CBCS SCHEME

USN 1CR22CI046

BCS502

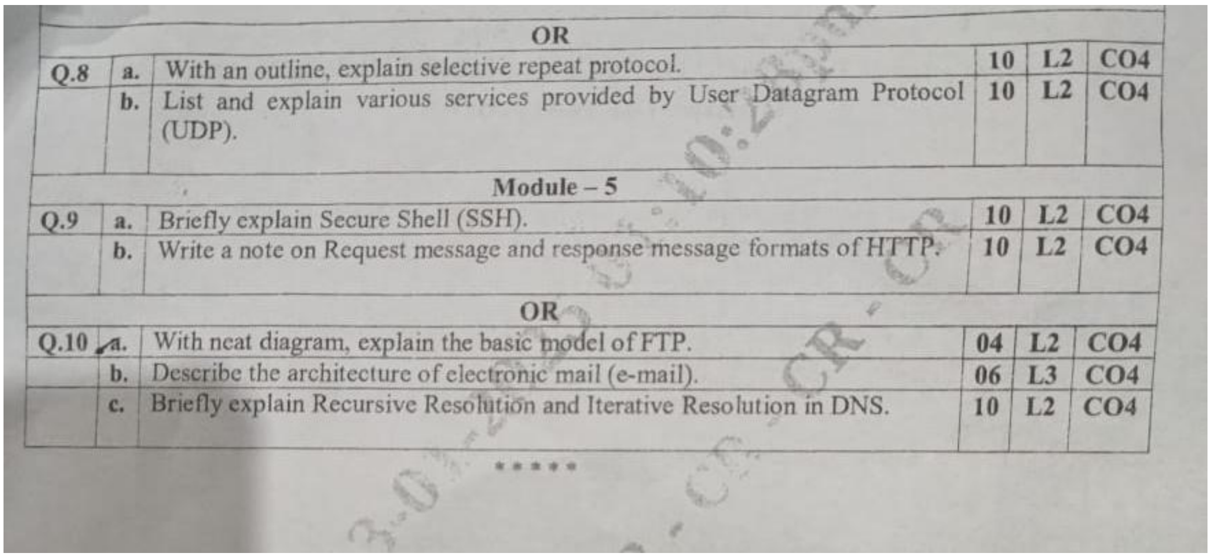
Fifth Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025
Computer Networks

Time: 3 hrs.

Max. Marks: 100

***Note:** 1. Answer any FIVE full questions, choosing ONE full question from each module.*
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	What is data communication? List and explain characteristics and components of communication model.	06	L1	CO1
	b.	Define switching. Explain Circuit Switched Network and Packet Switched Network.	06	L2	CO1
	c.	With neat sketch, explain different layers of TCP/IP protocol suite.	08	L2	CO1
		OR			
Q.2	a.	What are guided transmission media? Explain twisted pair cable in detail.	06	L1	CO1
	b.	What is Virtual Circuit Network (VCN)? With neat diagram, explain three phases involved in VCN.	08	L1	CO1
	c.	Write a note on Encapsulation and decapsulation at Source Host for TCP/IP protocol suite.	06	L2	CO1
		Module – 2			
Q.3	a.	Define Redundancy. Explain CRC encoder and CRC decoder operation with block diagram.	08	L2	CO2
	b.	Distinguish between Flow Control and Error Control. Explain Stop and Wait Protocol.	08	L2	CO2
	c.	List and explain Control Fields of I-frames, S-frames and U-frames.	04	L2	CO2
		OR			
Q.4	a.	What is Hamming distance? With example, explain Parity Check Code.	06	L1	CO2
	b.	Define Framing. Explain character oriented framing and bit-oriented framing.	06	L1	CO2
	c.	With flow diagram, explain CSMA/CA.	08	L2	CO2
		Module – 3			
Q.5	a.	Explain virtual-circuit approach to route the packets in packet-switched network.	10	L2	CO3
	b.	Illustrate the working of OSPF and BGP.	10	L3	CO3
		OR			
Q.6	a.	Explain IPv6 datagram format.	10	L2	CO3
	b.	Write an Dijikstra's algorithm to compute shortest path through graph.	06	L1	CO3
	c.	Write a note on Routing Information Protocol (RIP) algorithm.	04	L2	CO3
		Module – 4			
Q.7	a.	Explain Go-Back-N protocol working.	10	L2	CO4
	b.	With neat sketch, explain three-way handshaking of TCP connection establishment.	10	L2	CO4



		OR			
Q.8	a.	With an outline, explain selective repeat protocol.	10	L2	CO4
	b.	List and explain various services provided by User Datagram Protocol (UDP).	10	L2	CO4
		Module – 5			
Q.9	a.	Briefly explain Secure Shell (SSH).	10	L2	CO4
	b.	Write a note on Request message and response message formats of HTTP.	10	L2	CO4
		OR			
Q.10	a.	With neat diagram, explain the basic model of FTP.	04	L2	CO4
	b.	Describe the architecture of electronic mail (e-mail).	06	L3	CO4
	c.	Briefly explain Recursive Resolution and Iterative Resolution in DNS.	10	L2	CO4

*****

Q.1.a. What is data communication? List and explain characteristics and components of communication model

What is Data Communication?

Data communication refers to the exchange of data or information between two or more devices (e.g., computers, servers, or smartphones) through a communication medium like a wired or wireless network. The process involves transmitting, receiving, and processing data using established protocols.

Characteristics of Data Communication

1. **Delivery**: The data must be delivered to the correct destination. The recipient should be the intended device or user.
2. **Accuracy**: The communication system should ensure that the transmitted data is accurate without errors.
3. **Timeliness**: Data must be delivered in a timely manner to ensure it remains relevant.
4. **Jitter**: Jitter refers to variations in data packet arrival time. A good communication system minimizes jitter for smooth delivery, especially in multimedia.
5. **Bandwidth**: The capacity of the communication channel determines how much data can be transmitted in a given time frame.

Components of the Communication Model

The communication model typically consists of the following components:

1. **Sender**
   - **Role**: The device or entity that initiates the communication process by sending data.
   - **Example**: A computer or a smartphone sending an email.
2. **Receiver**

   - **Role**: The device or entity that receives the transmitted data.
   - **Example**: The recipient's computer or smartphone.

3. **Message**
   - **Role**: The actual information being transmitted.
   - **Example**: A text message, audio file, video stream, or email content.
4. **Transmission Medium**
   - **Role**: The physical path or channel through which data travels from sender to receiver.
   - **Examples**:
     - **Wired**: Ethernet cables, fiber optics.
     - **Wireless**: Wi-Fi, radio waves.
5. **Protocol**
   - **Role**: A set of rules or standards that define how data is formatted, transmitted, and received.
   - **Example**: HTTP, FTP, TCP/IP.
6. **Encoder/Decoder**
   - **Role**: Converts data into a transmittable format at the sender's end (encoding) and converts it back into a readable format at the receiver's end (decoding).
   - **Example**: Compression algorithms like MP3 for audio or H.264 for video.
7. **Feedback**
   - **Role**: Acknowledgment sent by the receiver to confirm successful data delivery.
   - **Example**: "Message received" notification or an HTTP acknowledgment packet.
8. **Noise**
   - **Role**: Any interference or distortion during the transmission process that can affect data integrity.
   - **Example**: Signal interference in wireless communication.

Q.1.b.Define switching. Explain Circuit Switched Network and Packet Switched Network.

Definition of Switching

Switching is a technique used in telecommunication and networking to route data or voice signals between devices in a network. It ensures efficient utilization of resources by directing data packets or establishing dedicated communication paths between the sender and receiver.

Types of Switching

1. **Circuit-Switched Network**
2. **Packet-Switched Network**

---

1. Circuit-Switched Network

In a circuit-switched network, a dedicated communication path or circuit is established between the sender and receiver for the duration of the communication session. This type of switching is commonly used in traditional telephone networks.

Features:

- **Dedicated Path**: A single, exclusive channel is established for communication.
- **Real-Time Communication**: Suitable for applications like voice calls.
- **Resource Usage**: Resources are reserved and cannot be shared with others until the session ends.
- **Latency**: Minimal latency once the circuit is established.

Advantages:

- Reliable and consistent data transfer.
- Predictable performance due to a fixed data transfer path.

Disadvantages:

- Inefficient use of resources as the dedicated channel remains idle when not in use.
- Establishing a circuit takes time.

Example:

Traditional Public Switched Telephone Network (PSTN).

---

2. Packet-Switched Network

In a packet-switched network, data is divided into packets, and each packet is transmitted independently over shared network resources. Packets may take different routes to reach the destination, where they are reassembled in the correct order.

Features:

- **No Dedicated Path**: Packets are routed dynamically based on availability.
- **Efficient Resource Use**: Resources are shared among multiple users.
- **Store-and-Forward Mechanism**: Routers store packets temporarily before forwarding them.
- **Data Integrity**: Packets are checked for errors and retransmitted if needed.

Advantages:

- Efficient resource utilization.
- Scalable and adaptable to varying traffic conditions.
- Lower cost compared to circuit-switched networks.

Disadvantages:

- Latency due to packet reassembly and routing delays.
- Potential for packet loss or out-of-order delivery.

Example:

The Internet, which uses protocols like TCP/IP.

Comparison: Circuit-Switched vs Packet-Switched Networks

Aspect	**Circuit-Switched Network**	**Packet-Switched Network**
Path	Dedicated	Dynamic
Resource Utilization	Inefficient	Efficient
Latency	Low (after setup)	Variable
Usage	Voice calls	Data communication (e.g., emails)
Example	PSTN	Internet

Q.1.c With neat sketch, explain different layers of TCP/IP protocol suite

Layers of TCP/IP Protocol Suite

The TCP/IP protocol suite is a set of communication protocols used to connect network devices on the internet. It is organized into four abstraction layers, each responsible for specific functionalities.

---

1. Application Layer

- **Function**: Provides network services to applications. It handles high-level protocols and user interfaces.
- **Protocols**:
    - HTTP, HTTPS (web browsing)
    - SMTP (email sending)
    - FTP (file transfer)
    - DNS (domain name resolution)
- **Examples**: Web browsers, email clients, and file transfer applications.

---

2. Transport Layer

- **Function**: Ensures reliable data delivery and provides end-to-end communication between devices.
- **Key Features**:
    - **Segmentation**: Divides large data into smaller segments.
    - **Flow Control**: Manages the data transmission rate to avoid congestion.
    - **Error Control**: Ensures data integrity by retransmitting lost or corrupted packets.
- **Protocols**:
    - TCP (reliable, connection-oriented)
    - UDP (faster, connectionless)

---

3. Internet Layer

- **Function**: Handles logical addressing and routing of data packets between devices.
- **Key Features**:
    - Logical IP addressing for unique identification of devices.
    - Routing of packets across networks.
- **Protocols**:
    - IP (IPv4, IPv6): Provides addressing and routing.
    - ICMP: Used for error reporting (e.g., ping command).
    - ARP: Resolves IP addresses to MAC addresses.

---

4. Network Access Layer

- **Function**: Deals with the hardware and physical transmission of data.
- **Key Features**:
    - Converts packets into frames for physical transmission.
    - Responsible for Media Access Control (MAC) addressing.
- **Protocols**:
    - Ethernet, Wi-Fi, DSL.
    - Manages data transmission over various physical media.

Diagram of TCP/IP Protocol Suite

plaintext

CopyEdit

```
+----------------------+
|  Application Layer   | <- HTTP, FTP, SMTP, DNS
+----------------------+
|    Transport Layer   | <- TCP, UDP
+----------------------+
|    Internet Layer    | <- IP, ICMP, ARP
+----------------------+
| Network Access Layer | <- Ethernet, Wi-Fi
+----------------------+
|    Physical Medium   | <- Copper, Fiber Optic, Wireless
+----------------------+
```

Key Points

- The TCP/IP model maps to the OSI model but has fewer layers:
  - **Application Layer** combines the Application, Presentation, and Session layers of OSI.
  - **Network Access Layer** corresponds to the Data Link and Physical layers of OSI.
- It is widely used because it is simpler and directly maps to internet communication.

Q.2.a.What are guided transmission media? Explain twisted pair cable in detail.

Guided Transmission Media

Guided transmission media refers to physical pathways that guide the transmission of data signals from one device to another. These media are tangible and require a physical connection. They are commonly used in wired communication systems.

Types of Guided Transmission Media

1. **Twisted Pair Cable**
2. **Coaxial Cable**
3. **Optical Fiber Cable**

Twisted Pair Cable

A twisted pair cable consists of pairs of insulated copper wires twisted together to reduce electromagnetic interference (EMI) and crosstalk. It is the most commonly used medium for telephone and LAN (Local Area Network) connections.

Structure of Twisted Pair Cable

- Two insulated copper wires are twisted together in a helical shape.
- The twisting reduces interference from external sources and between pairs.

Types of Twisted Pair Cables

1. **Unshielded Twisted Pair (UTP)**:
    - No additional shielding beyond the insulating cover.
    - Lighter, cheaper, and easier to install.
    - Used in Ethernet networks and telephone lines.
2. **Shielded Twisted Pair (STP)**:
    - Includes an additional metallic shielding to reduce interference.
    - More expensive and provides better performance than UTP.
    - Used in environments with high electromagnetic interference.

