Transport Layer

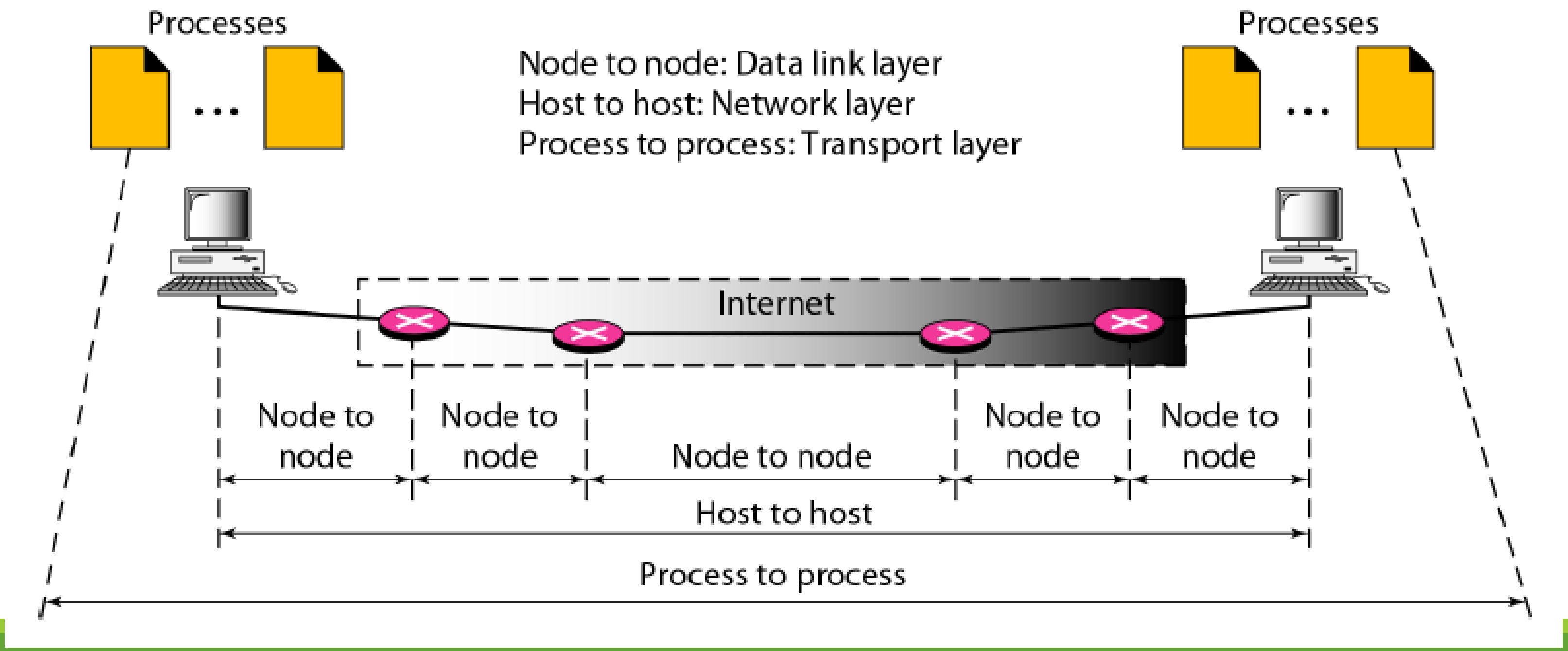
DR. NAVNEET KAUR

PROCESS-TO-PROCESS DELIVERY

- The **data link layer** is responsible for delivery of frames between two neighboring nodes over a link. This is called ***node-to-node delivery.***
- The **network layer** is responsible for delivery of datagrams between two hosts. This is called ***host-to-host delivery.***
- The transport layer is responsible for ***process-to-process delivery-****the* delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship.

Client/Server Paradigm

- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- Both processes (client and server) have the same name. For example, to get the day and time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on a remote machine.



Operating systems today support both multiuser and multiprogramming environments. A remote computer can run several server programs at the same time, just as local computers can run one or more client programs at the same time. For communication, we must define the following:

1. Local host

2. Local process

3. Remote host

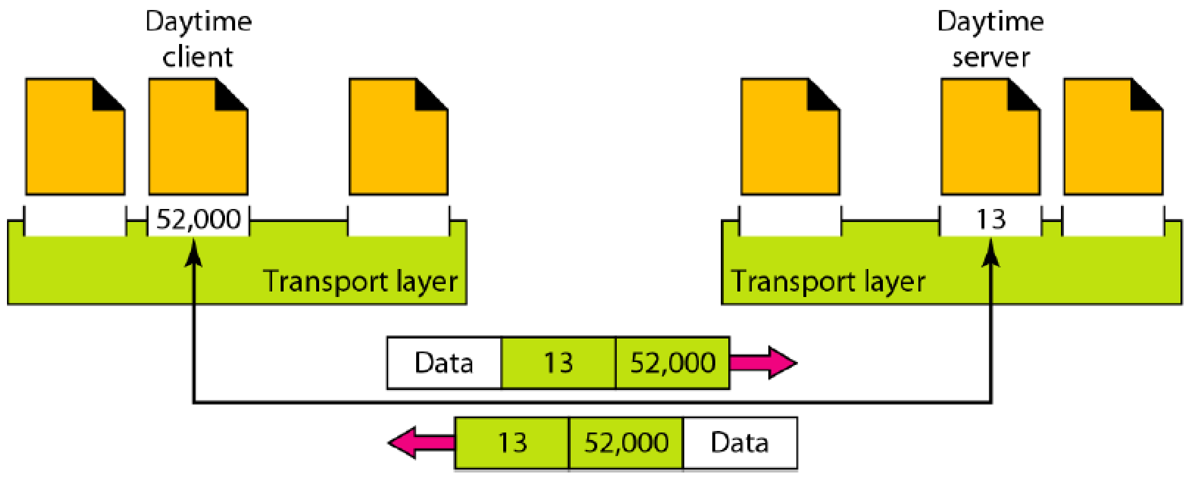
4. Remote process

Addressing

- Whenever we need to deliver something to one specific destination among many, we need an address.
- At the data link layer, we need **a MAC address** to choose one node among several nodes if the connection is not point-to-point. A frame in the data link layer needs a destination MAC address for delivery and a source address for the next node's reply.
- At the network layer, we need **an IP address** to choose one host among millions. A datagram in the network layer needs a destination IP address for delivery and a source IP address for the destination's reply.
- At the transport layer, we need **a transport layer address, called a port number**, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply.

In the Internet model, the port numbers are 16-bit integers between 0 and 65,535. The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral (temporary) port number. The Internet has decided to use universal port numbers for servers; these are called well-known port numbers.

