Transport Layer

The transport layer is responsible for the process-to-process delivery of the entire message. That means it delivers data from one program (like a browser) on one computer to the right program (like a web server) on another computer. The network layer only takes care of sending data from one device to another (source to destination), but it doesn't care which program the data belongs to. The transport layer ensures that the whole message arrives at the host application. Transport Layer Services & Responsibilities:

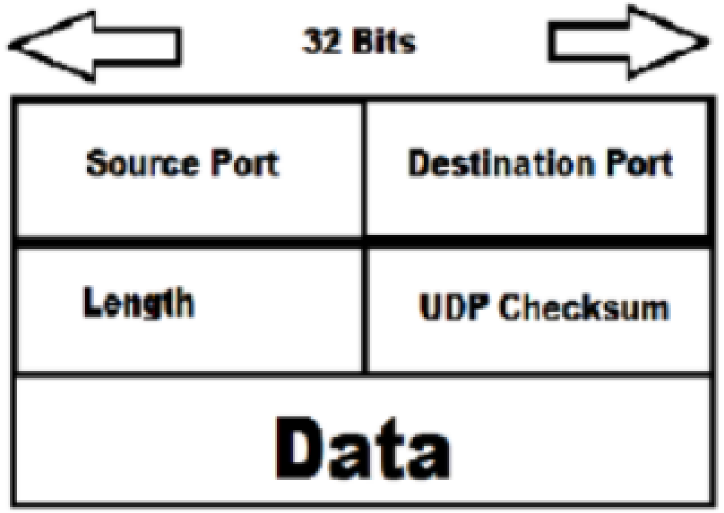
- **Process-to-Process Delivery**
  - Ensures data reaches the right program (process) on the destination device.
- **Multiplexing and Demultiplexing**
  - **Multiplexing:** Combines data from multiple applications for transmission.
  - **Demultiplexing:** Separates incoming data and delivers it to the correct application.
- **Segmentation and Reassembly**
  - Breaks large messages into smaller segments before sending.
  - Reassembles segments back into the original message at the destination.

Transport Layer

- **Congestion Control**
  - Helps prevent network overload by controlling the amount of data sent.
- **Connection Control**
  - Manages the setup and teardown of connections.
  - **TCP**: Connection-oriented (reliable)
  - **UDP**: Connectionless (faster but not guaranteed)
- **Flow Control**
  - Ensures the sender doesn't overwhelm the receiver by sending too much data at once.
- **Error Control**
  - Detects and corrects errors during data transmission using acknowledgments and retransmissions.

User Datagram Protocol (UDP):

The user datagram protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication. The UDP packet structure is as follows:



User Datagram Protocol (UDP):

The user datagram protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication. The UDP packet structure is as follows:

- Source Port **(16 bits):**
- This field indicates the port number of the sending application. It's used as a return address if a response is needed.
- Destination Port **(16 bits):**
- This field indicates the port number of the receiving application. It specifies which application on the receiving end should handle the incoming data.
- Length **(16 bits):**
- This field indicates the total length of the UDP datagram in bytes, including both the header and the data payload. The maximum theoretical size of a UDP datagram is 65,535 bytes, but the actual limit might be lower due to the underlying IP protocol.
- Checksum **(16 bits):**
- This field is used for error detection. It allows the receiving end to verify if the header and data have been corrupted during transmission.

UDP Operations:

UDP uses concepts common to the transport layer.

Connectionless Services: UDP provides connectionless service, meaning that each user datagram sent by UDP is independent. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination process. The user datagrams are not numbered and there is no connection establishment and no connection termination, which means each user datagram can travel on a different path. Stream of data cannot be sent; it should get fragmented.

Flow and Error Control: UDP is a very simple, unreliable transport protocol that does not provide flow control. The receiver may overflow with incoming messages. There is an error control mechanism in UDP, only uses **checksum** to detect errors., which means the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

Encapsulation and Decapsulation: To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages in an IP datagram.

Queuing: Queuing in UDP simply refers to requesting a port number for client processes and using that port number for the process-to-process delivery of messages.

UDP Characteristics and Applications:

• UDP allows very simple data transmission without any error detection. So, it can be used in networking applications like VOIP, streaming media, etc. where the loss of a few packets can be tolerated and still function appropriately.

• UDP is suitable for multicasting since multicasting capability is embedded in UDP software but not in TCP software.

• UDP is used for management processes such as SNMP.

• UDP is used for some route updating protocols such as RIP.

• UDP is used by Dynamic Host Configuration Protocol (DHCP) to assign IP addresses to systems dynamically.

• UDP is an ideal protocol for network applications where latency is critical such as gaming and voice and video communications

Transmission Control Protocol (TCP):

TCP is a transport layer protocol for process-to-process communication like UDP. It is a connection oriented, reliable transport protocol. TCP creates a virtual connection between two TCP clients to send data. In addition, TCP uses flow and error control mechanisms at the transport level.

TCP Operations: Various services and operations of TCP are as follows:

Process-to-Process Communication: Like UDP, TCP provides process-to-process communication using port numbers.

Stream Delivery Service: TCP is a stream-oriented protocol. It allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to have a dedicated connection that carries their data across the internet. For flow control, TCP uses sending and receiving buffer that provides some storage for data packets in case of overflow. This prevents the loss of packets by synchronizing the flow rate.

TCP Operations:

Full-Duplex Communication: TCP offers full-duplex services in which data can flow in both directions at the same time. Each TCP then has to send and receive a buffer, and segments move in both directions.

