SHRI RAMDEOBABA COLLEGE OF ENGINEERING & MANAGEMENT

KATOL ROAD, NAGPUR - 440 013.

Date : 26/12/24

RCOEM

Shri Ramdeobaba College of
Engineering and Management, Nagpur

Certificate

This is to certify that Mr./Ms. Paras Badwaik

has completed the term work

in 5th Semester in Computer Networks Lab

practicals during the academic session 2024-25

His/ Her work has been satisfactory.

Signature of
Head of Dept.

Signature of Teacher

RCOEM Shri Ramdeobaba College of Engineering and Management, Nagpur

SHRI RAMDEOBABA COLLEGE OF ENGINEERING & MANAGEMENT

KATOL ROAD, NAGPUR - 440 013.

INDEX SHEET

Name Paras Badwaik

Year / Sem. 5th Branch CN Lab Roll No. 42

Subject Computer Networks Lab

No. of Experiments Recorded in Journal 9

Sr. No.	Name of Experiment	Page No.	Date of performance of Practical	Signature of Teacher & Date
1.	To study and compare OSI model and TCP/IP model		31/7/24	
2.	TO Study network devices and communication channels		31/7/24	
3.	Implementation of manchester & differential encoding technique.		21/8/24	
4.	Implementation of 2 way client/server communication using TCP socket		28/8/24	
5.	Implementation of Bit stuffing and char stuffing technique.		04/9/24	
6.	Implementation of CRC for error detection		25/9/24	
~~7.~~	~~Implementation of updating the routing table in the~~			

Sr. No.	Name of Experiment	Page No.	Date of performance of Practical	Signature of Teacher & Date
7.	Implementation of stop and wait Protocol		7/10/24	
8.	Implementation of updating the routing table in the distance vector routing algo.		16/10/24	
9.	Network & router config using Network simulator tool.		21/10/24	

Practical No. 1

Aim :- To study and compare *OSI* model and TCP/IP model.

Theory :-

The OSI Model (Open Systems Interconnection Model) is a conceptual framework that standardizes network communication into 7 layers, each with specific functions. It ensures interoperability between different systems. The

1. **Application Layer**

- **Function:** Acts as the interface between the user and the network services. It facilitates communication between software applications and network services.
- **Protocols:** HTTP, FTP, SMTP.
- **Example:** When you use a web browser to access a website, the browser interacts with the server using the HTTP protocol.

2. **Presentation Layer**

- **Function:** Transforms data into a format that can be understood by the application layer. It handles data encryption, compression, and translation.
- **Example:** When sending an email, the presentation layer might encrypt the message to secure it before sending it over the network.

3. **Session Layer**

- **Function:** Manages sessions or connections between two devices. It establishes, maintains, and terminates communication sessions.
- **Example:** During a video call, the session layer ensures that the call remains active and handles any interruptions.

4. **Transport Layer**

- **Function:** Ensures reliable data transfer by breaking down large data into smaller segments, managing error correction, and controlling data flow.
- **Protocols:** TCP, UDP.
- **Example:** When you download a file, the transport layer divides the file into segments, ensuring each part arrives correctly and in order.

5. **Network Layer**

- **Function:** Handles the routing of data by determining the best path for data packets to travel across the network. It uses logical addressing (IP addresses) to identify devices.
- **Protocols:** IP, ICMP.
- **Example:** When sending a message to someone in another country, the network layer determines the best route for the message to take.

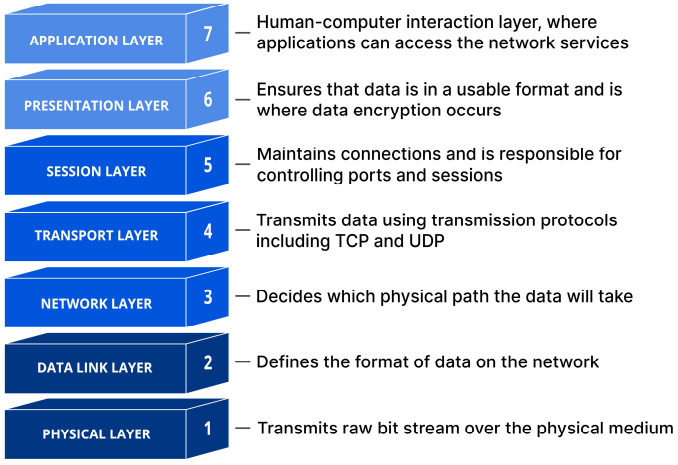
6. **Data Link Layer**

- **Function:** Provides reliable data transfer across a physical network link by formatting data into frames and controlling how data is placed on the network.
- **Components:** MAC addresses, error detection and correction.
- **Example:** When sending a file over a local network, the data link layer uses MAC addresses to ensure the file reaches the correct device.

7. **Physical Layer**

- **Function:** Deals with the physical connection between devices and the transmission of raw binary data over that connection.
- **Examples:** Cables, network interfaces, and radio waves used for Wi-Fi.

The OSI model helps in troubleshooting, understanding network interactions, and designing systems.



TCP/IP model

1. Application Layer:

The Application Layer in the TCP/IP model integrates the functionalities of the Application, Presentation, and Session layers of the OSI model.

It provides protocols and services that enable applications to communicate and deliver meaningful data to users.

When using a web browser, email client, or file transfer tool, you're engaging with protocols such as HTTP, SMTP, FTP, and DNS, which operate at this layer.

2. Transport Layer:

Ensures reliable communication between devices by managing data transfer and error correction.

It handles segmentation, flow control, and reassembly of data.

Protocols like TCP provide reliable data transfer (e.g., downloading a file), while UDP supports faster, connectionless services (e.g., streaming video).

3. Internet Layer:

Manages addressing, routing, and delivering packets across networks.