Port numbers

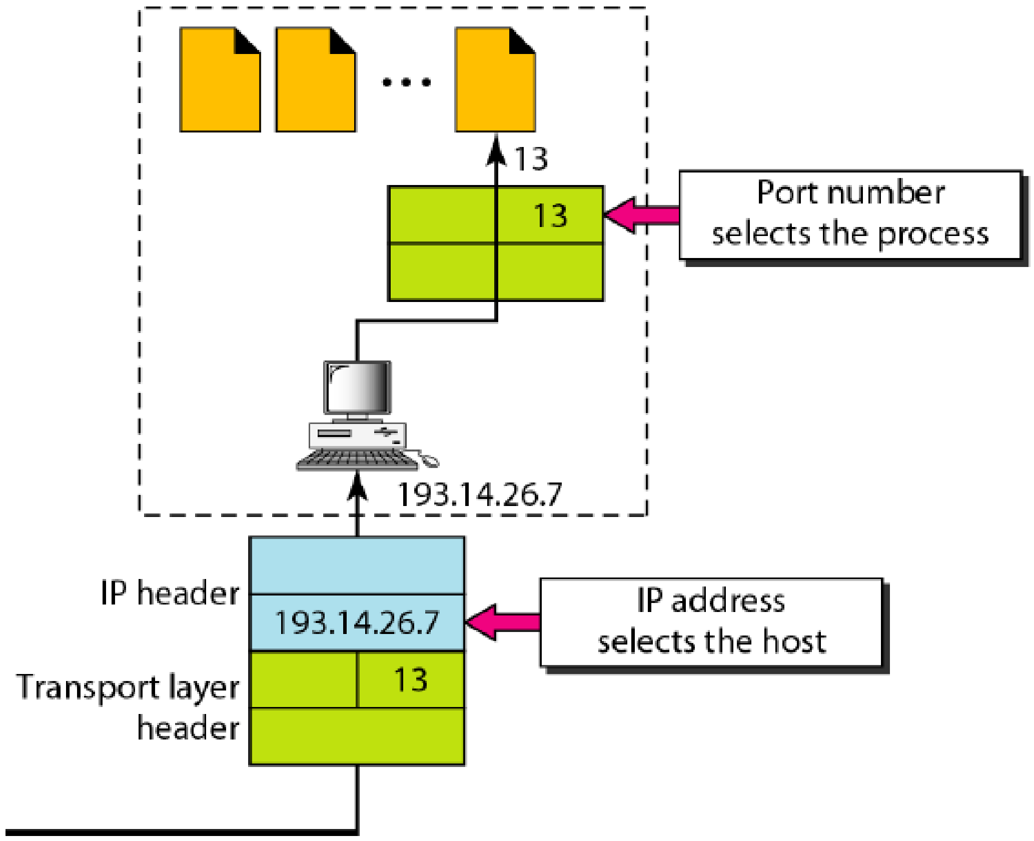


IP addresses versus port numbers

It should be clear by now that the IP addresses and port numbers play different roles in selecting the final destination of data.

The destination IP address defines the host among the different hosts in the world.

After the host has been selected, the port number defines one of the processes on this particular host (see Figure 23.3).



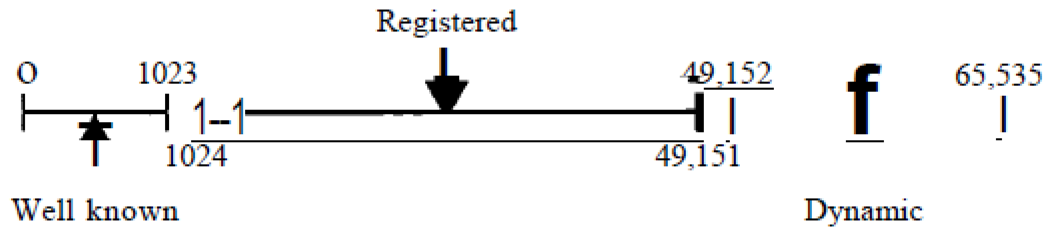
IANA Ranges

The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private), as shown in Figure.

o **Well-known ports.** The ports ranging from 0 to 1023 are assigned and controlled by IANA. These are the well-known ports.

o **Registered ports.** The ports ranging from 1024 to 49,151 are not assigned or controlled by IANA. They can only be registered with IANA to prevent duplication.

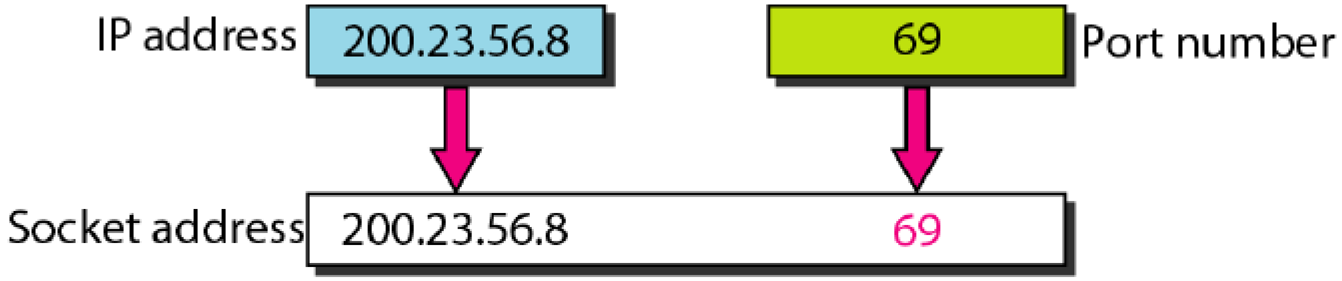
o **Dynamic ports.** The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.



Socket Addresses

Process-to-process delivery needs two identifiers, IP address and the port number, at each end to make a connection. The combination of an IP address and a port number is called a **socket address**. The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely.

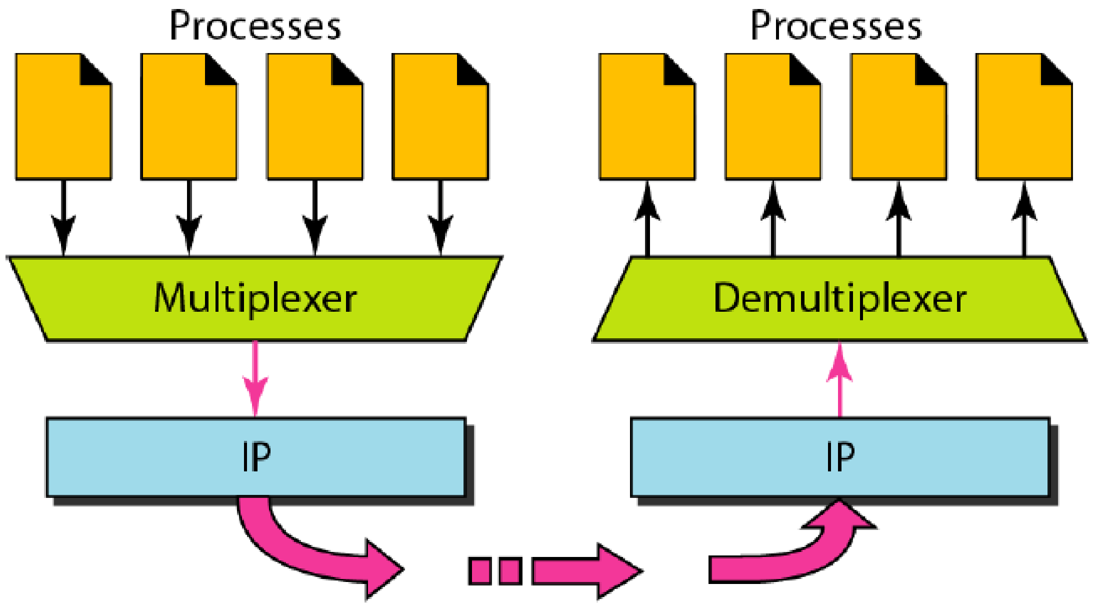
A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the IP header and the transport layer protocol header. The IP header contains the IP addresses; the UDP or TCP header contains the port numbers.



Multiplexing and Demultiplexing

Multiplexing

- At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time.
- This is a many-to-one relationship and requires multiplexing.
- The protocol accepts messages from different processes, differentiated by their assigned port numbers.
- After adding the header, the transport layer passes the packet to the network layer.



Demultiplexing

- At the receiver site, the relationship is one-to-many and requires demultiplexing.
- The transport layer receives datagrams from the network layer.
- After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.

Connectionless Versus Connection-Oriented Service

A transport layer protocol can either be connectionless or connection-oriented.

Connectionless Service

In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either. We will see shortly that one of the transport layer protocols in the Internet model, **UDP, is connectionless**.

Connection-Oriented Service

In a connection-oriented service, a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. We will see shortly that **TCP and SCTP are connection-oriented protocols.**

Reliable Versus Unreliable

The transport layer service can be reliable or unreliable. If the application layer program needs reliability, we use a reliable transport layer protocol by implementing flow and error control at the transport layer. This means a slower and more complex service.

