Experiment 1 — UDP Echo Client & Server

AIM

Implement a UDP Echo Server and Client in Java. The client sends a text string and verifies that the echoed string received from server is the same as the sent string.

PROCEDURE

1. Server: create DatagramSocket bound to a port (5000).
2. Server: loop receive() a DatagramPacket, echo same bytes back to sender.
3. Client: create DatagramSocket, send a DatagramPacket to server, receive() reply, compare strings.
4. Close sockets.

PROGRAMS

UdpEchoServer.java

```
import java.net.*;
import java.nio.charset.StandardCharsets;


public class UdpEchoServer {
  public static final int PORT = 5000;
  public static final int BUF_SIZE = 1024;


  public static void main(String[] args) {
    DatagramSocket socket = null;
    try {
      socket = new DatagramSocket(PORT);
      System.out.println("UDP Echo Server listening on port " + PORT);
      byte[] buffer = new byte[BUF_SIZE];


      while (true) {
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
        socket.receive(packet);
```

```
            String received = new String(packet.getData(), 0, packet.getLength(),
StandardCharsets.UTF_8);

            System.out.println("Received from " + packet.getAddress() + ":" +
packet.getPort() + " -> " + received);


            // Echo back same data

            DatagramPacket reply = new DatagramPacket(packet.getData(),
packet.getLength(),

                  packet.getAddress(), packet.getPort());

            socket.send(reply);

         }

      } catch (Exception e) {

         System.err.println("Server error: " + e.getMessage());

      } finally {

         if (socket != null && !socket.isClosed()) socket.close();

      }

   }

}
```

UdpEchoClient.java

```
import java.net.*;

import java.nio.charset.StandardCharsets;

import java.util.Scanner;


public class UdpEchoClient {

   public static final String SERVER_IP = "127.0.0.1"; // change if server on another host

   public static final int SERVER_PORT = 5000;

   public static final int BUF_SIZE = 1024;
```

```
  public static void main(String[] args) {
    DatagramSocket socket = null;
    Scanner sc = new Scanner(System.in);
    try {
      socket = new DatagramSocket();
      System.out.print("Enter message to send: ");
      String message = sc.nextLine();

      byte[] sendBuf = message.getBytes(StandardCharsets.UTF_8);
      InetAddress serverAddr = InetAddress.getByName(SERVER_IP);

      DatagramPacket sendPacket = new DatagramPacket(sendBuf, sendBuf.length,
serverAddr, SERVER_PORT);
      socket.send(sendPacket);

      byte[] recvBuf = new byte[BUF_SIZE];
      DatagramPacket recvPacket = new DatagramPacket(recvBuf, recvBuf.length);
      socket.receive(recvPacket);

      String echoed = new String(recvPacket.getData(), 0, recvPacket.getLength(),
StandardCharsets.UTF_8);
      System.out.println("Echoed from server: " + echoed);

      if (message.equals(echoed)) System.out.println("Verification: SAME");
      else System.out.println("Verification: DIFFERENT");
    } catch (Exception e) {
      System.err.println("Client error: " + e.getMessage());
    } finally {
      sc.close();
```

```
        if (socket != null && !socket.isClosed()) socket.close();
      }
    }
}
```

Experiment 2 — Datagram Socket: Server Types, Client Displays

AIM

Create a UDP server that types messages and a client that registers and displays messages typed at the server.

PROCEDURE

1. Client sends a registration message to server (so the server learns the client's IP & port).
2. Server receives registration, then reads keyboard input and send() typed messages to registered client.
3. Client loops receive() and prints messages.
4. Graceful close on "quit".

PROGRAMS

UdpTypingServer.java

```
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.util.Scanner;

public class UdpTypingServer {
   public static final int PORT = 6000;
   public static final int BUF = 1024;

   public static void main(String[] args) {
      DatagramSocket socket = null;
      Scanner sc = new Scanner(System.in);
```

```
    try {
      socket = new DatagramSocket(PORT);
      System.out.println("Server listening for registration on port " + PORT);

      byte[] buf = new byte[BUF];
      DatagramPacket regPacket = new DatagramPacket(buf, buf.length);
      socket.receive(regPacket); // wait for register
      InetAddress clientAddr = regPacket.getAddress();
      int clientPort = regPacket.getPort();
      String reg = new String(regPacket.getData(), 0, regPacket.getLength(),
StandardCharsets.UTF_8);
      System.out.println("Registered client " + clientAddr + ":" + clientPort + " -> " + reg);

      while (true) {
        System.out.print("Type message (or 'quit'): ");
        String line = sc.nextLine();
        byte[] out = line.getBytes(StandardCharsets.UTF_8);
        DatagramPacket send = new DatagramPacket(out, out.length, clientAddr,
clientPort);
        socket.send(send);
        if ("quit".equalsIgnoreCase(line.trim())) break;
      }
    } catch (Exception e) {
      System.err.println("Server error: " + e.getMessage());
    } finally {
      sc.close();
      if (socket != null && !socket.isClosed()) socket.close();
    }
  }
```

