If There is Maths Involve Verify on Your Own.

CO Shows Chapter No. Chap 3 to Chap 6

CO3 Explain path vector routing. Distinguish between Distance vector and Link state routing.

Path vector routing is a protocol that is helpful for routing between autonomous systems (AS). It is similar to distance vector routing, but instead of advertising the metric of the nodes, a speaker node advertises the path to nodes in its AS and other ASs. For example, if router A1 in AS1 receives a packet for A3, it knows the path is in AS1. If it receives a packet for D1, it knows the packet should go from AS1 to AS2 and then to AS3.

Path vector routing has some advantages over distance vector routing:

- **Loop prevention**: When a router receives a message, it checks if its AS is in the path list. If it is, it will ignore the message to prevent looping.
- **Optimal path**: Path vector routing allows each AS to choose a path that fits its organization, which may involve criteria such as security, safety, and reliability in addition to the number of hops.

Distance vector and link state routing are both intradomain routing protocols. They can be used inside an autonomous system, but not between them. **They are not suitable for interdomain routing primarily because of scalability**. Distance vector routing can become unstable in large networks. Link state routing requires a large number of resources to calculate routing tables and creates heavy traffic because of flooding.

Here is a table summarizing the differences between distance vector and link state routing:

Feature	Distance Vector Routing	Link State Routing
Algorithm	Each node shares its routing table with its neighbors and updates its table based on information from its neighbors.	Each node has a map of the entire network and uses Dijkstra's algorithm to calculate the shortest paths to all other nodes.

Information	Each node only knows the distance to its neighbors.	Each node has complete knowledge of the network topology.
Message Updates	Periodic and triggered updates	Flooding of link-state packets when there are changes in the network
Convergence	Can be slow and prone to routing loops	Fast and avoids routing loops
Scalability	Not suitable for large networks	More scalable but requires more resources
Example	RIP	OSPF

CO3 Suppose a network with IP Address 200.1.2.0 is divided into 4 subnets, find following things for each subnet.

i) IP Address of the subnet
ii) Total number of IP Addresses
iii) Total number of hosts that can be configured
iv) Range of IP Addresses

The network address **200.1.2.0** belongs to **Class C** because the first byte is between 192 and 223. Dividing a network with this address into four subnets requires borrowing two bits from the host portion of the address, resulting in a subnet mask of **/26** (the default mask for Class C is /24).

Here is the breakdown of the IP addresses for each subnet:

- **Subnet 1:**
  - **IP Address:** 200.1.2.0/26
  - **Total Number of IP Addresses:** 64 (232-26 = 26)
  - **Total Number of Hosts:** 62 (64 - 2; subtracting the network and broadcast addresses)
  - **Range of IP Addresses:** 200.1.2.1 - 200.1.2.62

- **Subnet 2:**
    - **IP Address:** 200.1.2.64/26
    - **Total Number of IP Addresses:** 64
    - **Total Number of Hosts:** 62
    - **Range of IP Addresses:** 200.1.2.65 - 200.1.2.126
- **Subnet 3:**
    - **IP Address:** 200.1.2.128/26
    - **Total Number of IP Addresses:** 64
    - **Total Number of Hosts:** 62
    - **Range of IP Addresses:** 200.1.2.129 - 200.1.2.190
- **Subnet 4:**
    - **IP Address:** 200.1.2.192/26
    - **Total Number of IP Addresses:** 64
    - **Total Number of Hosts:** 62
    - **Range of IP Addresses:** 200.1.2.193 - 200.1.2.254

CO4 Justify why TCP is reliable.

TCP is a reliable transport layer protocol because it uses a combination of mechanisms to ensure that data is delivered to the destination application without errors, duplication, or loss, and in the correct order. These mechanisms include:

- **Sequence Numbers**: TCP assigns a sequence number to each byte in the data stream, allowing the receiver to detect missing or duplicate segments and request retransmission if necessary.
- **Checksums**: TCP uses a checksum to detect data corruption in each segment. The receiver discards corrupted segments and the sender will retransmit the segment.
- **Acknowledgments (ACKs):** TCP uses acknowledgments to confirm receipt of segments. The receiver sends an ACK to the sender after

successfully receiving a segment. TCP acknowledgments are cumulative, acknowledging all bytes up to a given sequence number. This mechanism helps to ensure that the sender knows which segments have been successfully received and which need to be retransmitted. Duplicate ACKs help the sender detect missing segments and trigger faster retransmission (fast retransmit).

- **Timers**: TCP uses timers to detect lost segments. If an ACK is not received within a certain time (timeout interval), the sender assumes that the segment has been lost and retransmits it. TCP dynamically adjusts the timeout interval based on estimated round-trip time (RTT) to improve efficiency.
- **Retransmissions**: TCP retransmits segments that are either lost or corrupted.
- **Flow control**: TCP implements flow control to prevent the sender from overwhelming the receiver with data. The receiver advertises its window size, indicating the amount of data it can buffer, and the sender adjusts its sending rate accordingly. This mechanism helps to prevent data loss due to buffer overflow at the receiver.

In contrast to TCP, UDP does not implement these mechanisms and is therefore considered an unreliable protocol.

The reliability features of TCP make it a suitable choice for applications that require a guaranteed and ordered delivery of data.

CO4 For each of the following protocols, determine whether TCP or UDP is used as the transport layer protocol and explain the reason for your choice.

a. Telnet

b. VoIP

c. FTP

d. DNS

e. HTTP

- **a. Telnet: TCP** Telnet provides remote terminal access. A user at one host can access a terminal on a second host as if they were directly connected. This application requires that data is transmitted reliably and in the correct order to ensure that commands and responses are executed correctly. TCP is therefore a suitable choice.
- **b. VoIP: UDP VoIP (Voice over IP)** is a real-time application that is sensitive to delays. **UDP is therefore a more suitable choice because it does not implement congestion control mechanisms or retransmissions**, which can introduce delays. While **some data loss is tolerable**, excessive delays can severely degrade the quality of the voice communication. **Some VoIP applications use TCP as a backup if UDP communication fails.**
- **c. FTP: TCP FTP (File Transfer Protocol)** is used to transfer files between a client and a server. Because it is essential that the files are transferred without errors, **TCP is the preferred choice because it provides reliable data transfer**. FTP uses two parallel TCP connections: a control connection for control information, such as user identification, passwords, and commands, and a data connection to transfer the file.
- **d. DNS: UDP DNS (Domain Name System)** is used to resolve domain names to IP addresses. **It typically uses UDP because of its speed and low overhead. If a DNS query fails, the application can simply resend the query**, which is more efficient than waiting for TCP retransmission.
- **e. HTTP: TCP HTTP (Hypertext Transfer Protocol)** is used to transfer web pages and other content between a web server and a web browser. **It uses TCP because it requires reliable data transfer to ensure that the web pages are displayed correctly**.

CO5 Illustrate in detail DNS.

The Domain Name System (DNS) is a critical Internet application that provides a directory service for the Internet by translating hostnames to IP addresses.

DNS is essential to the functioning of many other Internet applications, such as the **World Wide Web**, **email**, and **file transfer** because these applications rely on DNS to translate user-supplied hostnames to IP addresses. **TCP and UDP rely on IP, which in turn relies on DNS**.