Advantages of Twisted Pair Cable

1. **Cost-Effective**: Inexpensive compared to coaxial and optical fiber cables.
2. **Flexible and Lightweight**: Easy to install and maintain.
3. **Reduces Crosstalk**: Twisting minimizes signal interference between adjacent wires.
4. **Widely Available**: Commonly used for many networking applications.

Disadvantages of Twisted Pair Cable

1. **Limited Bandwidth**: Cannot support very high data transfer rates.
2. **Limited Distance**: Signal degradation occurs over long distances without repeaters.
3. **Susceptible to Interference**: Especially for UTP in environments with high EMI.

Applications of Twisted Pair Cable

1. **Telephone Lines**: Used for voice communication.
2. **Local Area Networks (LANs)**: Widely used in Ethernet connections (e.g., Cat 5, Cat 6 cables).
3. **DSL Lines**: Used for broadband internet access.

Comparison: UTP vs STP

Aspect	UTP (Unshielded Twisted Pair)	STP (Shielded Twisted Pair)
Cost	Lower	Higher
Interference	More susceptible	Better protection from EMI
Performance	Adequate for basic applications	Suitable for high-interference areas
Installation	Easier	Slightly more difficult

Q.2.b.What is Virtual Circuit Network (VCN)? With neat diagram, explain three phases involved in VCN.

What is a Virtual Circuit Network (VCN)?

A **Virtual Circuit Network (VCN)** is a type of network communication model in which a logical path or circuit is established between the sender and receiver before data transmission begins. It combines features of circuit switching and packet switching, ensuring reliable and ordered delivery of packets over a predefined route.

Key Features of VCN:

1. **Logical Connection**: A virtual connection is established before data transfer.
2. **Packet Sequencing**: Data packets are delivered in order along the pre-defined path.
3. **Error Handling**: Reliable delivery with mechanisms to detect and correct errors.

Phases of Virtual Circuit Network

VCNs operate in three distinct phases:

1. Connection Establishment Phase

- A logical connection (virtual circuit) is created between the sender and receiver.
- The network nodes (routers/switches) determine the path to be used for the data transmission.
- Routing tables are updated to define the route.
- This phase is similar to setting up a circuit in circuit switching.

2. Data Transfer Phase

- Data is transmitted in the form of packets along the established virtual circuit.
- Each packet contains a **virtual circuit identifier (VCI)** instead of the full destination address.
- The network ensures:
    - **Orderly delivery**: Packets arrive in sequence.
    - **Error checking and correction**: If errors occur, the affected packets are retransmitted.

3. Connection Termination Phase

- Once the communication is complete, the virtual circuit is terminated.
- The resources allocated during the connection establishment are released.
- This ensures efficient utilization of network resources.

Diagram: Virtual Circuit Network and Phases

plaintext

CopyEdit

```
+-----------------+         +---------------+         +----------------+
|     Sender      | <----> | Intermediate  | <----> |    Receiver    |
|                 |         | Nodes (Switches)       |                |
+-----------------+         +---------------+         +----------------+


  Phase 1: Connection Establishment (Route setup and resource allocation)
  Phase 2: Data Transfer (Data packets travel along the virtual circuit)
  Phase 3: Connection Termination (Release of resources)
```

Advantages of Virtual Circuit Network

1. **Reliable Communication**: Guarantees ordered delivery of packets.
2. **Efficient Error Handling**: Reduces retransmission overhead due to packet loss.
3. **Fixed Path**: Simplifies routing during the data transfer phase.

Disadvantages of Virtual Circuit Network

1. **Setup Delay**: The connection establishment phase adds latency before transmission begins.
2. **Resource Dependency**: Resources are reserved for the duration of the connection.
3. **Scalability Issues**: May not be as efficient in highly dynamic or large-scale networks.

Applications

- Virtual Circuit Networks are commonly used in:
    - Asynchronous Transfer Mode (ATM).
    - Frame Relay.
    - MPLS (Multiprotocol Label Switching) for modern data communication.

Q.2.c.Write a note on Encapsulation and decapsulation at Source Host for TCP/IP 06 12 protocol suite.

Encapsulation and Decapsulation in TCP/IP Protocol Suite

Encapsulation and decapsulation are essential processes in the **TCP/IP protocol suite**, enabling data communication between devices across a network. These processes ensure that data is prepared for transmission at the source and correctly interpreted at the destination.

Encapsulation (Source Host)

Encapsulation is the process of adding headers (and sometimes trailers) to data as it passes through the layers of the TCP/IP protocol suite. Each layer encapsulates the data received from the layer above it with its own protocol-specific header.

Steps of Encapsulation:

1. **Application Layer**:
   - The application generates the data (e.g., a web page or an email) and passes it to the transport layer.
   - No headers are added at this layer in TCP/IP (unlike OSI).
2. **Transport Layer**:
   - Adds a **transport header**, which includes:
     - Port numbers to identify applications (e.g., HTTP uses port 80).
     - Sequence numbers for data reassembly.
     - Error detection codes for reliability.
   - The encapsulated unit is called a **segment** (TCP) or **datagram** (UDP).
3. **Internet Layer**:
   - Adds an **IP header**, which includes:
     - Source and destination IP addresses.
     - Packet identification and routing information.
   - The encapsulated unit is called a **packet**.
4. **Network Access Layer**:
   - Adds a **frame header and trailer**, which include:
     - MAC (Media Access Control) addresses.
     - Error-checking codes for the physical transmission.
   - The encapsulated unit is called a **frame**.
5. **Physical Layer**:

    - Converts the frame into electrical, optical, or radio signals for transmission over the physical medium.

Decapsulation (Destination Host)

Decapsulation is the reverse process of encapsulation, performed at the destination host. Each layer removes its corresponding header, processes the information, and passes the remaining data to the layer above.

Steps of Decapsulation:

1. **Physical Layer**:
    - Receives raw signals and converts them into frames.
    - Passes the frame to the network access layer.
2. **Network Access Layer**:
    - Removes the frame header and trailer.
    - Checks for errors using the trailer information.
    - Passes the packet to the internet layer.
3. **Internet Layer**:
    - Removes the IP header.
    - Processes the destination IP address to ensure the packet is intended for this host.
    - Passes the segment to the transport layer.
4. **Transport Layer**:
    - Removes the transport header.
    - Uses port numbers to determine the correct application for the data.
    - Passes the data to the application layer.
5. **Application Layer**:
    - Processes the data and presents it to the user or the corresponding application.

Encapsulation and Decapsulation in Diagram

plaintext

CopyEdit

```
Source Host (Encapsulation)          Destination Host (Decapsulation)

+-----------------------+            +-----------------------+

| Application Data      | ---> Application ---> | Application Data      |
+-----------------------+                    +-----------------------+
| Transport Header      | ---> Transport  ---> | Transport Header      |
+-----------------------+                    +-----------------------+
| Internet Header       | ---> Internet   ---> | Internet Header       |
+-----------------------+                    +-----------------------+
| Frame Header & Trailer| ---> Network    ---> | Frame Header & Trailer|
+-----------------------+                    +-----------------------+
| Physical Medium       | ---> Physical   ---> | Physical Medium       |
+-----------------------+                    +-----------------------+
```

Q.3.a.Define Redundancy. Explain CRC encoder and CRC decoder operation with block diagram.

Definition of Redundancy

Redundancy refers to the inclusion of extra bits or information in transmitted data to detect and/or correct errors during communication. These extra bits do not convey additional information but are used to enhance the reliability of data transmission.

Redundancy techniques ensure that errors introduced during transmission can be detected or corrected at the receiver's end.

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) is an error-detecting technique widely used in digital communication and storage systems. CRC uses polynomial division to generate a checksum, which is transmitted along with the data. The receiver performs the same division operation to verify data integrity.

Operation of CRC Encoder and Decoder

1. CRC Encoder

The encoder generates a checksum (remainder) using polynomial division and appends it to the data before transmission.

Steps in CRC Encoding:

1. **Data Representation**: Represent the message as a binary string.
2. **Generator Polynomial**: Use a predefined generator polynomial (e.g., G(x)G(x)G(x)).
3. **Padding**: Append n−1n-1n−1 zero bits to the message, where nnn is the degree of the generator polynomial.
4. **Division**: Perform binary division of the padded message by G(x)G(x)G(x) using modulo-2 arithmetic.

5. **Remainder**: The remainder from the division is the CRC checksum.
6. **Appending Checksum**: Append the checksum to the original message and transmit the result.

2. CRC Decoder

The decoder verifies the integrity of the received message by dividing it by the same generator polynomial.

Steps in CRC Decoding:

1. **Receive Data**: Retrieve the transmitted message, which includes the original data and the CRC checksum.
2. **Division**: Perform binary division of the received message by G(x)G(x)G(x) using modulo-2 arithmetic.
3. **Verification**:
   - If the remainder is zero, the data is error-free.
   - If the remainder is non-zero, an error has occurred.

Block Diagrams

CRC Encoder Block Diagram

plaintext

CopyEdit

```
+----------------+        +---------------+
| Input Message  |  ------> |   Shift       |
| (Data Bits)    |          | Register      |
+----------------+        +---------------+
                     |
                     v
             +---------------+
             | Modulo-2 XOR  |
             +---------------+
                     |
                     v
+----------------+        +---------------+
| Generator      |  ------>  | CRC Checksum  |
| Polynomial     |          | Generator     |
+----------------+         +---------------+
```

CRC Decoder Block Diagram

plaintext

CopyEdit

```
+----------------+         +---------------+
| Received Data  |  ------> |   Shift       |
| (Data + CRC)   |          | Register      |
+----------------+         +---------------+
                     |
                     v
               +---------------+
               | Modulo-2 XOR  |
               +---------------+
                     |
                     v
+----------------+         +---------------+
| Generator      |  ------>  | CRC Remainder |
| Polynomial     |          | Verifier      |
+----------------+         +---------------+
```

Advantages of CRC

1. **High Error Detection**: Detects single-bit and burst errors effectively.
2. **Simple Implementation**: Easy to implement using shift registers and modulo-2 arithmetic.
3. **Efficiency**: Adds minimal redundancy while ensuring high reliability.

Disadvantages of CRC

1. **No Error Correction**: CRC detects errors but cannot correct them.
2. **Limited Scope**: Cannot detect all types of errors, especially in high-noise environments.

Applications

- Networking protocols (e.g., Ethernet, Wi-Fi).
- Storage devices (e.g., hard drives, SSDs).

- Communication systems (e.g., digital TV, mobile networks).

Q.3.b.Distinguish between Flow Control and Error Control Explain Stop and Wait Protocol

Distinction Between Flow Control and Error Control

Aspect	Flow Control	Error Control
Definition	Regulates the rate of data transmission to prevent the sender from overwhelming the receiver.	Ensures the integrity and correctness of the transmitted data by detecting and correcting errors.
Purpose	Prevents buffer overflow at the receiver.	Detects and corrects errors caused by noise or interference during transmission.
Mechanism	Uses techniques like sliding window, stop-and-wait.	Uses techniques like CRC, parity bits, checksums, and ARQ.
Key Focus	Data flow management.	Data reliability.
Protocols	Stop-and-Wait, Sliding Window Protocols.	ARQ (Automatic Repeat Request), Hamming Code.

Stop-and-Wait Protocol

The **Stop-and-Wait Protocol** is a simple data flow and error control method where the sender transmits one data frame at a time and waits for an acknowledgment (ACK) before sending the next frame.

How Stop-and-Wait Protocol Works

1. **Sender Side**:
   - Transmits a single data frame to the receiver.
   - Waits for an acknowledgment before sending the next frame.
2. **Receiver Side**:
   - Receives the frame, processes it, and sends an acknowledgment back to the sender.
   - If the frame is damaged or missing, the receiver does not send an acknowledgment, prompting the sender to retransmit.

Flow Control in Stop-and-Wait

- Ensures that the sender waits until the receiver is ready to accept the next frame.
- Prevents the receiver's buffer from being overwhelmed.

Error Control in Stop-and-Wait

- Uses **ACKs** and **timeouts** for error detection and correction:
    - If the sender does not receive an acknowledgment within a specific time (timeout), it retransmits the frame.
    - If an acknowledgment is received, the sender proceeds to the next frame.

Advantages of Stop-and-Wait Protocol

1. Simple and easy to implement.
2. Ensures reliable communication with error recovery.

Disadvantages of Stop-and-Wait Protocol

1. **Low Efficiency**: The sender remains idle while waiting for acknowledgment, leading to poor utilization of the network.
2. **High Delay**: Increased latency due to waiting after each frame.

Illustration of Stop-and-Wait Protocol

plaintext

CopyEdit

```
Sender:    [Frame 1] ---->    Waits for ACK    ----> [Frame 2]

           |                         |

           v                         v

Receiver:  Acknowledges ----> Sends ACK ----> Acknowledges
```

If an error occurs:

- Receiver does not send ACK.
- Sender retransmits the same frame.

Q.3.c. List and explain Coutrol Fields of 1-frames, S-frames and U-frames.

Control Fields in Frames (1-frames, S-frames, and U-frames)

In the **HDLC (High-Level Data Link Control)** protocol, control fields are used to manage and control the communication between devices. The HDLC protocol defines three types of frames:

1. **I-Frames (Information Frames)**: Used for data transmission.
2. **S-Frames (Supervisory Frames)**: Used for flow control and error control.
3. **U-Frames (Unnumbered Frames)**: Used for network management and control.