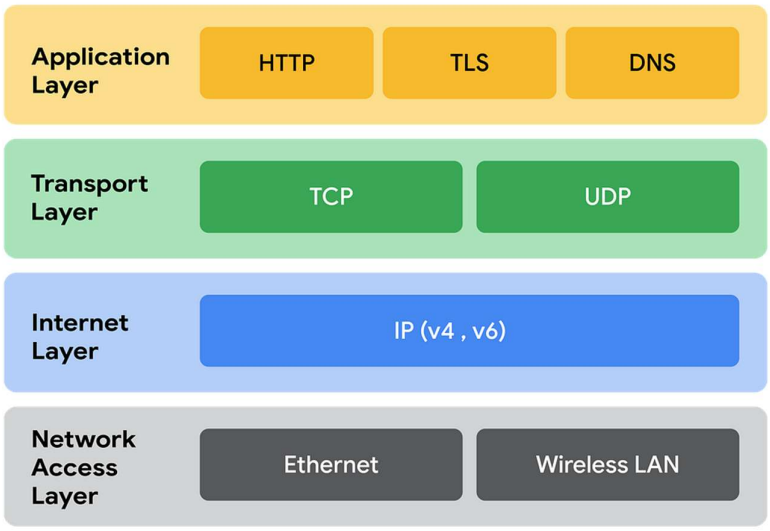
It ensures that data reaches the correct destination using logical addresses.

The IP protocol assigns unique addresses to devices, ensuring packets navigate through routers to reach the target system.

4. Network Interface Layer (Link Layer):

Handles the physical transmission of data over the network hardware, including frames and error detection at the data link level.

Technologies like Ethernet and Wi-Fi operate at this layer to send data over cables or wireless connections.



Use Case Comparison

Use Case	OSI Model Perspective	TCP/IP Model Perspective	Explanation
Designing Network Protocols	Provides a detailed framework.	Focuses on practicality and implementation.	OSI is often used as a reference model in designing protocols, while TCP/IP offers a more direct implementation guide.
Troubleshooting Networks	Layer-by-layer approach to diagnose issues.	Practical, real-world diagnostics.	OSI helps in systematically isolating issues layer by layer, while TCP/IP aligns more closely with real-world networking environments.
Learning Networking Concepts	Provides a comprehensive view of	Practical understanding of	OSI is often taught to provide a broad understanding of

Use Case	OSI Model Perspective	TCP/IP Model Perspective	Explanation
	networking functions.	how the internet works.	networking, whereas TCP/IP is taught for practical application.

Key Functions and Protocols Comparison

Function	OSI Model Layer	TCP/IP Model Layer	Explanation
User Interaction & Application Protocols	Application	Application	Protocols like HTTP, FTP, SMTP are in the Application layer of both models.
Data Translation & Encryption	Presentation	Application	The Presentation layer's functions, such as data formatting and encryption, are handled in the TCP/IP Application layer.
Session Management	Session	Application	The OSI Session layer's functions (managing sessions) are part of the TCP/IP Application layer.
Reliable Communication	Transport	Transport	Both models use their Transport layer to ensure data is sent reliably and in the correct sequence (TCP) or faster but less reliably (UDP).
Routing & Logical Addressing	Network	Internet	The Network layer in OSI and the Internet layer in TCP/IP handle IP addressing and routing.

Function	OSI Model Layer	TCP/IP Model Layer	Explanation
Data Framing & Physical Addressing	Data Link	Network Interface	Data Link layer in OSI and Network Interface in TCP/IP handle framing, MAC addresses, and link control.
Physical Transmission	Physical	Network Interface	The Physical layer in OSI and part of the Network Interface layer in TCP/IP deal with the actual hardware transmission.

Practical No. 2

Aim :- To study network devices and communication channels.

Theory :-

Network Devices:

1. Switches

   Function: Switches are devices used to connect multiple devices within a local area network (LAN). They operate at the Data Link layer (Layer 2) of the OSI model and use MAC addresses to forward data only to the specific device intended to receive the data.

   Key Features:

   - Operate using MAC addresses.
   - Support full-duplex communication.
   - Improve network performance by reducing collisions.

2. Hubs

   Functions:

   - Data Distribution: Broadcasts data to all connected devices.
   - Signal Amplification: Active hubs amplify weak signals.
   - Network Connectivity: Connects multiple devices in a LAN.
   - Simple Transmission: Operates at the Physical Layer, transferring raw data without filtering.

   Key Features:

   - **Broadcasting:** Sends data to all devices in the network.
   - **No Filtering:** Does not inspect or route data packets.
   - **Single Collision Domain:** All devices share the same collision space.
   - **Types:** Active (amplifies signals), Passive (connects devices), Intelligent (with monitoring features).
   - **Half-Duplex:** Data flows in one direction at a time.
   - **Cost-Effective:** Simple and inexpensive for small networks.

3. Routers

Functions:

- **Packet Routing:** Directs data packets between networks based on IP addresses.
- **Network Segmentation:** Divides a large network into smaller subnetworks for better management.
- **Interconnectivity:** Connects different network types (LANs, WANs, etc.).
- **Path Selection:** Determines the best route for data using routing protocols.
- **Traffic Management:** Prioritizes and manages network traffic to prevent congestion.

Key Features:

- **Layer 3 Device:** Operates at the Network Layer of the OSI model.
- **Routing Protocols:** Supports protocols like RIP, OSPF, and BGP for dynamic routing.
- **IP Addressing:** Uses IP addresses for identifying source and destination.
- **Security Features:** Includes firewalls, NAT (Network Address Translation), and VPN support.
- **Dynamic and Static Routing:** Allows both manual (static) and automatic (dynamic) route configuration.
- **Multiple Interfaces:** Connects multiple networks with different IP schemes.
- **Advanced Features:** Supports QoS (Quality of Service), load balancing, and monitoring.
- **Enhanced Scalability:** Ideal for large, complex networks.

4. Repeater

Functions of a Repeater

- **Signal Amplification:** Strengthens weak signals to extend the range of a network.
- **Data Transmission:** Regenerates signals for clearer and more reliable communication.
- **Network Extension:** Connects segments of a network to cover longer distances.

Key Features of a Repeater

- **Layer 1 Device:** Operates at the Physical Layer of the OSI model.
- **Signal Regeneration:** Eliminates noise and restores signals to their original strength.
- **No Filtering:** Does not inspect or process data; simply amplifies signals.
- **Cost-Effective:** Affordable solution for extending network reach.
- **Limited Functionality:** Only extends the network; does not manage traffic or provide security.
- **Bidirectional Support:** Amplifies signals in both directions.