How DNS Works

Let's examine how DNS works in more detail:

1. DNS Query: When a user wants to access a website, the user's computer sends a DNS query to its **local DNS server**. For example, if a user wants to access www.example.com, the user's computer will send a query to the local DNS server asking for the IP address of www.example.com. The query message includes the hostname that needs to be translated. The local DNS server acts as a proxy, forwarding the query into the DNS server hierarchy.

2. Root DNS Servers: The local DNS server first contacts one of the **root DNS servers**. The root servers are a network of replicated servers that are responsible for the top-level domains, such as .com, .org, and .net.

3. TLD DNS Servers: The root server responds to the local DNS server with the IP address of a **top-level domain (TLD) DNS server** that is responsible for the domain in question. For example, if the user is trying to access www.example.com, the root server would respond with the IP address of the .com TLD server.

4. Authoritative DNS Server: The local DNS server then contacts the TLD server. The TLD server responds with the IP address of the **authoritative DNS server** for the domain name in question. The authoritative DNS server is the DNS server that is responsible for managing the DNS records for a particular domain.

5. DNS Response: The local DNS server sends a query to the authoritative DNS server. The authoritative DNS server responds with the **IP address** for the hostname in question. The local DNS server caches this mapping and sends it back to the user's computer.

6. Accessing the Website: Once the user's computer has the IP address of the website, it can establish a TCP connection with the webserver and request the desired content.

DNS Records

DNS servers store **resource records (RRs)** in their databases to provide hostname-to-IP address mappings. A resource record is a four-tuple containing:

(Name, Value, Type, TTL)

where TTL is the time to live of the resource record.

DNS Caching

DNS caching is an important feature that improves DNS performance and reduces the number of DNS messages that need to be sent across the Internet. **When a DNS server receives a response to a query, it caches the information so that it can respond to future queries for the same hostname without having to contact other DNS servers.**

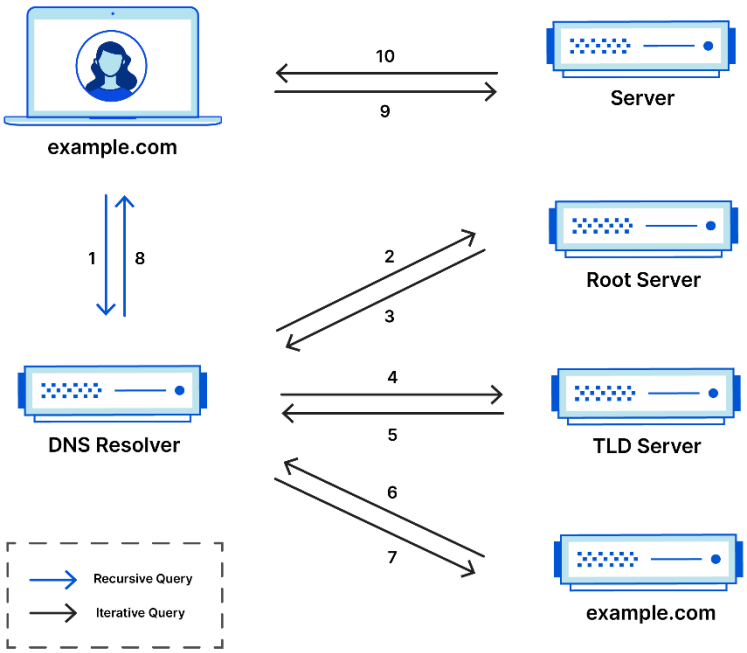
Types of DNS Queries

There are two types of DNS queries: **recursive** and **iterative**.

Recursive queries require the DNS server to provide the final answer to the query. If the server does not have the information, it will query other servers until it finds the answer.

Iterative queries, on the other hand, do not require the DNS server to provide the final answer. If the server does not have the information, it will provide the client with the address of another DNS server that may have the information. The client will then query that server, and so on.

Complete DNS Lookup and Webpage Query



CO6 Define following terms with example in terms of security

i) Integrity
ii) Confidentiality
iii) Availability
iv) Privacy

i) Integrity

Integrity ensures that data has not been altered or tampered with in an unauthorized manner. It guarantees that the information received is the same as the information that was sent, without any modifications, deletions, or additions.

Example:

- Imagine Alice sends a financial transaction message to Bob to transfer $100. An attacker, Trudy, could intercept and modify this message, changing the amount to $1,000 before sending it to Bob. **If the integrity of the message is not maintained, Bob could end up losing $900**. To ensure message integrity, cryptographic techniques like **hash functions and digital signatures** can be used, as discussed in our previous conversation about security.

ii) Confidentiality

Confidentiality ensures that data is accessible only to authorized parties and remains secret from unauthorized individuals. This involves protecting information from unauthorized disclosure, access, or eavesdropping during transmission or storage.

Example:

- **When Alice sends a confidential email to Bob, she wants to ensure that only Bob can read the message**. If an attacker, Trudy, intercepts the email, confidentiality would be breached if she could read the contents. Encryption, like the **Secure Sockets Layer (SSL)**, can be used to maintain confidentiality by transforming the message into an unreadable format during transmission.

iii) Availability

Availability ensures that data and systems are accessible to authorized users when needed. It involves protecting systems from disruptions, outages, or attacks that could prevent legitimate users from accessing the resources they require.

Example:

- If a web server hosting a critical application is attacked and becomes unavailable, users who depend on that application will experience a loss of availability. **Denial-of-service (DoS) attacks** are a common threat to availability, as they aim to flood a system with traffic, making it unresponsive to legitimate requests.

iv) Privacy

Privacy focuses on protecting personal information from unauthorized access, use, or disclosure. It encompasses an individual's right to control the collection, use, and sharing of their data.

Example:

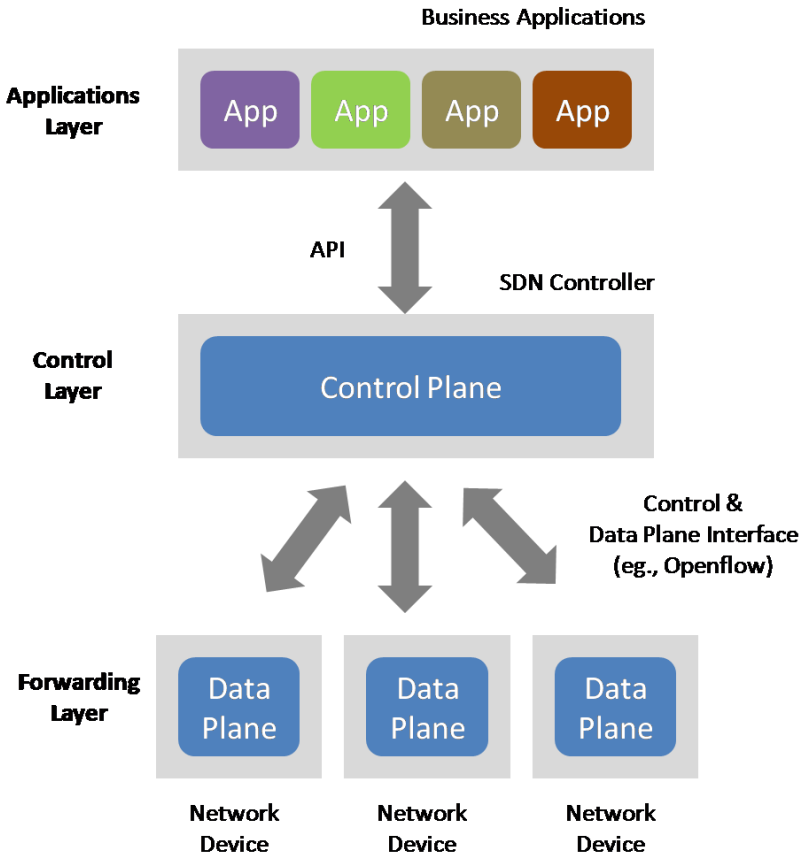
- **A user's browsing history, location data, or financial information are considered private**. Organizations that collect and store this data have a responsibility to implement appropriate security measures to protect user privacy and comply with relevant regulations. This can include using **encryption, access controls, and anonymization techniques**. describes how an anonymizing proxy can be used to protect user privacy when browsing a website.