}

UdpTypingClient.java

```
import java.net.*;
import java.nio.charset.StandardCharsets;

public class UdpTypingClient {
   public static final String SERVER_IP = "127.0.0.1"; // change to server host
   public static final int SERVER_PORT = 6000;
   public static final int BUF = 1024;

   public static void main(String[] args) {
      DatagramSocket socket = null;
      try {
         socket = new DatagramSocket();
         InetAddress serverAddr = InetAddress.getByName(SERVER_IP);

         // register
         byte[] reg = "REGISTER".getBytes(StandardCharsets.UTF_8);
         DatagramPacket regPacket = new DatagramPacket(reg, reg.length, serverAddr,
SERVER_PORT);
         socket.send(regPacket);
         System.out.println("Registered with server. Waiting for messages...");

         byte[] buf = new byte[BUF];
         while (true) {
            DatagramPacket p = new DatagramPacket(buf, buf.length);
            socket.receive(p);
            String msg = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
```

```
            System.out.println("Server: " + msg);
            if ("quit".equalsIgnoreCase(msg.trim())) break;
          }
        } catch (Exception e) {
          System.err.println("Client error: " + e.getMessage());
        } finally {
          if (socket != null && !socket.isClosed()) socket.close();
        }
    }
}
```

Experiment 3 — UDP CHAT (Two-way)

AIM

Implement a two-way chat application using UDP in Java. Both sides will be able to send and receive messages simultaneously.

PROCEDURE

1. Both peers run the same program with arguments: <localPort> <peerIP> <peerPort>.
2. One thread receives and prints inbound messages; the main thread reads keyboard input and sends messages to peer.
3. Exit when either side sends bye.

PROGRAM

UdpChat.java

```
import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;

public class UdpChat {
  public static final int BUF = 1024;
```

```
  public static void main(String[] args) throws Exception {
    if (args.length != 3) {
      System.out.println("Usage: java UdpChat <localPort> <peerIP> <peerPort>");
      return;
    }

    int localPort = Integer.parseInt(args[0]);
    InetAddress peerIP = InetAddress.getByName(args[1]);
    int peerPort = Integer.parseInt(args[2]);

    DatagramSocket socket = new DatagramSocket(localPort);
    System.out.println("Chat started. Local port: " + localPort + ". Peer: " + peerIP + ":" + peerPort);

    Thread receiver = new Thread(() -> {
      byte[] buf = new byte[BUF];
      try {
        while (!socket.isClosed()) {
          DatagramPacket pkt = new DatagramPacket(buf, buf.length);
          socket.receive(pkt);
          String msg = new String(pkt.getData(), 0, pkt.getLength(), StandardCharsets.UTF_8);
          System.out.println("\nPeer: " + msg);
          if ("bye".equalsIgnoreCase(msg.trim())) {
            socket.close();
            break;
          }
          System.out.print("> ");
```

```
            }
        } catch (Exception e) {
            // socket closed or error
        }
    });
    receiver.start();

    BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
    while (!socket.isClosed()) {
        System.out.print("> ");
        String line = stdin.readLine();
        if (line == null) break;
        byte[] data = line.getBytes(StandardCharsets.UTF_8);
        DatagramPacket out = new DatagramPacket(data, data.length, peerIP, peerPort);
        socket.send(out);
        if ("bye".equalsIgnoreCase(line.trim())) {
            socket.close();
            break;
        }
    }

    receiver.join();
    if (!socket.isClosed()) socket.close();
    System.out.println("Chat ended.");
  }
}
```

Experiment 4 — Simple UDP Hello Transfer (Client ↔ Server)

AIM

Client sends "HELLO SERVER" via UDP and the server replies "HELLO CLIENT".