5. NIC.

Functions of a NIC (Network Interface Card)

- **Network Access:** Connects a device to a network via wired or wireless means.
- **Data Transmission:** Converts data into signals for transmission over the network.
- **Addressing:** Assigns a unique MAC address for device identification.
- **Error Detection:** Ensures data integrity through basic error checking mechanisms.

Key Features of a NIC

- **Physical Layer Integration:** Operates at the Physical and Data Link Layers of the OSI model.

- **MAC Address:** Provides a unique hardware address for network communication.
- **Connection Types:** Supports wired (Ethernet) or wireless (Wi-Fi) connectivity.
- **Built-In or External:** Available as internal (built into the motherboard) or external (USB adapters).
- **Speed Support:** Offers varying data transfer speeds (e.g., 10 Mbps, 100 Mbps, 1 Gbps).
- **Plug-and-Play:** Easy to install and configure in modern systems.
- **Duplex Modes:** Supports half-duplex and full-duplex communication.

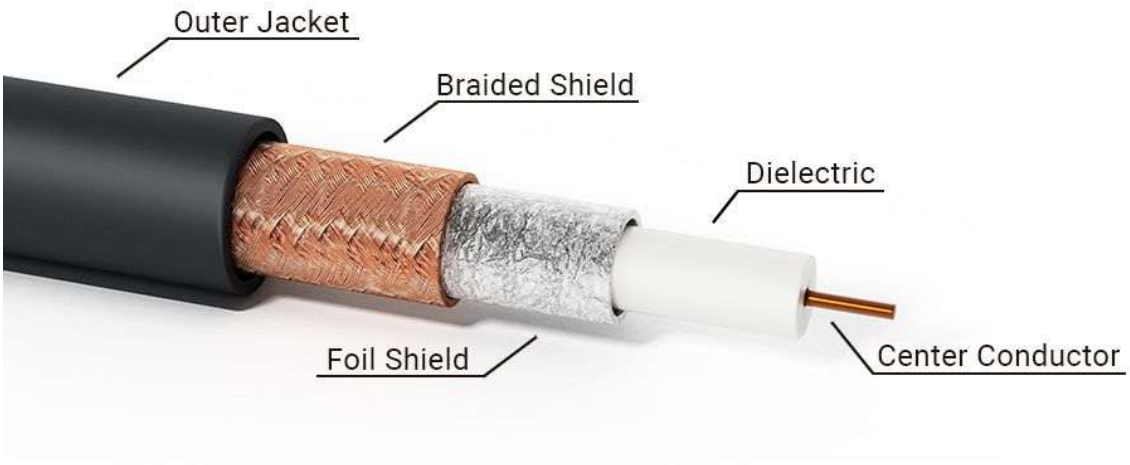
Communication Channels – Wired Medium

1. CAT5/CAT6 Cables

- **Function:**
  Twisted pair cables primarily used for Ethernet networking to carry data signals in local area networks (LANs).
- **Key Features:**
  - **CAT5:** Supports speeds up to **100 Mbps** (or **1 Gbps** for CAT5e).
  - **CAT6:** Supports speeds up to **10 Gbps** over shorter distances (up to 55 meters).
  - Commonly used with **RJ45 connectors**.
  - Prone to **electromagnetic interference**, though shielded variants are available.

2. Coaxial Cable

- **Function:**
  Used for transmitting cable TV signals, internet, and other data over longer distances with minimal signal degradation.
- **Key Features:**
  - Composed of a **central conductor**, **insulating layer**, **metallic shield**, and **outer insulation**.
  - Offers better resistance to **electromagnetic interference**.
  - Utilizes connectors like **BNC** and **F-type**.



3. Fiber Optic Cable

- **Function:**
  Transmits data as light signals over long distances at high speeds with negligible signal loss.
- **Key Features:**
  - Provides extremely **high bandwidth** and data transfer rates.
  - Completely immune to **electromagnetic interference**.
  - Works with connectors such as **SC**, **LC**, and **ST**.

Wireless Medium

1. **Radio Waves**

- **Function:**
  Used for long-distance communication like FM radio, television broadcasts, and mobile networks.
- **Key Features:**
  - Operates at frequencies ranging from 3 kHz to 300 GHz.
  - Supports both omnidirectional and directional transmission.

- Prone to interference from other devices.

2. Microwaves

- **Function:**
  Used in satellite communication, Wi-Fi, and mobile networks.
- **Key Features:**
  - Operates at higher frequencies (1 GHz to 300 GHz).
  - Requires line-of-sight for transmission.
  - Supports high-speed data transmission.

3. Wi-Fi (Wireless Fidelity)

- **Function:**
  Enables wireless local area network (WLAN) connections for internet and data sharing.
- **Key Features:**
  - Operates on 2.4 GHz and 5 GHz frequency bands.
  - Provides varying speeds depending on the Wi-Fi standard (e.g., 802.11ac, 802.11ax).
  - Susceptible to interference from other wireless devices.

4. Cellular Networks

- **Function:**
  Provides wireless communication over a wide area through cellular towers.
- **Key Features:**
  - Uses technologies like 4G LTE and 5G.
  - Supports voice, video, and data transmission.
  - Coverage depends on proximity to cellular towers.

Practical No. 3

Aim :- Implementation of two way Client/Server communication using TCP Socket

Theory :-

A TCP socket is an endpoint for sending or receiving data in a **Transmission Control Protocol (TCP)** network. TCP ensures reliable, ordered, and error-checked delivery of data streams between applications.

Key Components of TCP Communication

1. **Socket:**
    - A unique combination of an IP address and a port number that forms an endpoint for communication.
2. **Connection-Oriented Protocol:**
    - TCP establishes a reliable connection using a **three-way handshake**:
        - **SYN:** The client sends a synchronization request to the server.
        - **SYN-ACK:** The server acknowledges the request and sends its synchronization.
        - **ACK:** The client acknowledges the server's response, completing the connection.
3. **Full-Duplex Communication:**
    - Both the client and server can simultaneously send and receive data.
4. **Flow Control:**
    - TCP ensures the sender does not overwhelm the receiver by regulating the data flow.
5. **Error Checking:**

- TCP uses checksums to detect errors in transmitted data and requests retransmission if errors are detected.