Each frame type has a specific control field structure that serves different purposes.

---

1. I-Frames (Information Frames)

I-frames carry user data and control information for flow and error control.

Control Field Structure for I-Frames:

Bits	Description
N(S)	Sequence number of the transmitted frame.
N(R)	Acknowledgment number for the received frame.
P/F (Poll/Final)	Indicates whether the frame is a poll or final frame.

- **Purpose**:
    - Transmit user data between devices.
    - Include acknowledgment for previously received data frames.

---

2. S-Frames (Supervisory Frames)

S-frames are used for flow control and error control. They do not carry user data but manage the communication session.

Control Field Structure for S-Frames:

Bits	Description
Control Type (2 bits)	Indicates the type of S-frame: - **00**: Receive Ready (RR) - **01**: Receive Not Ready (RNR) - **10**: Reject (REJ) - **11**: Selective Reject (SREJ).
N(R)	Acknowledgment number for the last correctly received I-frame.
P/F	Poll/Final bit for session management.

- **Purpose**:
    - Manage flow control:
        - **RR**: Indicates the receiver is ready to accept more frames.
        - **RNR**: Indicates the receiver is not ready to accept frames.
    - Manage error control:
        - **REJ**: Signals a negative acknowledgment for a lost or erroneous frame.
        - **SREJ**: Used for selective retransmission of a specific frame.

---

3. U-Frames (Unnumbered Frames)

U-frames are used for network management and control functions, such as connection establishment, disconnection, and error reporting.

Control Field Structure for U-Frames:

Bits	**Description**
Control Type (5 bits)	Specifies the type of U-frame operation (e.g., SABME, DISC, UA).
P/F	Poll/Final bit for session management.
Command/Response	Differentiates between command and response frames.

- **Purpose**:
    - Establish, manage, and terminate communication sessions.
    - Handle special control signals like **Set Asynchronous Balanced Mode Extended (SABME)**, **Disconnect (DISC)**, and **Unnumbered Acknowledgment (UA)**.

Summary Table of Control Fields

Frame Type	**Purpose**	**Control Field Components**
I-Frame	Data transmission	N(S), N(R), P/F
S-Frame	Flow and error control	Control Type (RR, RNR, REJ, SREJ), N(R), P/F
U-Frame	Network management and control	Control Type (SABME, DISC, UA, etc.), P/F, Command/Response

Q.4.a.What is Hamming distance? With example, explain Party Check Code.

Hamming Distance

The **Hamming distance** between two strings of equal length is defined as the number of positions at which the corresponding symbols (bits) are different. In other words, it measures how many bits need to be changed to convert one binary string into another.

Formula:

Hamming Distance=Number of bit positions where the two strings differ\text{Hamming Distance} = \text{Number of bit positions where the two strings differ}Hamming Distance=Number of bit positions where the two strings differ

For example:

- Hamming distance between 10101 and 10011 is **2** because the bits at the second and fourth positions are different.

Parity Check Code

The **parity check code** is a simple error detection scheme in which an additional bit, called the **parity bit**, is added to the data to make the number of 1-bits either even or odd. The parity bit is used to check the integrity of data during transmission. There are two types of parity checks:

1. **Even Parity**: The number of 1-bits in the data, including the parity bit, should be even.
2. **Odd Parity**: The number of 1-bits in the data, including the parity bit, should be odd.

Working of Parity Check:

- **Sender Side**: The sender calculates the parity bit based on the data and appends it to the data.
- **Receiver Side**: The receiver checks the received data (including the parity bit) to verify whether the number of 1-bits is even or odd, depending on the parity scheme used. If the condition is violated, an error is detected.

Example of Parity Check Code:

Let's assume we are using **even parity**.

1. **Original Data**: 1011 (4 bits of data)
   - Number of 1s in 1011 is 3, which is odd.
   - To make the number of 1s even, we append a **parity bit** of 1 (because 3 + 1 = 4, which is even).
   - **Transmitted Data**: 10111 (data + parity bit)
2. **At Receiver Side**: The receiver checks the parity.
   - Number of 1s in 10111 is 4, which is even.
   - The data is correct as per the even parity rule.

If the transmitted data were 10110 (with an incorrect parity bit):

- Number of 1s in 10110 is 3, which is odd. Therefore, the receiver detects an error because it doesn't match the expected even parity.

Hamming Distance and Parity Check

The **Hamming distance** can be used to evaluate the effectiveness of error-detection schemes like parity checking. Parity checks can detect **single-bit errors** because they ensure that the total number of 1-bits (including the parity bit) matches the parity rule (even or odd). However, they cannot detect **multiple-bit errors** where the number of flipped bits does not change the overall parity.

Example:

- **Original Data**: 1011 (Even parity → Parity bit 1 → 10111)

- **Transmitted Data**: 11110 (Hamming distance from the original = **2**)

In this case, two bits are flipped, but since we are using only a single parity bit, the error will not be detected by the parity check (as the overall number of 1s remains even). This highlights the limitation of using only a **single-bit parity check** in detecting multiple-bit errors.

Q.4.b. Define Framing. Explain character oriented framing and bit-oriented framing

Framing

Framing is the process of dividing a stream of data into manageable units called **frames**. Frames are essential in data communication as they help the sender and receiver identify the beginning and end of a message. They also facilitate the efficient handling and error detection of data as it travels over the network.

Framing can be achieved in different ways, depending on how the data is represented and transmitted. Two common types of framing are **Character-Oriented Framing** and **Bit-Oriented Framing**.

Character-Oriented Framing

In **Character-Oriented Framing** (also called **Byte-Oriented Framing**), the data is treated as a sequence of characters (bytes), where each frame consists of one or more characters, and special characters are used to delimit the beginning and end of the frame.

Characteristics:

1. **Delimiter**: Specific characters (such as **DLE** - Data Link Escape or **STX** - Start of Text and **ETX** - End of Text) are used to mark the boundaries of a frame.
2. **Frame Structure**: A frame consists of a sequence of characters (including data and control characters).
3. **Handling Special Characters**: If the special delimiter characters appear within the data, they are **escaped** by adding an escape character before them to differentiate between data and control characters.

Example:

In character-oriented framing, we might use:

- **STX** (Start of Text): Marks the start of a frame.
- **ETX** (End of Text): Marks the end of a frame.

Frame structure:

STX <data> ETX

If the data contains special characters like **STX** or **ETX**, the escape character **DLE** is used to ensure proper framing.

Example frame:

STX HelloDLESTXWorldETX

Here, the character DLE is used to escape the **STX** character in the data.

Bit-Oriented Framing

Bit-Oriented Framing treats the data as a sequence of bits, and the boundaries of the frames are defined by bit patterns rather than characters. This method is more efficient for handling larger data streams.

Characteristics:

1. **Delimiter**: A special **bit pattern** (e.g., **01111110**) is used to indicate the start and end of a frame.
2. **Bit Stuffing**: If the special bit pattern appears in the data, extra bits (called **stuffed bits**) are inserted to differentiate between data and delimiters.
3. **Efficiency**: This framing method is more efficient than character-oriented framing, especially for binary data, as it doesn't rely on character sets or escape sequences.

Example:

In bit-oriented framing, the pattern **01111110** (known as the **Flag** sequence) is used as the delimiter to mark the beginning and end of a frame.

Frame structure:

Flag <data> Flag

If the data contains the flag pattern **01111110**, **bit stuffing** is used to insert a 0 after every sequence of five consecutive 1s in the data, preventing the flag sequence from occurring in the data.

Example:

- **Data**: 01111110 (which is the flag pattern)
- **Stuffed Data**: 011111101 (the extra 1 is stuffed to avoid confusion with the flag)

Frame Example:

Flag 011111101011111100 Flag

Summary of Differences

Aspect	**Character-Oriented Framing**	**Bit-Oriented Framing**
Data Representation	Data is treated as a sequence of characters (bytes).	Data is treated as a sequence of bits.
Delimiters	Uses special characters (e.g., STX, ETX).	Uses bit patterns (e.g., 01111110 flag).

Aspect	**Character-Oriented Framing**	**Bit-Oriented Framing**
Escape Mechanism	Uses escape characters (e.g., DLE) for special characters.	Uses **bit stuffing** to prevent delimiter patterns in the data.
Efficiency	Less efficient for large data or binary data.	More efficient for large or binary data.
Error Handling	Relies on special characters for framing.	Relies on bit patterns and stuffing for integrity.

Both **character-oriented** and **bit-oriented** framing methods are widely used in data communication, with **bit-oriented framing** being more suitable for handling raw binary data, while **character-oriented framing** is simpler and often used in text-based protocols.

Q.4.c. With low diagram, explain CSMA/CA

CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)

CSMA/CA is a network protocol used to manage how devices on a shared communication medium (like Wi-Fi or wireless networks) access and transmit data. The goal is to avoid data collisions by detecting the carrier signal before transmission (carrier sensing) and implementing methods to avoid collisions (collision avoidance).

How CSMA/CA Works:

1. **Carrier Sensing**: The device listens to the channel to see if it is free or busy (i.e., whether other devices are transmitting).
2. **Collision Avoidance**:
   - If the channel is **idle**, the device waits for a random amount of time (called **backoff**), then starts transmitting.
   - If the channel is **busy**, the device will **not transmit immediately** but will **wait** until the channel becomes idle and then apply backoff before transmitting.
3. **Random Backoff**: The device waits for a random period (calculated using a backoff algorithm, usually a binary exponential backoff) to reduce the chances of collisions, especially if multiple devices are trying to transmit at the same time.
4. **Transmission**: Once the device determines that the channel is idle and after waiting for the random backoff time, it starts transmitting the data.
5. **Acknowledgment**: After transmission, the receiver sends an acknowledgment (ACK) back to the sender to confirm that the transmission was successful.

Steps of CSMA/CA

1. **Carrier Sense**: The device listens to the medium to check if it's clear for transmission. If it's idle, it proceeds; otherwise, it waits.

2. **Backoff**: If the channel is busy, the device chooses a random backoff time and waits for that period before trying again.
3. **Transmission**: After the random backoff, if the channel is still clear, the device starts sending data.
4. **ACK (Acknowledgment)**: The receiver sends an acknowledgment (ACK) to inform the sender that the transmission was successful.

CSMA/CA Flow Diagram

sql

CopyEdit

```
+-----------------------+
|  Start               |
+-----------------------+
         |
         v
  +---------------+
  |  Carrier Sense |
  +---------------+
         |
    +-----+-----+
    |         |
   Idle       Busy
    |         |
    v         v
+-----------+   +-----------------+
| Transmit   |   | Wait for Idle   |
| Data       |   | & Backoff       |
+-----------+   +-----------------+
         |
         v
    +----------+
    |  ACK     |
+----------+
    |
    v
+------------+
|  End     |
+------------+
```

Detailed Explanation of the Diagram:

1. **Carrier Sense**:
    - The device first listens to the channel to check whether it's free or occupied.
2. **Idle Channel**:
    - If the channel is idle, the device immediately proceeds with **data transmission**.
3. **Busy Channel**:
    - If the channel is busy, the device **waits** and checks the channel again after a backoff period.
4. **Transmission**:
    - Once the channel becomes idle, the device waits for a small random backoff time before starting its transmission to reduce the chances of collision with other devices trying to transmit at the same time.
5. **Acknowledgment (ACK)**:
    - After successful transmission, the receiver sends an **acknowledgment** to the sender to confirm that the data was received correctly.

Advantages of CSMA/CA:

1. **Collision Avoidance**: By waiting for a random backoff time before transmission, CSMA/CA helps avoid direct collisions between devices.
2. **Efficient for Wireless Networks**: Since it helps manage transmissions in wireless systems where collisions are more challenging to detect, CSMA/CA is crucial for protocols like **Wi-Fi**.

Limitations of CSMA/CA:

1. **Hidden Node Problem**: CSMA/CA cannot detect collisions if devices are out of range of each other but both transmit to a common receiver (hidden nodes).
2. **Performance Degradation**: If too many devices use CSMA/CA, the random backoff times may lead to delays and reduce throughput, especially in dense networks.

Q.5.a. Explain virtual-circuit approach network. in packet-switched to route the packets

Virtual-Circuit Approach in Packet-Switched Networks

A **Virtual-Circuit (VC)** approach is a method used in **packet-switched networks** where a logical connection is established between the source and destination before any data is transmitted. Once the connection is set up, packets are sent across the network using a pre-defined path that has been established for the duration of the communication session. This logical connection is referred to as a **virtual circuit**, which simulates a dedicated point-to-point connection even though the physical connection may be shared by multiple users.

In packet-switched networks, each packet is forwarded independently without the need for a dedicated physical connection. However, with virtual circuits, all packets for a given session follow the same path, ensuring that the packets reach their destination in the correct order and with reduced routing complexity.

