5.1 The transport service: Services provided to the upper layers
5.2 Transport protocols: UDP, TCP
5.3 Port and Socket
5.4 Connection establishment, Connection release
5.5 Flow control & buffering
5.6 Multiplexing & de-multiplexing
5.7 Congestion control algorithm: Token Bucket and Leaky Bucket Transport Layer

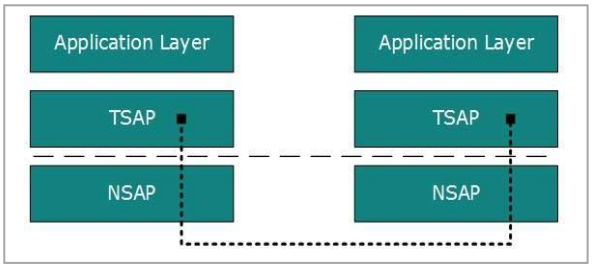
Next Layer in OSI Model is recognized as Transport Layer (Layer-4). All modules and procedures relating to transportation of data or data stream are categorized into this layer. As all other layers, this layer communicates with its peer Transport layer of the remote host. Transport layer offers peer-to-peer and end-to-end connection between two processes on remote hosts. Transport layer takes data from upper layer (i.e. Application layer) and then breaks it into smaller size segments, numbers each byte, and hands over to lower layer (Network Layer) for delivery.

Transport Layer Functions

- This Layer is the first one which breaks the information data, supplied by Application layer in to smaller units called segments. It numbers every byte in the segment and maintains their accounting.
- This layer ensures that data must be received in the same sequence in which it was sent.
- This layer provides end-to-end delivery of data between hosts which may or may not belong to the same subnet.
- All server processes aim to communicate over the network are equipped with well-known Transport Service Access Points (TSAPs) also known as port numbers.

End-to-End Communication

A Transport Services Access Point (**TSAP**) is an end-point for communication between the Transport layer (layer 4) and the Session layer in the OSI (Open Systems Interconnection) reference model to specify a destination address for a connection. Each TSAP is an address that uniquely identifies a specific instantiation of a service. TSAPs are created by concatenating the node's Network Service Access Point (**NSAP**) with a transport identifier, and sometimes a packet and/or protocol type. When using the OSI Network Layer (CONS or CLNS), the base for constructing an address for a network element is an NSAP address, similar in concept to an IP address.

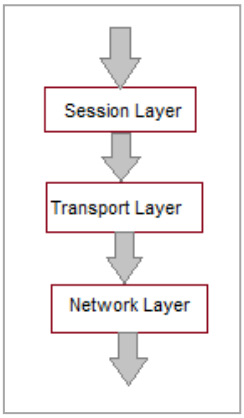


For example, *when a DHCP client wants to communicate with remote **DHCP** server, it always requests on port number **67**. When a **DNS** client wants to communicate with remote DNS server, it always requests on port number **53** (UDP).*

End to end indicates a communication happening between two applications (maybe you and *your friend using Skype*). It doesn't care what's in the middle, it just considers that the two ends are taking with one another. It generally is a Layer 4 (or higher) communication. **Point to point** is a Layer 2 link with two devices only on it. That is, two devices with an IP address have a cable going straight from a device into the other. A protocol used there is PPP, and HDLC is a legacy one.

How The Transport Layer Works

The transport layer receives information from the session layer, which keeps track of the user's identification and personal settings, and passes it to the network layer, which consists of hardware that is capable of sending the data to other devices. While the network layer is used to actually transport data from one location to another, the transport layer is used to assemble, compile, and encode data so that it is ready to be transported.



Applications

The transport layer is used for all software-related transportation of data between two or more applications or devices. *For example, VPNs, FTP sessions, and server-to-client downloads in order to transfer information*.

Advantages

The transport layer is advantageous because it can assemble data extremely fast and open a connection between two or more applications even if they are located on different devices. The transport layer is also advantageous because it provides error checking and data recovery as well as both connection-oriented and byte-oriented communication.

Steps on Transport Service

-***Connect***: executed by client to request the connection. It sends connection req. TPDU(transport protocol data unit) (packet) to server. If server is able to handle connection then it sends Connection accepted TPDU.
-***Listen***: executed by server. It means server is ready to accept connection. It blocks the process of server until connection request comes.
-***Send***: executed to send data to other end.
-***Receive***: whenever client or server is waiting for data, it executes Receive primitive. This is also blocking primitive.
-***Disconnect***: to disconnect connection. Two variants of disconnect:

- **Asymmetric**: either side issues disconnect, connection will be released.
- **Symmetric**: both the side need to separately execute disconnect.

5.1 The transport service : Services provided to upper layers

Goal of the transport layer is to efficient, reliable and cost-effective service to application layer. *To provide the service, transport layer uses service provided by network layer*. Hardware and software within the transport layer that provides service is called **transport layer entity**. Transport layer provides two types service to application layer.

✓ **Connection oriented service**: In connection oriented services a session connection is required before any data can be sent. Handshaking is done to setup the end-to-end connection. This system works only in the bidirectional environment. It doesn't work in the unidirectional environment. Due to connection establishment this type of services becomes more reliable. Due to connection establishment the services are slower than connectionless services. Example : TCP

✓ **Connectionless service**: In connectionless services session connection is not required before any data sent. It also doesn't require the connection termination. This type of service is not reliable but faster than the connection oriented services. Example : UDP

5.2 Transport Protocols :

The two main Transport layer protocols are:

- **Transmission Control Protocol (TCP):** It provides reliable communication between two hosts. *E.g. World Wide Web(HTTP), E-mail (SMTP TCP), File Transfer Protocol (FTP), Secure Shell (SSH). **Example** :***Email:** **Reason:*** suppose if some packet(words/statement) is missing we cannot understand the content. It should be reliable.
- **User Datagram Protocol (UDP):** It provides unreliable communication between two hosts. *E.g. Domain Name System (DNS), Streaming media applications such as movies, Online multiplayer games, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP). **Example : Video streaming: Reason:** Suppose if some packet(frame/sequence) is missing we can understand the content. Because video is collection of frames. For 1 second video, there should be 25 frames(image). Even though we can understand some frames are missing due to our imagination skills.*

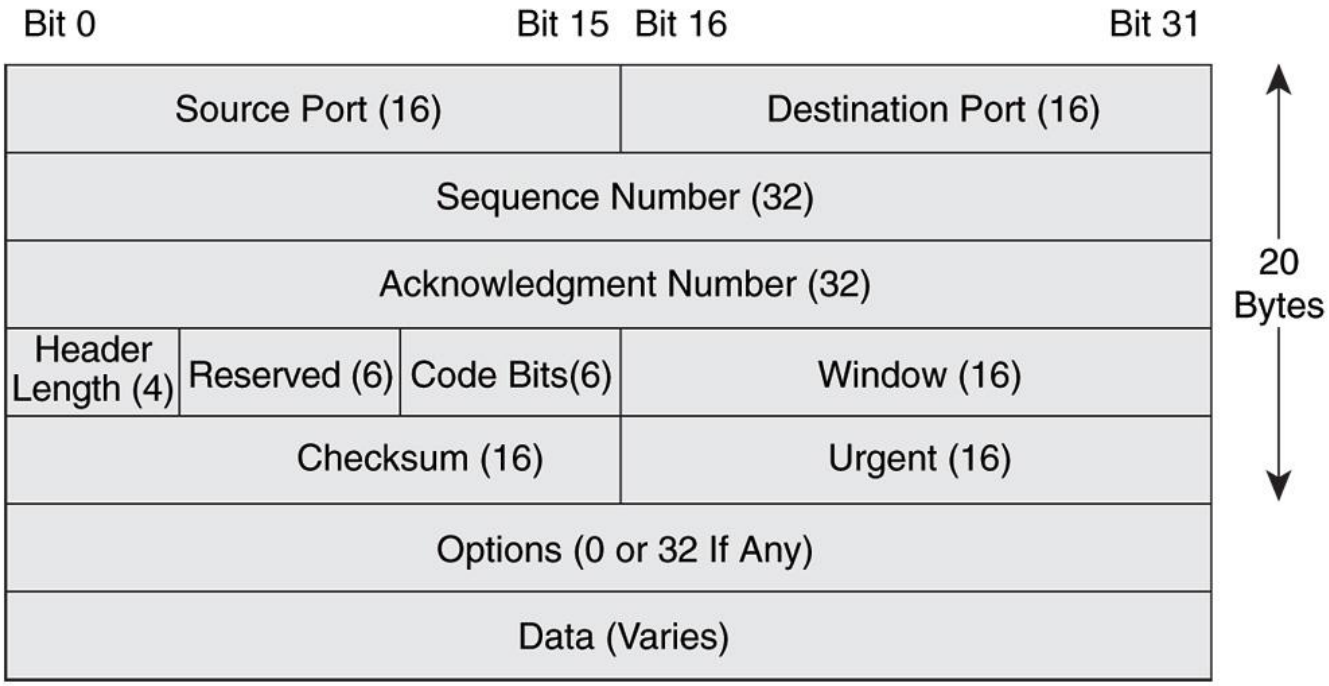
The Transmission Control Protocol (TCP) is one of the most important protocols of Internet Protocols suite. It is most widely used protocol for data transmission in communication network such as internet.