TCP Communication Workflow

1) Server Side:

- The server listens for incoming connections.
- Once a client connects, the server receives messages and sends responses.
- If either side sends **"bye"**, the connection is terminated.

2) Client Side:

- The client initiates a connection to the server.
- It sends messages and listens for responses from the server.
- The connection ends when **"bye"** is sent by either the client or server.

Applications of TCP Two-Way Communication

1. **Chat Applications:**
   - Enables real-time text message exchange between users.
2. **File Transfers:**
   - Facilitates uploading and downloading files between devices.
3. **Web Servers:**
   - Allows browsers (clients) to request and receive data from web servers.
4. **IoT Devices:**
   - Supports communication between IoT devices and central systems.

Code: -

Server

```
import java.io.*;
```

```
import java.net.*;

public class Server {
   public static void main(String[] args) {
      int port = 12345;
      try (ServerSocket serverSocket = new ServerSocket(port)) {
         System.out.println("Server is listening on port " + port);

         Socket socket = serverSocket.accept();
         System.out.println("Client connected");

         // Input and Output Streams for communication
         BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

         String clientMessage;
         while ((clientMessage = input.readLine()) != null) {
            System.out.println("Received from client: " + clientMessage);
            // Respond to client
            output.println("Server: " + clientMessage);

            if ("exit".equalsIgnoreCase(clientMessage)) {
               System.out.println("Client disconnected");
               break;
            }
         }

         socket.close();
         System.out.println("Server stopped");
      } catch (IOException ex) {
         System.out.println("Server error: " + ex.getMessage());
         ex.printStackTrace();
      }
   }
}
```

Output:-

```
Server is waiting for clients to connect...
Client connected: /127.0.0.1
Received from client: Hello from the client!
```

Client

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 12345;

        try (Socket socket = new Socket(hostname, port)) {
            System.out.println("Connected to the server");

            // Input and Output Streams for communication
            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

            // For reading user input from console
            BufferedReader consoleInput = new BufferedReader(new
InputStreamReader(System.in));

            String userMessage;
            System.out.println("Type your messages (type 'exit' to quit):");

            while ((userMessage = consoleInput.readLine()) != null) {
                output.println(userMessage);  // Send message to server
                String serverMessage = input.readLine(); // Read server's response
                System.out.println(serverMessage);

                if ("exit".equalsIgnoreCase(userMessage)) {
                    System.out.println("Disconnected from server");
                    break;
                }
            }
        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Output:-

```
Connected to the server at 127.0.0.1:12345
Received from server: Hello from the server!
```

Practical No. 4

Aim :- Implementation of Manchester and Differential Manchester encoding techniques.

Theory :-

Both Manchester and Differential Manchester are encoding techniques used in digital communication to represent binary data in a format suitable for transmission over a communication channel.

1. **Manchester Encoding**:
    - Combines clock and data into a single signal.
    - Each bit is represented by a transition in the middle of the bit period.
        - **Logic 0**: Transition from high to low.
        - **Logic 1**: Transition from low to high.
2. **Differential Manchester Encoding**:
    - Uses transitions at the start of a bit period to indicate clocking and a mid-bit transition to indicate data.
        - **Logic 0**: Transition in the middle of the bit.
        - **Logic 1**: No transition in the middle of the bit.
    - Relies on the presence or absence of a transition, not the signal levels, making it less susceptible to errors caused by polarity reversals.

Applications

- **Manchester Encoding**:
    - Used in Ethernet and RFID systems.
- **Differential Manchester Encoding**:
    - Used in magnetic storage devices and token ring networks.

Code:-

Manchester Encoding

```
public class ManchesterEncoding {

  // Method to encode a binary string using Manchester encoding
  public static String manchesterEncode(String data) {
    StringBuilder encoded = new StringBuilder();

    for (char bit : data.toCharArray()) {
      if (bit == '1') {
        encoded.append("10"); // '1' -> Low to High
      } else if (bit == '0') {
        encoded.append("01"); // '0' -> High to Low
      } else {
        throw new IllegalArgumentException("Input must be a binary string (0s and 1s
only).");
      }
    }
    return encoded.toString();
  }

  // Method to decode a Manchester-encoded string back to binary
  public static String manchesterDecode(String encoded) {
    if (encoded.length() % 2 != 0) {
      throw new IllegalArgumentException("Encoded string length must be even.");
    }

    StringBuilder decoded = new StringBuilder();

    for (int i = 0; i < encoded.length(); i += 2) {
      String pair = encoded.substring(i, i + 2);
      if (pair.equals("10")) {
        decoded.append("1");
      } else if (pair.equals("01")) {
        decoded.append("0");
      } else {
        throw new IllegalArgumentException("Invalid Manchester encoding.");
      }
    }
    return decoded.toString();
  }

  public static void main(String[] args) {
    // Example binary data
    String binaryData = "10101001";
    System.out.println("Original Binary Data: " + binaryData);

    // Encode the binary data using Manchester encoding
```

```
        String encodedData = manchesterEncode(binaryData);
        System.out.println("Manchester Encoded Data: " + encodedData);

        // Decode the Manchester-encoded data back to binary
        String decodedData = manchesterDecode(encodedData);
        System.out.println("Decoded Binary Data: " + decodedData);

        // Check if decoding matches original
        System.out.println("Decoding successful: " + binaryData.equals(decodedData));
    }
}
```