How Virtual-Circuit Networks Work:

1. **Connection Establishment**:
    - Before any data transmission occurs, a virtual circuit must be established between the sender (source) and receiver (destination).
    - This involves negotiating the path through the network, where routers and switches store the virtual circuit information, such as the identifiers for the circuit, the path to follow, and other control information.
    - This step is similar to setting up a **call** in a circuit-switched network.
    - During this phase, routing information is exchanged, and the network ensures that a path is available between the sender and receiver.
2. **Data Transfer**:
    - Once the virtual circuit is established, data (in the form of packets) is sent along the established path.
    - The packets contain a **virtual circuit identifier (VCI)** instead of a full destination address. This identifier tells each router or switch along the path how to forward the packet.
    - All packets belonging to the same virtual circuit follow the same path, ensuring that they reach the destination in order, and eliminating the need for reassembly.
3. **Connection Termination**:
    - After the data transfer is complete, the virtual circuit is terminated, and resources along the path are freed.
    - This step involves sending termination signals to each router or switch that participated in the virtual circuit to release the allocated resources.

Components of a Virtual-Circuit Network:

1. **Virtual Circuit Identifier (VCI)**:
   - A unique identifier for each virtual circuit established between two devices.
   - The VCI is used by routers and switches to determine how to forward packets along the established path.
2. **Routing Tables**:
   - Routers and switches use routing tables that store the virtual circuit information (VCI) to forward packets.
   - The routing tables are set up during the connection establishment phase.
3. **Switches/Routers**:
   - Devices in the network (like routers or switches) are responsible for forwarding packets based on the VCI, ensuring that packets from the same virtual circuit follow the same route.

Virtual Circuit Types:

1. **Permanent Virtual Circuit (PVC)**:
   - A type of virtual circuit that is always available and is established for long-term use.
   - It remains active until explicitly terminated by the network provider (e.g., Frame Relay, ATM networks).
2. **Switched Virtual Circuit (SVC)**:
   - A virtual circuit that is dynamically set up and torn down as needed, similar to a telephone call.
   - The network creates and destroys the virtual circuit based on demand, typically used in packet-switched networks like MPLS.

Routing in Virtual-Circuit Networks:

Routing in virtual-circuit networks is different from traditional packet-switching because the path for the data transfer is pre-determined and established before transmission. Here's how routing works in virtual-circuit networks:

1. **Path Setup**:
   - Before data transmission begins, a path is set up between the sender and receiver.
   - The routers along the path store information about the virtual circuit, including the **VCI** and the path to follow.
2. **Packet Forwarding**:

    - Each packet in a virtual circuit carries a **VCI**.
    - When a router receives a packet, it checks the VCI, looks up its routing table, and forwards the packet along the established path.
    - The packet travels from router to router, following the same path each time, using the VCI to ensure proper delivery.

3. **Forwarding Table**:
    - Routers maintain a forwarding table that maps the VCI to the next hop. This table is populated during the path setup phase.
4. **No Need for Reassembly**:
    - Since all packets of a virtual circuit follow the same path, they are delivered in order and can be processed by the destination without reordering.
5. **Connection Termination**:
    - Once the communication is complete, the virtual circuit is torn down, and the VCI information is erased from the routers and switches.

Advantages of Virtual-Circuit Networks:

1. **Reliable Communication**:
    - Since the path is established in advance, there is less chance of packet loss or out-of-order delivery, making it reliable.
2. **Congestion Control**:
    - Virtual-circuit networks can provide better congestion control because the resources (routers, bandwidth) for a virtual circuit are reserved, reducing the chances of congestion during transmission.
3. **Reduced Overhead**:
    - Unlike traditional packet-switched networks, there is no need for the destination address to be included in every packet since the VCI takes care of routing.
4. **Efficient Bandwidth Use**:
    - Virtual circuits allow the network to manage resources efficiently by reserving them for the duration of the connection, preventing other users from using those resources.

Disadvantages of Virtual-Circuit Networks:

1. **Setup Delay**:
    - Establishing a virtual circuit takes time, which can introduce delay in the initial connection setup phase.

2. **Resource Allocation**:
    - Virtual circuits require the reservation of resources along the path, which may lead to inefficient resource utilization if circuits are idle for long periods.
3. **Scalability**:
    - As the network grows, managing a large number of virtual circuits may become complex and resource-intensive.

Example of Virtual-Circuit Approach (Connection Setup and Data Transfer):

1. Connection Establishment:

- **Source**: Initiates the connection request.
- **Routers**: Create routing tables to track the path.
- **Destination**: Acknowledges the connection request.
- **VCI**: A unique identifier for this connection is created.

2. Data Transfer:

- Data packets are sent from the source, and each packet is labeled with the VCI.
- Routers forward the packets along the established path, using the VCI to make forwarding decisions.

3. Connection Termination:

- After the data transfer is complete, a **tear-down message** is sent to release resources and delete the VCI information from the routers.

Q.5.b. Illustrate the working of OSPF and BGP.

Working of OSPF (Open Shortest Path First)

OSPF is a **link-state routing protocol** used to find the best path for packets as they pass through a set of connected networks. It is commonly used within **Autonomous Systems (AS)**, which are networks or groups of networks under the control of a single organization.

Working of OSPF:

1. **Link-State Advertisements (LSAs)**:
    - OSPF routers exchange **Link-State Advertisements (LSAs)** to share information about their network links (interfaces, routers, etc.) and their status (up or down).
    - Each router sends LSA packets to its neighbors about the status of its links, which helps other routers in the network create a consistent map of the entire topology.
2. **Building the Link-State Database**:

    - Routers use the received LSAs to build a **link-state database** (LSDB), which holds a detailed map of the entire network's topology.
    - This database is the same for all routers in the same OSPF area.
3. **Dijkstra's Algorithm**:
    - Once the LSDB is built, OSPF routers use **Dijkstra's Shortest Path First (SPF) algorithm** to calculate the shortest path to every possible destination.
    - This algorithm computes the best possible route from the router to all other routers based on the link state information.
4. **Routing Table**:
    - After calculating the shortest paths, each router updates its **routing table** with the best routes (shortest paths).
    - This routing table is then used to forward packets to the appropriate destination based on the shortest path.
5. **Periodic Updates**:
    - OSPF routers periodically exchange LSAs to reflect changes in the network topology (e.g., new links, link failures, etc.).
    - This ensures that the network topology remains accurate and up to date.
6. **Areas and Hierarchical Structure**:
    - OSPF divides large networks into **areas** to limit the scope of LSAs, reducing network overhead.
    - The **Backbone Area (Area 0)** connects all other areas in OSPF, making it central to the design of an OSPF network.
    - Hierarchical design helps optimize routing and manage large-scale networks.

Working of OSPF – Key Steps:

1. **Router Discovery**: Routers discover each other through **Hello packets** to form adjacencies.
2. **LSA Flooding**: Routers flood LSAs to share their link-state information.
3. **Shortest Path Calculation**: Routers use Dijkstra's algorithm to compute the shortest paths based on the LSDB.
4. **Routing Table Update**: After calculating the paths, routers update their routing tables.

OSPF Example:

Imagine a network with three routers: **R1**, **R2**, and **R3** connected in a triangular topology. Each router will send LSAs about its links, and using Dijkstra's algorithm, each router will calculate the shortest path to the other routers. Once the tables are populated, **R1** will know the best path to **R2** and **R3**, and so on.

Working of BGP (Border Gateway Protocol)

BGP is a **path vector routing protocol** used to exchange routing information between **different Autonomous Systems (ASes)**. It is the protocol used on the **internet** to route data between different networks, making it the backbone of the Internet's routing structure.

Working of BGP:

1. **BGP Peerings**:
   - BGP routers (called **BGP speakers**) establish a peering relationship with other BGP routers using a **TCP connection** (port 179).
   - Peers exchange **BGP UPDATE messages** containing routing information about network prefixes, next-hop addresses, and path attributes.
2. **Routing Information Exchange**:
   - BGP routers exchange routing updates to learn about network prefixes and reachability information from their peers.
   - Unlike OSPF, BGP does not flood the entire network with information. Instead, it only exchanges information between directly connected ASes.
3. **AS Path and Attributes**:
   - BGP uses **AS Path** to track the path that data takes across different ASes.
   - BGP also uses several other **attributes** (like **Next Hop**, **Local Preference**, **MED**, **AS Path**, and **Community**) to determine the best path. The AS Path is a list of ASes that the route has traversed.
4. **Best Path Selection**:
   - BGP routers use a set of rules to select the **best route** from multiple available routes. Some of the key attributes used in this decision process are:
     - **Longest prefix match**: Choose the route with the longest matching prefix.
     - **AS Path length**: Prefer routes with shorter AS paths.
     - **Next Hop**: Prefer routes with a reachable next hop.
     - **Local Preference**: Higher preference is given to routes with a higher local preference value.
5. **Route Aggregation**:
   - BGP supports **route aggregation**, which reduces the number of routes exchanged between ASes. Instead of advertising individual IP addresses, BGP can advertise a range of addresses.
6. **Route Updates and Withdrawals**:

    - **BGP UPDATE messages** are used to announce new routes, modify existing routes, or withdraw previously advertised routes.
    - **BGP KEEPALIVE messages** are used to maintain the session and ensure that the peering relationship is alive.
7. **Loop Prevention**:
    - BGP uses the **AS Path** to prevent routing loops. If a route advertisement contains an AS that has already been traversed, the route is rejected to prevent a loop.

BGP Example:

Consider three ASes: **AS1**, **AS2**, and **AS3** connected as follows:

- **AS1** is connected to **AS2**.
- **AS2** is connected to **AS3**.

If **AS1** wants to route data to **AS3**, BGP will advertise the route information to **AS2** and **AS3**. **AS2** will advertise its path information back to **AS1** and **AS3**. **AS1** then selects the best route based on BGP's path selection rules, considering attributes like AS Path, Next Hop, and Local Preference.

Key Differences Between OSPF and BGP:

Feature	**OSPF**	**BGP**
Protocol Type	Link-state protocol	Path-vector protocol
Scope	Interior Gateway Protocol (IGP) – within an AS	Exterior Gateway Protocol (EGP) – between ASes
Routing Information	Link-state advertisements (LSAs)	Routing updates with path attributes
Routing Algorithm	Dijkstra's Shortest Path First (SPF)	Path selection based on AS Path and other attributes
Convergence Time	Fast convergence	Slower convergence compared to OSPF
Network Topology	Suitable for smaller, single-AS networks	Suitable for large, multi-AS networks (Internet)
Metric	Cost (based on link bandwidth)	Path length (number of ASes)

Q.6.a Explain IPv6 datagram format.

IPv6 Datagram Format

The **IPv6** (Internet Protocol version 6) datagram is the unit of data that is transmitted across an IPv6 network. It is designed to overcome the limitations of IPv4, especially regarding address space. An IPv6 datagram consists of a **header** and **payload**, with the payload containing the data being transmitted, such as application data.

IPv6 Datagram Structure

An IPv6 datagram has a fixed header size of **40 bytes**, which is significantly more streamlined compared to the IPv4 header (which is 20 bytes but can be larger with options). The IPv6 header includes fields that help in routing, fragmentation, and addressing. Below is a detailed breakdown of the **IPv6 datagram format**:

IPv6 Datagram Header Format:

Field Name	**Length (Bits)**	**Description**
Version	4	Specifies the IP version (IPv6 = 6).
Traffic Class	8	Used for Differentiated Services (QoS) to prioritize packets.
Flow Label	20	Used to label packets belonging to the same flow for special handling.
Payload Length	16	Specifies the length of the payload (data) in bytes, excluding the header.
Next Header	8	Identifies the type of the next header (e.g., TCP, UDP, ICMP, etc.).
Hop Limit	8	Specifies the maximum number of hops the packet can take before being discarded.
Source Address	128	The IPv6 address of the sender.
Destination Address	128	The IPv6 address of the recipient.

Explanation of IPv6 Header Fields:

1. **Version (4 bits)**:
   - This field indicates the version of the IP protocol. For IPv6, this value is **6**. The first 4 bits of an IPv6 datagram are always set to 0110, which corresponds to version 6.
2. **Traffic Class (8 bits)**:
   - This field is used for **Differentiated Services (DS)**, allowing the network to distinguish and prioritize traffic based on type (e.g., voice, video, or general data).
   - It is similar to the Type of Service (TOS) field in IPv4.

3. **Flow Label (20 bits)**:
   - The **Flow Label** is used to identify packets that belong to the same flow. A flow is a sequence of packets from a source to a destination that requires special handling.
   - It can be used for tasks like real-time data stream management, making it useful in applications like VoIP and video conferencing.
4. **Payload Length (16 bits)**:
   - This field indicates the length of the payload (data) in the IPv6 datagram.
   - The payload length excludes the size of the IPv6 header, so this field specifies how much data follows the IPv6 header.
5. **Next Header (8 bits)**:
   - This field identifies the protocol of the layer above IPv6 (e.g., TCP, UDP, ICMP).
   - It indicates which protocol should be processed next. For example:
     - **6** for TCP
     - **17** for UDP
     - **58** for ICMPv6
6. **Hop Limit (8 bits)**:
   - This field is used to prevent packets from circulating indefinitely in the network due to routing loops.
   - The hop limit field is decremented by 1 by each router that processes the packet. When the hop limit reaches 0, the packet is discarded.
7. **Source Address (128 bits)**:
   - This field specifies the IPv6 address of the sender of the packet. It allows the recipient to know where the packet came from.
   - IPv6 addresses are represented in hexadecimal format (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).
8. **Destination Address (128 bits)**:
   - This field contains the destination IPv6 address where the packet is being sent. It specifies the end-point for the packet.
   - Like the source address, it is written in hexadecimal format and has 128 bits.