CO6 Explain SDN Architecture and communication.

Software-defined networking (SDN) is a new approach to network management that separates the control plane from the data plane. This allows network administrators to have more control over the network and to implement new features more easily.

SDN Architecture

In traditional networks, each network device, such as a router or switch, has its own control plane that is responsible for making decisions about how to forward traffic. This can make it difficult to manage the network because changes have to be made on each device individually. In SDN, the control plane is moved to a central controller, which is typically a PC running Linux. **The SDN controller communicates with the network devices using a protocol such as OpenFlow.** The data plane in the SDN network device consists of hardware, such as a classification engine, which examines packets to decide how to forward them. The data plane does not contain control plane software, so the functionality inside a network element is much less than in traditional networks.



SDN Communication

The SDN controller can communicate with the network devices using several different protocols. One common protocol is **OpenFlow**. OpenFlow is a southbound protocol, which means that it is used for communication between the controller and the network devices. SDN also uses northbound protocols for communication between the controller and

network management applications. **SDN follows the paradigm pioneered by SNMP (Simple Network Management Protocol), sending management traffic over the same network that carries data traffic**.

An SDN system needs two communication protocols:

- **Controller-to-element communication**: This communication allows the controller to configure and control the network devices in the SDN domain. OpenFlow, a southbound protocol that specifies the format and meaning of messages, is one example.
- **Controller-to-controller communication**: The management software on the controllers needs to coordinate configurations to ensure that the configurations on all network elements are consistent.

CO3 How many bits are allocated for Network ID and Host ID in 23.192.157.234 address?

The provided IP address (23.192.157.234) falls under **Class A** in the classful IP addressing scheme. In Class A, **8 bits** are allocated for the Network ID (netid) and **24 bits** are allocated for the Host ID (hostid).

CO3 You have a network that needs 29 subnets while maximizing the number of host addresses available on each subnet. How many bits must you borrow from the host field to provide the correct subnet mask?

To determine the number of bits to borrow from the host field for 29 subnets, you need to find the smallest power of 2 that is greater than or equal to 29.

- 25 = 32, which is greater than 29.

Therefore, you need to borrow **5 bits** from the host field to create enough subnets. This will give you 32 possible subnets (25) while still allowing for a reasonable number of host addresses per subnet.

CO3 The network address of 172.16.0.0/19 provides how many subnets and hosts?

The network address 172.16.0.0/19 allows for **8,192 subnets** and **8,190 hosts per subnet**. Here's the breakdown:

- The /19 prefix length indicates that **19 bits are used for the network portion** of the address, leaving **13 bits for the host portion**.
- With 13 bits available for hosts, each subnet can accommodate 213 = **8,192 host addresses**. However, two of these addresses are reserved: the network address and the broadcast address, leaving **8,190 usable host addresses per subnet**.
- Since you borrowed 5 bits from the host portion to create subnets, you have 25 = **32 possible subnets**.

CO3 For the following IP Addresses 1.2.3.4 , 172.15.20.60 Identify the Class, Network IP Address, broadcast address.

Here is the class, network IP address, and broadcast address for each IP address:

- **1.2.3.4**
  - **Class:** A
  - **Network IP Address:** 1.0.0.0
  - **Broadcast Address:** 1.255.255.255
- **172.15.20.60**
  - **Class:** B
  - **Network IP Address:** 172.15.0.0
  - **Broadcast Address:** 172.15.255.255

CO3 Represent the following IPv4 addresses from binary notation to dotted-decimal notation

i. 10000001 00001011 00001011 11101111

ii. 11000001 10000011 00011011 11111111

To convert this binary representation to dotted-decimal notation, divide the 32-bit address into four 8-bit octets and convert each octet to its decimal equivalent:

- 10000001 = 129
- 00001011 = 11
- 00001011 = 11
- 11101111 = 239

Therefore, the dotted-decimal notation for this IP address is **129.11.11.239**.

- **ii. 11000001 10000011 00011011 11111111**

Similarly, for this address:

- 11000001 = 193
- 10000011 = 131
- 00011011 = 27
- 11111111 = 255

Therefore, the dotted-decimal notation is **193.131.27.255**.

CO3 A host with IP Address 10.100.100.100 wants to use loopback testing. What will be Source IP Address Destination IP Address

- **Source IP Address**: 10.100.100.100
- **Destination IP Address**: 127.0.0.1

During loopback testing, a programmer runs two applications on a single computer and instructs them to use a loopback address for communication. This allows testing of program logic without needing two computers or sending packets across a network. **The loopback address never appears in a packet traveling across a network.**

CO4 For each of the following applications, determine whether TCP or UDP is used as the transport layer protocol and explain the reason for your choice.

a. Whatsapp text messaging

b. Online games

c. Web browsing

d. Live streaming

e. Email

a. Whatsapp Text Messaging

- **Protocol: TCP**
- **Reason:** Whatsapp text messaging requires **reliable delivery** of messages to ensure that all messages are received by the recipient. TCP provides a connection-oriented, reliable data transfer service that guarantees the delivery of messages without errors. Although speed is important, reliability is the priority to ensure that messages are not lost.

b. Online Games

- **Protocol: UDP**
- **Reason:** Online games are highly sensitive to **latency**, requiring quick responses to user actions. UDP, being a connectionless protocol, offers **lower overhead** and faster transmission speeds compared to TCP. While some packet loss can be tolerated in many online games, minimizing delay is crucial for a smooth and responsive gaming experience.

c. Web Browsing

- **Protocol: TCP**
- **Reason:** Web browsing relies on the **accurate and complete retrieval of web pages**, which often consist of multiple objects (text, images, etc.). TCP's reliable data transfer service ensures that all data is received correctly and in the proper order. This reliability is essential for rendering web pages correctly and providing a consistent browsing experience.

d. Live Streaming

- **Protocol: UDP (commonly) or TCP**

- **Reason:** Live streaming prioritizes **real-time delivery** over absolute reliability. UDP's lower overhead and faster transmission speed make it suitable for handling the continuous flow of audio and video data with minimal delay. However, some live streaming services may opt for TCP, especially for on-demand content or when operating within networks with low packet loss rates. The choice between UDP and TCP for live streaming involves a trade-off between speed and reliability, and factors like network conditions and content type influence the decision.

e. Email

- **Protocol: TCP**
- **Reason:** Email requires **reliable delivery** to ensure that messages arrive at their destination without errors or loss. TCP's connection-oriented, reliable data transfer service is well-suited for this purpose. The potential consequences of lost or corrupted emails necessitate a protocol that prioritizes reliability over speed in this application.

CO3 What is the DHCP process for client machine?