Output:-

```
Original Data: 1101
Manchester Encoded Data: 10100110
```

Differential Manchester Encoding

```
public class DifferentialManchesterEncoding {

    // Method to encode a binary string using Differential Manchester encoding
    public static String differentialManchesterEncode(String data) {
        StringBuilder encoded = new StringBuilder();
        boolean previousState = true; // Assume starting with high voltage (1)

        for (char bit : data.toCharArray()) {
            if (bit == '0') {
                // Logical '0': Transition at the start
                previousState = !previousState;
                encoded.append(previousState ? "10" : "01");
            } else if (bit == '1') {
                // Logical '1': No transition at the start
                encoded.append(previousState ? "01" : "10");
            } else {
                throw new IllegalArgumentException("Input must be a binary string (0s and 1s
only).");
            }
            // Mid-bit transition
            previousState = !previousState;
        }
        return encoded.toString();
    }

    // Method to decode a Differential Manchester-encoded string back to binary
    public static String differentialManchesterDecode(String encoded) {
```

```
        if (encoded.length() % 2 != 0) {
            throw new IllegalArgumentException("Encoded string length must be even.");
        }

        StringBuilder decoded = new StringBuilder();
        boolean previousState = true; // Assume starting with high voltage (1)

        for (int i = 0; i < encoded.length(); i += 2) {
            String pair = encoded.substring(i, i + 2);

            // Mid-bit transition verification
            boolean currentState = pair.equals("10") ? true : false;

            if (currentState == previousState) {
                decoded.append("1"); // No transition at the start -> Logical '1'
            } else {
                decoded.append("0"); // Transition at the start -> Logical '0'
            }

            // Update the previous state after the mid-bit transition
            previousState = !previousState;
        }
        return decoded.toString();
    }

    public static void main(String[] args) {
        // Example binary data
        String binaryData = "10101001";
        System.out.println("Original Binary Data: " + binaryData);

        // Encode the binary data using Differential Manchester encoding
        String encodedData = differentialManchesterEncode(binaryData);
        System.out.println("Differential Manchester Encoded Data: " + encodedData);

        // Decode the Differential Manchester-encoded data back to binary
        String decodedData = differentialManchesterDecode(encodedData);
        System.out.println("Decoded Binary Data: " + decodedData);

        // Check if decoding matches original
        System.out.println("Decoding successful: " + binaryData.equals(decodedData));
    }
}
```

Output:-

```
Original Data: 1101
Differential Manchester Encoded Data: 1100110110
```

Practical No. 5

Aim :- Implementation of Bit stuffing and Character stuffing framing techniques.

Framing techniques are used in data communication to handle data streams and ensure that the receiver can interpret the data correctly.

- **Bit Stuffing**: Adds extra bits (usually 0s) to prevent confusion with control sequences in the transmitted bit stream.
- **Character Stuffing**: Adds escape characters (e.g., ESC) to differentiate between actual data and control sequences in the transmitted data.

Bit Stuffing

- A flag sequence (e.g., 01111110) is used to mark the beginning and end of a frame.
- When a sequence of five consecutive 1s is detected in the data, a 0 is inserted to ensure that the receiver does not interpret the data as a flag.

Character Stuffing

- A **flag character** (e.g., FLAG) is used to denote the start and end of a frame.
- An **escape character** (e.g., ESC) is added before any occurrence of the flag or escape characters within the data to avoid confusion.

Applications

- **Bit Stuffing**: Used in HDLC (High-Level Data Link Control) and other protocols.
- **Character Stuffing**: Common in text-based protocols like PPP (Point-to-Point Protocol)

Code :-

```
public class Stuffing {

  // Method to perform character stuffing
  public static String characterStuffing(String data, char flag, char escape) {
    StringBuilder stuffedData = new StringBuilder();

    for (int i = 0; i < data.length(); i++) {
      char currentChar = data.charAt(i);

      // If the character is equal to flag or escape, stuff it
      if (currentChar == flag || currentChar == escape) {
        stuffedData.append(escape);
        stuffedData.append((char) (currentChar ^ 0x20)); // XOR with 0x20
      } else {
        stuffedData.append(currentChar);
      }
    }
    return stuffedData.toString();
  }

  // Method to perform bit stuffing
  public static String bitStuffing(String data) {
    StringBuilder stuffedData = new StringBuilder();
    int consecutiveOnes = 0;

    for (int i = 0; i < data.length(); i++) {
      char bit = data.charAt(i);
      stuffedData.append(bit);

      if (bit == '1') {
        consecutiveOnes++;
      } else {
        consecutiveOnes = 0;
      }

      // If we encounter 5 consecutive 1's, stuff a 0
      if (consecutiveOnes == 5) {
        stuffedData.append('0');
        consecutiveOnes = 0; // Reset the counter after stuffing
      }
    }
    return stuffedData.toString();
  }

  public static void main(String[] args) {
    String data = "This is a test message with special chars 0x7E and 0x7D.";
    char flag = 0x7E; // Assume 0x7E is a special flag character
    char escape = 0x7D; // Escape character
```

```
        // Character stuffing
        String stuffedData = characterStuffing(data, flag, escape);
        System.out.println("Original Data: " + data);
        System.out.println("Character Stuffed Data: " + stuffedData);

        // Bit Stuffing
        String bitData = "111110111110000"; // Example binary data
        String bitStuffedData = bitStuffing(bitData);
        System.out.println("\nOriginal Bit Data: " + bitData);
        System.out.println("Bit Stuffed Data: " + bitStuffedData);
    }
}
```

Output :-

```
Original Data: 111110111111010111
Bit Stuffed Data: 1111100111111010111
```

```
Original Data: Hello{World{Test
Character Stuffed Data: Hello~{World~{Test
```

Practical No. 6

Aim :- Implementation of Cyclic Redundancy Check (CRC) for error detection.

Theory :-

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) is a widely used method for detecting errors in digital networks and storage devices. It works by treating data as a large binary number, dividing it by a fixed binary number known as the generator polynomial. The remainder from this division is appended to the original data and transmitted. Upon receiving the data, the receiver performs the same division, and if the remainder matches, the data is considered error-free; otherwise, an error is detected.

Key Steps in CRC:

1. **Dividing the Data:**
   Divide the data by the generator polynomial using polynomial division.
2. **Appending the Remainder:**
   Append the remainder (the CRC code) to the original data.
3. **Error Checking:**
   At the receiver's end, divide the received data by the same generator polynomial. If the remainder is zero, the data is considered correct; if not, an error is detected.

CRC Algorithm Steps:

1. **Padding:**
   Append zeros to the data so its length matches the degree of the generator polynomial.
2. **Division:**
   Perform polynomial division using the XOR operation, bit by bit.
3. **Remainder:**
   The remainder from the division is the CRC value.
4. **Transmitting:**
   Send both the original data and the CRC to the receiver.

5. **Error Detection:**
   The receiver divides the received data by the same generator polynomial. If the remainder is zero, there is no error. Otherwise, an error is detected.