IPv6 Datagram Format Example:

Let's assume we have the following values for the IPv6 header:

Field Name	Value
Version	6
Traffic Class	0x00
Flow Label	0x00000
Payload Length	0x0044 (68 bytes)
Next Header	0x06 (TCP)
Hop Limit	0x40 (64 hops)
Source Address	2001:0db8:85a3:0000:0000:8a2e:0370:7334
Destination Address	2001:0db8:85a3:0000:0000:8a2e:0370:1234

In this case:

- The datagram version is **6** (IPv6).
- The **Traffic Class** is set to **0** (no special handling).
- The **Flow Label** is **0x00000**.
- The **Payload Length** is **68 bytes**.
- The **Next Header** is **TCP** (protocol 6).
- The **Hop Limit** is set to **64** hops.
- The **Source Address** is 2001:0db8:85a3:0000:0000:8a2e:0370:7334.
- The **Destination Address** is 2001:0db8:85a3:0000:0000:8a2e:0370:1234.

IPv6 Datagram Payload:

The payload of the IPv6 datagram contains the data being transmitted (such as TCP, UDP, or ICMP data). The exact structure of the payload depends on the protocol indicated in the **Next Header** field. For example:

- If the **Next Header** field indicates TCP (protocol 6), the payload will contain a TCP segment.
- If the **Next Header** field indicates UDP (protocol 17), the payload will contain a UDP datagram.

Summary of IPv6 Datagram Format:

- The **IPv6 datagram** has a **fixed 40-byte header**, followed by a **payload**.
- It contains essential fields for routing, traffic management, and data delivery.

- Key fields include **Source Address**, **Destination Address**, **Flow Label**, **Hop Limit**, and **Next Header**, which are used for routing and data forwarding across the IPv6 network.
- IPv6 allows for a larger address space (128-bit addresses) and better handling of different types of traffic compared to IPv4.

Q.6.b. Write an Dijikstra's algorithm to compute shortest path through graph

Dijkstra's algorithm is a well-known algorithm used to find the shortest path from a source node to all other nodes in a graph with non-negative edge weights. Here's the step-by-step explanation and the Python code for Dijkstra's algorithm:

Dijkstra's Algorithm – Explanation

1. **Initialization**:
   - Start with a source node. Initialize the shortest distance from the source to itself as 0 and all other nodes as infinity (∞).
   - Keep track of nodes whose shortest distance is already finalized.
2. **Relaxation**:
   - Select the node with the smallest tentative distance (initially the source node).
   - For each neighbor of this node, check if the tentative distance through the current node is smaller than the previously recorded distance for that neighbor. If so, update the shortest distance for that neighbor.
3. **Repeat**:
   - Mark the current node as visited (finalized).
   - Select the next unvisited node with the smallest tentative distance and repeat the relaxation step until all nodes have been visited.
4. **Termination**:
   - The algorithm terminates when all nodes have been visited, and the shortest distance from the source node to every other node has been calculated.

Dijkstra's Algorithm – Python Implementation

```
def dijkstra(graph, start):
  # Number of nodes in the graph
  num_nodes = len(graph)

  # Distance table (initialize all distances as infinity, except the start node)
  dist = {node: float('inf') for node in range(num_nodes)}
  dist[start] = 0
```

```
    # Priority queue (min-heap) for selecting the node with the smallest distance
    pq = [(0, start)]  # (distance, node)

    while pq:
        # Get the node with the smallest tentative distance
        current_dist, current_node = heapq.heappop(pq)

        # If the current distance is greater than the already found shortest distance, skip it
        if current_dist > dist[current_node]:
            continue

        # Explore the neighbors of the current node
        for neighbor, weight in graph[current_node]:
            # Calculate the tentative distance to the neighbor
            new_dist = current_dist + weight

            # If a shorter path to the neighbor is found, update the distance and push to the priority
queue
            if new_dist < dist[neighbor]:
                dist[neighbor] = new_dist
                heapq.heappush(pq, (new_dist, neighbor))

    return dist

# Example graph represented as an adjacency list
# graph[node] = [(neighbor, weight), ...]
graph = {
    0: [(1, 4), (2, 1)],
    1: [(0, 4), (2, 2), (3, 5)],
    2: [(0, 1), (1, 2), (3, 8), (4, 10)],
    3: [(1, 5), (2, 8), (4, 2)],
    4: [(2, 10), (3, 2)]
}


# Compute the shortest paths from node 0
shortest_paths = dijkstra(graph, 0)


# Display the shortest distance from node 0 to every other node
print("Shortest distances from node 0:")
for node, dist in shortest_paths.items():
    print(f"Node {node}: {dist}")
```

Explanation of Code:

1. **Graph Representation**:
   - The graph is represented using an adjacency list. Each node has a list of tuples, where each tuple contains a neighboring node and the weight of the edge connecting them.
2. **Initialization**:
   - dist: This dictionary stores the shortest distance from the start node to each node, initialized to infinity (float('inf')), except for the start node, which is set to 0.
3. **Priority Queue**:
   - A priority queue (min-heap) is used to always expand the node with the smallest tentative distance. This ensures the algorithm always processes the closest unvisited node.
4. **Relaxation**:
   - For each node processed, the algorithm checks its neighbors and updates their tentative distances if a shorter path is found.
5. **Output**:
   - After running the algorithm, the shortest distances from the source node to all other nodes are printed.

Example Execution:

Given the graph:

scss

CopyEdit

```
0 --(4)-- 1 --(2)-- 2 --(1)-- 0
```

```
\     |   |   /
 (1)   (2)  (8) (10)
  \    |    |  /
   2 --(5)-- 3
      |    |    |
     (2) (10) (8)
```

For the source node 0, the output would be:

yaml

CopyEdit

```
Shortest distances from node 0:
Node 0: 0
Node 1: 4
Node 2: 1
Node 3: 9
Node 4: 11
```

This shows that the shortest path from node 0 to:

- Node 1 is 4
- Node 2 is 1
- Node 3 is 9
- Node 4 is 11

Time Complexity:

- **Time Complexity**: O(Elog⁡V)O(E \log V)O(ElogV), where EEE is the number of edges and VVV is the number of nodes.
    - The priority queue operations (insert and extract-min) take logarithmic time.
    - For each node, we process each of its neighbors.
- **Space Complexity**: O(V+E)O(V + E)O(V+E), where VVV is the number of nodes and EEE is the number of edges, due to the storage of the graph and the distance table.

Q.6.c. Write a note on Routing Information Protocol (RIP) algorithm.

Routing Information Protocol (RIP)

The **Routing Information Protocol (RIP)** is one of the oldest distance-vector routing protocols, widely used in small and medium-sized networks. RIP uses hop count as its routing metric, where the number of hops between the source and the destination determines the best route. It is a simple and easy-to-configure protocol, but it has some limitations in terms of scalability and efficiency.

Key Characteristics of RIP:

1. **Routing Metric**:
    - RIP uses **hop count** as the metric to determine the shortest path between routers. Each hop represents one router passed through to reach the destination.
    - The maximum allowed hop count is **15**. This means that a destination is considered unreachable if the hop count exceeds 15, which limits RIP to small networks.
2. **Distance Vector Protocol**:
    - RIP is a **distance-vector protocol**, which means that each router periodically shares its routing table with its immediate neighbors.
    - A router updates its routing table based on the information received from its neighbors. The router will adopt the shortest path (lowest hop count) for each destination.
3. **Periodic Updates**:
    - RIP routers send **updates** to their neighbors every **30 seconds**. This ensures that routing information stays updated, but it can lead to network congestion and overhead if the network is large.
    - RIP also supports **triggered updates**, which are sent immediately when a significant change in the network occurs (e.g., a route failure).
4. **Routing Tables**:
    - Each router maintains a **routing table** that contains the destination network, the next-hop router, and the metric (hop count) to reach that destination.
    - The routing table is periodically updated based on the information received from neighboring routers.
5. **Convergence**:
    - **Convergence** refers to the process of all routers in the network agreeing on the best routes. RIP can take a long time to converge in the case of network topology changes, especially in larger networks.
6. **Limitations**:
    - **Scalability**: RIP is not well-suited for large networks due to its hop count limit and the overhead of periodic updates.
    - **Slow Convergence**: RIP can experience slow convergence, which might cause routing loops or suboptimal routing during network changes.
    - **Loop Prevention**: RIP uses mechanisms like **split horizon**, **route poisoning**, and **hold-down timers** to prevent routing loops.

RIP Versions:

There are two main versions of RIP:

1. **RIP version 1 (RIP v1)**:
   - RIP v1 was the original version and operates purely on **classful IP addressing**. It does not support subnetting and cannot carry subnet mask information in the routing updates.
   - RIP v1 only supports **IPv4** addresses.
   - It uses **broadcast** for sending routing updates to all neighboring routers.
2. **RIP version 2 (RIP v2)**:
   - RIP v2 was introduced to address the limitations of RIP v1, including support for **classless routing** (CIDR) and the inclusion of **subnet mask** information in routing updates.
   - RIP v2 also supports **multicast** (using address 224.0.0.9) instead of broadcast for routing updates, which reduces unnecessary traffic.
   - It supports both **IPv4** and **IPv6**.
3. **RIPng (RIP next generation)**:
   - RIPng is an extension of RIP v2 and is designed to support **IPv6**.
   - It is essentially RIP v2 modified to support IPv6 addressing and other features related to IPv6.

Working of RIP:

1. **Initialization**:
   - When a router starts up, it initializes its routing table with directly connected networks and the hop count is set to 0 for these routes.
   - The router sends a **routing update** to its neighbors, sharing the list of its direct routes and their associated hop counts.
2. **Route Calculation**:
   - When a router receives a routing update from a neighbor, it updates its routing table by checking if any new routes or better routes (with a lower hop count) are available.
   - The router then recalculates the best route for each destination and updates its neighbors with the new information.
3. **Update and Propagation**:
   - Every 30 seconds, each router sends a **routing update** to its neighbors.
   - The update contains the full routing table, including information about the current best path to each destination.
4. **Loop Prevention**:
   - **Split Horizon**: This technique prevents a router from advertising a route back to the router from which it learned that route.

- **Route Poisoning**: When a route becomes invalid, the router advertises the route with a hop count of 16 (which is considered "infinity" and unreachable).
- **Hold-down Timers**: When a route is marked as unreachable, it is placed in a "hold-down" state for a specified time, during which the router does not accept updates for that route.

RIP Packet Format:

RIP routing updates are sent in **RIP packets**, and the format includes:

- **Command**: Indicates whether the packet is a request or a response.
- **Version**: The RIP version being used (1, 2, or RIPng).
- **Authentication**: Optional authentication field for security purposes (supported in RIP v2).
- **Routing Entries**: A series of entries containing:
    - Destination network address
    - Metric (hop count)
    - Next-hop router (in some cases)

Example of RIP Route Table:

Destination Network	**Next Hop**	**Metric (Hop Count)**
192.168.1.0/24	Directly connected	0
192.168.2.0/24	192.168.1.1	1
192.168.3.0/24	192.168.2.1	2
192.168.4.0/24	192.168.3.1	3

- In this table:
    - **192.168.1.0/24** is directly connected (hop count 0).
    - **192.168.2.0/24** can be reached by one hop through **192.168.1.1**.
    - **192.168.3.0/24** is reached by two hops, first through **192.168.2.1**, then through **192.168.1.1**.
    - **192.168.4.0/24** is reached by three hops.

Advantages of RIP:

- **Simplicity**: RIP is easy to configure and understand.
- **Widely Supported**: RIP is supported by most routers and network devices.
- **Compatibility**: It is a well-established protocol and works well for small networks.

Disadvantages of RIP:

- **Limited Scalability**: The 15-hop limit makes RIP unsuitable for larger networks.

- **Slow Convergence**: RIP can take a long time to react to network topology changes.
- **Bandwidth Intensive**: Frequent updates (every 30 seconds) can lead to unnecessary bandwidth consumption in larger networks.

Q.7.a.Explain Go-Back-N protocol working.

Go-Back-N Protocol (GBN)

The **Go-Back-N (GBN)** protocol is a type of **Automatic Repeat reQuest (ARQ)** protocol used in reliable data communication. It is a ** sliding window protocol** in which the sender can send several frames before needing an acknowledgment for the first one, but the receiver is required to acknowledge frames in order.

In Go-Back-N, the sender can transmit multiple frames in a single go (without waiting for an acknowledgment for each individual frame), but the receiver can only accept frames in order. If any frame is lost or corrupted, all subsequent frames need to be retransmitted, starting from the lost/corrupted frame.