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.

TCP Header Format

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 or Header Length (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.
- **Code Bits or Flags (1-bit each)**
  - **NS** - Nonce Sum bit is used by Explicit Congestion Notification signaling process.
  - **CWR** - When a host receives packet with ECE bit set, it sets Congestion Windows Reduced to acknowledge that ECE received.
  - **ECE** -It has two meanings:
    - If SYN bit is clear to 0, then ECE means that the IP packet has its CE (congestion experience) bit set.
    - If SYN bit is set to 1, ECE means that the device is ECT capable.
  - **URG** - It indicates that Urgent Pointer field has significant data and should be processed.
  - **ACK** - It indicates that Acknowledgement field has significance. If ACK is cleared to 0, it indicates that packet does not contain any acknowledgement.
  - **PSH** - 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** - This flag is used to set up a connection between hosts.
  - **FIN** - This flag is used to release a connection and no more data is exchanged thereafter. Because packets with SYN and FIN flags have sequence numbers, they are processed in correct order.
- **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 Data 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.

Feature of TCP

i. Addressing

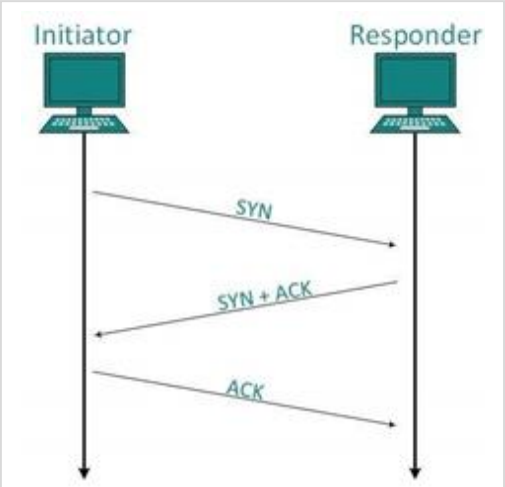
TCP communication between two remote host services is done by means of port numbers (TSAPs). Ports numbers can range from 0 – 65535 which are divided as:

- System Ports (0 – 1023)
- User Ports (1024 – 49151)
- Private/Dynamic Ports (49152 – 65535)

ii.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.

- **Establishment:** Client initiates the connection and sends the segment with a Sequence number. Server acknowledges it back with its own Sequence number and ACK of client's segment which is one more than client's Sequence number. Client after receiving ACK of its segment sends an acknowledgement of Server's response.
- **Release:** Either of server and client can send TCP segment with FIN flag set to 1. When the receiving end responds it back by ACKnowledging FIN, that direction of TCP communication is closed and connection is released.



TCP Connection

iii. 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 slow start phase starts again.

iv. Error Control and Flow Control

TCP uses port numbers to know what application process it needs to handover the data segment. Along with that, it 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.

v. Multiplexing

The technique to combine two or more data streams in one session is called Multiplexing. When a TCP client initializes a connection with Server, it always refers to a well-defined port number which indicates the application process. The client itself uses a randomly generated port number from private port number pools.

Using TCP Multiplexing, a client can communicate with a number of different application process in a single session. For example, a client requests a web page which in turn contains different types of data (HTTP, SMTP, FTP etc.) the TCP session timeout is increased and the session is kept open for longer time so that the three-way handshake overhead can be avoided.

This enables the client system to receive multiple connection over single virtual connection. These virtual connections are not good for Servers if the timeout is too long.

vi. 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

viii. 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).

viii. Crash Recovery

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

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

The User Datagram Protocol (UDP)

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.

Requirement of UDP

A question may arise, why do we need an unreliable protocol to transport the data? We deploy UDP where the acknowledgement packets share significant amount of bandwidth along with the actual data. For example, in case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amount of bandwidth wastage. The

best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost, the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features

- UDP is used when acknowledgement of data does not hold any significance.
- UDP is 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 Format

To transmit a UDP datagram, a computer completes the appropriate fields in the UDP header (PCI) and forwards the data together with the header for transmission by the IP network layer. The port numbers identify the sending process and the receiving process

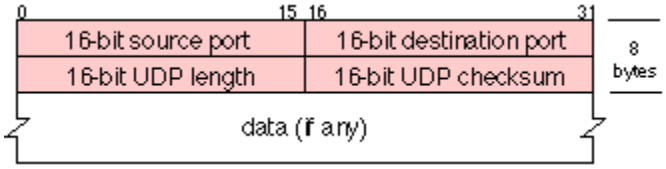


Fig. *The UDP protocol header consists of 8 bytes of Protocol Control Information (PCI)*

UDP header contains four main parameters:

- **Source Port** - This 16 bits information is used to identify the source port of the packet.
- **Destination Port** - This 16 bits information, is used identify application level service on destination machine.
- **Length** - Length field specifies the entire length of UDP packet (including header). It is 16-bits field and minimum value is 8-byte, i.e. the size of UDP header itself.
- **Checksum** - This field stores the checksum value generated by the sender before sending and verified by receiver may be used for error-checking of the header and data.

Why UDP Faster Than TCP ?

- smaller header size
- minimum error checking mechanism
- less number of fields

Difference between TCP and UDP

	TCP	UDP
Acronym for	Transmission Control Protocol	User Datagram Protocol or Universal Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	As a message makes its way across the internet from one computer to another. This is connection based.	UDP is also a protocol used in message transport or transfer. This is not connection based which means that one program can send a load of packets to another and that would be the end of the relationship.
Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Use by other protocols	HTTP, HTTPs, FTP, SMTP, Telnet	DNS, DHCP, TFTP, SNMP, RIP, VOIP.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer.
Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because error recovery is not attempted. It is a "best effort" protocol.
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.

Common Header Fields	Source port, Destination port, Check Sum	Source port, Destination port, Check Sum
Streaming of data	Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.	Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.
Weight	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.
Data Flow Control	TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP does not have an option for flow control
Error Checking	TCP does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.	UDP does error checking but simply discards erroneous packets. Error recovery is not attempted.
Fields	1. Sequence Number, 2. AcK number, 3. Data offset, 4. Reserved, 5. Control bit, 6. Window, 7. Urgent Pointer 8. Options, 9. Padding, 10. Check Sum, 11. Source port, 12. Destination port	1. Length, 2. Source port, 3. Destination port, 4. Check Sum
Acknowledgement	Acknowledgement segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake (connectionless protocol)

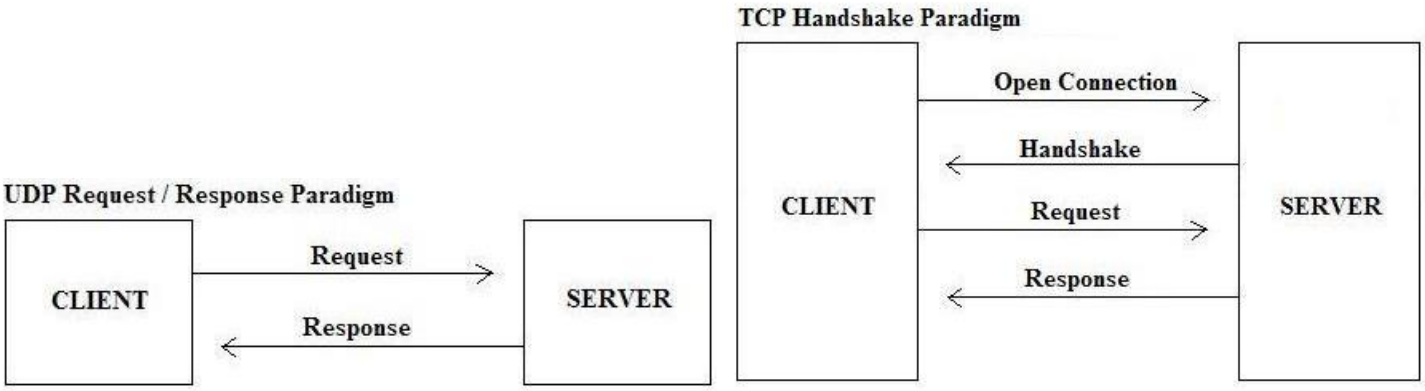


Figure 4. UDP versus TCP Request/Response Models

5.3 Port and Socket

PORT

- One of the circuit connection points on a front end processor or local intelligent controller
- The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer
- At the transport layer, an address is needed to choose among multiple processes running on the destination host called **Port Number**
  - Destination Port Number for delivery
  - Source Port Number for reply
- **Port** is represented by a positive (16-bit) integer value between 0 and 65,535
- Some **ports** have been reserved to support common / well known services
- Ports run from **1..65535**. **1..1024** require root privileges to use and ports **1..255** are reserved for common processes like: 80: HTTP, 110: POP, 25: SMTP, 22: SSH

SOCKET

If the IP address is like an office building main phone number, a socket is like the extension numbers for offices. So the combination of IP and Port called the socket, uniquely identify an "office" (server process). You will see unique identifiers like 192.168.2.100:80 where 80 is the port. Just like in an office, it is possible no process is listening at a port. That is, there is no server waiting for requests at that port.