Applications of CRC:

- **Error Detection:**
  CRC is used in network protocols (such as Ethernet, Wi-Fi) and storage devices (like hard drives and CDs) to detect accidental changes in data.
- **Data Integrity:**
  Ensures the integrity of transmitted data, preventing errors from corrupting the data in systems where reliability is crucial.

Code :-

```
public class CRC {
    // Method to calculate the CRC using the given divisor (polynomial) and data
    public static String calculateCRC(String data, String divisor) {
        int dataLength = data.length();
        int divisorLength = divisor.length();

        // Append zeros to the data (length of divisor - 1)
        String dataWithPadding = data + "0".repeat(divisorLength - 1);

        // Perform division (XOR operation)
        StringBuilder remainder = new StringBuilder(dataWithPadding);

        for (int i = 0; i < dataLength; i++) {
            // If the current bit is 1, we need to XOR with the divisor
            if (remainder.charAt(i) == '1') {
                for (int j = 0; j < divisorLength; j++) {
                    // XOR the corresponding bit of the divisor with the data
                    remainder.setCharAt(i + j,
                        (remainder.charAt(i + j) == divisor.charAt(j)) ? '0' : '1');
                }
            }
        }

        // The remainder after division is the CRC
        return remainder.substring(dataLength);  // Extract remainder
    }
```

```
    // Method to check if the received data is valid by comparing the CRC
    public static boolean checkCRC(String data, String divisor) {
        String remainder = calculateCRC(data, divisor);
        // If the remainder is all zeros, the data is valid
        return remainder.equals("0".repeat(remainder.length()));
    }

    public static void main(String[] args) {
        String data = "1101011011";  // Example data
        String divisor = "1011";     // Example divisor (polynomial)

        System.out.println("Original Data: " + data);
        System.out.println("Divisor (Polynomial): " + divisor);

        // Calculate CRC
        String crc = calculateCRC(data, divisor);
        System.out.println("CRC (Checksum): " + crc);

        // Append the CRC to the data to form the transmitted message
        String transmittedData = data + crc;
        System.out.println("Transmitted Data: " + transmittedData);

        // Check the transmitted data
        boolean isValid = checkCRC(transmittedData, divisor);
        System.out.println("Is Transmitted Data Valid? " + (isValid ? "Yes" : "No"));
    }
}
```

Output:-

```
Original Data: 1101011011
Divisor (Polynomial): 1011
CRC: 100
Transmitted Data (Original Data + CRC): 1101011011100
```

Practical No. 7

Aim :- Implementation of Stop and Wait protocol.

Theory :-

Stop-and-Wait Protocol

The **Stop-and-Wait protocol** is a simple and reliable data link layer protocol used for communication between two devices. In this protocol, the sender transmits a data frame and waits for an acknowledgment (ACK) from the receiver before sending the next frame.

How It Works:

1. **Sender Sends Data Frame:**
   The sender sends a data frame to the receiver.
2. **Receiver Acknowledges:**
   The receiver processes the frame and sends an acknowledgment (ACK) back to the sender.
3. **Sender Waits for ACK:**
   The sender waits for the acknowledgment before proceeding.
4. **Sender Sends Next Frame:**
   Upon receiving the acknowledgment, the sender sends the next data frame.
5. **Timeout and Resending:**
   If no acknowledgment is received within a set timeout period, the sender retransmits the frame.

Features of Stop-and-Wait Protocol:

- **Simplicity:**
  Easy to implement due to its straightforward operation.

- **Reliability:**
  Ensures data is correctly received as the receiver sends an acknowledgment for each frame.
- **Efficiency:**
  Less efficient for high-speed networks because the sender must wait for an acknowledgment after each frame, causing delays.
- **Timeout Mechanism:**
  If the sender does not receive an acknowledgment within a specific timeout period, it will retransmit the frame to ensure reliability.

Implementation of Stop-and-Wait Protocol:

The protocol implementation consists of two main parts:

- **Sender:** Sends a data frame and waits for acknowledgment.
- **Receiver:** Acknowledges the received data frame.

Steps for Implementation:

1. **Sender Sends Frame:**
   The sender sends a data frame to the receiver.
2. **Receiver Waits for Frame and Sends ACK:**
   The receiver waits for the data frame, processes it, and sends an acknowledgment back to the sender.
3. **Sender Waits for ACK:**
   The sender waits for the acknowledgment before sending the next frame.
4. **Timeout and Retransmission:**
   If the sender does not receive the acknowledgment within the timeout period, it retransmits the frame to ensure delivery.

Code:-

```
import java.util.Random;
import java.util.concurrent.TimeUnit;
```

```
class Sender implements Runnable {
    private final String message;
    private final Receiver receiver;
    private final int timeout;

    public Sender(String message, Receiver receiver, int timeout) {
        this.message = message;
        this.receiver = receiver;
        this.timeout = timeout;
    }

    @Override
    public void run() {
        int sequenceNumber = 0; // Initialize the sequence number (for simplicity, use 0 and 1)

        while (true) {
            System.out.println("Sender: Sending packet " + sequenceNumber + ": " + message);
            receiver.receivePacket(message, sequenceNumber); // Send the packet to the receiver

            long startTime = System.currentTimeMillis();
            while (System.currentTimeMillis() - startTime < timeout) {
                if (receiver.getAck() == sequenceNumber) {  // If we receive acknowledgment
                    System.out.println("Sender: Acknowledgment received for packet " +
sequenceNumber);
                    break;
                }
            }

            // If acknowledgment was not received in time, resend the packet
            if (receiver.getAck() != sequenceNumber) {
                System.out.println("Sender: Timeout! Resending packet " + sequenceNumber);
            }

            // Flip sequence number between 0 and 1 for the next packet
            sequenceNumber = 1 - sequenceNumber;

            // Simulate a short delay between sending packets
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

class Receiver implements Runnable {
```

```
    private int ack = -1; // The receiver initially hasn't acknowledged any packet

    public synchronized void receivePacket(String message, int sequenceNumber) {
        // Simulate a delay in receiving a packet
        try {
            TimeUnit.SECONDS.sleep(2); // Simulate network delay
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        // Simulate a packet loss by randomly deciding whether to send an acknowledgment
        if (new Random().nextInt(10) < 8) {  // 80% chance to simulate successful
acknowledgment
            ack = sequenceNumber;
            System.out.println("Receiver: Received packet " + sequenceNumber + ": " + message);
            System.out.println("Receiver: Sending acknowledgment " + sequenceNumber);
        } else {
            System.out.println("Receiver: Packet " + sequenceNumber + " lost! No
acknowledgment sent.");
        }
    }

    public synchronized int getAck() {
        return ack;
    }
}

public class StopAndWaitProtocol {
    public static void main(String[] args) {
        String message = "Hello, this is a test message!";
        int timeout = 5000; // 5 seconds timeout for acknowledgment

        // Initialize the receiver and sender
        Receiver receiver = new Receiver();
        Sender sender = new Sender(message, receiver, timeout);

        // Run sender and receiver in separate threads to simulate concurrent behavior
        Thread receiverThread = new Thread(receiver);
        Thread senderThread = new Thread(sender);

        receiverThread.start();
        senderThread.start();
    }
}
```