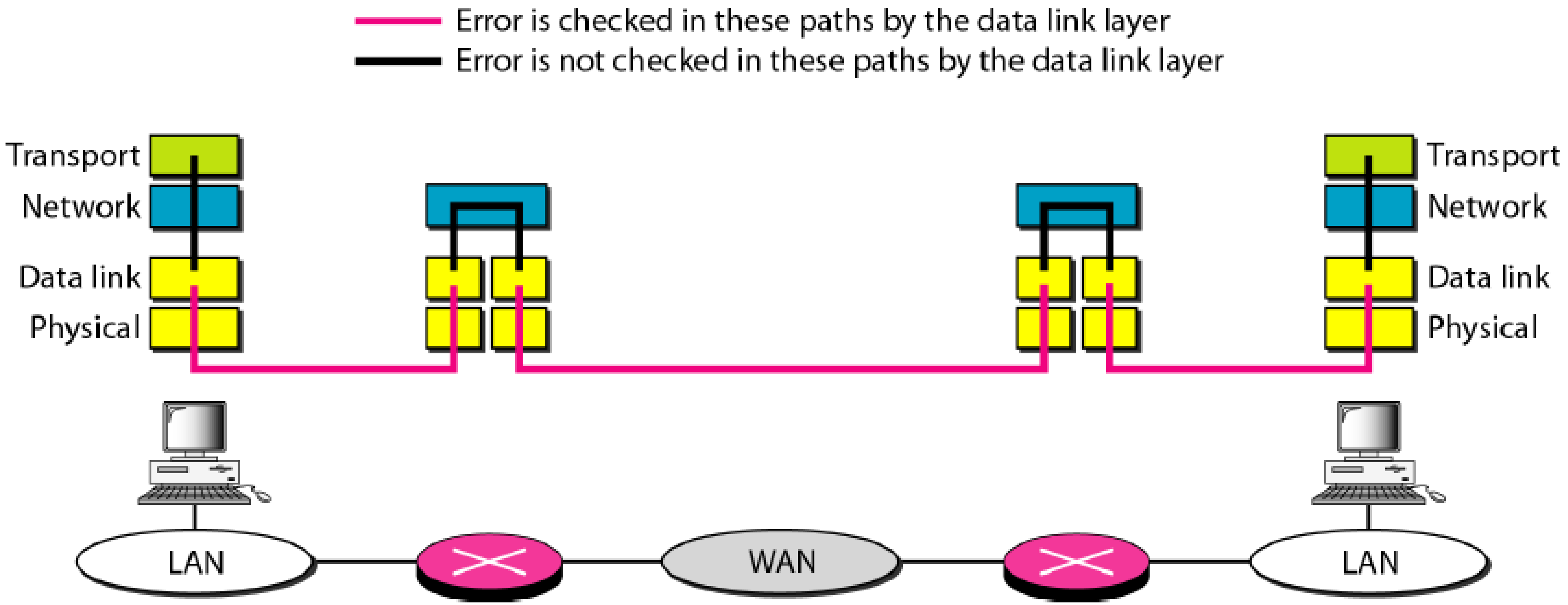
On the other hand, if the application program does not need reliability because it uses its own flow and error control mechanism or it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.

In the Internet, there are three common different transport layer protocols, as we have already mentioned. UDP is connectionless and unreliable; TCP and SCTP are connection-oriented and reliable. These three can respond to the demands of the application layer programs.

One question often comes to the mind. If the data link layer is reliable and has flow and error control, do we need this at the transport layer, too? The answer is yes. Reliability at the data link layer is between two nodes; we need reliability between two ends. Because the network layer in the Internet is unreliable (best-effort delivery), we need to implement reliability at the transport layer.

Reliable Vs Unreliable

Error control

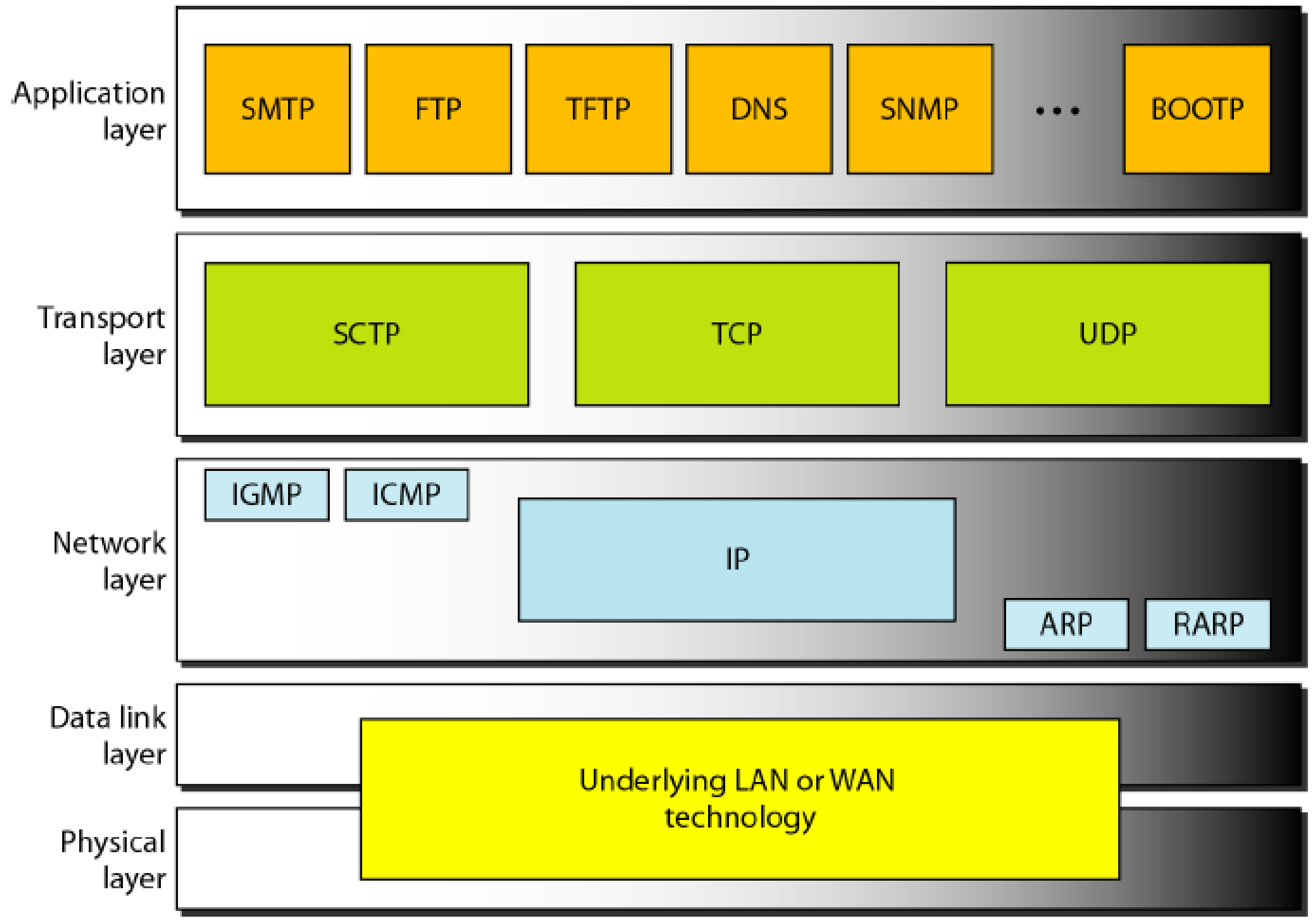


Error is checked in these paths by the data link layer
Error is not checked in these paths by the data link layer
Transport
Network
Data link
Physical
Transport
Network
Data link
Physical
LAN
WAN
LAN

Three Protocols

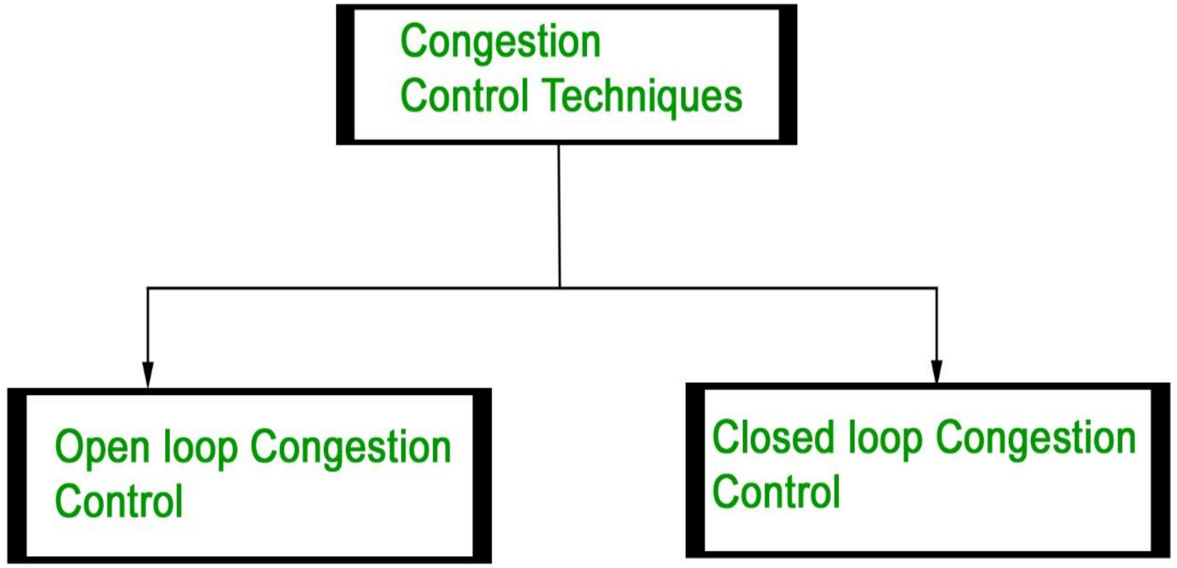
The original TCP/IP protocol suite specifies two protocols for the transport layer: UDP and TCP. We first focus on UDP, the simpler of the two, before discussing TCP. A new transport layer protocol, SCTP, has been designed. Figure shows the position of these protocols in the TCP/IP protocol suite.