Key Features of Go-Back-N (GBN):

1. **Windowing Mechanism**: Both the sender and receiver maintain a **sliding window**. The size of the window is denoted by **N**, where **N** is the number of frames that can be sent by the sender before receiving an acknowledgment. The sender can send **N** frames at a time, but the receiver can only accept frames in the correct order.
2. **Sequence Numbers**: Each frame transmitted by the sender is assigned a sequence number. The sequence number helps the receiver distinguish between frames and ensures that they can be processed in the correct order.
3. **Cumulative Acknowledgments**: The receiver sends an acknowledgment (ACK) for the last correctly received frame. If a frame is lost or an error occurs, the receiver will not acknowledge the frame, and the sender will retransmit all frames starting from the missing one.
4. **Retransmissions**: If the sender does not receive an acknowledgment for a frame within a certain time (due to loss or errors), it will **retransmit the frame** along with all subsequent frames, regardless of whether they were correctly received.
5. **Timer**: The sender uses a **timer** for each frame it sends. If the timer expires before an acknowledgment is received, the sender will retransmit the frame, and all frames after it.

Working of Go-Back-N Protocol:

The working of Go-Back-N ARQ involves the sender and receiver maintaining a sliding window, sending frames, receiving acknowledgments, and retransmitting lost or erroneous frames.

Here is how it works step by step:

1. Sender Operation:

- The sender has a window of size **N**, and it can send up to **N** frames before receiving an acknowledgment. For example, if **N = 4**, the sender can send frames **0, 1, 2, 3** without waiting for an acknowledgment.

- The sender keeps a timer for each frame sent. When a frame is sent, the timer starts.
- If the acknowledgment for a frame is received within the timeout period, the sender slides the window forward by one frame.
- If the acknowledgment is not received within the timeout period, the sender **retransmits the frame and all subsequent frames**.

2. Receiver Operation:

- The receiver has a **window size of 1**, meaning it can only accept frames in order. For example, the receiver will accept frame **0**, then frame **1**, then frame **2**, and so on.
- The receiver sends an **acknowledgment (ACK)** for the highest numbered frame that has been received in sequence.
- If the receiver detects a missing or out-of-order frame, it discards the frame and sends an ACK for the last correctly received frame, indicating that the sender needs to retransmit from the missing frame.

3. Acknowledgments:

- **Cumulative ACK**: The receiver acknowledges the last successfully received frame. For instance, if frames **0, 1, and 2** are received correctly, the receiver sends an acknowledgment for frame **2**, indicating that the sender can move on to the next set of frames.
- **Retransmission of Lost Frames**: If any frame is lost or corrupted, the receiver will send an acknowledgment for the last correctly received frame (e.g., frame **2**). The sender, upon receiving the acknowledgment for frame **2**, knows that frame **3** and any subsequent frames must be retransmitted.

Example of Go-Back-N Protocol:

Assume **N = 4**, and the sender wants to transmit **frames 0 to 6**.

1. The sender sends frames **0, 1, 2, 3** (since N=4).
2. The receiver correctly receives **frames 0, 1, 2**, but **frame 3** is lost.
3. The receiver sends an **ACK for frame 2** (the last correctly received frame).
4. The sender, upon receiving **ACK for frame 2**, realizes that **frame 3** is missing and retransmits **frames 3, 4, 5, 6**.
5. The receiver correctly receives **frames 3, 4, 5, 6** and sends **ACK for frame 6**.
6. The sender slides its window and transmits the next set of frames (if necessary).

Diagram of Go-Back-N Protocol:

```
Sender                          Receiver

+----------------------------------+  +-----------------------------+

| Window Size: 4                  |  | Window Size: 1              |

| Frame 0  |  Frame 1  |  Frame 2  |  Frame 3  |                 |
+---------------------------------+  +----------------------------+
|  [ 0, 1, 2, 3 ]   --->          |  ---> Frame 0 is received     |
|   Timer starts for each frame   |  ---> Frame 1 is received     |
|    (t0)   (t1)   (t2)   (t3)     |  ---> Frame 2 is received     |
|  ---> Acknowledgment for frame 2 |  ---> Frame 3 is lost         |
|  ---> Retransmit frames 3,4,5,6  |  ---> Acknowledgment for 2    |
+---------------------------------+  +----------------------------+
|   Frames 3, 4, 5, 6              |   ---> Retransmit frames 3,4,5,6|
|  ---> Window slides to 1,2,3,4   |  ---> Acknowledgment for 6    |
+---------------------------------+  +----------------------------+
```

Key Concepts in Go-Back-N:

- **Window Size (N)**: The number of frames the sender can send before needing an acknowledgment.
- **Sequence Numbering**: Sequence numbers help distinguish between different frames.
- **Cumulative Acknowledgments**: The receiver acknowledges all frames up to the highest one correctly received.
- **Retransmissions**: If a frame is lost or corrupted, all subsequent frames need to be retransmitted.

Advantages of Go-Back-N:

- **Efficient Use of Bandwidth**: The sender can send multiple frames without waiting for each individual acknowledgment.
- **Simple to Implement**: Go-Back-N is easy to understand and implement.

Disadvantages of Go-Back-N:

- **Retransmission Overhead**: If one frame is lost, all subsequent frames must be retransmitted, which can be inefficient.
- **Wasted Bandwidth**: The sender may retransmit frames that were correctly received by the receiver.
- **Limited by Window Size**: Larger window sizes require more buffer space and greater complexity.

Conclusion:

Go-Back-N is a reliable, sliding window protocol that uses cumulative acknowledgments. While it offers simplicity and some bandwidth efficiency, its main drawback is the inefficiency in retransmitting frames, especially in the case of lost or corrupted frames. This makes it suitable for scenarios where packet loss is minimal but less efficient in high-loss environments.

Q.7.b.With neat sketch, explain three-way handshaking of ICP connection establishment

Three-Way Handshaking for TCP Connection Establishment

The **three-way handshake** is the process used to establish a connection between a client and a server in **TCP (Transmission Control Protocol)**. It ensures both sides are ready to communicate and that they agree on the parameters for the communication, such as sequence numbers. The three-way handshake involves three steps:

1. **SYN (Synchronize)**: The client initiates the connection by sending a SYN message to the server.
2. **SYN-ACK (Synchronize-Acknowledge)**: The server responds to the client's SYN message by sending a SYN-ACK.
3. **ACK (Acknowledge)**: The client acknowledges the server's SYN-ACK message, completing the handshake.

Step-by-Step Explanation of the Three-Way Handshake:

Step 1: SYN (Client to Server)

- The client sends a **SYN** (Synchronize) packet to the server to begin the connection.
- The SYN packet contains a randomly generated **Initial Sequence Number (ISN)**, which is a unique identifier for the connection.

Client → Server:

- Packet: SYN, Sequence Number = X

Step 2: SYN-ACK (Server to Client)

- The server receives the SYN packet from the client.
- The server responds by sending a **SYN-ACK** (Synchronize-Acknowledge) packet back to the client.
- The SYN-ACK packet contains:
    - The **Acknowledgment Number** (the client's ISN + 1) to acknowledge the receipt of the client's SYN.
    - The server's own **ISN**.

Server → Client:

- Packet: SYN, Acknowledgment Number = X + 1, Sequence Number = Y

Step 3: ACK (Client to Server)

- The client receives the SYN-ACK packet from the server.
- The client sends an **ACK** (Acknowledge) packet back to the server to confirm the establishment of the connection.

- The ACK packet contains:
  - The **Acknowledgment Number** (the server's ISN + 1) to acknowledge the receipt of the server's SYN.
  - The **Sequence Number** (the client's ISN + 1) to acknowledge the receipt of the server's SYN.

Client → Server:

- Packet: Acknowledgment Number = Y + 1, Sequence Number = X + 1

Once the server receives the ACK packet from the client, the connection is fully established, and data transfer can begin.

Illustration of Three-Way Handshake:

```
Client                    Server

  |                         |

  |------------ SYN (X) ---------------> |  (Client sends SYN to server with ISN = X)

  |                         |

  | <----------- SYN-ACK (X+1, Y) -------|  (Server responds with SYN and Acknowledgment of SYN)

  |                         |

  |------------ ACK (Y+1, X+1) ---------> |  (Client acknowledges with ACK, completing the handshake)

  |                         |

  |        Connection Established    |

  |                         |
```

Details of the Three-Way Handshake:

1. **SYN from Client**: The client sends a SYN packet to the server with a random initial sequence number (X). This marks the beginning of the connection request.
2. **SYN-ACK from Server**: Upon receiving the SYN, the server responds with a SYN-ACK. It acknowledges the client's sequence number by sending **X + 1** as the acknowledgment number. The server also includes its own initial sequence number (Y).
3. **ACK from Client**: The client acknowledges the server's SYN-ACK with an ACK packet, which contains the server's sequence number **Y + 1** as the acknowledgment number and **X + 1** as its own sequence number. This confirms that the client has received the SYN-ACK.

After the three-way handshake is completed, the client and server are synchronized, and the connection is established, allowing data transfer to begin.

Purpose of the Three-Way Handshake:

- **Synchronization of Sequence Numbers**: Both the client and server agree on the sequence numbers for data transmission.

- **Ensuring Reliability**: Both sides confirm that they are ready to send and receive data.
- **Establishing Communication**: The handshake helps in establishing a reliable communication channel, ensuring both sides are aware of each other and can handle the connection.

Q.8.a. With an outline, explain selective repeat protocol.

Selective Repeat Protocol

The **Selective Repeat Protocol** (SR) is a type of **Automatic Repeat reQuest (ARQ)** protocol used for reliable data communication. It is an enhancement of the **Go-Back-N Protocol** and provides more efficient handling of lost or corrupted frames by only retransmitting the specific frames that were lost or erroneous. In Selective Repeat, the sender can send multiple frames without waiting for an acknowledgment for each individual frame, but only the lost or erroneous frames are retransmitted.

Key Features of Selective Repeat Protocol:

1. **Sliding Window Mechanism**:
   - Both the sender and receiver maintain a **sliding window**. The sender's window size is **N**, which is the number of frames it can send before receiving an acknowledgment. The receiver's window size is typically **N/2** (for efficiency), and it can accept out-of-order frames.
2. **Sequence Numbers**:
   - Frames are assigned **sequence numbers** to differentiate between them. The sender uses these sequence numbers to identify each frame, and the receiver uses them to detect whether a frame is missing or out-of-order.
3. **Acknowledgments**:
   - The receiver sends **individual acknowledgments (ACKs)** for each correctly received frame. It can acknowledge a frame out-of-order if the window allows.
   - If a frame is lost or corrupted, the receiver will not acknowledge it, prompting the sender to retransmit only that specific frame.
4. **Retransmission of Lost Frames**:
   - The sender maintains a **buffer** of frames that have been sent but not acknowledged.
   - If the sender does not receive an acknowledgment for a particular frame within a given time (due to loss or corruption), it will **retransmit only that specific frame**, rather than retransmitting all subsequent frames like in Go-Back-N.

Working of Selective Repeat Protocol:

1. Sender's Operation:

- The sender can send multiple frames without waiting for an acknowledgment for each individual frame. The sender's window size (N) determines how many frames can be sent at once.
- The sender starts a **timer** for each transmitted frame.

- When the sender receives an acknowledgment for a specific frame, it slides the window forward.
- If the acknowledgment for a frame is not received within a timeout period, the sender retransmits only that specific frame (not all frames like Go-Back-N).
- The sender can also manage its window to avoid sending frames for which there may be insufficient buffer space on the receiver side.

2. Receiver's Operation:

- The receiver can accept frames out of order, but it will store them in a buffer until the missing frames are received.
- The receiver sends an acknowledgment for each frame received in order, and each acknowledgment contains the sequence number of the next expected frame.
- If a frame is missing, the receiver will not send an acknowledgment for it, prompting the sender to retransmit only the missing frame.
- The receiver can buffer out-of-order frames, which will be processed once the missing frames arrive.

3. Acknowledgment:

- Each correctly received frame is acknowledged by the receiver.
- The acknowledgment packet sent by the receiver includes the **next expected sequence number**. For example, if the receiver has correctly received frames 0, 1, and 3, it will send an acknowledgment for **frame 2** (the next expected frame).
- If any frame is missing or corrupted, the receiver does not acknowledge it, which indicates that the sender must retransmit only that frame.

Illustration of Selective Repeat Protocol:

Assume the sender's window size is 4 and the receiver's window size is 3. The sender wants to send frames **0, 1, 2, 3, 4, 5**.

```
Sender                        Receiver

+---------------------------------+  +-----------------------------+

| Window Size: 4                |  | Window Size: 3            |

| Frame 0  | Frame 1  | Frame 2  | Frame 3  | Frame 4  | Frame 5  |  |

+---------------------------------+  +-----------------------------+

| ---> Send Frame 0, 1, 2, 3      |  ---> Frame 0 is received     |

| ---> Send Frame 4, 5            |  ---> Frame 1 is received     |

| ---> Frame 2 is lost            |  ---> Frame 3 is lost         |

| ---> Wait for Acknowledgments    |  ---> Frame 4 is received     |

| ---> Retransmit Frame 2 (only)   |  ---> Acknowledgment for Frame 1 |
| ---> Frame 3 is retransmitted    |  ---> Acknowledgment for Frame 4 |
+----------------------------------+  +-----------------------------+
| ---> Frame 2 and 3 are received |  ---> Frame 2 is received     |
| ---> Acknowledgment for 3       |  ---> Acknowledgment for Frame 2 |
+----------------------------------+  +-----------------------------+
```

Advantages of Selective Repeat Protocol:

1. **Efficient Use of Bandwidth**: Selective Repeat is more efficient than Go-Back-N, as only the lost or corrupted frames are retransmitted, not all subsequent frames.
2. **Lower Retransmission Overhead**: Since only specific frames are retransmitted, this reduces the amount of retransmission, thus saving bandwidth.
3. **Better for High Error Environments**: Selective Repeat works better than Go-Back-N in environments with a higher rate of packet loss, as it avoids retransmitting frames that have been successfully received.