- A Socket is one endpoint of a two-way communication link between two processes running on the network
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent
- TCP connection can be uniquely identified by its two endpoints, multiple connections are possible between host and the server
- Sockets provide an interface for programming networks at the transport layer
- Process to Process delivery of data needs two identifiers, IP Address and Port Number at each endpoint
- **Socket Address** → combination of **IP address** and a **Port number**
- Transport Layer Protocol needs a pair of Socket addresses
  - **Client Socket Address** : Uniquely defines the Client Process
  - **Server Socket Address** : Uniquely defines Server Process
- Both Socket Addresses contain IP Header and Transport Layer Protocol Header
- IP Header contains IP Addresses, TCP & UDP Header contains the Port Numbers

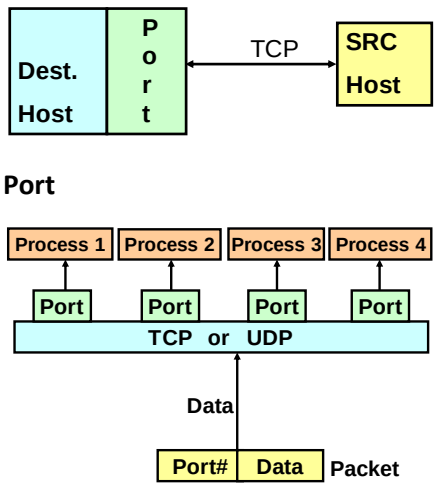


Fig. Port

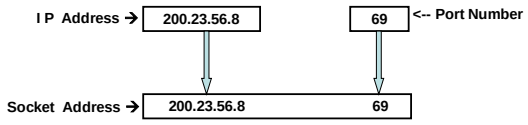


Fig. Socket Example

Types of Socket

I. **Stream socket** :(connection- oriented socket) :It provides reliable, connected networking service Error free; no out- of- order packets (uses TCP) applications: telnet/ ssh, http, …
server : socket(), bind(), **listen(), accept(),** receive(), send()
client : socket(), **connect(),** send(), receive()

II. **Datagram socket** :(connectionless socket) : It provides unreliable, best- effort networking service Packets may be lost; may arrive out of order (uses UDP) applications: streaming audio/ video (real player),
server : socket(), bind(), receive-from(), send-to()
client : socket(), send-to(), receive-from()

bind( ) is used particularly by server programs, and **connect**( ) by client programs. The others are used by both.

- bind( ) applies a "name" to an existing socket, so that it can be referred to. The essence of this "name" or identifier, is an IP-address-and-port-number pair.
- connect( ) searches out an already named (or "already bind'ed") program out on the network, by name. So calls to bind( ) supply, as parameters, information that tells which socket and what name. And calls to connect( ) supply a local socket, and the name of a remote socket to search for, locate, and connect to.

Once a server program has created a socket and named it with **bind**( ) giving it an IP address and port number, should any program anywhere on the network give that same name to the **connect**( ) function, that program will find our server program and they will link up. The server program uses the **accept**( ) function to react to the client's connect( ). So the accept( ) must be called before the connect( ) is issued. Once all this connecting and accepting is done, both sides can freely use **read/write or receivev/send** to get stuff shipped to each other

5.4 Connection Establishment and Connection Release

A TCP uses 3-way handshaking mechanism to establish the connection between the nodes. The basic steps followed by TCP to establish the connection are as follows :

Step 1: Client end system sends **SYN**chronize packet to server

Step 2: Server end system receives SYN, replies with SYNchronize-ACKnowledgement
- Allocates buffers
- ACKs received SYN

Step 3: Client receives SYN-ACK and sends ACKnowledge to server and Server receive ACK.

The TCP socket connection is now ESTABLISHED.
- client starts the "real work"

SYNchronize and **ACK**nowledge messages are indicated by a either the SYN bit, or the ACK bit inside the TCP header, and the **SYN-ACK** message has both the SYN and the ACK bits turned on (set to 1) in the TCP header.

TCP knows whether the network TCP socket connection is opening, synchronizing, established by using the **SYN**chronize and **ACK**nowledge messages when establishing a network TCP socket connection.

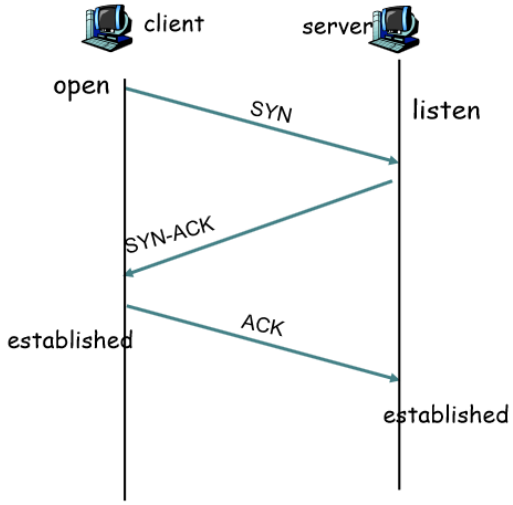


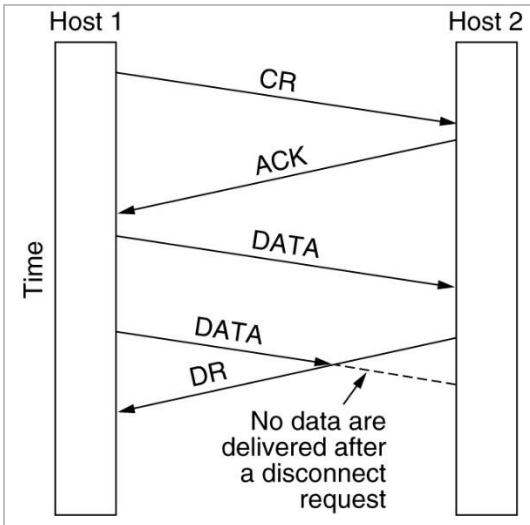
Fig. Connection establishment

Connection Release

Connection at transport can be released in two ways:-

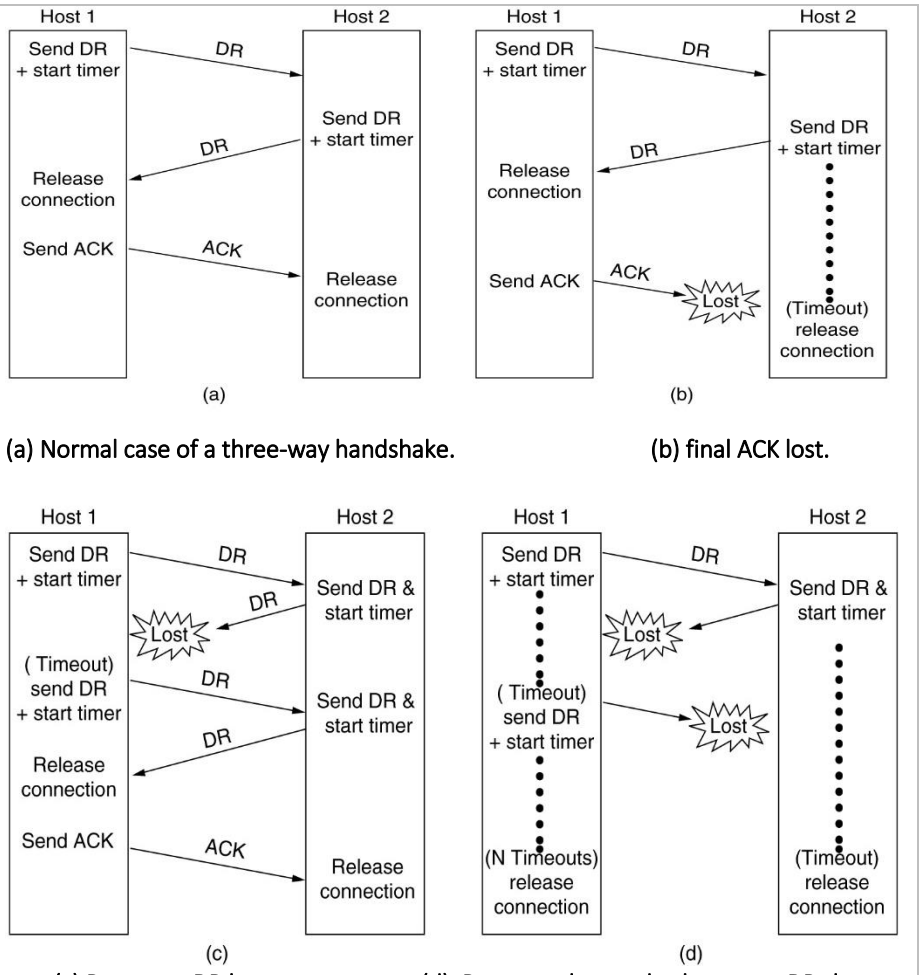
1. **asymmetric**: if one of host terminates connection, then in both the direction, data communication will be terminated.
2. **symmetric**: if one of the host disconnects connection, then it cannot send the data but it can receive it.