Position of UDP, TCP, and SCTP in TCP/IP suite



Congestion Control techniques in Computer Networks

Congestion control refers to the techniques used to control or prevent congestion. Congestion control techniques can be broadly classified into two categories:


Congestion Control Techniques
Open loop Congestion Control
Closed loop Congestion Control

Open Loop Congestion Control

Open loop congestion control policies are applied to prevent congestion before it happens. The congestion control is handled either by the source or the destination.

Policies adopted by open loop congestion control –

- **Retransmission Policy:**

It is the policy in which retransmission of the packets are taken care of. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. This transmission may increase the congestion in the network. To prevent congestion, retransmission timers must be designed to prevent congestion and also able to optimize efficiency.

- **Admission Policy:**

In admission policy a mechanism should be used to prevent congestion. Switches in a flow should first check the resource requirement of a network flow before transmitting it further. If there is a chance of a congestion or there is a congestion in the network, router should deny establishing a virtual network connection to prevent further congestion. All the above policies are adopted to prevent congestion before it happens in the network.

- **Window Policy:**
  The type of window at the sender’s side may also affect the congestion. Several packets in the Go-back-n window are re-sent, although some packets may be received successfully at the receiver side. This duplication may increase the congestion in the network and make it worse. Therefore, Selective repeat window should be adopted as it sends the specific packet that may have been lost.

- **Discarding Policy:**
  A good discarding policy adopted by the routers is that the routers may prevent congestion and at the same time partially discard the corrupted or less sensitive packages and also be able to maintain the quality of a message. In case of audio file transmission, routers can discard less sensitive packets to prevent congestion and also maintain the quality of the audio file.

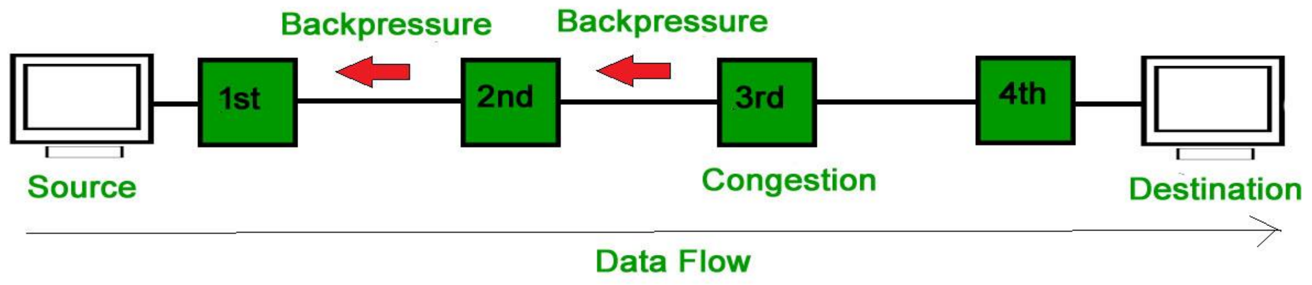
- **Acknowledgment Policy:**
  Since acknowledgements are also the part of the load in the network, the acknowledgment policy imposed by the receiver may also affect congestion. Several approaches can be used to prevent congestion related to acknowledgment. The receiver should send acknowledgement for N packets rather than sending acknowledgement for a single packet. The receiver should send an acknowledgment only if it has to send a packet or a timer expires.

Closed Loop Congestion Control

Closed loop congestion control techniques are used to treat or alleviate congestion after it happens. Several techniques are used by different protocols; some of them are:

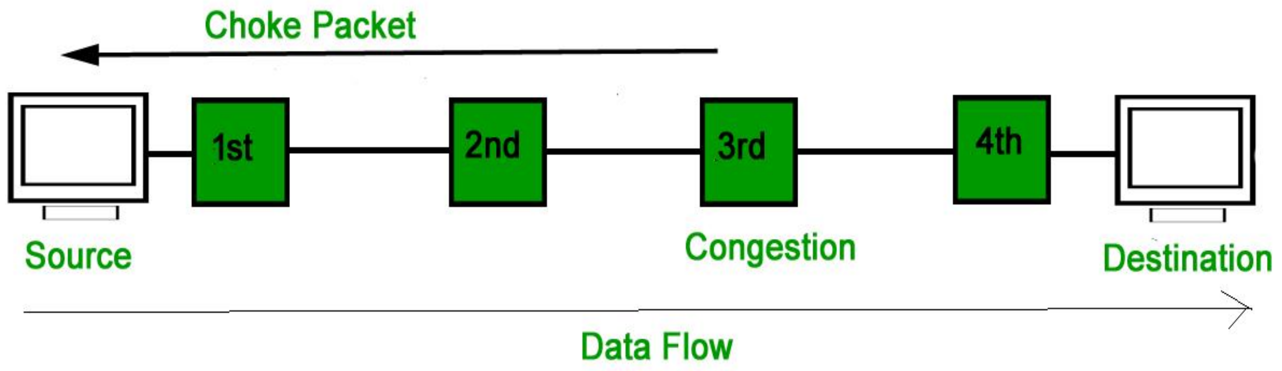
1. Backpressure:
Backpressure is a technique in which a congested node stops receiving packets from upstream node. This may cause the upstream node or nodes to become congested and reject receiving data from above nodes. Backpressure is a node-to-node congestion control technique that propagate in the opposite direction of data flow. The backpressure technique can be applied only to virtual circuit where each node has information of its above upstream node.

In above diagram the 3rd node is congested and stops receiving packets as a result 2nd node may be get congested due to slowing down of the output data flow. Similarly 1st node may get congested and inform the source to slow down.



2. Choke Packet Technique:

Choke packet technique is applicable to both virtual networks as well as datagram subnets. A choke packet is a packet sent by a node to the source to inform it of congestion. Each router monitors its resources and the utilization at each of its output lines. Whenever the resource utilization exceeds the threshold value which is set by the administrator, the router directly sends a choke packet to the source giving it a feedback to reduce the traffic. The intermediate nodes through which the packets has travelled are not warned about congestion.



3. Implicit Signaling :

In implicit signaling, there is no communication between the congested nodes and the source. The source guesses that there is congestion in a network. For example when sender sends several packets and there is no acknowledgment for a while, one assumption is that there is a congestion.

4. Explicit Signaling :

In explicit signaling, if a node experiences congestion it can explicitly sends a packet to the source or destination to inform about congestion. The difference between choke packet and explicit signalling is that the signal is included in the packets that carry data rather than creating a different packet as in case of choke packet technique.

Explicit signaling can occur in either forward or backward direction.

- **Forward Signaling :** In forward signaling, a signal is sent in the direction of the congestion. The destination is warned about congestion. The receiver in this case adopt policies to prevent further congestion.
- **Backward Signaling :** In backward signaling, a signal is sent in the opposite direction of the congestion. The source is warned about congestion and it needs to slow down.

Congestion Control Algorithm
Leaky bucket algorithm

In the network layer, before the network can make Quality of service guarantees, it must know what traffic is being guaranteed. One of the main causes of congestion is that traffic is often bursty.

To understand this concept first we have to know little about traffic shaping. **Traffic Shaping** is a mechanism to control the amount and the rate of traffic sent to the network. Approach of congestion management is called Traffic shaping. Traffic shaping helps to regulate the rate of data transmission and reduces congestion.