The **Dynamic Host Configuration Protocol (DHCP)** is a **client-server protocol** that enables a host to obtain network configuration information, including an **IP address**, automatically. This makes it a key component in plug-and-play networking. Here's a breakdown of the DHCP process for a client machine:

1. **DHCP Server Discovery:** When a client machine boots up, it needs to find a DHCP server on the network. The client broadcasts a **DHCP discover message** within a UDP packet to port 67. This packet is encapsulated in an IP datagram with a source IP address of 0.0.0.0 and a broadcast destination IP address of 255.255.255.255. The data link layer then broadcasts the frame containing this datagram to all nodes on the subnet.
2. **DHCP Server Offer(s):** DHCP servers on the subnet that receive the DHCP discover message respond with a **DHCP offer message**. This message is also broadcast to all nodes on the subnet using the IP broadcast address. Each offer message includes the transaction ID

from the discover message, a proposed IP address for the client, the network mask, and a lease time for the IP address.

3. **DHCP Request:** The client chooses from the available server offers and sends a **DHCP request message** to the selected server. The request message echoes back the configuration parameters from the chosen offer.
4. **DHCP ACK:** The server responds to the client's request with a **DHCP ACK message**, confirming the requested parameters.

Once the client receives the DHCP ACK, it can use the assigned IP address for the duration of the lease. DHCP also includes a mechanism for lease renewal, allowing the client to continue using the IP address beyond the initial lease time.

The use of DHCP simplifies network administration by automating the process of assigning IP addresses and providing essential network configuration information to client machines.

CO4 Compare TCP with UDP protocol.

TCP (Transmission Control Protocol) and **UDP (User Datagram Protocol)** are two fundamental transport layer protocols in the Internet protocol suite, each providing a different set of services to applications. Here is a comparison of the two:

TCP

- **Connection-oriented:** TCP establishes a **virtual connection** between the sending and receiving hosts before data transfer begins, similar to a phone call. This connection involves a **three-way handshake** process to synchronize the connection state between the two endpoints.
- **Reliable:** TCP provides **reliable data transfer**, guaranteeing that data is delivered to the destination in the correct order and without errors. This is achieved through mechanisms like sequence numbers, acknowledgments, checksums, and retransmissions. If a segment is lost or corrupted, TCP will resend it until it is successfully received.

- **Stream-oriented:** TCP treats data as a continuous stream of bytes, without preserving message boundaries. This is suitable for applications that require the transmission of large amounts of data as a single unit, such as file transfer.
- **Flow control:** TCP implements flow control to prevent the sender from overwhelming the receiver with data. This ensures that the receiver's buffer does not overflow and that data is transmitted at a rate the receiver can handle.
- **Congestion control:** TCP uses congestion control mechanisms to regulate the rate at which data is sent into the network, aiming to prevent network congestion. This helps ensure fair bandwidth allocation among different TCP connections.

UDP

- **Connectionless:** UDP is a connectionless protocol, meaning that there is no connection establishment phase before data transfer. Datagrams are sent independently and may arrive out of order, be lost, or duplicated.
- **Unreliable:** UDP does not guarantee reliable data delivery. There are no mechanisms for retransmission or error correction. Applications using UDP must handle any required reliability themselves.
- **Message-oriented:** UDP preserves message boundaries, treating each datagram as a separate unit. This is suitable for applications that send short, self-contained messages, such as DNS queries or multimedia streaming.
- **No flow control or congestion control:** UDP does not implement flow control or congestion control. This can lead to higher transmission speeds but also to potential packet loss and network congestion.
- **Low overhead:** UDP has a smaller header size compared to TCP, resulting in lower overhead and faster transmission.

Applications: As we have discussed, the choice between TCP and UDP depends on the specific requirements of the application.

- **TCP** is typically used for applications that prioritize reliability and data integrity, such as web browsing, email, and file transfer. These applications require that all data is received correctly and in the proper order.
- **UDP** is often chosen for applications where speed and low latency are critical, even at the expense of some reliability, such as online gaming, live streaming, and VoIP. These applications can tolerate occasional packet loss, and minimizing delay is more important than ensuring every packet is delivered.

CO4 What are the types of sockets? Explain various socket primitives used in the connection-oriented client-server approach.

Socket Types

The two basic communication paradigms used in the Internet map to two socket types:

- **Stream Sockets:** These sockets use the **SOCK_STREAM** type and are designed for connection-oriented communication, similar to a telephone call. Before communication can occur, two applications must request that a connection be created between them. This type of socket is typically used with the **TCP (Transmission Control Protocol)** transport protocol. TCP is a connection-oriented protocol that offers a reliable byte stream between two end systems. TCP handles segmenting long messages into shorter segments and provides congestion control, ensuring that a sender adjusts its transmission rate when the network is congested.
- **Datagram Sockets:** These sockets use the **SOCK_DGRAM** type and are designed for connectionless communication. They are typically used with the **UDP (User Datagram Protocol)** transport protocol. UDP is a connectionless, no-frills protocol that does not offer reliability, flow control, or congestion control. Each UDP datagram is treated as an independent packet, and their delivery is not guaranteed.

Socket Primitives in Connection-Oriented Client-Server Approach

The **socket API** provides a set of functions for creating and using sockets. In a connection-oriented client-server model, like the one typically used with TCP, several socket primitives are employed on both the client and server sides.

Client Side

A client, in a connection-oriented model, must actively initiate contact with a server. Here's a typical sequence of socket calls used by a client:

- **socket():** This function is used to create a socket. The client specifies the address family (**AF_INET** for IPv4) and the socket type (**SOCK_STREAM** for a TCP socket).
- **connect():** The client uses this function to establish a connection with the server. It requires the server's IP address and port number. This function initiates the three-way TCP handshake process to establish the connection.
- **send():** After establishing the connection, the client can use this function to send data to the server.
- **recv():** This function allows the client to receive data from the server.
- **close():** Once the communication is complete, the client closes the connection using this function.

Server Side

A server passively waits for incoming connection requests from clients. Here's a typical sequence of socket calls used by a server in a connection-oriented model:

- **socket():** Similar to the client, the server first creates a socket, specifying the address family and socket type.
- **bind():** The server uses this function to associate a specific port number with the socket. This is the port on which the server will listen for incoming client connections.
- **listen():** The server then uses this function to put the socket in a passive listening mode, making it ready to accept incoming connection requests from clients.

- **accept():** This function blocks the server process until a client connection request arrives. It creates a new socket dedicated to the connected client.
- **recv():** The server can use this function to receive data from the client.
- **send():** This function allows the server to send data to the client.
- **close():** After completing the data exchange, the server closes the connection.

The connection-oriented client-server paradigm, facilitated by socket primitives, provides a structured and reliable way for applications to communicate over a network, ensuring the ordered and error-free delivery of data.

CO3 Compare Circuit Switching and Packet Switching techniques.

Circuit Switching

Circuit switching establishes a **dedicated communication path** between two devices before any data is transmitted. This dedicated path, called a circuit, reserves a **fixed bandwidth** for the duration of the communication session. Circuit switching is analogous to a traditional telephone call where a dedicated line is established between two parties before the conversation begins.

Key characteristics of circuit switching:

- **Connection-oriented:** A connection must be established before data transfer can begin. This involves a setup phase where a path is determined and resources are allocated.
- **Dedicated resources:** A fixed bandwidth is reserved for the entire duration of the connection, even during periods of silence or inactivity. This can lead to inefficient use of network resources, particularly for bursty traffic.