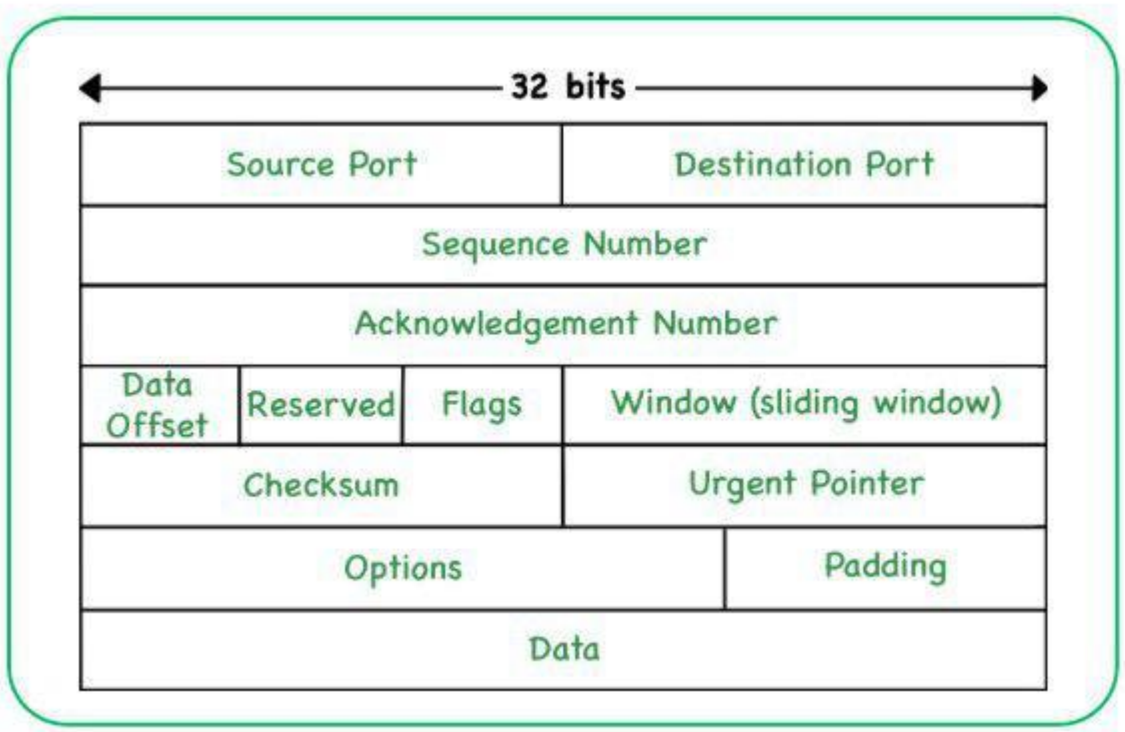
Connection-Oriented Service: When a process at site A wants to send and receive data from another process at site B, the following occurs:

- The two TCPs establish a connection between them.
- Data are exchanged in both directions.
- The connection is terminated.

Note that this is a virtual connection, not a physical connection.

Reliable Service: TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe arrival of data. This is possible due to efficient error control mechanisms

TCP Packet Format:



TCP Packet Format:

•**Source Port(16 bits):** It holds the source/transmitting application's port number and helps in determining the application where the data delivery is planned.

•**Destination Port (16 bits):** This field has the port number of the transmitting application and helps to send the data to the appropriate application.

•**Sequence Number (32 bits):** It ensures that the data is received in proper order by ordered segmenting and reassembling them at the receiving end.

•**Acknowledgment Number (32 bits):** This field contains the upcoming sequence number and it acknowledges the feedback up to that.

•**Data Offset (4 bits):** The data offset field indicates the starting point of the TCP data payload also storing the size of the TCP header.

TCP Packet Format:

•**Control Flags (9 bits):** TCP uses a few control flags to regulate communication. Some of the important flags include:

- **SYN (Synchronize):** Responsible for connecting the sender and receiver.
- **ACK (Acknowledgment):** Its purpose is transferring the acknowledgement of whether the sender has received data.
- **FIN (Finish):** It informs whether the TCP connection is terminated or not.
- **RST (Reset):** Mainly used to reset the connection when an error occurs.

•**Window Size (16 bits):** The size of the sender's receive window is specified by this property.

•**Checksum (16 bits):** It reveals if the header was damaged during transportation.

•**Urgent Pointer (16 bits):** This field points to the packet's first byte of urgent data.

•**Options (Variable length):** This field represents the different TCP options.

•**Data Payload:** This field mainly contains the information which is the actual application data that is being transmitted.

TCP Features and Characteristics:

To support the services of TCP, the following are some features:

Numbering System: TCP keeps track of segments being transmitted or received. There are two fields called the sequence number and the acknowledgment number for numbering the bytes within the segments and acknowledgments respectively.

Flow control: TCP provides a flow control mechanism. The receiver of the data controls the amount of data that is to be sent by the sender. This is done to prevent the receiver from being overloaded with data. The numbering system allows TCP to use byte-oriented flow control.

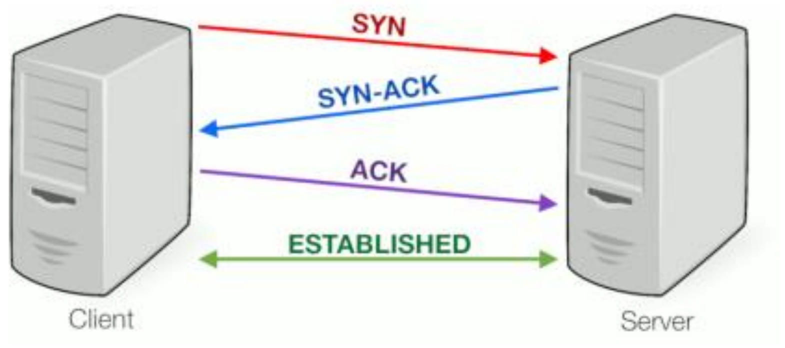
Error Control: To provide reliable service, TCP implements an error control mechanism. Although the error control mechanism considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented.

Congestion Control: TCP takes into account the congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control) but is also determined by the level of congestion in the network.

Three-Way Handshaking:

Since TCP is a connection-oriented service, it requires three phases: connection establishment, data transfer, and connection termination.

The connection establishment in TCP is called three-way handshaking. The figure below shows the handshaking process.



Similarly, three-way handshaking can also be used for connection termination.

Three-Way Handshaking:

Steps in the Three-Way Handshake:

1. The **client** sends a **SYN** packet to the server.
   - Client says: "Hello, I want to talk!“
2. The **server** responds with a **SYN-ACK** packet.
   - Server replies: "Hello, I got your message. I'm ready to talk!
3. The **client** sends an **ACK** packet back to the server.
   - Client responds: "Great! Let's start talking!“

After these three steps:

- The **connection is established**.
- Both sides are ready to send and receive data.
- It's a **3-step process** used by **TCP** to **start a connection** between a client and a server.