- *host A opens the connection with an ISN*
- *host B accepts the connect request by sending a TCP segment which*
  - *acknowledges host A's request (ACK flag on)*
  - *sets acknowledgement number to ISN+1*
  - *makes its own connection request (SYN flag on) with an ISN*
- *host A acknowledges this request*
- *note that the SYN flag "consumes" one byte of sequence space so that it can be acknowledged unambiguously*



Asymmetric Release

Unexpected disconnection with loss of data.



(a) Normal case of a three-way handshake. (b) final ACK lost.

(c) Response DR lost (d) Response lost and subsequent DRs lost.

Four protocol scenarios for releasing a connection.

Steps for symmetric Release:

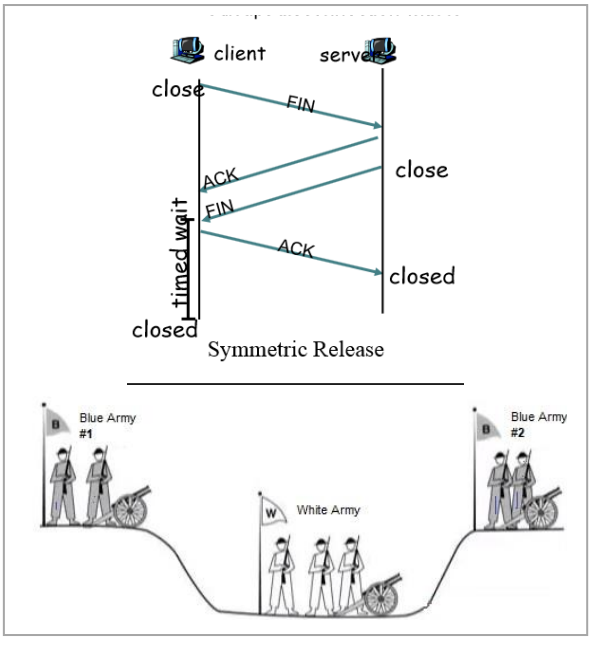
• **Step 1:** Client end system sends **TCP FIN**ish message or control segment to server which serves as connection termination.

• **Step 2:** Server receives **FIN**, replies with **ACK**. Closes connection, sends **FIN**.

• **Step 3:** Client receives **FIN**, replies with **ACK**.

Enters "timed wait" - will respond with **ACK** to received **FIN**s

• **Step 4:** server, receives **ACK**. Connection closed.



Connection Release(Problem): The two-army problem

• For blue army to win, both troops(1&2)must attack white army at same time

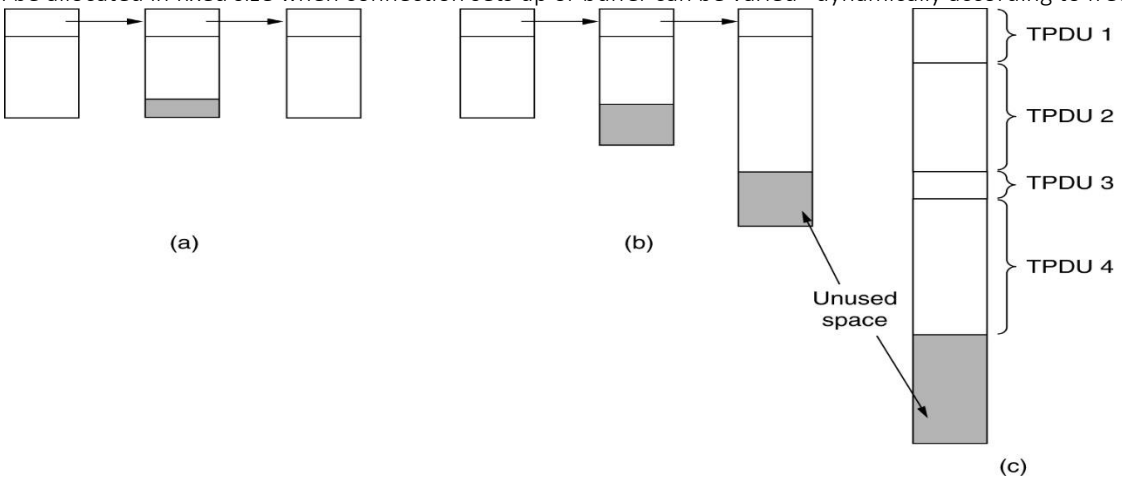
• For sync., #1 sends msg. to #2 and waits for ACK. Even though #1 receives ACK, attack is not possible as #2 doesn't know whether #1 has received its ACK or not

• Even, 3-way handshaking (#1 again sending ACK) will not guarantee successful attack as now #1 is not sure if #2 has received its ACK

• It can be proved that no protocol exists that works, but little risk can be taken while releasing the connection.

5.5 Flow Control and Buffering

- Transport layer manages end to end to flow. If the receiver is not able to manage with the flow of data, then data flow should be control from sender side, that part is done on Transport layer.
- Data link layer is also doing flow control, but it controls flow of data between adjacent nodes in path from source to destination.
- Reasons of packet loss at receiver is slow processing speed or insufficient buffer to store the data.
- Buffer are allocated at sender and receiver side. If the network service is reliable, so every send TPDU sent will be delivered to receiver and will be buffered and processes at receiver, so no need to keep buffer at sender.
- But if network service is unreliable and receiver may not able to handle every incoming TPDU then sender should also keep a buffer, where copy of TPDU resides until it's ACK comes.
- Buffers can be allocated in fixed size when connection sets up or buffer can be varied dynamically according to free memory.



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

Dynamic Buffer Allocation: as connection are opened and closed, memory available changes, so sender and receiver dynamically adjust buffer allocations.

In dynamic buffer allocation, initially sender will request certain number of buffers based on perceive need. receiver will grant as many buffers as it can.

Type of traffic carried by a connection also influences buffering strategy.

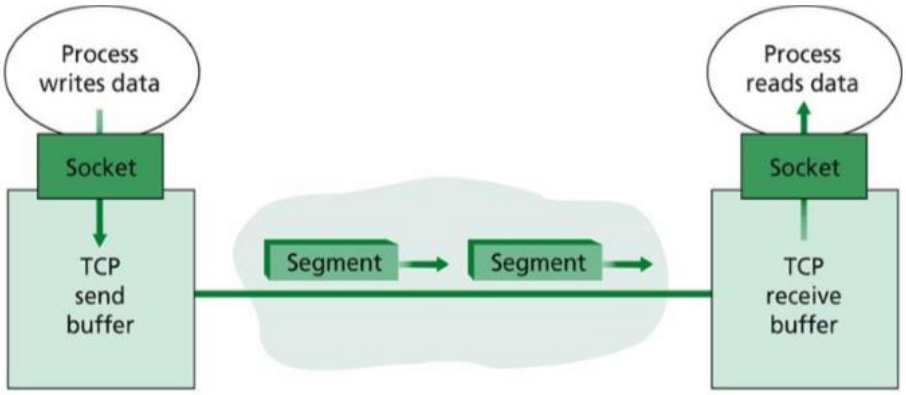


Figure : TCP sends and receives buffer

5.6 Multiplexing and Demultiplexing

In computer network, **multiplexing** is a technique by which multiple analog or digital signals are combined into one signal over a shared medium for transmission.

Similarly **De-multiplexing** is a technique by which a signal received from shared medium is identified and transmitted to the particular medium or devices.

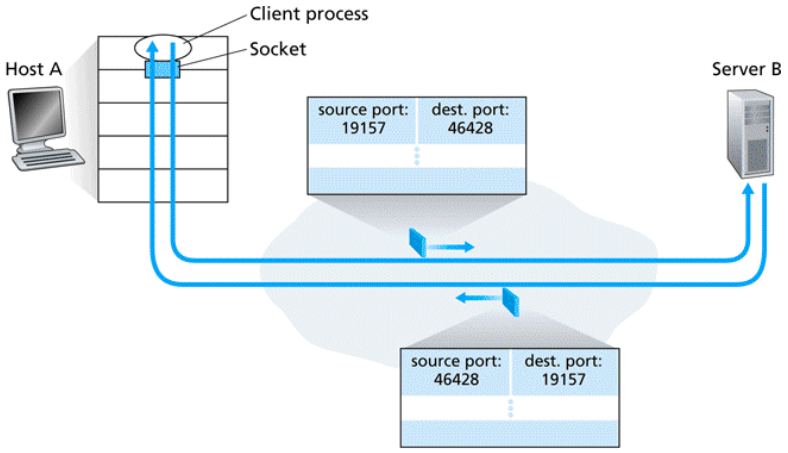
Generally, multiplexing means sending the different channel data through a common channel by switching the channel using the different technique. Multiplexing here means multiplexing multiple transport connections on a single network layer connection, which is generally required if the available bandwidth is more than or equal to the integration of individual requirement of each connection, thus making an effective utilization of the available bandwidth.