- **Guaranteed performance:** Once the circuit is established, data transmission occurs at a constant rate with minimal delay and jitter, making it suitable for real-time applications like voice calls.
- **Suitable for analog transmission:** Circuit switching was initially designed for analog telephone networks, where voice signals are transmitted at a constant rate.

Example: Traditional telephone networks are a classic example of circuit-switched networks.

Packet Switching

Packet switching breaks data into small units called **packets**, each containing a header with addressing information and a payload with the actual data. These packets are transmitted independently across the network and may follow different paths to reach their destination.

Key characteristics of packet switching:

- **Connectionless:** No dedicated connection is established before data transfer. Packets are sent on demand, and network resources are utilized only when packets are being transmitted.
- **Shared resources:** Packets from different sources compete for network resources like bandwidth and buffer space. This allows for more efficient use of resources compared to circuit switching, especially for bursty traffic.
- **Variable performance:** Packet transmission is subject to variable delays and potential packet loss due to congestion or network failures. This makes it less suitable for real-time applications requiring strict delay bounds.
- **Suitable for digital transmission:** Packet switching is well-suited for digital data transmission, where data can be easily divided into packets.

Types of packet switching:

- **Datagram networks:** Each packet is treated independently, with no prior connection establishment. Routers make forwarding decisions based on the destination address in each packet header. The Internet is a prime example of a datagram network.

- **Virtual-circuit networks:** A logical connection, called a virtual circuit, is established between the source and destination before data transfer begins. However, unlike circuit switching, this virtual circuit does not reserve dedicated resources. Packets belonging to a virtual circuit follow the same path, ensuring in-order delivery. Frame Relay and ATM are examples of virtual-circuit networks.

Example: The Internet is the most prominent example of a packet-switched network.

Comparison

Feature	**Circuit Switching**	**Packet Switching**
Connection	Connection-oriented	Connectionless (Datagram) or Virtual Circuit
Resource Allocation	Dedicated	Shared on demand
Data Transmission	Constant rate	Variable rate, potential for delay and loss
Efficiency	Lower for bursty traffic	Higher for bursty traffic
Suitability	Analog, real-time applications	Digital, applications tolerant of delay variations
Examples	Telephone network	Internet, Frame Relay, ATM

CO3 Explain classful addressing scheme of IPv4.

In the early days of **IPv4**, a system known as **classful addressing** was used to organize IP addresses. This scheme divided the 32-bit IPv4 address space into **five distinct classes**: A, B, C, D, and E. Each class served a different purpose and had a specific structure for its IP addresses. The class of an address could be easily determined by examining the first byte of the address in dotted-decimal notation or by looking at the first few bits in binary notation.

- **Classes A, B, and C** were primarily designed for **unicast communication**, where data is sent from one host to another specific host.
- **Class D** addresses were reserved for **multicast communication**, allowing data to be sent to a group of hosts.
- **Class E** addresses were designated as **reserved** for future use and were not allocated to hosts or networks.

Classful addressing suffered from several limitations. One of the main problems was **address depletion**. The fixed block sizes for each class led to inefficient allocation of IP addresses. For instance, Class A addresses, intended for large organizations, were largely wasted because few organizations required such a large number of addresses. Similarly, class C addresses were too small for many mid-sized organizations.

- The need for more flexible address allocation schemes, coupled with the rapid growth of the Internet, led to the development of **classless addressing** (also known as CIDR), which replaced the rigid class-based structure with a more flexible approach to address assignment.

Here's a breakdown of the classful addressing scheme:

- **Class A:** The first bit of a Class A address is always 0. The network ID is represented by the first 8 bits, and the host ID is represented by the remaining 24 bits. This allowed for a maximum of 128 networks (2^7) and 16,777,216 hosts (2^24) per network. A significant portion of the address space in Class A was wasted due to the large block size allocated to each network.
- **Class B:** The first two bits of a Class B address are 10. The network ID is represented by the first 16 bits, and the host ID is represented by the remaining 16 bits. This allowed for 16,384 networks (2^14) and 65,536 hosts (2^16) per network.
- **Class C:** The first three bits of a Class C address are 110. The network ID is represented by the first 24 bits, and the host ID is represented by the remaining 8 bits. This allowed for 2,097,152 networks (2^21) and 256 hosts (2^8) per network. The small size of Class C blocks was insufficient for many organizations.

- **Class D:** The first four bits of a Class D address are 1110. These addresses are used for multicasting, which allows data to be sent to a group of hosts. The entire 32-bit address is used to represent the multicast group address.
- **Class E:** The first four bits of a Class E address are 1111. These addresses were reserved for future use and are not assigned to hosts or networks.

Classful addressing was a significant step in the development of the Internet, but its limitations in address allocation and its inability to accommodate the growing demands of the Internet eventually led to its replacement by **classless addressing**.

CO3 Compare the TCP header and the UDP header. List the fields in the TCP header that are not part of the UDP header. Give the reason for each missing field.

TCP and **UDP** are two transport layer protocols that provide different services to applications. TCP is a connection-oriented protocol that provides reliable data transfer, while UDP is a connectionless protocol that does not guarantee delivery. The headers of TCP and UDP segments reflect these differences in functionality.

Here's a comparison of the fields present in **TCP headers** but absent in **UDP headers**:

- **Sequence Number:** TCP uses a 32-bit sequence number field to keep track of the order of bytes transmitted in a connection. This is crucial for TCP's reliable data transfer service, as it allows the receiver to reassemble the data stream in the correct order, even if segments arrive out of order or are lost. UDP does not guarantee in-order delivery and therefore does not require sequence numbers.
- **Acknowledgment Number:** The 32-bit acknowledgment number field in TCP headers is used to confirm the receipt of segments. When a receiver successfully receives a segment, it sends an acknowledgment back to the sender, indicating the next byte it expects to receive. This acknowledgment mechanism is essential for

reliable data transfer in TCP. UDP, being an unreliable protocol, does not utilize acknowledgments.

- **Header Length:** The 4-bit header length field in TCP specifies the length of the TCP header in 32-bit words. This is necessary because TCP headers can have variable lengths due to the presence of optional fields. UDP headers have a fixed size and therefore do not require a header length field.
- **Receive Window:** TCP employs flow control to prevent the sender from overwhelming the receiver with data. The 16-bit receive window field is used to communicate the amount of buffer space available at the receiver. This mechanism ensures that the sender only transmits data that the receiver can handle. UDP does not provide flow control, relying instead on the application to manage the rate of data transmission.
- **Urgent Data Pointer:** The 16-bit urgent data pointer field in TCP is used to indicate data that requires immediate processing by the receiver. This field is only valid if the URG flag in the TCP header is set. UDP does not support the concept of urgent data.
- **Flags:** The flag field in the TCP header contains 6 bits that control various aspects of the TCP connection, including connection establishment, termination, and data flow. The key flags are SYN (synchronization), ACK (acknowledgment), FIN (finish), RST (reset), PSH (push), and URG (urgent). UDP does not have any flags, as it does not manage the connection state.
- **Options:** TCP utilizes options to negotiate various parameters of the connection, such as the maximum segment size (MSS) and window scaling factors. The options field is variable in length, allowing for flexibility in adding new options in the future. UDP does not use options and relies on a fixed header format.

CO6 What is Data Centric Networks (DCN). List all the types data centric networks and give at least two examples of each DCN.