Key Differences Between TCP and UDP

- TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol.
- The speed for TCP is slower while the speed of UDP is faster.
- TCP is reliable as it guarantees data delivery while UDP is not reliable.
- TCP uses handshake protocols like SYN, SYN-ACK, and ACK while UDP uses no handshake protocols.
- TCP does error checking and also makes error recovery, on the other hand, UDP performs error checking, but it discards erroneous packets.
- TCP has acknowledgment segments, but UDP does not have any acknowledgment segments.
- TCP is heavy-weight, and UDP is lightweight.
- Data packets arrive in order at the receiver in TCP while no sequencing in UDP. • TCP doesn’t support broadcasting while UDP does.
- TCP is used by HTTP, HTTPS, FTP, SMTP and TELNET while UDP is used by DNS, DHCP, SNMP, RIP and VoIP.

Connection-Oriented Services:

A connection-oriented service needs an established connection between peers before data can be sent between the connection terminals. This method is often called a "reliable" network service. This handles real-time traffic more efficiently than connectionless protocols because data arrives in the same order as it was sent. Connection-oriented protocols are also less error-prone. There is a sequence of operations to be followed by the users of connection-oriented services. These are:

1. Connection is established.
2. Information is sent.
3. Connection is released.

In connection-oriented service, we have to establish a connection before starting the communication. When a connection is established, we send the message or the information and then we release the connection. An example of connection-oriented is TCP (Transmission Control Protocol) protocol.

Connection-less Services:

Connectionless service means that a terminal or node can send data packets to its destination without establishing a connection to the destination. A session connection between the sender and the receiver is not required, the sender just starts sending the data. The message or datagram is sent without prior arrangement, which is less reliable but faster transaction than a connection-oriented service. This works because of error handling protocols, which allow for error correction like requesting retransmission. It is similar to the postal services, as it carries the full address where the message (letter) is to be carried. Each message is routed independently from source to destination. The order of message sent can be different from the order received. LANs are actually connectionless systems with each computer able to transmit data packets as soon as it can access the network. The Internet is a large connectionless packet network in which all packet delivery is handled by Internet providers. Example of Connectionless service is UDP (User Datagram Protocol) protocol.

Congestion Control:

Too many packets present in (a part of) the network causes packet delay and loss that degrades performance. This situation is called congestion. In other words, congestion in a network may occur if the load on the network, the number of packets sent to the network, is greater than the capacity of the network (the number of packets a network can handle). The network and transport layers share the responsibility for handling congestion. Since congestion occurs within the network, it is the network layer that directly experiences it and must ultimately determine what to do with the excess packets. However, the most effective way to control congestion is to reduce the load that the transport layer is placing on the network. Congestion control refers to the mechanisms and techniques to control congestion and keep the load below capacity.

Effects of Congestion:

- As delay increases, performance decreases.
- If delay increases, retransmission occurs, making the situation even worse.

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens or remove congestion after it has happened.

Congestion Control Principle:

The General Principles of Congestion Control are as follows:

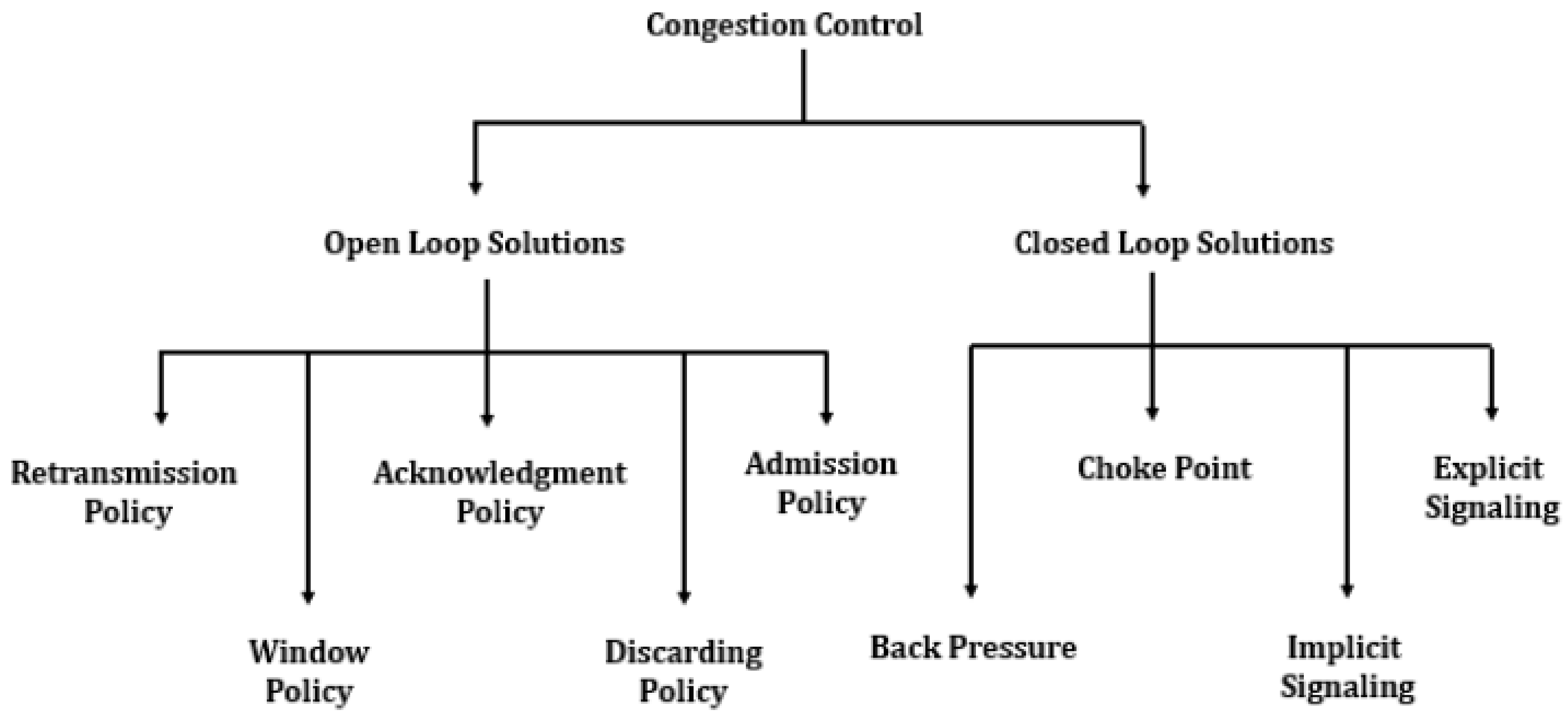
Open Loop Principle:

- attempt to prevent congestion from happening
- after the system is running, no corrections are made

Closed Loop Principle:

- monitor the system to detect congestion
- pass information to where the action is taken
- adjust system operation to correct the problem

Congestion Control Principle:



Open Loop Congestion Control:

In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination.