UDP or Connectionless Multiplexing and Demultiplexing

- Assume, a process on Host A (Sender), with port number **19157**, wants to send data to a process with port **46428** on Host B (Server)
- Host A creates a segment containing source port, destination port, and data passes it to the network layer in Host A
- Host B examines destination port number and delivers segment to server socket identified by port **46428**
  - note: a UDP socket is fully identified by a two-tuple consisting of

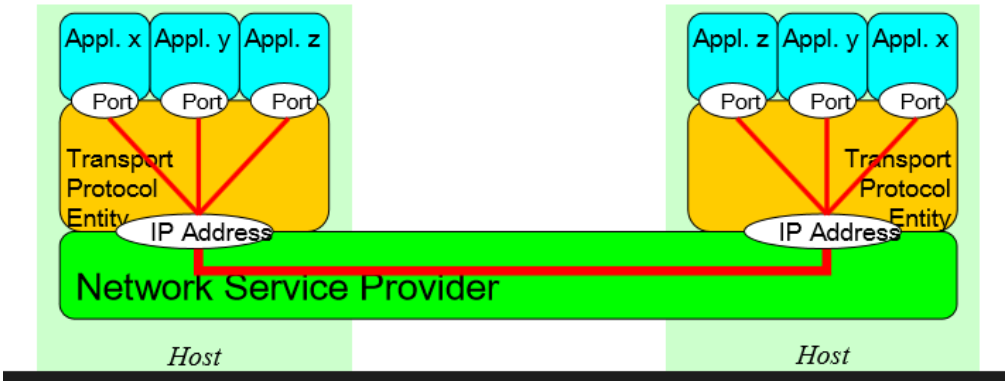
- a destination IP address
- a destination port number

- source port number from Host A is used at Host B as "return address":



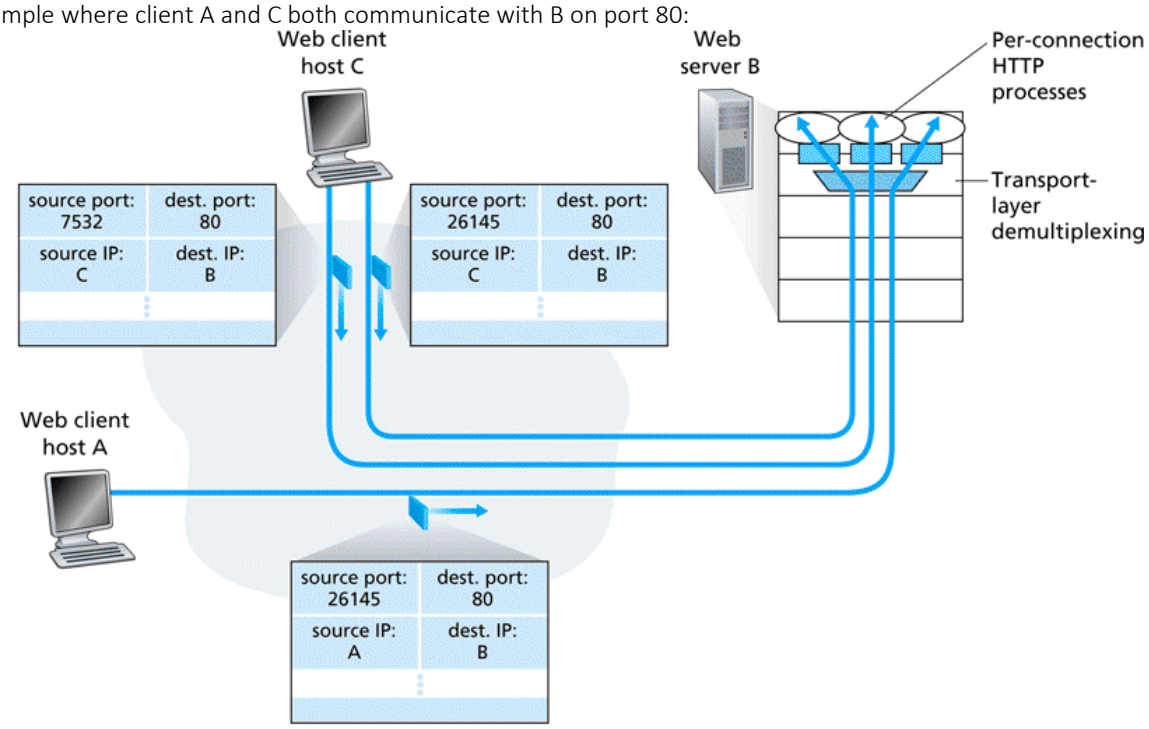
TCP or Connection-Oriented Multiplexing and Demultiplexing

- Each TCP connection has exactly two end-points
- this means that two arriving TCP segments with different source IP addresses or source port numbers will be directed to two *different* sockets, *even if they have the same destination port number*
- so, a TCP socket is identified by a four-tuple: (Source IP address and source port #, Destination IP address and destination port #)
- recall UDP uses only (destination IP address, destination port #)



Multiplexing and Demultiplexing Example

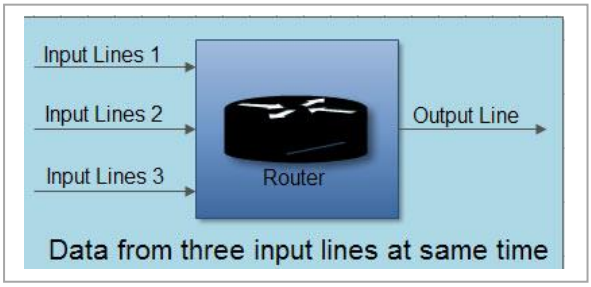
- an example where client A and C both communicate with B on port 80:



5.7 Congestion Control Algorithm: Token Bucket and Leaky Bucket Transport Layer

Congestion is an important issue that can arise in packet switched network. Congestion is a situation in Communication Networks in which too many packets are present in a part of the subnet, performance degrades. Congestion in a network may occur when the load on the network *(i.e.* the number of packets sent to the network) is greater than the capacity of the network *(i.e.* the number of packets a network can handle.)

Causing of Congestion:



1.The input traffic rate exceeds the capacity of the output lines. If suddenly, a stream of packet start arriving on three or four input lines and all need the same output line. In this case, a queue will be built up. If there is insufficient memory to hold all the packets, the packet will be lost. Increasing the memory to unlimited size does not solve the problem. This is because, by the time packets reach front of the queue, they have already timed out (as they waited the queue). When timer goes off source transmits duplicate packet that are also added to the queue. Thus, same packets are added again and again, increasing the load all the way to the destination.

2.The **routers are too slow to** perform bookkeeping tasks (queuing buffers, updating tables, etc.).

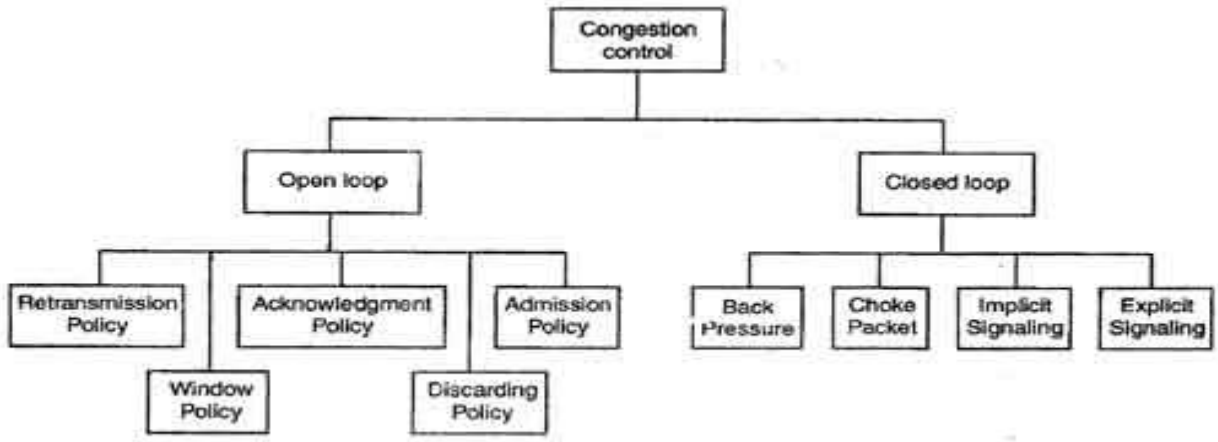
3.The **routers' buffer is too limited.**

4.Congestion in a subnet can occur **if the processors are slow.** Slow speed CPU at routers will perform the routine tasks such as queuing buffers, updating table etc. slowly. As a result of this, queues are built up even though there is excess line capacity.