Data-centric networking (DCN) is a new networking paradigm that shifts the focus from location-based addressing to content-based addressing. In

traditional networks like the internet, communication is based on the IP addresses of the sender and receiver. DCN, on the other hand, focuses on the data itself, allowing users to request content by name or by attributes rather than by its location on a specific server. This approach offers several advantages, including:

- **Improved Content Delivery**: By decoupling content from specific locations, DCN enables efficient content distribution and retrieval from multiple sources. This can enhance performance, reduce congestion, and improve scalability.
- **Simplified Content Management**: DCN makes it easier to manage, update, and distribute content across a network. Users can access the latest versions of data without worrying about its physical location.
- **Enhanced Security and Privacy**: DCN architectures can incorporate fine-grained access control mechanisms based on content attributes, improving security and privacy for sensitive information.

DCN architectures can be classified into several types based on their approach to content addressing and distribution:

1. Content Delivery Networks (CDNs)

CDNs are geographically distributed networks of servers that cache and deliver content closer to end-users. This reduces latency and improves content delivery performance. CDNs are widely used to deliver web content, streaming media, and software updates.

- **Examples**: Akamai, Cloudflare, Fastly

2. Peer-to-Peer (P2P) Networks

P2P networks enable direct communication and content sharing between devices without relying on central servers. Participants in a P2P network act as both clients and servers, contributing to content distribution. P2P is commonly used for file sharing, video streaming, and distributed computing.

- **Examples**: BitTorrent, eMule, Gnutella

3. Named Data Networking (NDN)

NDN replaces IP addresses with content names, allowing users to request data by name rather than by location. NDN employs a hierarchical naming scheme and relies on in-network caching to improve content delivery efficiency.

- **Examples**: Content-Centric Networking Project, NDN Testbed

4. Information-Centric Networking (ICN)

ICN is a broader term that encompasses various data-centric networking approaches. It focuses on the efficient and secure delivery of content based on its attributes. ICN architectures often incorporate in-network caching and content routing mechanisms.

- **Examples**: Publish-Subscribe Systems, Data-Oriented Routing

5. Data Center Networks

Data center networks are specialized networks designed to interconnect large-scale server infrastructure and handle the massive amounts of data generated by cloud applications. Data center networks often employ data-centric principles to optimize content distribution and storage.

- **Examples**: Google Data Center Network, Facebook Data Center Network

CO5 Distinguish between DNS, HTTP,SMTP and FTP w.r.t Stateful / Stateless, Transport Layer Protocol Used, Connectionless / Connection Oriented, Persistent / Non-persistent and Port Number Used.

Protocol	Stateful/Stateless	Transport Layer Protocol	Connectionless/Connection-Oriented	Persistent/ Non-persistent	Port Number
DNS	**Stateless**	**UDP**	**Connectionless**	**N/A**	**53**
HTTP	**Stateless**	**TCP**	**Connection-Oriented**	**Both**	**80**
SMTP	**Stateful**	**TCP**	**Connection-Oriented**	**Persistent**	**25**

FTP	**Stateful**	**TCP**	**Connection-Oriented**	**N/A**	**20 (Data), 21 (Control)**

DNS (Domain Name System)

- DNS is a **stateless protocol**. It does not retain any information about past client requests. Each DNS query is treated independently.
- It primarily uses **UDP** as the transport layer protocol due to its speed and efficiency.
- DNS is a **connectionless protocol** that does not require establishing a connection before sending a query.
- The concept of persistent/non-persistent connections is not applicable to DNS.
- DNS typically uses port number **53**.

HTTP (Hypertext Transfer Protocol)

- HTTP is inherently a **stateless protocol**. However, mechanisms like cookies and sessions can be employed to maintain state information across multiple requests.
- HTTP uses **TCP** as the transport layer protocol to ensure reliable data transfer for web pages and other resources.
- HTTP is a **connection-oriented protocol** that establishes a TCP connection before transmitting data.
- HTTP can support both **persistent and non-persistent connections**. Persistent connections allow multiple requests and responses to be sent over a single TCP connection, improving performance. Non-persistent connections require a new TCP connection for each request/response pair.
- HTTP typically uses port number **80**.

SMTP (Simple Mail Transfer Protocol)

- SMTP is a **stateful protocol** designed for sending email messages. It maintains connection state information during the process of transferring an email message.
- SMTP relies on **TCP** as the transport layer protocol to provide reliable delivery of email messages.
- SMTP is a **connection-oriented protocol** that establishes a TCP connection between the sender and receiver before transferring messages.
- SMTP uses **persistent connections** to send multiple email messages over a single TCP connection.
- SMTP uses port number **25**.

FTP (File Transfer Protocol)

- FTP is a **stateful protocol** that requires authentication and maintains state information about the current directory and transfer mode.
- FTP uses **TCP** as the underlying transport protocol.
- FTP is a **connection-oriented protocol** that requires establishing a TCP connection before file transfer can begin.
- FTP does not use the concept of persistent/non-persistent connections in the same way as HTTP. Instead, it establishes two separate TCP connections: a **control connection** for commands and responses, and a **data connection** for file transfer.
- FTP uses port number **21** for the control connection and port number **20** for the data connection.

CO5 illustrate in detail Simple Mail Transfer Protocol [SMTP] and File Transfer Protocol [FTP].

Simple Mail Transfer Protocol (SMTP) is the standard protocol for transferring email messages across the Internet. SMTP is an application-layer protocol that uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server.

- **SMTP follows a stream paradigm** using a persistent connection to send multiple email messages over a single TCP connection. This means that a sequence of bytes flows from one application program to another.
- **SMTP uses textual control messages**. This means that commands and replies between the client and server are sent as ASCII text.
- **SMTP can only transfer text messages**. If a message contains non-ASCII characters or binary data, such as an image, it must be encoded into 7-bit ASCII before being sent.

SMTP utilizes a push protocol. The sending mail server pushes the file to the receiving mail server. The TCP connection is initiated by the machine that wants to send the file. SMTP uses port number **25**.

To understand the process of SMTP, consider Alice sending an email to Bob:

1. Alice's user agent (email client) sends the message to her mail server.
2. The client side of SMTP, running on Alice's mail server, opens a TCP connection to the SMTP server on Bob's mail server.
3. After a handshaking process where the client and server introduce themselves, the client sends Alice's message.
4. Bob's mail server receives the message and places it in Bob's mailbox.

File Transfer Protocol (FTP) is a standard mechanism provided by TCP/IP for copying a file from one host to another. FTP is an application-layer protocol that uses TCP as the underlying transport protocol. **FTP is unique because it uses two parallel TCP connections:** a control connection and a data connection.

- **Control Connection**: This connection is used for sending control information, such as user identification, password, commands to change the remote directory, and commands to "put" and "get" files. It uses port number **21**. The control connection remains open throughout the duration of the user session.
- **Data Connection**: This connection is used to actually send the file. It uses port number **20**. A new data connection is created for each file

transferred within a session, meaning data connections are **non-persistent**.

FTP is a stateful protocol that requires authentication and maintains state information about the current directory and transfer mode.

Consider a user transferring files from their local host to a remote host:

1. The user interacts with FTP through an FTP user agent.
2. The FTP client process on the local host establishes a control TCP connection with the FTP server process on the remote host.
3. The user provides authentication information (username and password), which is sent over the control connection.
4. Once authenticated, the user can issue commands to transfer files.
5. For each file transfer, the server initiates a data connection to the client.
6. After the file is transferred, the data connection is closed.

CO5 Justify why HTTP's persistent connection is reliable. List the differences between persistent and Nonpersistent connection.