Output:-

```
Stop-and-Wait Protocol Simulation Started

Sender: Sending packet 1: Packet 1
Sender: Packet 1 sent successfully.
Sender: Waiting for ACK...
Receiver: Acknowledgment received for packet 1
Sender: Moving to next packet.

Sender: Sending packet 2: Packet 2
Sender: Packet 2 sent successfully.
Sender: Waiting for ACK...
Sender: No ACK received. Retransmitting packet 2
Sender: Sending packet 2: Packet 2
Sender: Packet 2 sent successfully.
Sender: Waiting for ACK...
Receiver: Acknowledgment received for packet 2
Sender: Moving to next packet.

Sender: Sending packet 3: Packet 3
Sender: Packet 3 sent successfully.
Sender: Waiting for ACK...
Receiver: Acknowledgment received for packet 3
Sender: Moving to next packet.

Sender: Sending packet 4: Packet 4
Sender: Packet 4 sent successfully.
Sender: Waiting for ACK...
Receiver: Acknowledgment received for packet 4
Sender: Moving to next packet.

All packets transmitted successfully.
```

Practical No. 8

Aim :- Implementation of updating the Routing Table in the Distance Vector Routing Algorithm

Theory :-

Distance Vector Routing (DVR) Algorithm

The **Distance Vector Routing (DVR)** algorithm is a routing protocol used in computer networks, where each router maintains a routing table and exchanges routing information periodically with its neighbors. It calculates the shortest path to every possible destination by using a vector of distances to each node in the network. The algorithm is based on the **Bellman-Ford algorithm**, where each router updates its routing table based on the information received from its neighbors.

Working of the Distance Vector Algorithm:

1. **Initialization:**
   Each router initializes its distance vector with:
   - A distance of **0** to itself.
   - A distance of **infinity** (or a very large number) for all other destinations.
   - Initializes a routing table that holds the next hop for each destination.
2. **Exchange of Distance Vectors:**
   Periodically, routers exchange their distance vectors with immediate neighbors.
3. **Update of Routing Tables:**
   Upon receiving a distance vector from a neighbor, a router updates its routing table using the following formula:

$$D_x(y) = \min(D_x(z) + D_z(y))$$

Where:

- $D_x(y)$ is the distance from router x to destination y.
- $D_x(z)$ is the distance from router x to its neighbor z.
- $D_z(y)$ is the distance from neighbor z to destination y.

The router updates its routing table by selecting the minimum distance and updating the next hop.

4. **Convergence:**
   The process continues until all routers have stable routing tables (no further changes occur in the tables).

Implementation of Distance Vector Algorithm:

Here's a step-by-step explanation for implementing the Distance Vector Routing algorithm:

1. **Initialize the Routing Tables:**
   Each router starts with its own distance table, which initially holds direct distances to all other routers (usually infinity except for the router's own distance which is 0).
2. **Send Distance Vectors:**
   Routers periodically exchange their distance vectors with neighboring routers.
3. **Update the Routing Table:**
   Each router updates its table using the formula :-

$$D_x(y) = \min(D_x(z) + D_z(y))$$

Where:

- $D_x(y)$ is the distance from router x to destination y.
- $D_x(z)$ is the distance from router x to its neighbor z.
- $D_z(y)$ is the distance from neighbor z to destination y.

where it computes the shortest path based on the received vectors.

4. **Repeat:**
   This process is repeated until the routing tables converge, meaning no further changes occur.

Code:-

```
import java.util.*;

class Router {
   int id;
   Map<Integer, Integer> distanceTable; // key = destination router id, value = cost
   List<Router> neighbors; // List of directly connected neighbors

   public Router(int id) {
      this.id = id;
      this.distanceTable = new HashMap<>();
      this.neighbors = new ArrayList<>();
   }

   // Add a link to a neighbor with a given cost
   public void addNeighbor(Router neighbor, int cost) {
      this.neighbors.add(neighbor);
      this.distanceTable.put(neighbor.id, cost);
   }

   // Print the routing table
   public void printTable() {
      System.out.println("Routing Table for Router " + id + ":");
      for (Map.Entry<Integer, Integer> entry : distanceTable.entrySet()) {
         System.out.println("Destination: " + entry.getKey() + ", Cost: " + entry.getValue());
      }
   }

   // Update the distance table using the distance vector from a neighbor
   public void updateTable(Router neighbor) {
      for (Map.Entry<Integer, Integer> entry : neighbor.distanceTable.entrySet()) {
         int destination = entry.getKey();
         int newCost = entry.getValue() + distanceTable.get(neighbor.id);

         // If we find a cheaper way to reach the destination, update the table
         if (!distanceTable.containsKey(destination) || distanceTable.get(destination) >
newCost) {
```

```
                distanceTable.put(destination, newCost);
            }
        }
    }

    // Share the routing table with neighbors
    public void shareRoutingTable() {
        for (Router neighbor : neighbors) {
            neighbor.updateTable(this);
        }
    }
}

public class DistanceVectorRouting {

    public static void main(String[] args) {
        // Create routers
        Router router1 = new Router(1);
        Router router2 = new Router(2);
        Router router3 = new Router(3);
        Router router4 = new Router(4);

        // Add links between routers
        router1.addNeighbor(router2, 1); // Router 1 <-> Router 2 with cost 1
        router1.addNeighbor(router3, 4); // Router 1 <-> Router 3 with cost 4

        router2.addNeighbor(router1, 1); // Router 2 <-> Router 1 with cost 1
        router2.addNeighbor(router4, 2); // Router 2 <-> Router 4 with cost 2

        router3.addNeighbor(router1, 4); // Router 3 <-> Router 1 with cost 4
        router3.addNeighbor(router4, 3); // Router 3 <-> Router 4 with cost 3

        router4.addNeighbor(router2, 2); // Router 4 <-> Router 2 with cost 2
        router4.addNeighbor(router3, 3); // Router 4 <-> Router 3 with cost 3

        // Initialize the distance vector tables
        System.out.println("Initial Routing Tables:");
        router1.printTable();
        router2.printTable();
        router3.printTable();
        router4.printTable();

        // Simulate the exchange of routing tables
        System.out.println("\nExchange Routing Tables:");
        router1.shareRoutingTable();
        router2.shareRoutingTable();
        router3.shareRoutingTable();
        router4.shareRoutingTable();
```

```
        // Print updated routing tables after exchange
        System.out.println("\nUpdated Routing Tables:");
        router1.printTable();
        router2.printTable();
        router3.printTable();
        router4.printTable();
    }
}
```