There are 2 types of traffic shaping algorithms:

- Leaky Bucket
- Token Bucket

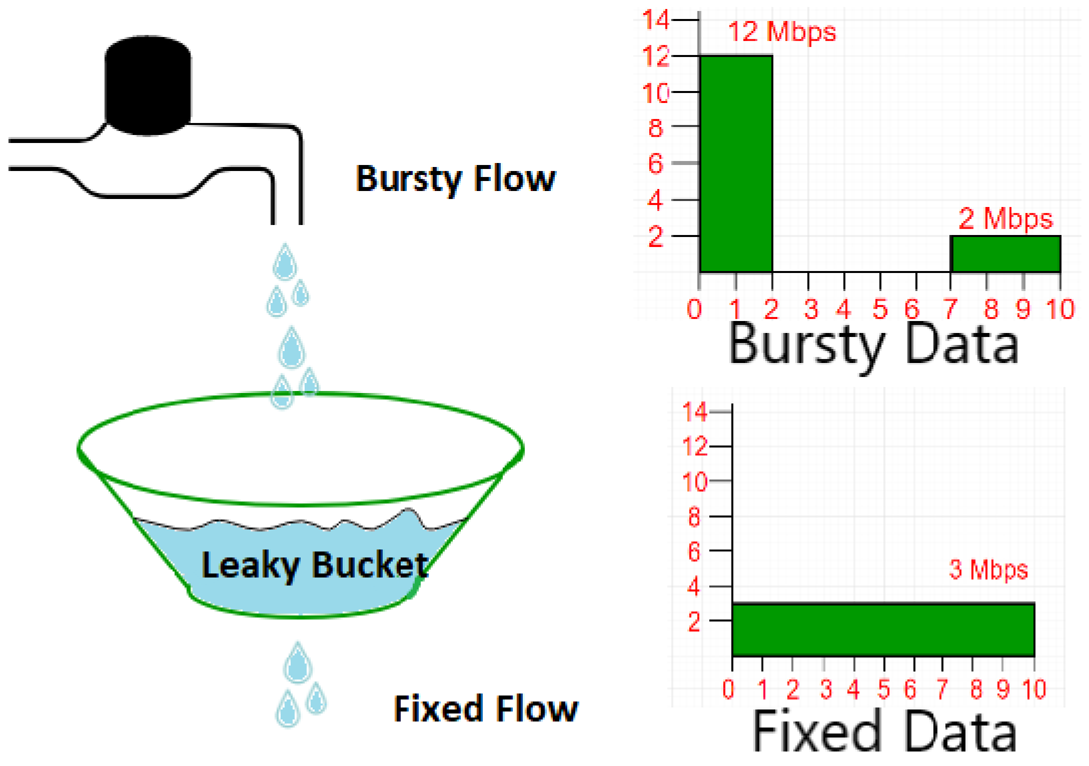
Suppose we have a bucket in which we are pouring water, at random points in time, but we have to get water at a fixed rate, to achieve this we will make a hole at the bottom of the bucket. This will ensure that the water coming out is at some fixed rate, and also if the bucket gets full, then we will stop pouring water into it.

The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.

In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In the figure, the host sends a burst of data at a rate of 12 Mbps for 2s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10 s. The leaky bucket smooths out the traffic by sending out data at a rate of 3 Mbps during the same 10 s.

Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion.

A simple leaky bucket algorithm can be implemented using FIFO queue. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM networks), the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.



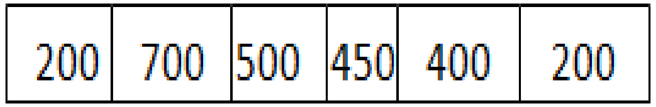
The following is an algorithm for variable-length packets:

```
Initialize a counter to n at the tick of the clock.
Repeat until n is smaller than the packet size of the packet at the head of the queue.
    Pop a packet out of the head of the queue, say P.
    Send the packet P, into the network
    Decrement the counter by the size of packet P.
Reset the counter and go to step 1.
```

Note: *In the below examples, the head of the queue is the rightmost position and the tail of the queue is the leftmost position.*

Example: Let n=1000

Packet=

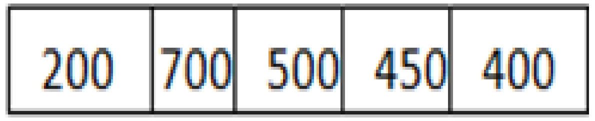


200	700	500	450	400	200

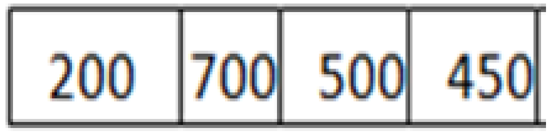
Since n > size of the packet at the head of the Queue, i.e. n > 200

Therefore, n = 1000-200 = 800

Packet size of 200 is sent into the network.



Now, again n > size of the packet at the head of the Queue, i.e. n > 400
Therefore, n = 800-400 = 400
Packet size of 400 is sent into the network.



Since, n < size of the packet at the head of the Queue, i.e. n < 450
Therefore, the procedure is stopped.

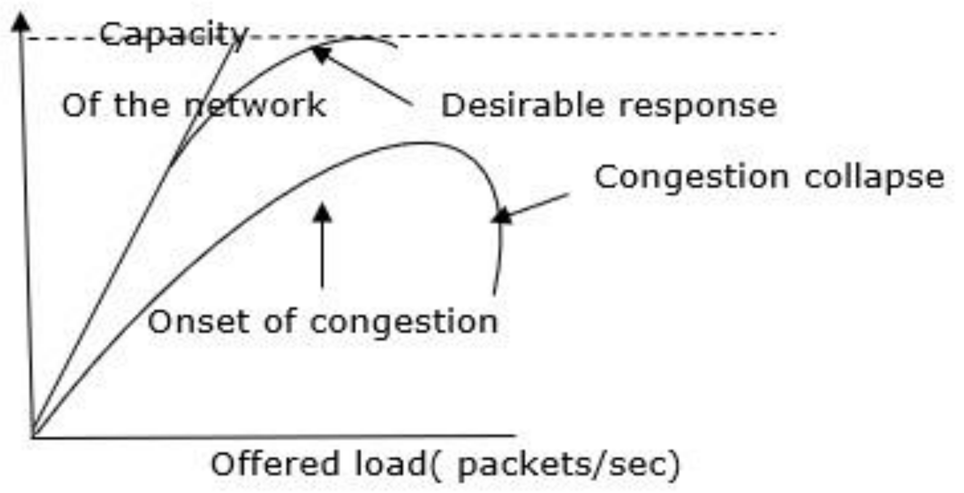
Initialise n = 1000 on another tick of the clock.
This procedure is repeated until all the packets are sent into the network.

Token Bucket algorithm

Token bucket algorithm is one of the techniques for congestion control algorithms. When too many packets are present in the network it causes **packet delay and loss of packet** which degrades the performance of the system. This situation is called **congestion**.

The **network layer** and **transport layer** share the responsibility for handling congestions. One of the most effective ways to control congestion is trying to reduce the load that transport layer is placing on the network. To maintain this network and transport layers have to work together.

The Token Bucket Algorithm is diagrammatically represented as follows −



With too much traffic, performance drops sharply.