It is important to distinguish the reliability of **HTTP** from the reliability of the underlying transport layer protocol it uses, which is **TCP**.

TCP provides a reliable data transfer service to HTTP. This means that each HTTP request message sent by a client will eventually arrive intact at the server, and each HTTP response message sent by the server will eventually arrive intact at the client. TCP achieves this reliability through mechanisms like sequence numbers, acknowledgments, and timers.

HTTP itself is a stateless protocol, meaning it does not retain any information about past client requests. **Therefore, the concept of reliability in HTTP is tied to the reliable data transfer service provided by TCP, not to any inherent reliability mechanism within HTTP itself.**

Here's a breakdown of the differences between persistent and non-persistent connections in HTTP:

Non-Persistent Connections

- **A separate TCP connection is established for each requested object**. This means that if a web page contains multiple objects (HTML file, images, etc.), multiple TCP connections will be created and closed.
- **Each object suffers a delivery delay of two round-trip times (RTTs)**. One RTT is needed to establish the TCP connection, and another RTT is needed to request and receive the object.
- **Higher overhead on the server** due to the need to allocate buffers and variables for each new connection.

Persistent Connections

- **The server leaves the TCP connection open after sending a response, allowing subsequent requests and responses between the same client and server to be sent over the same connection**. This reduces the overhead of establishing new connections for each object.
- **Multiple objects can be sent over a single TCP connection, improving performance**.
- **Reduced latency** because objects can be requested back-to-back without waiting for replies to pending requests (pipelining).

CO3 Distinguish between IPv4 and IPv6 with respect to address space, security features, address length, address representation, message transmission scheme, fragmentation.

IPv4 and **IPv6** are both versions of the Internet Protocol (IP), which is responsible for addressing and routing packets across the internet. However, there are significant differences between the two versions.

Address Space

- **IPv4 uses 32-bit addresses**, providing a total of 232 (approximately 4.3 billion) unique addresses. This limited address space has led to address depletion issues.

- **IPv6 uses 128-bit addresses**, offering a vast address space of 2128 addresses. This massive address space ensures that the internet won't run out of addresses in the foreseeable future.

Address Length

- **IPv4 addresses are 32 bits long**.
- **IPv6 addresses are 128 bits long**.

Address Representation

- **IPv4 addresses are typically represented in dotted-decimal notation**, where each of the four bytes is expressed as a decimal number between 0 and 255, separated by dots (e.g., 192.168.1.1).
- **IPv6 addresses are represented in hexadecimal colon notation**, where the 128-bit address is divided into eight 16-bit sections, each represented by four hexadecimal digits, separated by colons (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

Security Features

- **IPv4 does not inherently include security features**. Security mechanisms like IPsec (Internet Protocol Security) were developed as add-ons.
- **IPv6 has built-in support for security features like authentication and encryption**, providing confidentiality and integrity of packets.

Message Transmission Scheme

- **Both IPv4 and IPv6 are connectionless protocols**. This means that they do not require establishing a connection before sending data.
- They both provide **best-effort delivery**, meaning that they do not guarantee delivery of packets or the order in which they arrive.

Fragmentation

- **In IPv4, fragmentation (dividing a datagram into smaller units) can be performed by routers along the path if a datagram is larger than the Maximum Transmission Unit (MTU) of the network**.

- **In IPv6, fragmentation can only be performed by the sending host**. Routers in IPv6 simply drop datagrams that are too large and send an error message to the sender.

CO4 Distinguish between TCP and UDP w.r.t connection setup, data unit, data delivery retransmission, reliability, flow control.

Transmission Control Protocol (TCP) and **User Datagram Protocol (UDP)** are two distinct transport layer protocols that provide different services to applications.

Connection Setup

- **TCP:** TCP is a **connection-oriented** protocol that requires a three-way handshake to establish a connection before data transfer can begin. This process involves the exchange of SYN, SYN-ACK, and ACK segments between the client and server.
- **UDP:** UDP is a **connectionless** protocol, meaning no prior connection setup is required. Applications can send data immediately without any handshaking.

Data Unit

- **TCP:** The basic unit of data transfer in TCP is called a **segment**. TCP segments have a header containing various control information and a data field containing the payload.
- **UDP:** The data unit in UDP is called a **datagram** or **user datagram**. UDP datagrams also have a header and data field, but the header is simpler than a TCP segment header.

Data Delivery

- **TCP:** TCP provides **reliable, in-order delivery** of data. It uses sequence numbers to ensure that data arrives in the correct order and retransmits any lost or corrupted segments.
- **UDP:** UDP offers **unreliable, best-effort delivery**. Datagrams may arrive out of order, be duplicated, or lost without any notification or retransmission attempts.

Retransmission

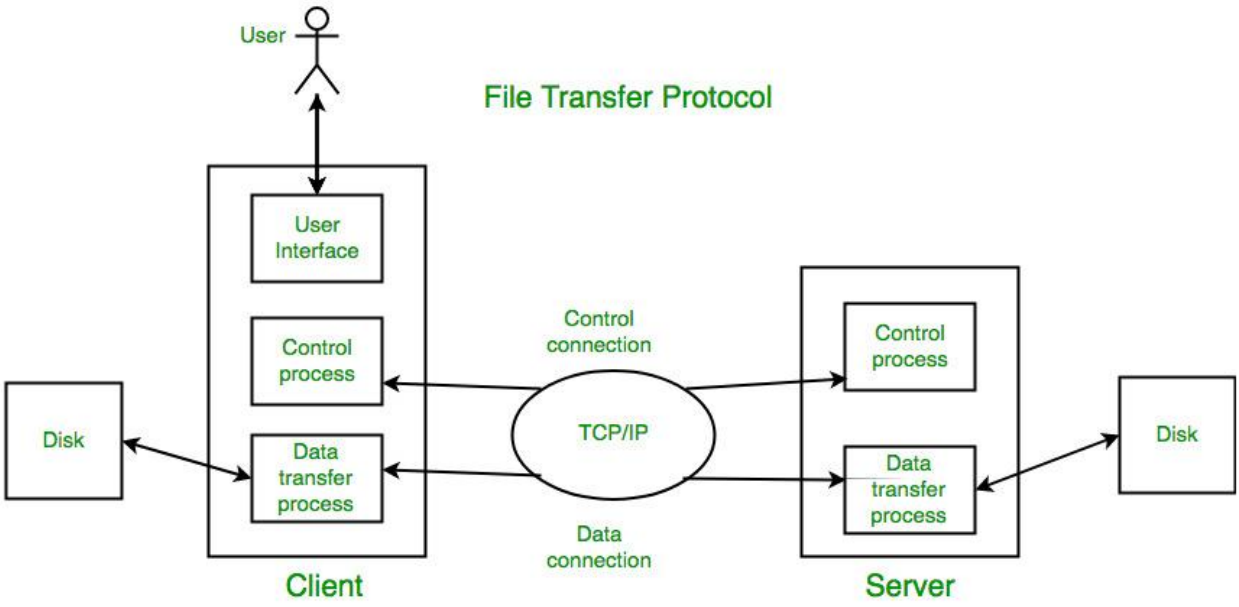
- **TCP:** TCP implements **retransmission mechanisms** to handle lost or corrupted segments. It uses timers and acknowledgments to detect missing segments and retransmit them until acknowledged.
- **UDP: UDP does not perform retransmissions.** Lost or corrupted datagrams are simply discarded.