The list of policies that can prevent congestion are:

Retransmission Policy: It is the policy in which retransmission of the packets is taken care. 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.

Window Policy: The type of window at the sender side may also affect the congestion. Several packets in the Go-back-n window are resent, 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, a Selective repeat window should be adopted as it sends the specific packet that may have been lost.

Acknowledgment Policy: Since acknowledgments are also 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

Open Loop Congestion Control:

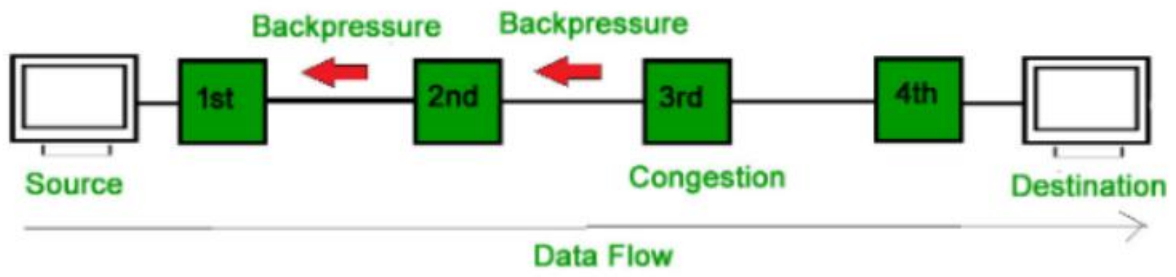
Discarding Policy: A good discarding policy adopted by the routers is that the routers may prevent congestion and at the same time partially discards the corrupted or less sensitive package and also be able to maintain the quality of a message. In the case of audio file transmission, routers can discard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

Admission Policy: In the 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 congestion or there is congestion in the network, a router should deny establishing a virtual network connection to prevent further congestion.

Closed Loop Congestion Control:

Closed-Loop congestion control mechanisms try to reduce the effects of congestion after it happens.

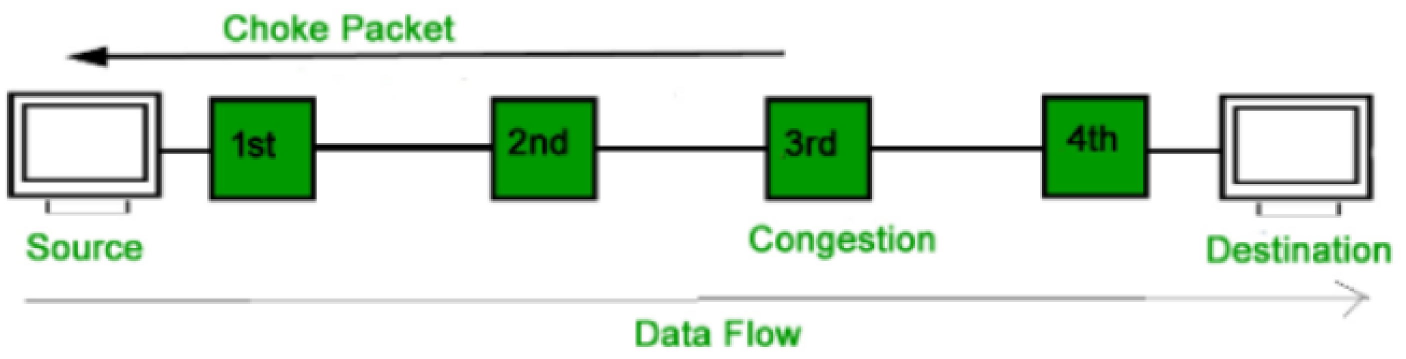
Back Pressure: Backpressure is a technique in which a congested node stops receiving packets from the upstream node. This may cause the upstream node or nodes to become congested and rejects receiving data from the above nodes. Backpressure is a node-to-node congestion control technique that propagates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



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

Closed Loop Congestion Control:

Choke Packet: 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 have traveled are not warned about congestion.



Closed Loop Congestion Control:

Implicit Signaling: In this method, there is no communication between congested nodes or nodes and the source. The source guesses that there is congestion somewhere in the network from other symptoms. For example, when a source sends several packets and there is no acknowledgment for a while, one assumption is that the network is congested and the source should slow down.

Explicit Signaling: In explicit signaling, if a node experiences congestion, it can explicitly send a packet to the source or destination to inform about congestion. The difference between choke packet and explicit signaling is that the signal is included in the packets that carry data rather than creating different packets as in the case of the choke packet technique. Explicit signaling can occur in either a forward or backward direction.

Forward Signaling: In forward signaling, the signal is sent in the direction of the congestion. The destination is warned about congestion. The receiver in this case adopts policies to prevent further congestion.

Backward Signaling: In backward signaling, the signal is sent in the opposite direction of the congestion. The source is warned about congestion and it needs to slow down

TCP Congestion Control:

Congestion Control in TCP (Virtual-Circuit Subnet) is a closed loop-based design for connection-oriented services which can be done during connection setup. The basic principle is that when setting up a virtual circuit, we have to make sure that congestion is avoided.

The following method is used for congestion control in TCP:

- **Admission Control**

Once the congestion has been signaled, no newer virtual circuits can be set up until the problem has been solved.

This type of approach is often used in normal telephone networks. When the exchange is overloaded, then no new calls are established.

TCP Congestion Control:

Another Approach: Alternative routes

To allow new virtual connections, route these carefully so that none of the congested routers (or none of the problem area) is a part of this route i.e., to avoid the part of the network that is overloaded.