Token Bucket Algorithm

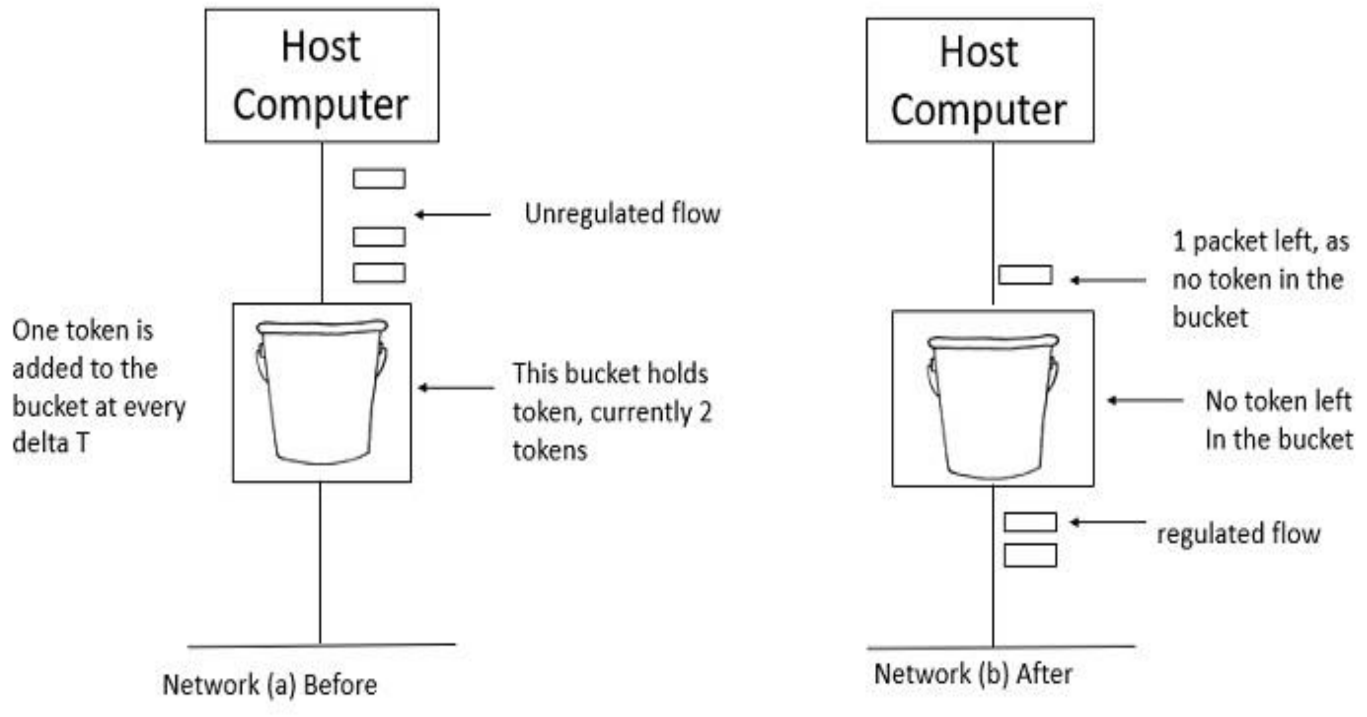
The leaky bucket algorithm enforces output patterns at the average rate, no matter how busy the traffic is. So, to deal with the more traffic, we need a flexible algorithm so that the data is not lost. One such approach is the token bucket algorithm.

Let us understand this algorithm step wise as given below −

- **Step 1** − In regular intervals tokens are thrown into the bucket f.
- **Step 2** − The bucket has a maximum capacity f.
- **Step 3** − If the packet is ready, then a token is removed from the bucket, and the packet is sent.
- **Step 4** − Suppose, if there is no token in the bucket, the packet cannot be sent.

Example

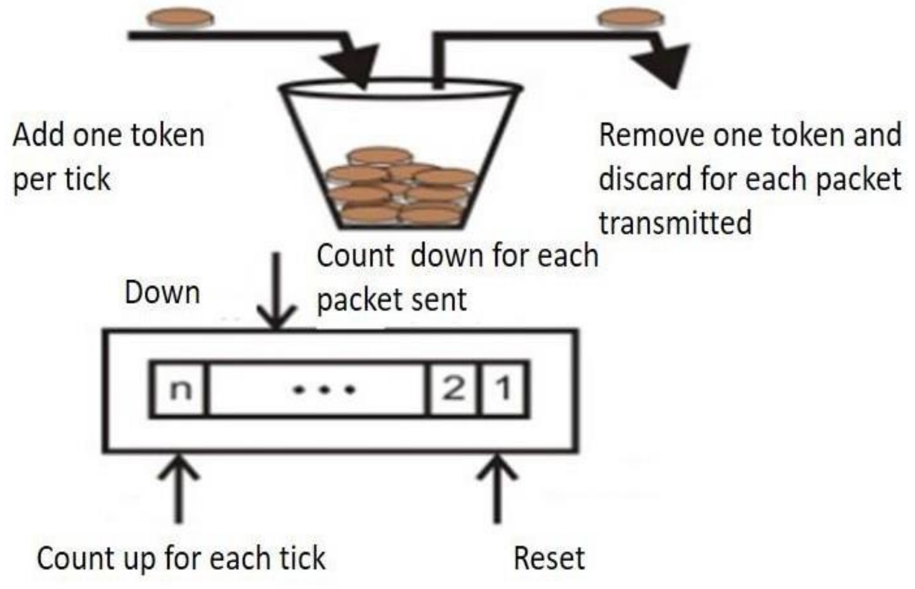
Let us understand the Token Bucket Algorithm with an example –


Host Computer
Unregulated flow
One token is added to the bucket at every delta T
This bucket holds token, currently 2 tokens
Network (a) Before
Host Computer
1 packet left, as no token in the bucket
No token left In the bucket
regulated flow
Network (b) After

In figure (a) the bucket holds two tokens, and three packets are waiting to be sent out of the interface.
In Figure (b) two packets have been sent out by consuming two tokens, and 1 packet is still left.

When compared to Leaky bucket the token bucket algorithm is less restrictive that means it allows more traffic. The limit of busyness is restricted by the number of tokens available in the bucket at a particular instant of time.

The implementation of the token bucket algorithm is easy − a variable is used to count the tokens. For every t seconds the counter is incremented and then it is decremented whenever a packet is sent. When the counter reaches zero, no further packet is sent out.
This is shown in below given diagram −


Add one token per tick
Remove one token and discard for each packet transmitted
Down
Count down for each packet sent
n
• • •
2
1
Count up for each tick
Reset

What is Transmission Control Protocol (TCP)?

- TCP stands for **Transmission Control Protocol**. It is a transport layer protocol that facilitates the transmission of packets from source to destination.
- It is a connection-oriented protocol that means it establishes the connection prior to the communication that occurs between the computing devices in a network.
- This protocol is used with an IP protocol, so together, they are referred to as a TCP/IP.
- The main functionality of the TCP is to take the data from the application layer.
- Then it divides the data into a several packets, provides numbering to these packets, and finally transmits these packets to the destination.
- The TCP, on the other side, will reassemble the packets and transmits them to the application layer.
- As we know that TCP is a connection-oriented protocol, so the connection will remain established until the communication is not completed between the sender and the receiver.

Features of TCP protocol

The following are the features of a TCP protocol:

- **Transport Layer Protocol**

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

- **Reliable**

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

- **Order of the data is maintained**

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

- **Connection-oriented**

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

- **Full duplex**

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

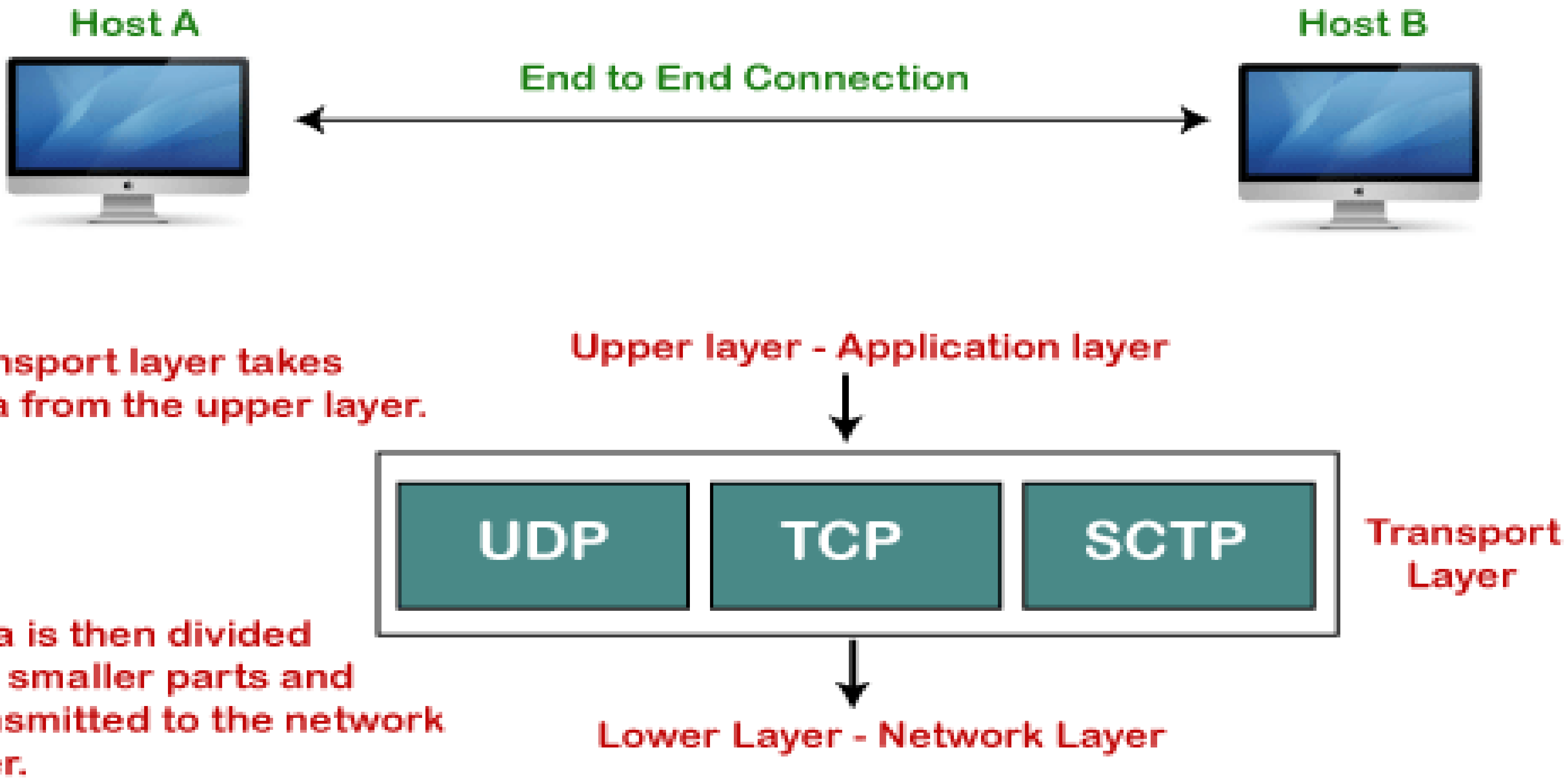
- **Stream-oriented**

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

Need for Transport Control Protocol

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

Purpose of Transport Layer

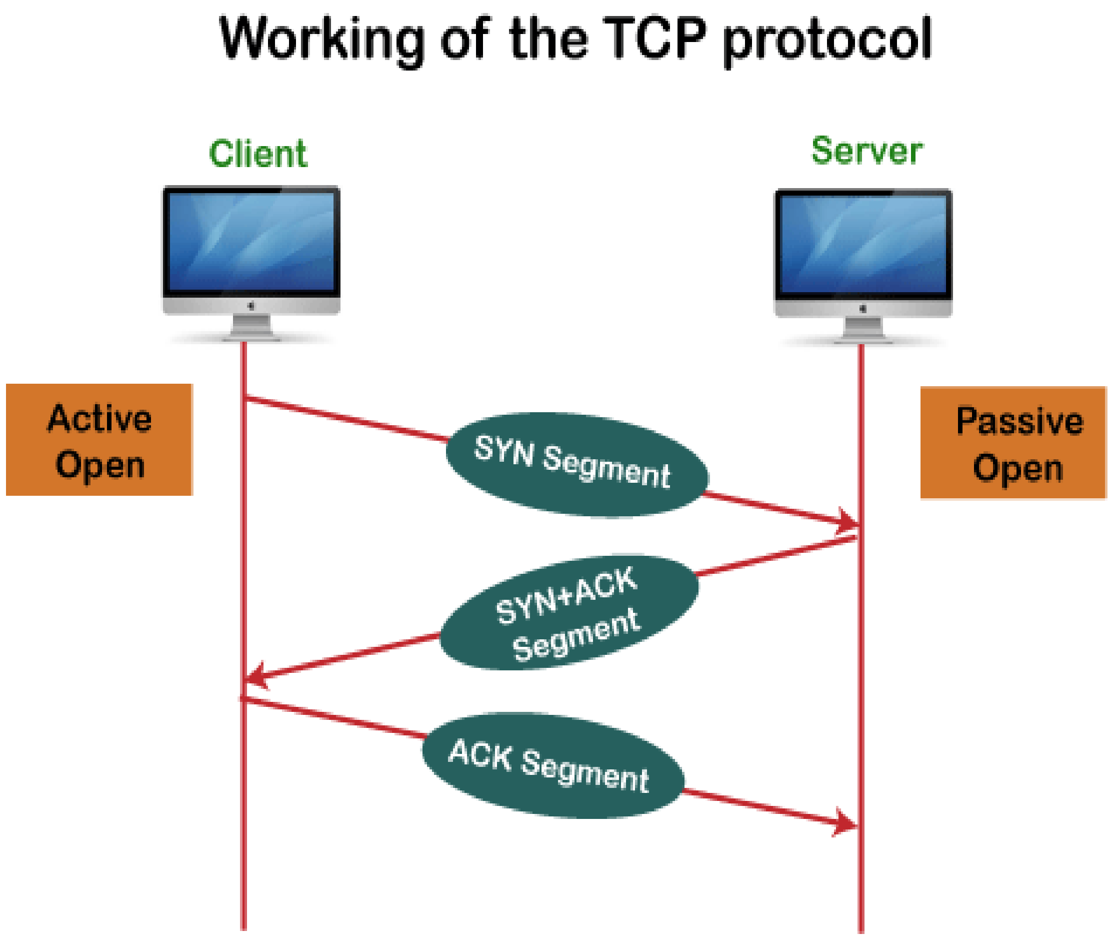


Transport layer takes data from the upper layer.

Data is then divided into smaller parts and transmitted to the network layer.

Working of TCP

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



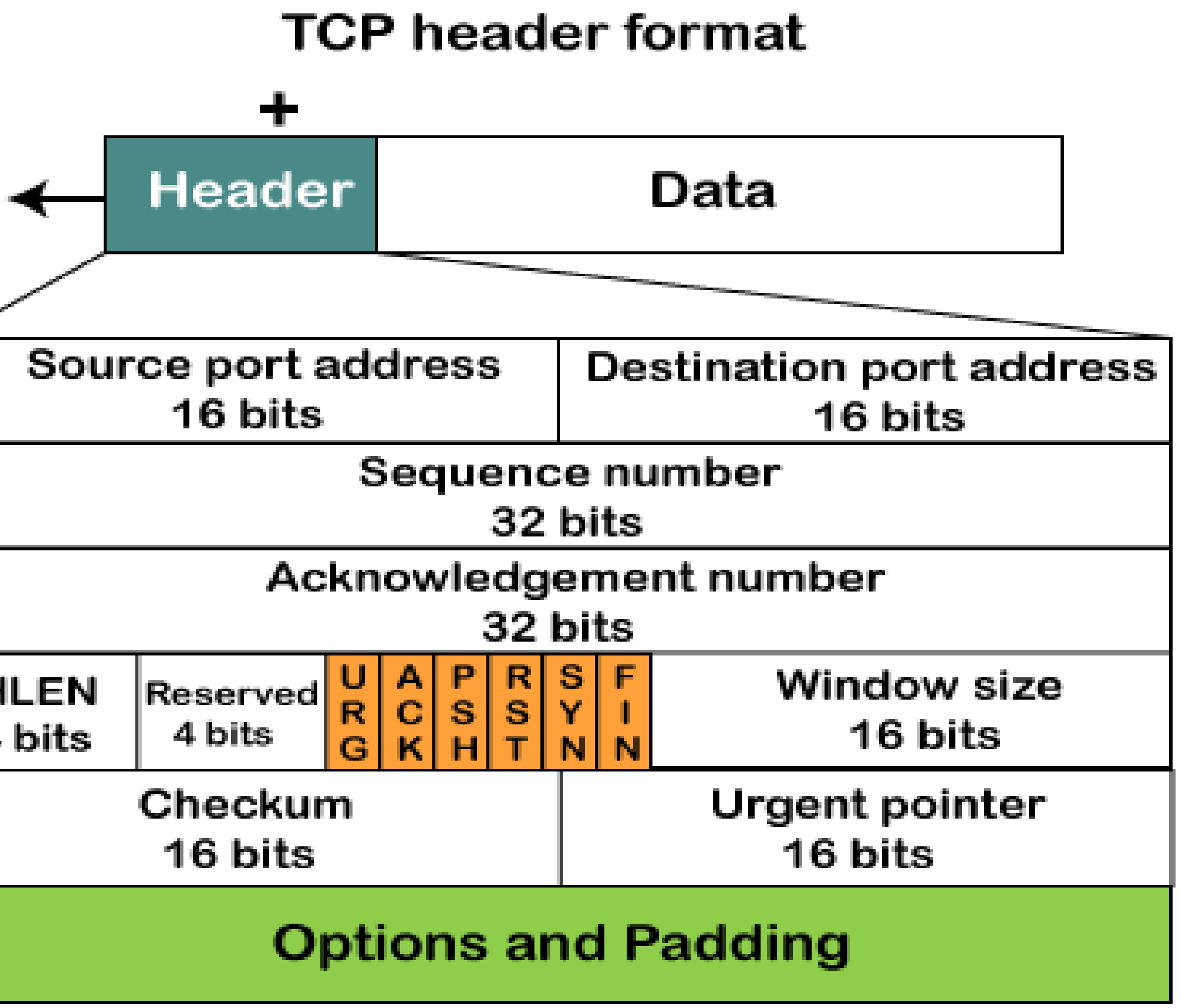
Advantages of TCP

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

Disadvantage of TCP

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

TCP Header Format



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

Flags

There are six control bits or flags:

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

- **Window size**

It is a 16-bit field. It contains the size of data that the receiver can accept. This field is used for the flow control between the sender and receiver and also determines the amount of buffer allocated by the receiver for a segment. The value of this field is determined by the receiver.