Reliability

- **TCP:** TCP is a **reliable protocol** that guarantees data delivery and integrity. It employs sequence numbers, acknowledgments, and retransmissions to ensure data arrives correctly at the destination.
- **UDP:** UDP is an **unreliable protocol** that does not guarantee data delivery or order.

Flow Control

- **TCP:** TCP implements **flow control** to prevent the sender from overwhelming the receiver with data. It uses a sliding window mechanism to regulate the amount of data that can be sent at a time. The receiver advertises its available buffer space to the sender, allowing for dynamic adjustment of the window size.
- **UDP: UDP does not have flow control mechanisms.** The sender can transmit data at any rate without considering the receiver's capacity.



CO5 In your practical lab session you are performing your practical. You want to save your program on college server, How FTP will be used in this scenario.

In a practical lab session, you can use **File Transfer Protocol (FTP)** to save your program on the college server. Here's how the process would work:

1. **Initiate an FTP Connection:** On your local machine, you would use an FTP client application to initiate a connection to the college server's FTP server. You would need to provide the server's hostname or IP address, your username, and password for authentication. The FTP client establishes two connections with the server:
    - **Control Connection:** This connection, using port 21, is used for sending commands and receiving responses between your FTP client and the FTP server. For example, you'll send commands to navigate directories, list files, and initiate file transfers.
    - **Data Connection:** This connection, using port 20, is used for the actual transfer of files.
2. **Navigate to the Target Directory:** Once authenticated, you would use FTP commands to navigate to the desired directory on the server where you want to save your program. Commands like cd (change directory) and pwd (print working directory) can be used to navigate the server's directory structure.
3. **Transfer the Program File:** You would then use the STOR command to upload your program file from your local machine to the server. This command tells the FTP server to receive a file from the client. The FTP client will then initiate a data connection on port 20 to transfer the file. You might need to specify the file type (e.g., ASCII, binary) before initiating the transfer.
4. **Verify the Transfer:** After the transfer is complete, you can use the LIST command to verify that your program file has been successfully saved in the correct directory on the server.
5. **Close the Connection:** Once you have finished transferring the file, you can use the QUIT command to close the FTP connection.

CO4 Name the protocols which uses the either TCP or UDP protocol for establishing the connection for correct answer

The choice of using **TCP** or **UDP** depends on the specific requirements of the application. Here are some protocols that use either TCP or UDP:

Applications that typically use TCP:

- **HTTP (Hypertext Transfer Protocol)**: Used for transferring web pages and other content over the internet. TCP's reliability ensures that web pages are delivered completely and in order.
- **SMTP (Simple Mail Transfer Protocol)**: Used for sending email messages. TCP's reliability is crucial for ensuring emails are delivered without errors.
- **FTP (File Transfer Protocol)**: Used for transferring files between computers. TCP's reliability ensures that files are transferred completely and without corruption.
- **Telnet (Terminal Network)**: Provides a remote terminal connection, allowing users to access and control a computer from a distance. TCP's reliability is important for maintaining a stable connection.
- **SSH (Secure Shell)**: Used for secure remote login and other secure network services. Relies on TCP's reliable connection for security and data integrity.

Applications that typically use UDP:

- **DNS (Domain Name System)**: Used for translating domain names into IP addresses. UDP's speed and efficiency are preferred for DNS queries, as a small delay in resolving a domain name is usually acceptable.
- **SNMP (Simple Network Management Protocol)**: Used for monitoring and managing network devices. UDP's simplicity and low overhead are suitable for SNMP messages.
- **DHCP (Dynamic Host Configuration Protocol)**: Used for automatically assigning IP addresses to devices on a network. UDP's connectionless nature allows for efficient broadcast of DHCP messages to multiple devices.

- **RIP (Routing Information Protocol)**: A routing protocol used to exchange routing information between routers. UDP is commonly used for RIP due to its simplicity.
- **RTP (Real-time Transport Protocol)**: Used for transmitting real-time data, such as audio and video streaming. While UDP is inherently unreliable, RTP adds mechanisms for sequence numbering, timestamps, and payload type identification, enabling applications to handle packet loss and jitter effectively in real-time applications.

CO5 What is the role of SMTP protocol in email system? With neat diagram explain the working of SMTP. What is the range of SMTP protocol?9

The **Simple Mail Transfer Protocol (SMTP)** is the standard protocol used to transfer email messages across the internet. SMTP operates at the application layer and relies on the reliable data transfer service of TCP to ensure email messages are delivered without errors. It uses a **push protocol** to send messages from a sender's mail server to a recipient's mail server.

Working of SMTP

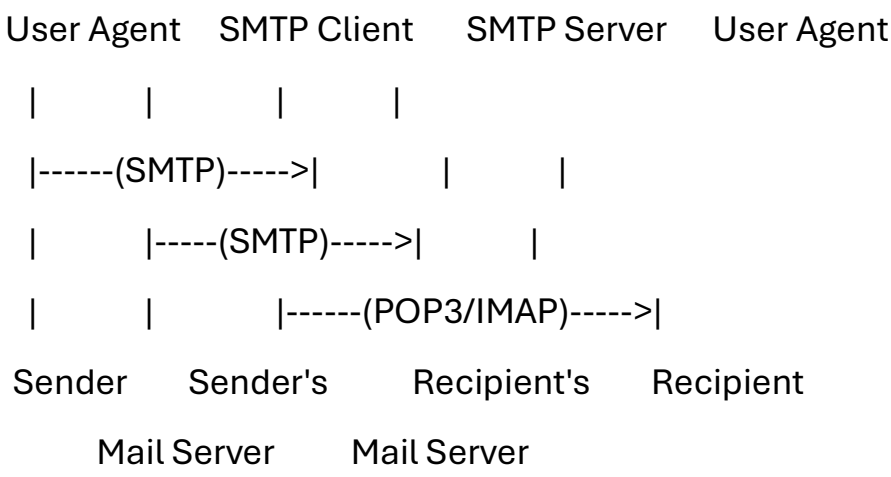
Here's how SMTP works to transfer an email message, illustrated with a diagram:

1. **User Agent to Sender's Mail Server:**
    - The sender composes an email using a user agent (email client).
    - The user agent sends the message to the sender's mail server using SMTP.
    - The sender's mail server places the message in a message queue.
2. **Sender's Mail Server to Recipient's Mail Server:**
    - The SMTP client on the sender's mail server opens a TCP connection to the SMTP server on the recipient's mail server.
    - The SMTP client and server perform a handshake, exchanging information like the sender's and recipient's email addresses.

- The SMTP client sends the email message through the TCP connection.
- The SMTP server on the recipient's mail server receives the message and places it in the recipient's mailbox.

3. **Recipient's Mail Server to User Agent:**
    - The recipient uses their user agent to retrieve the message from their mailbox on the mail server. This is typically done using a mail access protocol like POP3 or IMAP.

Diagram



Range of SMTP

SMTP is used **twice** in the email delivery process:

- **Between the sender's user agent and the sender's mail server.**
- **Between the sender's mail server and the recipient's mail server.**

Limitations of SMTP:

- SMTP traditionally only supports the transmission of **text messages**. To send binary files (like images, audio, or video), they must be encoded into a text format like MIME.
- SMTP does not handle the final delivery of the message to the recipient's user agent; a separate mail access protocol like POP3 or IMAP is required for this step.

Overall, SMTP plays a crucial role in the email system by providing a standardized and reliable way to transfer email messages between mail servers.