5.Congestion is also caused by **slow links.** This problem will be solved when high speed links are used. But it is not always the case. Sometimes increase in link bandwidth can further deteriorate the congestion problem as higher speed links may make the network more unbalanced.

How to correct the Congestion Problem:

- Congestion Control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.
- Congestion control mechanisms are divided into two categories, (i) prevents the congestion from happening and (ii) removes congestion after it has taken place.



Types of Congestion Control Methods

These two categories are:

1. Open loop: policies are used to prevent the congestion before it happens.
2. Closed loop: used to remove the congestion after it happens.

Open Loop Congestion Control

- In this method, policies are used to prevent the congestion before it happens.
- Congestion control is handled either by the source or by the destination.
- The various methods used for open loop congestion control are:

1. Retransmission Policy

- The sender retransmits a packet, if it feels that the packet it has sent is lost or corrupted.
- However, retransmission in general may increase the congestion in the network. But we need to implement good retransmission policy to prevent congestion. The retransmission policy and the retransmission timers need to be designed to optimize efficiency and at the same time prevent the congestion.

2. Window Policy

- To implement window policy, selective reject window method is used for congestion control.

• Selective Reject method is preferred over Go-back-n window as in Go-back-n method, when timer for a packet times out, several packets are resent, although some may have arrived safely at the receiver. Thus, this duplication may make congestion worse.
• Selective reject method sends only the specific lost or damaged packets.

3. Acknowledgement Policy

• The acknowledgement policy imposed by the receiver may also affect congestion.
• If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion.
• Acknowledgments also add to the traffic load on the network. Thus, by sending fewer acknowledgements can reduce load on the network. To implement it, several approaches can be used:
  - A receiver may send an acknowledgement only if it has a packet to be sent.
  - A receiver may send an acknowledgement when a timer expires.
  - A receiver may also decide to acknowledge only *N* packets at a time.

4. Discarding Policy

• A router may discard less sensitive packets when congestion is likely to happen.
• Such a discarding policy may prevent congestion and at the same time may not harm the integrity of the transmission.

5. Admission Policy

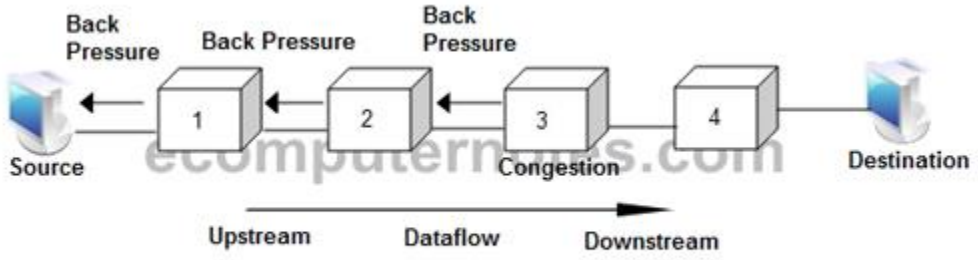
• An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual circuit networks.
• Switches in a flow first check the resource requirement of a flow before admitting it to the network.
• A router can deny establishing a virtual circuit connection if there is congestion in the "network or if there is a possibility of future congestion.

Closed Loop Congestion Control

• Closed loop congestion control mechanisms try to remove the congestion after it happens.
• The various methods used for closed loop congestion control are:

1. Backpressure

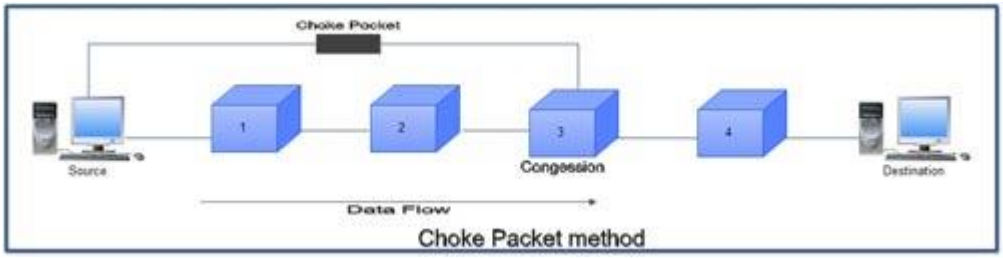
• Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow.
• The backpressure technique can be applied only to virtual circuit networks. In such virtual circuit each node knows the upstream node from which a data flow is coming.
• In this method of congestion control, the congested node stops receiving data from the immediate upstream node or nodes.
• As shown in fig node 3 is congested and it stops receiving packets and informs its upstream node 2 to slow down. Node 2 in turns may be congested and informs node 1 to slow down. Now node 1 may create congestion and informs the source node to slow down. In this way the congestion is alleviated. Thus, the pressure on node 3 is moved backward to the source to remove the congestion.



Backpressure Method

2. Choke Packet

• In this method of congestion control, congested router or node sends a special type of packet called choke packet to the source to inform it about the congestion.
• Here, congested node does not inform its upstream node about the congestion as in backpressure method.
• In choke packet method, congested node sends a warning directly to the source station *i.e.* the intermediate nodes through which the packet has traveled are not warned.



Choke Packet method

3. Implicit Signaling

• In implicit signaling, there is no communication between the congested node or nodes and the source.

• The source guesses that there is congestion somewhere in the network when it does not receive any acknowledgment. Therefore, the delay in receiving an acknowledgment is interpreted as congestion in the network.
• On sensing this congestion, the source slows down.
• This type of congestion control policy is used by TCP.

4. Explicit Signaling

• In this method, the congested nodes explicitly send a signal to the source or destination to inform about the congestion.
• Explicit signaling is different from the choke packet method. In choke packed method, a separate packet is used for this purpose whereas in explicit signaling method, the signal is included in the packets that carry data.
• Explicit signaling can occur in either the forward direction or the backward direction.
• In backward signaling, a bit is set in a packet moving in the direction opposite to the congestion. This bit warns the source about the congestion and informs the source to slow down.
• In forward signaling, a bit is set in a packet moving in the direction of congestion. This bit warns the destination about the congestion. The receiver in this case uses policies such as slowing down the acknowledgements to remove the congestion.

Congestion control algorithms: Leaky Bucket and Token Bucket

- packet loss typically results from buffer overflow in routers as the network becomes *congested*
- congestion results from too many senders trying to send data at too high a rate
- packet retransmission treats a symptom of congestion, but not the cause
- to treat the cause, senders must be "regulated" (reduce their rate)
- TCP implements a congestion control algorithm based on perceived congestion by the sender:
    - if it perceives little congestion, it increases its send rate
    - if it perceives there is congestion, it reduces its send rate

(i) Leaky Bucket Algorithm

✓ It is a traffic shaping mechanism that controls the amount and the rate of the traffic sent to the network.
✓ A leaky bucket algorithm shapes bursty traffic into fixed rate traffic by averaging the data rate.
✓ The leaky bucket enforces a constant output rate (average rate) regardless of the burs-ness of the input. Does nothing when input is idle.
✓ The Leaky Bucket Algorithm used to control rate in a network.
✓ It is implemented as a single-server queue with constant service -me.
✓ If the bucket (buffer) overflows then packets are discarded.

Analogy

- Imagine a bucket with a small hole at the bottom.
- The rate at which the water is poured into the bucket is not fixed and can vary but it leaks from the bucket at a constant rate. Thus (as long as water is present in bucket), the rate at which the water leaks does not depend on the rate at which the water is input to the bucket.
- Also, when the bucket is full, any additional water that enters into the bucket spills over the sides and is lost.