- **Checksum**

It is a 16-bit field. This field is optional in UDP, but in the case of TCP/IP, this field is mandatory.

- **Urgent pointer**

It is a pointer that points to the urgent data byte if the URG flag is set to 1. It defines a value that will be added to the sequence number to get the sequence number of the last urgent byte.

- **Options**

It provides additional options. The optional field is represented in 32-bits. If this field contains the data less than 32-bit, then padding is required to obtain the remaining bits.

Examples-

- If header length field contains decimal value 5 (represented as 0101), then-

  Header length = 5 x 4 = 20 bytes

- If header length field contains decimal value 10 (represented as 1010), then-

  Header length = 10 x 4 = 40 bytes

- If header length field contains decimal value 15 (represented as 1111), then-

  Header length = 15 x 4 = 60 bytes

UDP Protocol-

- UDP is short for **User Datagram Protocol**.
- It is the simplest transport layer protocol.
- It has been designed to send data packets over the Internet.
- It simply takes the datagram from the network layer, attaches its header and sends it to the user.

Characteristics of UDP-

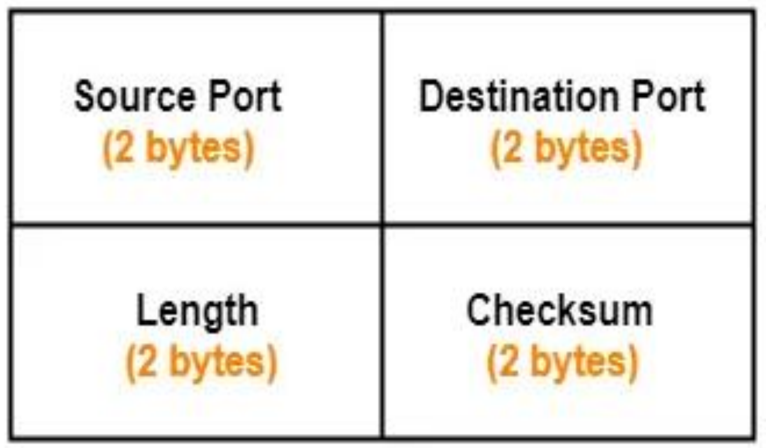
- It is a connectionless protocol.
- It is a stateless protocol.
- It is an unreliable protocol.
- It is a fast protocol.
- It offers the minimal transport service.
- It is almost a null protocol.
- It does not guarantee in order delivery.
- It does not provide congestion control mechanism.
- It is a good protocol for data flowing in one direction.

Need of UDP-

- TCP proves to be an overhead for certain kinds of applications.
- The **Connection Establishment** Phase, **Connection Termination** Phase etc of TCP are time consuming.
- To avoid this overhead, certain applications which require fast speed and less overhead use UDP.

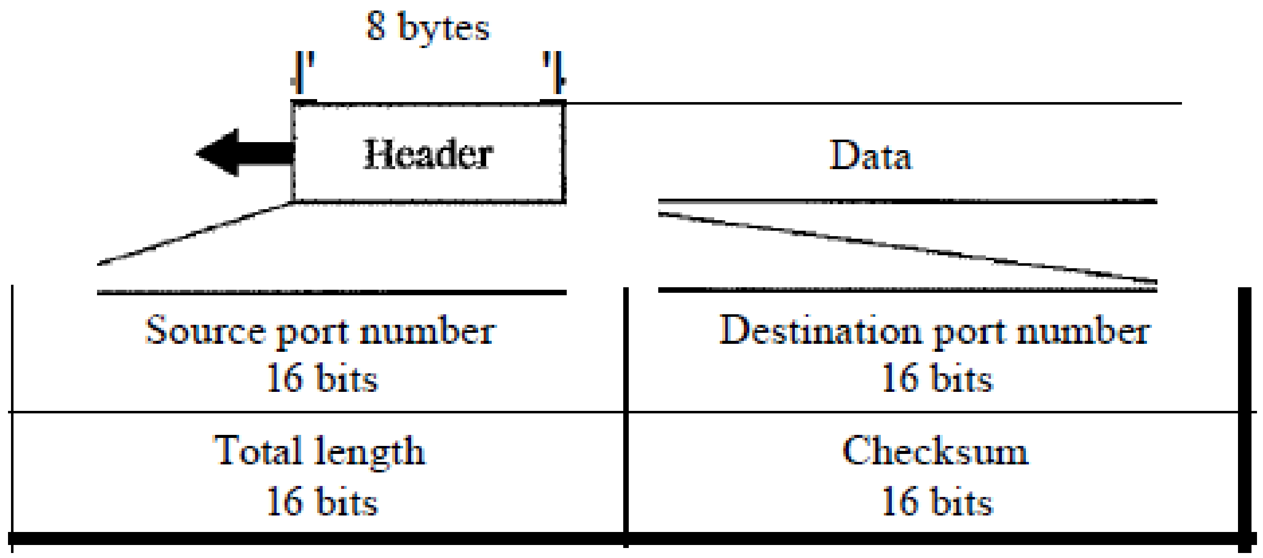
UDP Header-

The following diagram represents the UDP Header Format-



Source Port (2 bytes)	Destination Port (2 bytes)
Length (2 bytes)	Checksum (2 bytes)

UDP Header



1. Source Port-

- Source Port is a 16 bit field.
- It identifies the port of the sending application.

2. Destination Port-

- Destination Port is a 16 bit field.
- It identifies the port of the receiving application.

3. Length-

- Length is a 16 bit field.
- It identifies the combined length of UDP Header and Encapsulated data.

Length = Length of UDP Header + Length of encapsulated data

4. Checksum-

- Checksum is a 16 bit field used for error control.
- It is calculated on UDP Header, encapsulated data and IP pseudo header.
- Checksum calculation is not mandatory in UDP.

Applications Using UDP-

Following applications use UDP-

- Applications which require one response for one request use UDP. Example- **DNS**.
- Routing Protocols like RIP and OSPF use UDP because they have very small amount of data to be transmitted.
- Trivial **File Transfer Protocol** (TFTP) uses UDP to send very small sized files.
- Broadcasting and multicasting applications use UDP.
- Streaming applications like multimedia, video conferencing etc use UDP since they require speed over reliability.
- Real time applications like chatting and online games use UDP.
- Management protocols like SNMP (Simple Network Management Protocol) use UDP.
- Bootp / DHCP uses UDP.
- Other protocols that use UDP are- Kerberos, Network Time Protocol (NTP), Network News Protocol (NNP), Quote of the day protocol etc.

Question: The following is a dump of a UDP header in hexadecimal format.
0632000DOOlCE217
a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?

a. The source port number is first four hexadecimal values i.e. $(0632)_{16}$. Convert it into decimal number. The source port number will be 1586.
b. The destination port number is second four hexadecimal values i.e. $(000D)_{16}$. Convert it into decimal number. The destination port number will be 13.
c. The total length of user datagram is third four hexadecimal values i.e. $(OO1C)_{16}$. Convert it into decimal number. The total length of user datagram is 28.
d. The length of the data= Length of user datagram-length of header i.e. 28-8= 20 bytes.
e. The designation port is 13. and well known ports lie between 0 to 1023. So, it is well known port. So, the packet is directed from a client to a server.