Disadvantages of Selective Repeat Protocol:

1. **Complexity**: The protocol is more complex to implement than Go-Back-N, as it requires the receiver to buffer out-of-order frames and manage acknowledgments for each frame individually.
2. **Receiver Buffer Requirements**: The receiver must have sufficient buffer space to store out-of-order frames, which can increase memory requirements, especially in high-throughput scenarios.
3. **Potential for Out-of-Order Delivery**: Since frames can be received out of order, the protocol needs mechanisms to ensure that they are processed in the correct order.

Q.8.b List and explain various services provided by User Datagram Protocol (UDP)

Services Provided by User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a connectionless, lightweight protocol that operates at the transport layer of the OSI model. Unlike Transmission Control Protocol (TCP), UDP does not establish a connection before transmitting data and does not guarantee reliable delivery. Despite its lack of reliability, UDP provides several services that are useful in specific use cases where speed is more important than reliability.

Here are the key services provided by **UDP**:

1. Connectionless Communication

- **Definition**: UDP is a **connectionless protocol**, meaning there is no need to establish a connection between the sender and receiver before data transmission. Each data packet (datagram) is sent independently.
- **Implication**: This service minimizes the overhead required for establishing and maintaining a connection, making UDP ideal for applications where quick transmission is more critical than reliability (e.g., real-time applications like video streaming, online gaming).

2. Unreliable Delivery

- **Definition**: UDP does **not guarantee** the delivery of packets. There is no mechanism for acknowledgment, retransmission, or sequencing of packets. If a packet is lost or corrupted, it is not retransmitted by UDP.
- **Implication**: This service results in faster communication but at the cost of reliability. Applications that use UDP are typically designed to handle packet loss and can tolerate some loss of data, such as voice or video communication.

3. No Flow Control

- **Definition**: UDP does not perform any flow control. It does not regulate the pace at which data is sent and does not ensure that the receiver can handle incoming data at the sender's rate.
- **Implication**: This means that UDP allows the sender to transmit data as fast as possible without waiting for feedback from the receiver, potentially leading to network congestion or packet loss in scenarios where the receiver is overwhelmed.

4. No Error Recovery

- **Definition**: UDP provides **error detection** but does not provide error recovery. The protocol includes a **checksum** to detect errors in the data being transmitted, but there is no mechanism to request retransmissions or repair corrupted data.
- **Implication**: While UDP can detect some errors in the data, it relies on higher layers (such as the application layer) to handle any necessary corrections if needed.

5. Message Integrity (Checksum)

- **Definition**: UDP includes a **checksum** field in its header to ensure the integrity of the data. The checksum is used to verify that the data has not been altered during transmission.
- **Implication**: If the checksum fails, the receiver will discard the packet. However, the protocol does not request retransmission of the data, leaving it up to the application to handle any necessary recovery.

6. Multiplexing and Demultiplexing

- **Definition**: UDP provides multiplexing and demultiplexing services, allowing multiple applications to send and receive data over the network simultaneously. This is achieved by using **ports** to distinguish different application data streams.
- **Implication**: UDP supports communication between different processes or applications running on the same device by providing each application with a unique port number, enabling multiple applications to share the same network connection.

7. Low Overhead

- **Definition**: UDP has a **small header size** (8 bytes) compared to TCP, which makes it a low-overhead protocol. This is because UDP does not include features like sequencing, acknowledgment, or flow control that add extra information to the packet.

- **Implication**: The low overhead makes UDP more efficient for transmitting small amounts of data or data that does not require reliability, making it suitable for real-time or latency-sensitive applications.

8. Low Latency

- **Definition**: Since UDP does not establish a connection, does not guarantee delivery, and does not provide acknowledgment or flow control, the protocol has a **low latency**.
- **Implication**: UDP is commonly used in applications where timely delivery is more important than reliable delivery, such as in voice over IP (VoIP), online gaming, and live streaming, where a slight delay in transmission can significantly degrade the user experience.

9. Broadcast and Multicast Support

- **Definition**: UDP supports **broadcast** and **multicast** communication, allowing a sender to transmit data to multiple receivers at once.
- **Implication**: This feature is useful for applications that need to send data to multiple receivers simultaneously, such as network discovery protocols, video conferencing, or streaming services.

Summary of Services Provided by UDP:

Service	**Description**
Connectionless	No need for connection establishment.
Unreliable Delivery	No guarantee of data delivery, no retransmissions.
No Flow Control	No regulation of data transmission rate.
No Error Recovery	No retransmission of lost or corrupted packets.
Message Integrity (Checksum)	Error detection with checksum for data integrity.
Multiplexing/Demultiplexing	Allows communication between multiple applications using unique ports.
Low Overhead	Minimal header size for efficient data transfer.
Low Latency	No connection setup, allowing faster data transmission.
Broadcast/Multicast	Supports communication to multiple receivers.

Use Cases of UDP:

- **Real-time applications**: Such as video and voice streaming, where speed is more important than ensuring every packet is delivered.
- **Online gaming**: Where small data loss is often acceptable and fast transmission is critical.
- **DNS (Domain Name System)**: For quick queries where time-sensitive responses are required.

- **Simple request-response protocols**: For applications that send a short request and receive a quick response without establishing a full connection.

Q 9.a. Briefly explain Secure Shell (SSH).

Secure Shell (SSH)

Secure Shell (SSH) is a cryptographic network protocol used to provide secure communication over an unsecured network, such as the internet. SSH allows users to securely access remote systems, execute commands, transfer files, and perform administrative tasks. It is widely used for secure remote login and other secure network services, replacing older protocols like **Telnet** and **rsh**, which transmitted data (including passwords) in plain text and were vulnerable to interception.

Key Features of SSH:

1. **Encryption**:
   - SSH encrypts the data transmitted between the client and the server, ensuring that it cannot be read by unauthorized users. This protects sensitive information like usernames, passwords, and commands from being exposed to attackers.
2. **Authentication**:
   - **Public Key Authentication**: SSH supports authentication using a pair of cryptographic keys (public and private keys). The server holds the **public key**, and the client uses the **private key** to prove its identity.
   - **Password Authentication**: In addition to public-key authentication, SSH can also use passwords for authentication, though this is considered less secure.
3. **Integrity**:
   - SSH ensures data integrity by using cryptographic hash functions to verify that data has not been altered during transmission.
4. **Port Forwarding**:
   - SSH can be used to forward network ports, enabling users to securely tunnel data from one network to another. This allows for secure browsing, file sharing, and accessing services that are otherwise restricted or unsafe.
5. **Secure File Transfer**:
   - SSH includes tools like **SFTP (Secure File Transfer Protocol)** and **SCP (Secure Copy Protocol)**, which are used for securely transferring files between systems over an SSH connection.
6. **Session Management**:
   - SSH allows multiple commands or applications to run over the same secure session, reducing the need to repeatedly authenticate.
7. **X11 Forwarding**:

    - SSH supports forwarding of X11 graphical applications. This allows users to run graphical applications on a remote server and have them display locally, securely, over the SSH connection.

Components of SSH:

1. **SSH Client**: The software used by the client to initiate a connection to an SSH server. Examples of SSH clients are **OpenSSH**, **PuTTY**, and **WinSCP**.
2. **SSH Server**: The software running on the remote machine that listens for incoming SSH connections. **OpenSSH Server** is a popular implementation on Unix-like systems.
3. **SSH Protocol**: The protocol defines how communication is established, authenticated, and encrypted between the client and server. SSH uses TCP port 22 by default.

How SSH Works (Overview):

1. **Session Initialization**:
    - The client initiates the connection to the SSH server.
2. **Key Exchange**:
    - The client and server exchange cryptographic keys to establish a secure channel. This process involves negotiating encryption algorithms and generating shared secrets.
3. **Authentication**:
    - The client authenticates itself to the server using either public key authentication or password authentication.
4. **Data Exchange**:
    - Once authenticated, the client can send encrypted commands or files to the server. All communication is securely encrypted and transmitted over the established SSH session.
5. **Termination**:
    - The session ends when the client or server closes the connection, ensuring that no residual data is left exposed.

Common Use Cases of SSH:

- **Remote Login**: SSH is commonly used to remotely log into Unix-like servers and systems to perform administrative tasks.
- **Secure File Transfers**: Using tools like **SFTP** and **SCP**, SSH enables secure file transfers between remote systems.
- **Remote Command Execution**: System administrators use SSH to remotely execute commands on servers without the need to physically access them.
- **Tunneling and Port Forwarding**: SSH is often used to create secure tunnels for other services, such as securing database connections or web traffic.

Q.9.b. Write a note on Request message and response message formats of HTTP

HTTP Request and Response Message Formats

The **Hypertext Transfer Protocol (HTTP)** is the foundation of data communication on the World Wide Web. It follows a client-server model where the client sends requests to the server, and the server responds with appropriate data. The communication between the client and the server is facilitated by two types of messages: **Request Messages** and **Response Messages**.

1. HTTP Request Message Format

An HTTP **request message** is sent by the client (usually a browser or application) to the server to request a resource or perform an action. The request message consists of the following components:

1. **Request Line**:
    - The request line indicates the type of HTTP request (method), the resource being requested, and the HTTP version.
    - **Format**:

<Method> <Request-URI> <HTTP-Version>

- **Method**: Specifies the HTTP method used. Common methods include:
    - GET: Requests a resource.
    - POST: Sends data to the server (e.g., form submission).
    - PUT: Uploads data to a specified resource.
    - DELETE: Removes a specified resource.
    - HEAD: Requests only the headers of a resource.
    - OPTIONS: Requests supported HTTP methods for a resource.
- **Request-URI**: The URI (Uniform Resource Identifier) specifies the resource being requested. For example, /index.html or /api/data.
- **HTTP-Version**: The version of HTTP being used (e.g., HTTP/1.1).

Example:

GET /index.html HTTP/1.1

2. **Headers**:
    - The headers contain additional metadata about the request, such as the type of content being sent, the type of data accepted, or authentication credentials.
    - Each header is a key-value pair, and multiple headers can be included in a request.
    - Common headers include:
        - Host: Specifies the domain name of the server.
        - User-Agent: Identifies the client making the request (browser, app, etc.).

            - Accept: Specifies the media types the client can accept (e.g., text/html, application/json).
            - Content-Type: Specifies the media type of the body of the request (e.g., application/x-www-form-urlencoded for form data).
            - Authorization: Contains credentials for authenticating the client (e.g., Bearer token).

Example:

Host: www.example.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8

3. **Body** (Optional):
    - The body of the request contains the data being sent to the server, and it is used in methods like POST, PUT, and PATCH to send data (e.g., form submissions or JSON payloads).
    - For example, in a POST request, the body may contain data such as a form's inputs or JSON data.
    - **Example** (in a POST request):

name=JohnDoe&email=john@example.com

4. **Example of a complete HTTP Request**:
5. makefile
6. CopyEdit
7. POST /submitForm HTTP/1.1
8. Host: www.example.com
9. User-Agent: Mozilla/5.0
10. Content-Type: application/x-www-form-urlencoded
11. Content-Length: 27
12.
13. name=JohnDoe&email=john@example.com

---

2. HTTP Response Message Format

An HTTP **response message** is sent by the server to the client in response to a request. It consists of the following components:

1. **Status Line**:

- The status line consists of the HTTP version, a status code, and a description of the status code.
- **Format**:

<HTTP-Version> <Status-Code> <Status-Message>

  - **HTTP-Version**: The version of HTTP being used (e.g., HTTP/1.1).
  - **Status-Code**: A 3-digit number that indicates the result of the request (e.g., 200, 404, 500).
  - **Status-Message**: A short textual description of the status code (e.g., "OK", "Not Found", "Internal Server Error").

Example:

HTTP/1.1 200 OK

2. **Headers**:
   - The response headers provide metadata about the response, such as the type of content being returned, the length of the content, and caching instructions.
   - Common response headers include:
     - Content-Type: Specifies the media type of the response body (e.g., text/html, application/json).
     - Content-Length: Specifies the size of the response body in bytes.
     - Cache-Control: Defines caching policies (e.g., no-cache, private).
     - Location: Used in redirection responses to specify the URL to which the client should be redirected.
     - Server: Identifies the software running on the server (e.g., Apache, nginx).

Example:

Content-Type: text/html; charset=UTF-8

Content-Length: 1234

3. **Body** (Optional):
   - The body of the response contains the actual data being returned to the client, such as HTML content, JSON data, images, or other resources.
   - In a 200 OK response, this is typically the requested resource (e.g., a webpage, API response, etc.).
   - **Example** (in an HTML response body):

```
html

<html>

  <head><title>Welcome</title></head>
```

  <body><h1>Hello, World!</h1></body>

</html>

4. **Example of a complete HTTP Response**:
5. php
6. CopyEdit
7. HTTP/1.1 200 OK
8. Content-Type: text/html; charset=UTF-8
9. Content-Length: 138
10.
11. <html>
12.   <head><title>Welcome</title></head>
13.   <body><h1>Hello, World!</h1></body>
14. </html>

Summary of HTTP Request and Response Message Format:

Component	Request Message	Response Message
Request Line / Status Line	Method, URI, HTTP Version	HTTP Version, Status Code, Status Message
Headers	Key-value pairs (e.g., Host, Content-Type)	Key-value pairs (e.g., Content-Type, Content-Length)
Body	Data sent from client to server (optional)	Data sent from server to client (optional)

Q.10.a.. With neat diagram, explain the basic model of FTP.