Yet another approach can be: To negotiate different parameters between the host and the network when the connection is set up. During the setup time itself, Host specifies the volume and shape of traffic, quality of service, maximum delay, and other parameters, related to the traffic it would be offering to the network. Once the host specifies its requirement, the resources needed are reserved along the path, before the actual packet follows

Traffic Shaping:

Traffic Shaping is a technique used to **control the rate of traffic** sent into the network. It helps to smooth out bursty traffic and ensure network stability and fairness. Two key algorithms used in traffic shaping are:

1. Leaky bucket
2. Token bucket.

Traffic Shaping:

Leaky bucket algorithm:

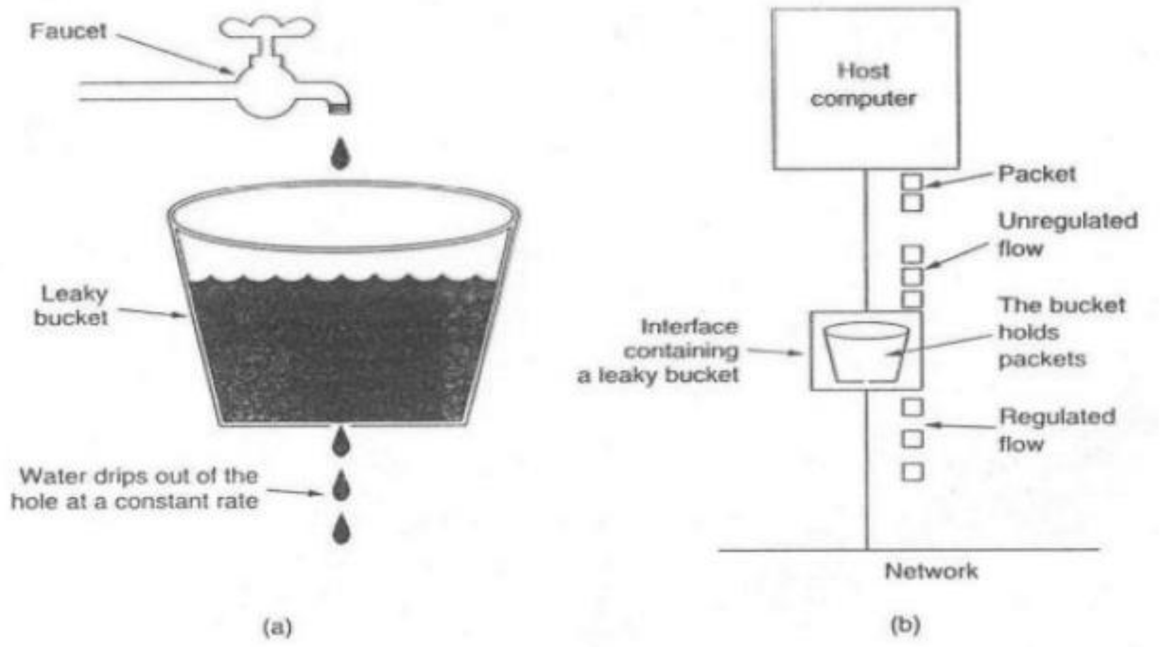
❑ **Concept:**

- Think of a bucket with a **small hole** at the bottom.
- Water (packets/data) is poured into the bucket at any rate.
- The bucket **leaks at a constant rate**.
- If the bucket overflows, the extra water (packets) is **discarded.**

❑ **Mechanism:**

- A **fixed-size queue (bucket)** holds the incoming packets.
- Packets are removed and transmitted at a **constant rate**, r.
- If the queue is full and new packets arrive, they are **dropped** (i.e., traffic is shaped by **dropping** excess traffic).

Traffic Shaping:



Traffic Shaping:

Token bucket.

❑ **Concept:**

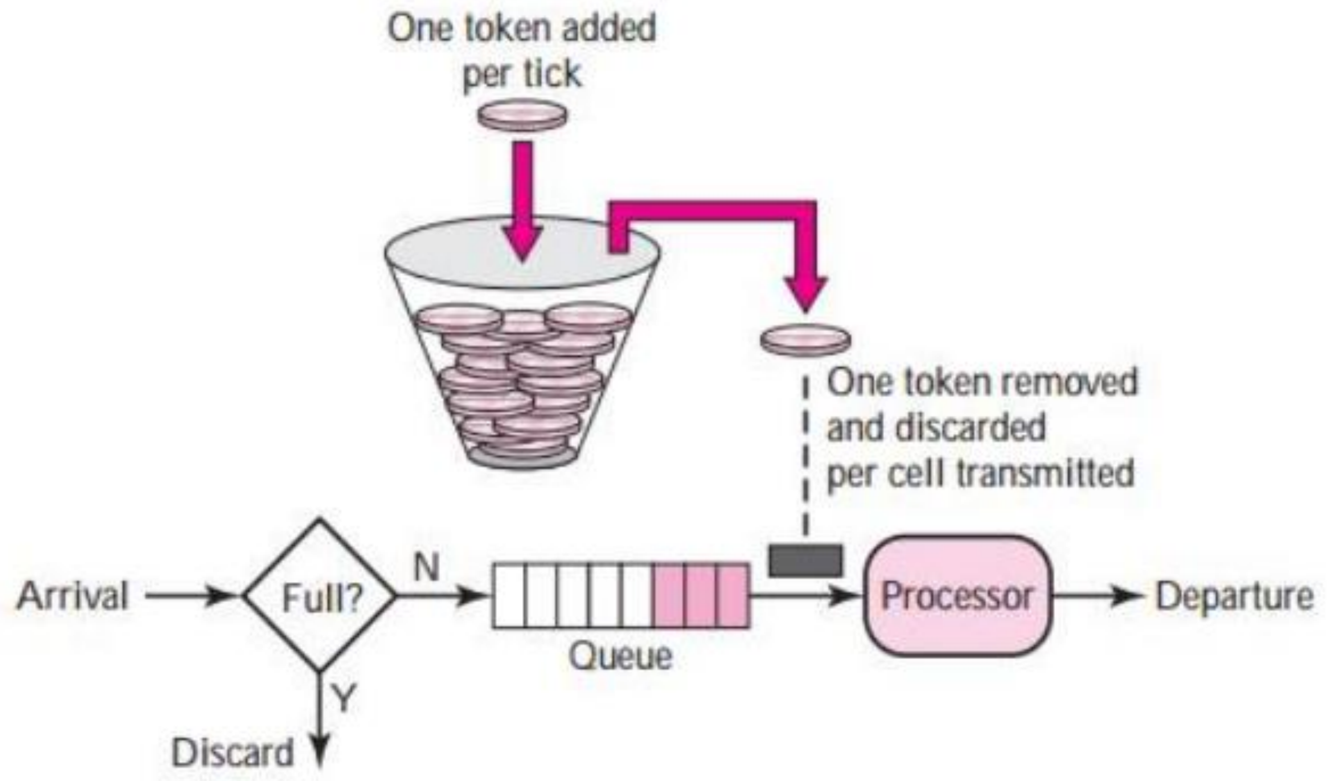
- A bucket holds **tokens**, which are generated at a fixed rate.
- **To send a packet**, the sender must **remove a token**.
- If enough tokens exist, packets can be sent immediately.
- If not enough tokens, packets must **wait** (or are dropped if the system is strict).