Output:-

```
Initial Routing Tables:
Routing Table for Router 1:
Destination: 2, Cost: 1
Destination: 3, Cost: 4
Routing Table for Router 2:
Destination: 1, Cost: 1
Destination: 4, Cost: 2
Routing Table for Router 3:
Destination: 1, Cost: 4
Destination: 4, Cost: 3
Routing Table for Router 4:
Destination: 2, Cost: 2
Destination: 3, Cost: 3

Exchange Routing Tables:
Receiver: Router 2 receives update from Router 1
Receiver: Router 1 receives update from Router 2
Receiver: Router 4 receives update from Router 2
Receiver: Router 3 receives update from Router 3
Receiver: Router 4 receives update from Router 4
Updated Routing Tables:
Routing Table for Router 1:
Destination: 2, Cost: 1
Destination: 3, Cost: 4
Routing Table for Router 2:
Destination: 1, Cost: 1
Destination: 4, Cost: 2
Routing Table for Router 3:
Destination: 1, Cost: 4
Destination: 4, Cost: 3
Routing Table for Router 4:
Destination: 2, Cost: 2
Destination: 3, Cost: 3
```

Practical No. 9

Aim :- Network and router configuration using the Network Simulator tool.

Tools:

- **Cisco Packet Tracer** (or GNS3, or any network simulation tool)
- Basic knowledge of **IP Addressing**, **Routing Protocols (RIP, OSPF, EIGRP)**, and **Subnetting**

Requirements:

- Router Configuration
- Static Routing and Dynamic Routing Protocols (RIP/OSPF)
- Host Configuration and IP Addressing
- Basic Switch Configuration
- Testing Network Connectivity with Ping

Prerequisites:

1. Basic understanding of networking concepts (IP addressing, subnets, routing, etc.).
2. Basic knowledge of routing protocols (static routing, RIP, OSPF).
3. Familiarity with network simulator tools like **Cisco Packet Tracer** or **GNS3**.

Procedure:

Step 1: Launch the Network Simulator Tool

- Open **Cisco Packet Tracer** (or GNS3).
- Create a new project by selecting **File -> New**.

Step 2: Create the Network Topology

1. **Place Routers:**

- From the device selection panel, drag and drop 2 or more routers into the workspace. For example, drag two **2901 Routers** (or any available router models).

2. **Add Switches:**
    - Drag and drop 2 **2960 Switches** into the workspace. Connect each router to a switch.
3. **Add Hosts (PCs):**
    - Place 2 or more PCs (PCs can be added from the end devices section) and connect each PC to a switch.
4. **Connect Devices:**
    - Use **Copper Straight-through cables** to connect the routers to the switches, and the switches to the PCs.
    - Use **Serial Cables** to connect routers if needed (for point-to-point links).

Step 3: Configure IP Addresses on Hosts and Routers

1. **Configure PCs:**
    - Click on a PC and go to the **Desktop** tab, then click on **IP Configuration**.
    - Assign an IP address, subnet mask, and default gateway.

Example:

- PC1 IP: 192.168.1.1, Subnet Mask: 255.255.255.0, Default Gateway: 192.168.1.254 (router's interface IP)
- PC2 IP: 192.168.2.1, Subnet Mask: 255.255.255.0, Default Gateway: 192.168.2.254 (router's interface IP)

2. **Configure Router Interfaces:**
    - Click on the router and go to the **CLI** (Command Line Interface).
    - Configure IP addresses on the router’s interfaces.

Step 4: Configure Static Routing (Optional for Testing)

1. To enable communication between PCs on different networks, you can configure static routing on the routers.

Step 5: Test Connectivity Using Ping

1. From **PC1**, open the **Command Prompt** (Desktop tab -> Command Prompt).
2. Test the connection by pinging **PC2**
3. If the configuration is correct, you should see replies from 192.168.2.1.

Step 6: Configure Routing Protocols (RIP or OSPF)

1. **RIP Configuration:**
    - On each router, enable RIP and specify the network to advertise.
2. **OSPF Configuration:**
    - To configure OSPF, enable OSPF on the router and configure the area.

Step 7: Verify Routing Table and Connectivity

1. **Check Routing Table:**
    - On each router, use the following command to verify the routing table:
2. **Test Connectivity:**
    - From **PC1**, test the connectivity by pinging **PC2** again:

    If the routing protocol has been configured properly, you should get replies.

Step 8: Save the Configuration

1. After completing the configurations and ensuring the network is functioning correctly, save the configuration on each router.

Conclusion:

- This practical demonstrated how to configure and simulate a network with routers, switches, and hosts using a network simulator.
- We learned how to configure IP addresses, static routing, and dynamic routing protocols (RIP or OSPF).

- We also verified connectivity using the ping command and examined routing tables to ensure proper routing information exchange.

This type of practical provides a foundation for understanding real-world network configurations and routing protocols that are fundamental in any network setup.