)
  - Quote of the day protocol
  - TFTP, RTSP, RIP, OSPF.

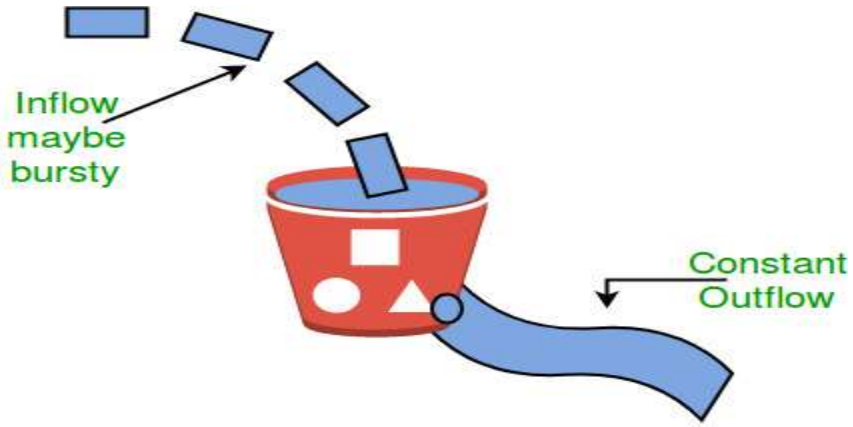
- Application layer can do some of the tasks through UDP-
  - Trace Route
  - Record Route
  - Time stamp
- UDP takes datagram from Network Layer, attach its header and send it to the user. So, it works fast.
- Actually UDP is null protocol if you remove checksum field.

What is **congestion**

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

Congestion control algorithms

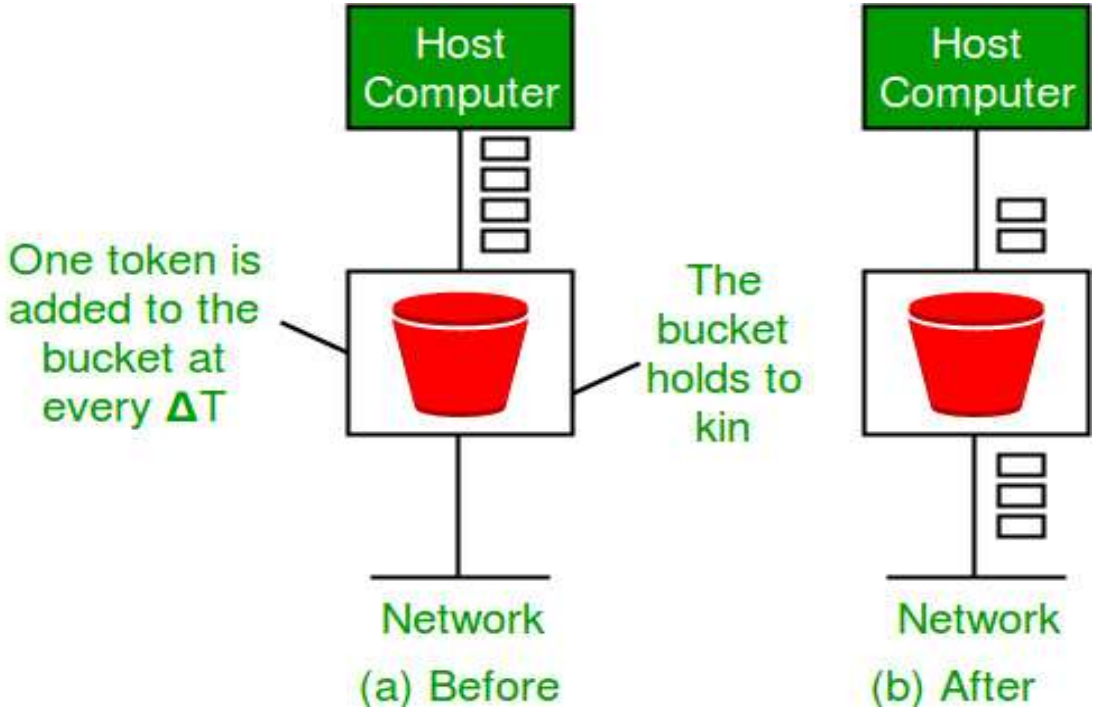
- **Leaky Bucket Algorithm**
- Let us consider an example to understand
- Imagine a bucket with a small hole in the bottom. No matter at what rate water enters the bucket, the outflow is at constant rate. When the bucket is full with water additional water entering spills over the sides and is lost.



- Similarly, each network interface contains a leaky bucket and the following **steps** are involved in leaky bucket algorithm:
- When host wants to send packet, packet is thrown into the bucket.
- The bucket leaks at a constant rate, meaning the network interface transmits packets at a constant rate.
- Bursty traffic is converted to a uniform traffic by the leaky bucket.
- In practice the bucket is a finite queue that outputs at a finite rate.

- **Token bucket Algorithm**
- **Need** of token bucket Algorithm:-
- The leaky bucket algorithm enforces output pattern at the average rate, no matter how bursty the traffic is. So in order to deal with the bursty traffic we need a flexible algorithm so that the data is not lost. One such algorithm is token bucket algorithm.
- **Steps** of this algorithm can be described as follows:
- In regular intervals tokens are thrown into the bucket. ƒ
- The bucket has a maximum capacity. ƒ
- If there is a ready packet, a token is removed from the bucket, and the packet is sent.
- If there is no token in the bucket, the packet cannot be sent.

- Let’s understand with an example,
- In figure we see a bucket holding three tokens, with five packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. In figure (B) We see that three of the five packets have gotten through, but the other two are stuck waiting for more tokens to be generated.



Ways in which token bucket is superior to leakybucket

The leaky bucket algorithm controls the rate at which the packets are introduced in the network, but it is very conservative in nature.

- Some flexibility is introduced in the token bucket algorithm. In the token bucket, algorithm tokens are generated at each tick (up to a certain limit).
- For an incoming packet to be transmitted, it must capture a token and the transmission takes place at the same rate.
- Hence some of the busty packets are transmitted at the same rate if tokens are available and thus introduces some amount of flexibility in the system.

- **Formula:** $M * s = C + \rho * s$
  where S – is time taken
  M – Maximum output rate
  ρ – Token arrival rate
  C – Capacity of the token bucket in byte

Leaky Bucket vs Token Bucket

- LB discards packets; TB does not. TB discards tokens.
- With TB, a packet can only be transmitted if there are enough tokens
- LB sends packets at an average rate. TB allows for large bursts to be sent faster by speeding up the output.
- TB allows saving up tokens (permissions) to send large bursts. LB does not allow saving.