❑ **Mechanism:**

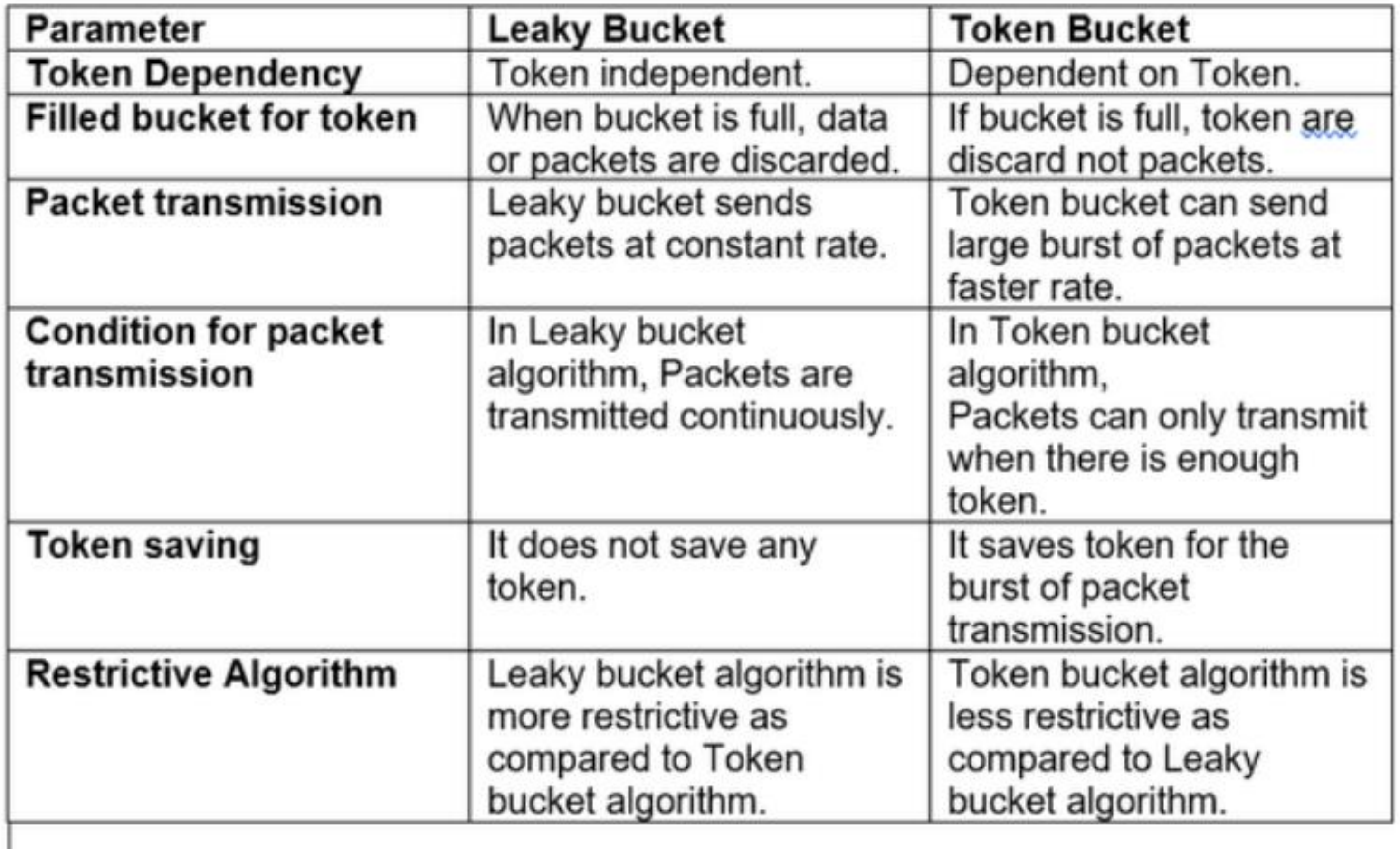
- Tokens are added to the bucket at a rate of r tokens/sec.
- The bucket has a **maximum capacity**, B.
- To transmit a packet of size n, n tokens are needed.
- If n tokens are available, the packet is sent immediately.
- If not, the packet is delayed or discarded.

Traffic Shaping:

Token bucket algorithm:



Traffic Shaping:

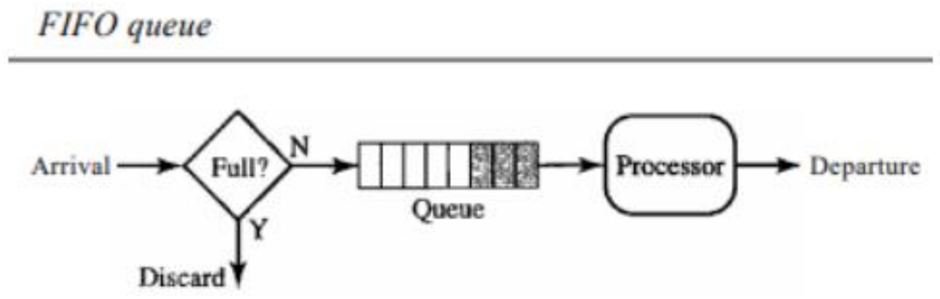


Parameter	Leaky Bucket	Token Bucket
Token Dependency	Token independent.	Dependent on Token.
Filled bucket for token	When bucket is full, data or packets are discarded.	If bucket is full, token are discard not packets.
Packet transmission	Leaky bucket sends packets at constant rate.	Token bucket can send large burst of packets at faster rate.
Condition for packet transmission	In Leaky bucket algorithm, Packets are transmitted continuously.	In Token bucket algorithm, Packets can only transmit when there is enough token.
Token saving	It does not save any token.	It saves token for the burst of packet transmission.
Restrictive Algorithm	Leaky bucket algorithm is more restrictive as compared to Token bucket algorithm.	Token bucket algorithm is less restrictive as compared to Leaky bucket algorithm.