Example

- The same concept can be applied to packets in the network. Consider that data is coming from the source at variable speeds. Suppose that a source sends data at 12 Mbps for 4 seconds. Then there is no data for 3 seconds. The source again transmits data at a rate of 10 Mbps for 2 seconds. Thus, in a time span of 9(4+3+2) seconds, 68(12x4+10x2) Mb data has been transmitted.
- If a leaky bucket algorithm is used, the data flow will be 8 Mbps for 9 seconds. **Thus, constant flow is maintained.**

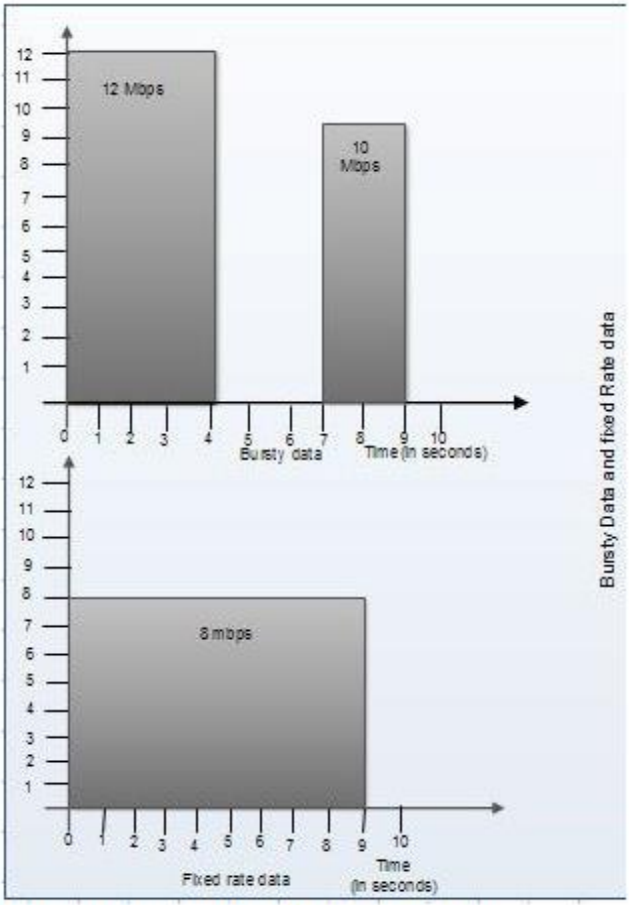


Fig. Constant Flow by Leaky Bucket Algorithm

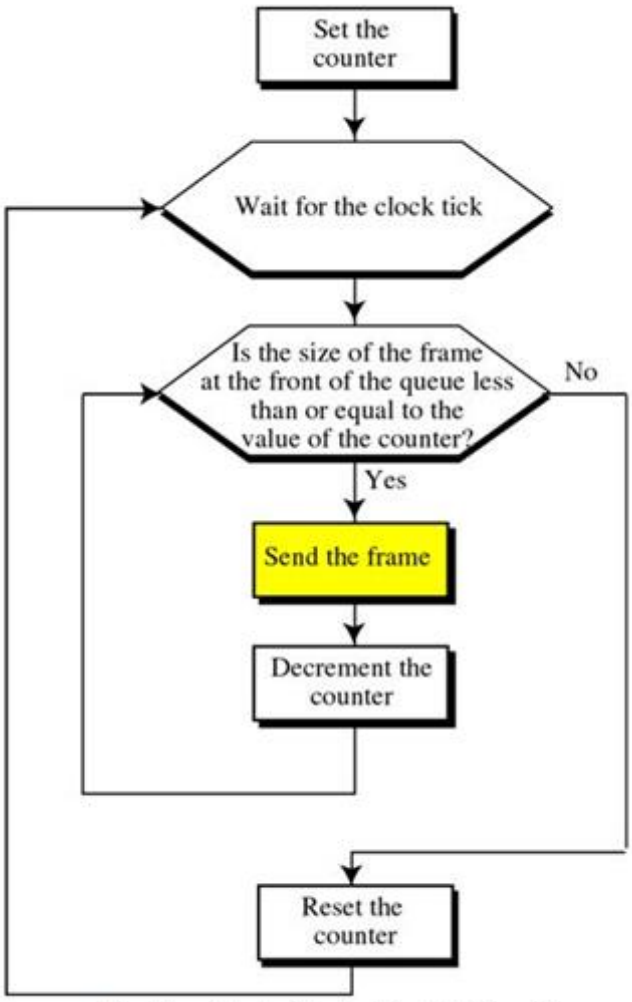
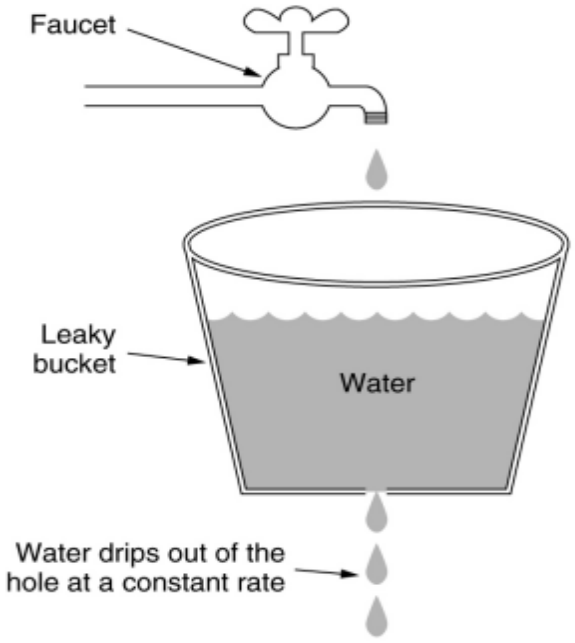
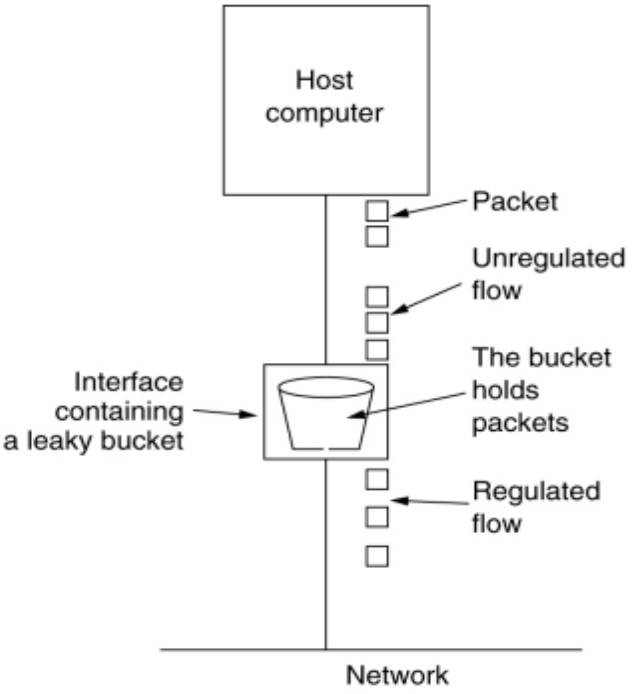


Fig. Flow Chart of Leaky Bucket Algorithm



(a) Leaky bucket with water



(b) Leaky bucket with packets

(ii) Token Bucket (TB) Algorithm

- ✓ The leaky bucket algorithm allows only an average (constant) rate of data flow. Its major problem is that it cannot deal with bursty data.
- ✓ A token bucket algorithm is a modification of leaky bucket in which leaky bucket contains tokens.
- ✓ In this algorithm, a token(s) are generated at every clock tick. For a packet to be transmitted, system must remove token(s) from the bucket. Thus, a token bucket algorithm allows idle hosts to accumulate credit for the future in form of tokens.

Analogy:

- ✓ A leaky bucket algorithm does not consider the idle time of the host. For example, if the host was idle for 10 seconds and now it is willing to send data at a very high speed for another 10 seconds, the total data transmission will be divided into 20 seconds and average data rate will be maintained. The host is having no advantage of sitting idle for 10 seconds.
- ✓ To overcome this problem, a token bucket algorithm is used. A token bucket algorithm allows bursty data transfers.

Example:

- ✓ if a system generates 100 tokens in one clock tick and the host is idle for 100 ticks. The bucket will contain 10,000 tokens. Now, if the host wants to send bursty data, it can consume all 10,000 tokens at once for sending 10,000 cells or bytes.
- ✓ Thus, a host can send bursty data as long as bucket is not empty.