Basic Model of FTP (File Transfer Protocol)

The **File Transfer Protocol (FTP)** is a standard network protocol used to transfer files between a client and a server over a TCP/IP network. FTP operates on a client-server model, where the client requests files, and the server provides those files or receives files from the client.

FTP uses two separate channels for communication: one for **command control** and the other for **data transfer**. This separation ensures efficient file transfers while maintaining the structure of the protocol.

Basic FTP Model

The FTP model involves two main entities:

1. **FTP Client**: A user application or program that initiates the request to retrieve or send files to the server.
2. **FTP Server**: A system that stores files and responds to the client's requests, either providing requested files or accepting uploads.

FTP uses **two channels** to establish communication:

- **Command Channel (Control Channel)**:
    - This is a persistent connection that remains open throughout the FTP session. It is used for sending control commands (e.g., USER, PASS, LIST, GET, PUT).
    - The command channel uses **Port 21** by default for communication between the FTP client and server.
- **Data Channel**:
    - This channel is used for the actual file data transfer (sending or receiving files). The data channel is opened and closed dynamically as needed, depending on the file transfer operations.
    - The data channel can use either **Port 20** (active mode) or a dynamically assigned port (passive mode) for file transfer.

FTP Connection Phases

1. **Connection Establishment**:
    - The client connects to the FTP server on **Port 21** and establishes the control connection.
2. **Authentication**:
    - The FTP client sends a USER command to provide the username and a PASS command to provide the password for authentication.
3. **Command Transmission**:
    - After authentication, the client sends various FTP commands through the command channel (e.g., LIST, RETR, STOR).
4. **Data Transfer**:
    - When the client requests a file (e.g., RETR), or uploads a file (e.g., STOR), the server establishes the data channel for the actual file transfer.
5. **Connection Termination**:
    - After the transfer is complete, the control channel is closed by sending a QUIT command, and the session ends.

FTP Active vs Passive Mode

1. **Active Mode (PORT Mode)**:
   - The client sends a PORT command to the server, informing it of the port on which the client will listen for the incoming data connection.
   - The server then opens a connection to that port from its **Port 20** and transfers data to the client.
2. **Passive Mode (PASV Mode)**:
   - The client sends a PASV command to the server, which causes the server to open a random high-numbered port for the data transfer.
   - The client then connects to this port to receive or send data.

Diagram: Basic FTP Model

```
+-----------------+              +------------------+
 |   FTP Client  |               |   FTP Server   |
 |-----------------|              |------------------|
 |              |  <--- Control --->|              |
 |  Port 21      |  (Port 21)    |  Port 21       |
 |              |             |              |
 +-----------------+              +------------------+
       |                        ^
       |                        |
       |                        |
       |  <--- Data (File Transfer) --->  |
       |                        |
       |    (Port 20 or Dynamic Port)    |
      +---------------------------------+
```

1. **Control Channel**: Established on Port 21, used for communication commands.
2. **Data Channel**: Established dynamically on either Port 20 (active mode) or a high-numbered port (passive mode), used for transferring files.

Key Commands in FTP:

1. **USER**: Sends the username to the server.
2. **PASS**: Sends the password for authentication.

3. **LIST**: Requests a directory listing from the server.
4. **RETR**: Retrieves (downloads) a file from the server.
5. **STOR**: Stores (uploads) a file to the server.
6. **QUIT**: Terminates the FTP session.

Summary of FTP Model:

- FTP operates on a **client-server model** where the client requests files from the server, and the server responds with the requested data.
- **Two channels** are used for communication: the **command channel** (Port 21) and the **data channel** (Port 20 or dynamic port).
- FTP can operate in two modes:
    - **Active mode (PORT)**, where the server initiates the data connection.
    - **Passive mode (PASV)**, where the client initiates the data connection.

Q.10.b. Describe the architecture of electronic mail (e-mail)

Architecture of Electronic Mail (E-Mail)

The **electronic mail (e-mail)** system is a method of exchanging digital messages between people using electronic devices connected via a network, such as the internet. The e-mail system follows a **client-server architecture**, where users send and receive messages using a variety of e-mail clients, and these messages are stored and managed by e-mail servers. Below is a detailed description of the architecture of e-mail:

Basic Components of E-Mail System:

1. **E-Mail Client**:
    - The **e-mail client** is the application or software that allows users to interact with the e-mail system. It is used to send, receive, and manage e-mail messages.
    - Popular e-mail clients include applications like **Outlook**, **Mozilla Thunderbird**, **Apple Mail**, and web-based clients like **Gmail** and **Yahoo Mail**.
2. **E-Mail Server**:
    - An **e-mail server** is a system that stores and manages e-mail messages for users. It ensures that messages are delivered to the correct recipient and supports the protocols for message retrieval and delivery.
    - The e-mail server typically uses two main types of protocols:
        - **SMTP (Simple Mail Transfer Protocol)**: Used for sending e-mails from the client to the server and between servers.

- **POP3 (Post Office Protocol version 3) / IMAP (Internet Message Access Protocol)**: Used for retrieving e-mails from the server to the client.

E-Mail Architecture

The e-mail system follows a layered architecture where each layer serves a specific function in the process of sending, routing, and retrieving messages. Here's how the architecture is structured:

1. Sender Side (Sending an E-Mail)

1. **E-Mail Client (Sender's End)**:
   - The user creates an e-mail message using an e-mail client, specifying the recipient’s address, subject, body, and attachments.
   - Once the message is ready, the e-mail client sends it to the **SMTP server** for processing.
2. **SMTP Server**:
   - The **SMTP (Simple Mail Transfer Protocol)** server is responsible for forwarding the e-mail to the recipient's mail server.
   - The sender's SMTP server checks the recipient’s domain (e.g., example.com), looks up the recipient’s **MX (Mail Exchange) record** via DNS (Domain Name System), and forwards the message to the recipient's SMTP server.

2. Mail Transfer Process (Between Servers)

1. **Mail Transfer Agents (MTAs)**:
   - These are the **SMTP servers** that relay e-mails between the sender's and recipient's mail servers.
   - The SMTP server ensures that the message is routed correctly to the recipient's **Mail Delivery Agent (MDA)** or **Mail Retrieval Agent (MRA)** for final delivery.
2. **DNS (Domain Name System)**:
   - The DNS system plays a vital role in determining the mail server responsible for receiving e-mails for a specific domain (e.g., example.com).
   - The sending SMTP server queries the DNS for the **MX records** associated with the recipient's domain.

3. Recipient Side (Receiving an E-Mail)

1. **Recipient's Mail Server**:
   - The recipient's mail server receives the e-mail message sent by the sender's SMTP server and stores it in the recipient's mailbox.
   - The e-mail server uses a **Mail Delivery Agent (MDA)** to place the e-mail into the appropriate mail folder or mailbox of the recipient.
2. **POP3/IMAP**:

   - To retrieve the e-mail, the recipient uses a client (such as **Outlook** or **Thunderbird**) that communicates with the mail server using either **POP3** or **IMAP**.
     - **POP3** (Post Office Protocol version 3) allows users to download e-mails from the server to their local machine and typically removes the message from the server.
     - **IMAP** (Internet Message Access Protocol) allows users to view e-mails directly on the server without downloading them, keeping them synchronized across multiple devices.
   - IMAP is more suitable for users accessing e-mails from multiple devices.

4. Final Retrieval by E-Mail Client:

- The e-mail client retrieves the message from the mail server using **POP3** or **IMAP**.
- The client displays the message for the recipient to read and interact with.

E-Mail Architecture Diagram:

```
+---------------------+       +---------------------+      +-------------------+
 |                 |        |                 |       |               |
 |  Sender's E-Mail   |        |   Mail Transfer    |      | Recipient's E-Mail|
 |     Client        | -----> |     Agent (MTA)    |-----> |      Server      |
 |                 |        |                 |       |               |
 +---------------------+       +---------------------+      +-------------------+
                                  |
                                  v
                           +-----------------+
                           |  Mail Delivery |
                           |     Agent (MDA)|
                           +-----------------+
                                  |
                                  v
                          +-------------------+
                          |               |
                          | Recipient's E-Mail|
                          |     Client      |
                          |               |
                  +-------------------+
```

Key E-Mail Protocols:

1. **SMTP (Simple Mail Transfer Protocol)**:
   - Used for sending and routing e-mails between servers.
   - Operates on **Port 25** by default.
   - Enables communication between the client and server as well as between mail servers.
2. **POP3 (Post Office Protocol version 3)**:
   - Used for retrieving e-mails from the mail server to the client.
   - Operates on **Port 110** by default.
   - Downloads e-mails to the client and removes them from the server (unless configured otherwise).
3. **IMAP (Internet Message Access Protocol)**:
   - Used for retrieving e-mails from the server while keeping them stored on the server for multiple device access.
   - Operates on **Port 143** by default.
   - Allows the e-mail client to view, move, and manage messages on the server.

Process Flow of E-Mail System:

1. **Sending**:
   - The user creates an e-mail message.
   - The e-mail client sends it to the SMTP server.
   - The SMTP server forwards the message to the recipient's mail server.
2. **Routing**:
   - The e-mail travels through the mail transfer agent (MTA) and is routed using DNS.
   - The e-mail is delivered to the recipient's mail server.
3. **Retrieving**:
   - The recipient accesses their e-mail client and connects to the mail server using either POP3 or IMAP to retrieve the e-mail.
   - The e-mail is downloaded or viewed based on the protocol used.

Q.10.c. Briefly explain Recursive Resolution and Iterative Resolution in DNS.

Recursive Resolution vs Iterative Resolution in DNS

The **Domain Name System (DNS)** is used to translate human-readable domain names (like www.example.com) into IP addresses that computers use to communicate over the internet. The process of resolving domain names involves querying DNS servers to obtain the corresponding IP address. DNS resolution can occur in two primary ways: **recursive resolution** and **iterative resolution**.

---

1. Recursive Resolution:

Recursive resolution is when a DNS server takes on the responsibility of fully resolving a domain name query. In this case, the client (usually the end user's computer) sends the query to a DNS resolver (often provided by the user's ISP or a third-party service like Google DNS). The DNS resolver then queries other DNS servers on behalf of the client until it finds the required information, such as the IP address for the requested domain name.

Steps in Recursive Resolution:

1. The **client** (e.g., the browser) sends a DNS query to a DNS **resolver**.
2. The DNS resolver checks its local cache for the domain's IP address. If the answer is not cached, it queries other DNS servers.
3. The resolver may query the **root DNS server** if it doesn't know the IP of the authoritative DNS server for the domain.
4. The root server directs the resolver to the appropriate **TLD (Top-Level Domain)** DNS server (e.g., .com or .org).
5. The TLD server then points the resolver to the **authoritative DNS server** for the specific domain.
6. Finally, the authoritative DNS server provides the **IP address** associated with the domain.
7. The DNS resolver returns the **final answer** (IP address) to the client.

In recursive resolution, the client is relieved of having to send multiple queries. The DNS resolver handles all the queries and returns the final result to the client.

---

2. Iterative Resolution:

In **iterative resolution**, the DNS client (or resolver) queries each DNS server in sequence to obtain the required IP address. Unlike recursive resolution, the DNS server does not take full responsibility for resolving the query. Instead, it returns the best possible answer it knows at the time, and it is up to the client to follow up with further queries.

Steps in Iterative Resolution:

1. The **client** (e.g., browser or local DNS resolver) sends a DNS query to a DNS **resolver**.
2. The resolver checks if it has the IP address cached. If not, it queries the **root DNS server**.

3. The root server responds with a **reference** to the appropriate **TLD DNS server** for the domain (e.g., .com).
4. The resolver then queries the TLD server, which responds with a **reference** to the authoritative DNS server for the domain.
5. The resolver then queries the **authoritative DNS server**, which finally returns the IP address.
6. The resolver then returns the IP address to the client.

In iterative resolution, the resolver doesn't fully resolve the domain name query. Instead, it provides the client with referrals (references) to other DNS servers until the final answer is found.

Key Differences:

Aspect	**Recursive Resolution**	**Iterative Resolution**
Responsibility	DNS resolver is responsible for fully resolving the query.	The client or resolver is responsible for querying multiple DNS servers.
Process	The DNS resolver handles all queries and returns the final result to the client.	The resolver provides references and the client follows up with additional queries.
Speed	Generally faster for the client, as the resolver does all the work.	Might be slower as the client has to query multiple servers.
Load on DNS Servers	Puts more load on the DNS resolver.	Puts less load on DNS resolvers, as they just provide referrals.
Example	A client asks a DNS resolver to find www.example.com, and the resolver does all the work.	A client asks a DNS resolver to find www.example.com, and the resolver refers the client to the root server, which refers it to the TLD server, and so on.