Queuing Techniques for Scheduling:

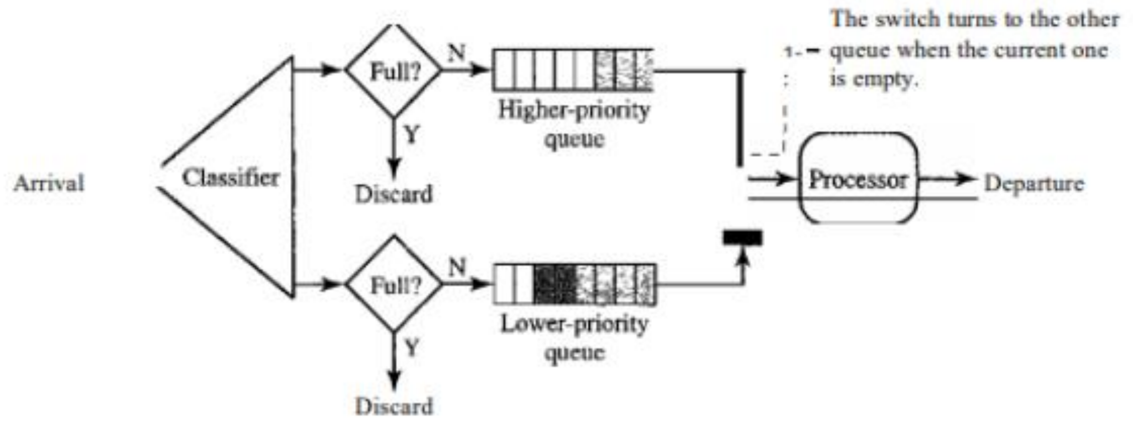
QoS traffic scheduling is a scheduling methodology of network traffic based on QoS (Quality of Service). Packets from different flows arrive at a switch or router for processing. A good scheduling technique treats the different flows in a fair and appropriate manner. Several scheduling techniques are designed to improve the quality of service. Major scheduling techniques are FIFO Queuing, Priority Queuing, and Weight Fair Queuing.

FIFO Queuing: In first- in first-out (FIFO) queuing, packets wait in a buffer (queue) until the node (router or switch) is ready to process them. If the average arrival rate is higher than the average processing rate, the queue will fill up and new packets will be discarded. A FIFO queue is familiar to those who have had to wait for a bus at a bus stop.



Queuing Techniques for Scheduling:

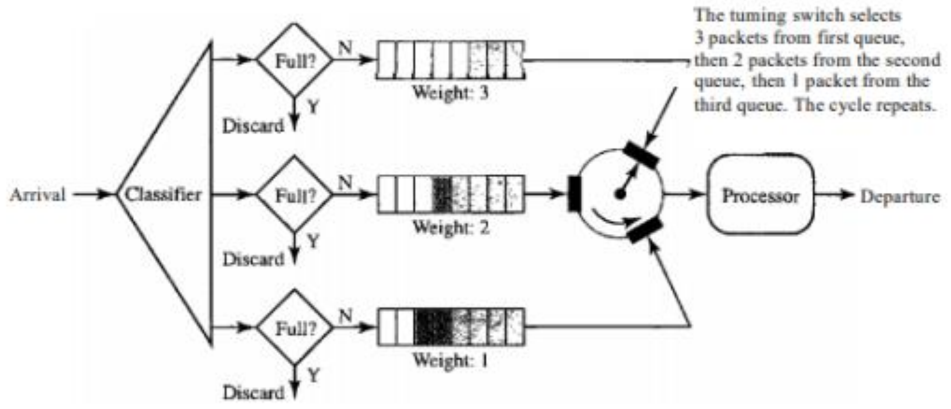
Priority Queuing: In priority queuing, packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving a queue until it is empty.



A priority queue can provide better QoS than the FIFO queue because higher priority traffic, such as multimedia, can reach the destination with less delay. However, there is a potential drawback. If there is a continuous flow in a high-priority queue, the packets in the lower-priority queues will never have a chance to be processed. This is a condition called starvation

Queuing Techniques for Scheduling:

Weighted Fair Queuing: A better scheduling method is weighted fair queuing. In this technique, the packets are still assigned to different classes and admitted to different queues. The queues, however, are weighted based on the priority of the queues; higher priority means a higher weight. The system processes packets in each queue in a round-robin fashion with the number of packets selected from each queue based on the corresponding weight. For example, if the weights are 3, 2, and 1, three packets are processed from the first queue, two from the second queue, and one from the third queue. If the system does not impose priority on the classes, all weights can be equal. In this way, we have fair queuing with priority.

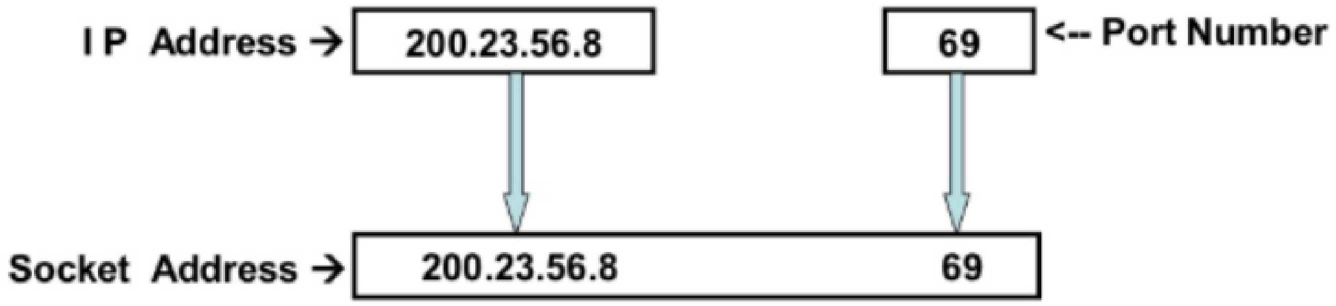

The turning switch selects 3 packets from first queue, then 2 packets from the second queue, then 1 packet from the third queue. The cycle repeats.
Arrival
Classifier
Full?
N
Y
Discard
Weight: 3
Weight: 2
Weight: 1
Processor
Departure

Introduction to Ports and Sockets: (Port Addressing/Socket Addressing)

The IP address and the physical address are necessary for a quantity of data to travel from a source to the destination host. However, arrival at the destination host is not the final objective of data communications on the Internet. A system that sends nothing but data from one computer to another is not complete. Today, computers are devices that can run multiple processes at the same time. The end objective of Internet communication is a process of communicating with another process. For example, computer A can communicate with computer C by using TELNET. At the same time, computer A communicates with computer B by using the File Transfer Protocol (FTP). For these processes to receive data simultaneously, we need a method to label the different processes. In other words, they need addresses. In the TCP/IP architecture, the label assigned to a process is called a port address. A port address in TCP/IP is 16 bits in length. Source and destination addresses are found in the IP packet, belonging to the network layer. A transport layer datagram or segment that uses port numbers is wrapped into an IP packet and transported by it. The network layer uses the IP packet information to transport the packet across the network (routing). Arriving at the destination host, the host's IP stack uses the transport layer information (port number) to pass the information to the application.