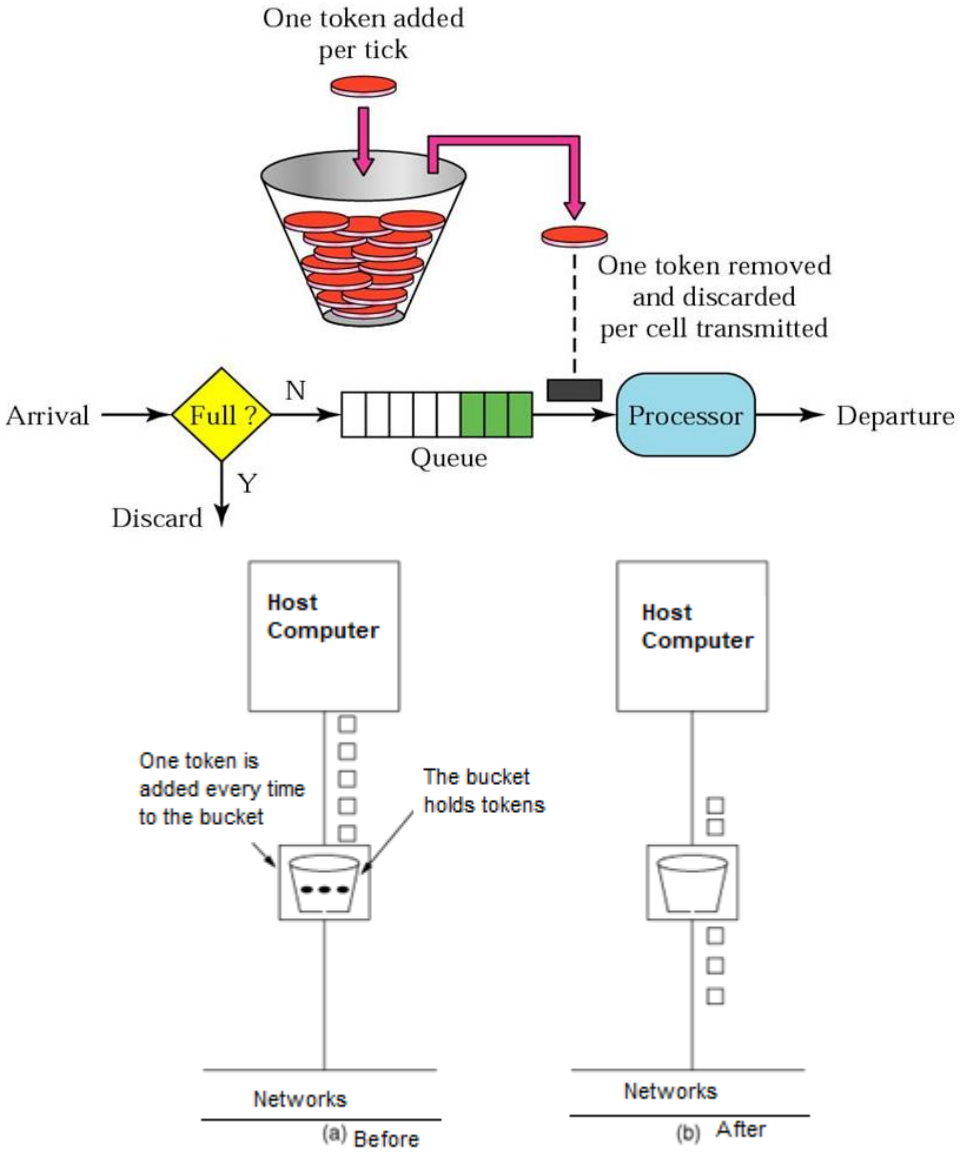


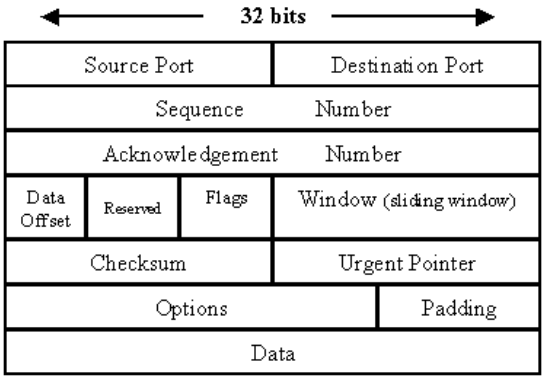
Fig. Toket Bucket Algorithm

Like the leaky bucket algorithm, the token bucket algorithm queues up packets. But instead of discarding packets when the queue is full, the bucket holds a number of tokens--say, 3. This means that 3 packets could be sent at once, while (for example) two packets are queued up. As tokens are regenerated at some regular time intervals, packets can be sent along. This algorithm allows for some burstiness in traffic, which the leaky bucket algorithm does not, and does not discard packets

Difference between TCP and UDP

1) Connection oriented vs Connection less

First and foremost, difference between them is TCP is a connection oriented protocol, and UDP is connection less protocol. This means a connection is established between client and server, before they can send data. Connection establishment process is also known as TCP hand shaking where control messages are interchanged between client and server. Attached image describe the process of TCP handshake, for example which control messages are exchanged between client and server. Client, which is initiator of TCP connection, sends SYN message to server, which is listening on a TCP port. Server receives and sends a SYN-ACK message, which is received by client again and responded using ACK. Once server receive this ACK message, TCP connection is established and ready for data transmission. On the other hand, UDP is a connection less protocol, and point to point connection is not established before sending messages. That's the reason, UDP is more suitable for multicast distribution of message, one to many distribution of data in single transmission.



TCP Header Format

2) Reliability

TCP provides delivery guarantee, which means a message sent using TCP protocol is guaranteed to be delivered to client. If message is lost in transits then its recovered using resending, which is handled by TCP protocol itself. On the other hand, UDP is unreliable, it doesn't provide any delivery guarantee. A datagram package may be lost in transits. That's why UDP is not suitable for programs which requires guaranteed delivery.

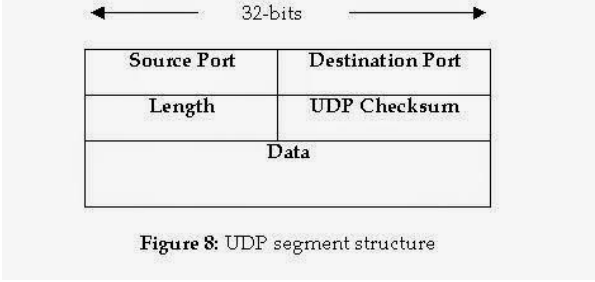
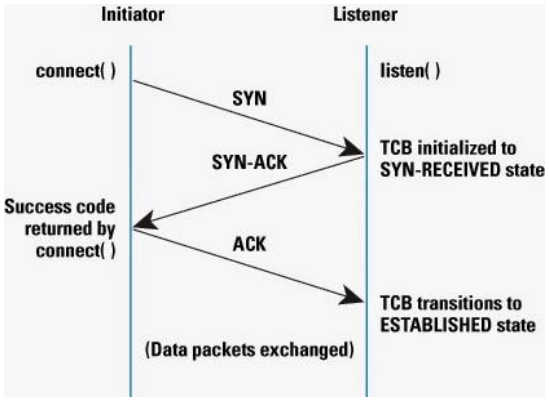


Figure 8: UDP segment structure

UDP Header Format

3) Ordering

Apart from delivery guarantee, TCP also guarantees order of message. Message will be delivered to client in the same order, server has sent, though its possible they may reach out of order to the other end of the network. TCP protocol will do all sequencing and ordering for you. UDP doesn't provide any ordering or sequencing guarantee. Datagram packets may arrive in any order. That's why TCP is suitable for application which need delivery in sequenced manner, though there are UDP based protocol as well which provides ordering and reliability by using sequence number and redelivery e.g. TIBCO Rendezvous, which is actually a UDP based application.



Connection oriented

4) Data Boundary

TCP does not preserve data boundary, UDP does. In Transmission control protocol, data is sent as a byte stream, and no distinguishing indications are transmitted to signal message (segment) boundaries. On UDP, Packets are sent individually and are checked for integrity only if they arrived. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent. Though TCP will also deliver complete message after assembling all bytes. Messages are stored on TCP buffers before sending to make optimum use of network bandwidth.

5) Speed

In one word, TCP is slow and UDP is fast. Since TCP does has to create connection, ensure guaranteed and ordered delivery, it does lot more than UDP. This cost TCP in terms of speed, that's why UDP is more suitable where speed is a concern, for example online video streaming, telecast or online multi-player games.

6) Heavy weight vs Light weight

Because of the overhead mentioned above, Transmission control protocol is considered as heavy weight as compared to light weight UDP protocol. Simple mantra of UDP to deliver message without bearing any overhead of creating connection and guaranteeing delivery or order guarantee keeps it light weight. This is also reflected in their header sizes, which is used to carry meta data.

7) Header size

TCP has bigger header than UDP. Usual header size of a TCP packet is 20 bytes which is more than double of 8 bytes, header size of UDP datagram packet. TCP header contains Sequence Number, Ack number, Data offset, Reserved, Control bit, Window, Urgent Pointer, Options, Padding, Check Sum, Source port, and Destination port. While UDP header only contains Length, Source port, Destination port, and Check Sum.

8) Congestion or Flow control

TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. On the other hand, UDP does not have an option for flow control.

9) Usage and application

Where does TCP and UDP are used in internet? After knowing key differences between TCP and UDP, we can easily conclude, which situation suits them. Since TCP provides delivery and sequencing guaranty, it is best suited for applications that require high reliability, and transmission time is relatively less critical. While UDP is more suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. In practice, TCP is used in finance domain e.g. FIX protocol is a TCP based protocol, UDP is used heavily in gaming and entertainment sites

10) TCP and UDP based Protocols

One of the best example of TCP based higher end protocol is HTTP and HTTPS, which is every where on internet. In fact most of the common protocol you are familiar of e.g. Telnet, FTP and SMTP all are based over Transmission Control Protocol. UDP don't have any thing as popular as HTTP but UDP is used in protocol like DHCP (Dynamic Host Configuration Protocol) and DNS (Domain Name System). Some of the other protocol, which is based over User Datagram protocol is Simple Network Management Protocol (SNMP), TFTP, BOOTP and NFS (early versions).