Introduction to Ports and Sockets: (Port Addressing/Socket Addressing)

Port Number	Assignment
21	File Transfer Protocol (FTP)
23	TELNET remote login
25	SMTP
80	HTTP
53	DNS



IP address and Port Number is combinedly called Socket Address which identifies the host along with the networking application running in the host. A socket is one endpoint of a two-way communication link between two programs 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. An endpoint is a combination of an IP address and a port number.

Socket Programming:

- **Socket Programming**
- In network communication, applications usually have **two parts**:
- A **client program** (which requests something)
- A **server program** (which responds to the request)
- These programs run on **two different computers** (called end systems), and when they start, they create **processes** that talk to each other using **sockets**.

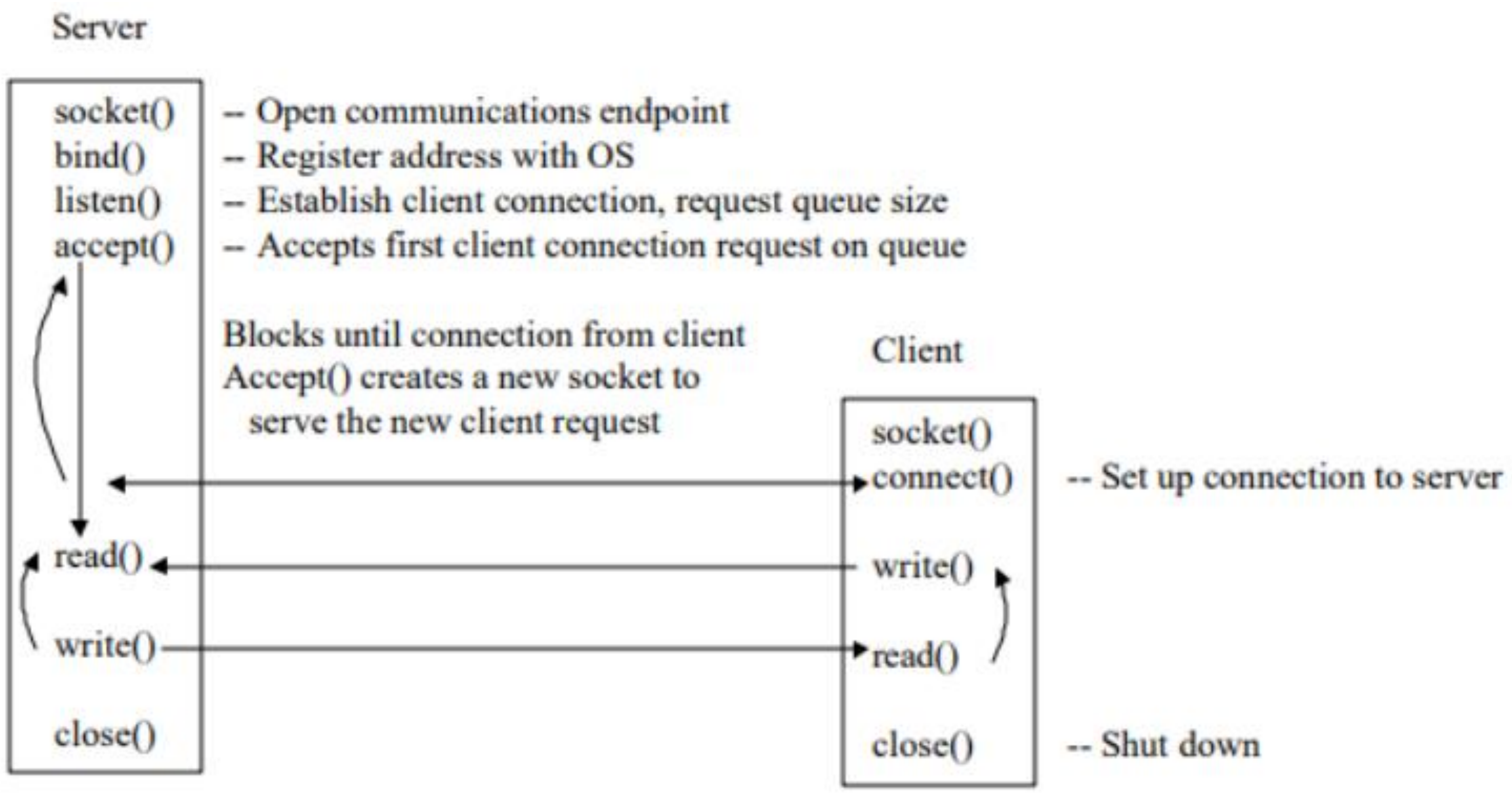
- **What is a Socket?**
- A **socket** is like a **door** between a computer and the network.
- Programs **send and receive data** through this door.
- Think of it as a **connection point** that helps two programs talk.

Socket Programming:

- **Developer's Job in Socket Programming**
- When creating a network application, the main job of a developer is to **write code** for:
- The **client-side program**
- The **server-side program**
- This whole process is called **socket programming**.

- **TCP vs UDP (Choosing the Protocol)**
- Before writing the program, the developer must decide:
- Use **TCP** (reliable, connection-based) or
- Use **UDP** (faster, no connection, less reliable)

Socket Programming:


Server
socket()
bind()
listen()
accept()
– Open communications endpoint
– Register address with OS
– Establish client connection, request queue size
– Accepts first client connection request on queue
Blocks until connection from client
Accept() creates a new socket to
serve the new client request
read()
write()
close()
Client
socket()
connect()
-- Set up connection to server
write()
read()
close()
-- Shut down

Thank You