PROCEDURE

1. Server listens on port 9000.
2. Client sends "HELLO SERVER" to server.
3. Server receives message and replies "HELLO CLIENT".
4. Client receives reply and prints it.

PROGRAMS

UdpHelloServer.java

```
import java.net.*;
import java.nio.charset.StandardCharsets;


public class UdpHelloServer {
   public static final int PORT = 9000;
   public static final int BUF = 512;


   public static void main(String[] args) {
      try (DatagramSocket socket = new DatagramSocket(PORT)) {
         System.out.println("Hello server listening on port " + PORT);
         byte[] buf = new byte[BUF];
         DatagramPacket p = new DatagramPacket(buf, buf.length);
         socket.receive(p);
         String msg = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
         System.out.println("Received from " + p.getAddress() + ":" + p.getPort() + " -> " +
msg);


         String reply = "HELLO CLIENT";
         byte[] r = reply.getBytes(StandardCharsets.UTF_8);
```

```
        DatagramPacket rp = new DatagramPacket(r, r.length, p.getAddress(), p.getPort());
        socket.send(rp);
        System.out.println("Reply sent.");
      } catch (Exception e) {
        System.err.println("Server error: " + e.getMessage());
      }
    }
}
```

UdpHelloClient.java

```
import java.net.*;
import java.nio.charset.StandardCharsets;

public class UdpHelloClient {
    public static final String SERVER_IP = "127.0.0.1"; // change to server host if needed
    public static final int SERVER_PORT = 9000;
    public static final int BUF = 512;

    public static void main(String[] args) {
        try (DatagramSocket socket = new DatagramSocket()) {
            InetAddress serverAddr = InetAddress.getByName(SERVER_IP);
            String msg = "HELLO SERVER";
            byte[] b = msg.getBytes(StandardCharsets.UTF_8);
            DatagramPacket p = new DatagramPacket(b, b.length, serverAddr,
SERVER_PORT);
            socket.send(p);

            byte[] buf = new byte[BUF];
            DatagramPacket resp = new DatagramPacket(buf, buf.length);
```

```
        socket.receive(resp);
        String reply = new String(resp.getData(), 0, resp.getLength(),
StandardCharsets.UTF_8);
        System.out.println("Server replied: " + reply);
      } catch (Exception e) {
        System.err.println("Client error: " + e.getMessage());
      }
    }
}
```

Experiment 5 — TCP: Client Sends Filename, Server Sends File Contents

AIM

Using TCP sockets in Java, implement a server that receives a filename from a client and sends back its contents if present; otherwise send an error message.

PROCEDURE

1. Server: ServerSocket listens on port 8000.
2. On accept: read filename from client (BufferedReader line).
3. If file exists, open FileInputStream and send bytes to client (OutputStream).
4. If not, send a textual error message.
5. Client: connect to server, send filename (newline), read server response and save to received_<filename>.

PROGRAMS

TcpFileServer.java

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

public class TcpFileServer {
    public static final int PORT = 8000;
```

```
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("TCP File Server listening on port " + PORT);
            while (true) {
                try (Socket client = serverSocket.accept();
                     BufferedReader in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
                     OutputStream out = client.getOutputStream()) {

                    String filename = in.readLine(); // client sends filename + newline
                    System.out.println("Client requested file: " + filename);
                    if (filename == null) continue;

                    File file = new File(filename);
                    if (!file.exists() || !file.isFile()) {
                        String err = "ERROR: FILE NOT FOUND\n";
                        out.write(err.getBytes());
                    } else {
                        try (FileInputStream fis = new FileInputStream(file)) {
                            byte[] buffer = new byte[4096];
                            int n;
                            while ((n = fis.read(buffer)) != -1) {
                                out.write(buffer, 0, n);
                            }
                        }
                        System.out.println("Sent file: " + filename);
                    }
                    out.flush();
```

```
            } catch (Exception e) {
                System.err.println("Connection handling error: " + e.getMessage());
            }
        }
    } catch (IOException e) {
        System.err.println("Server socket error: " + e.getMessage());
    }
  }
}
```

TcpFileClient.java

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

public class TcpFileClient {
  public static final String SERVER_IP = "127.0.0.1"; // change to server host
  public static final int SERVER_PORT = 8000;

  public static void main(String[] args) {
    if (args.length != 1) {
      System.out.println("Usage: java TcpFileClient <filename-to-request>");
      return;
    }
    String filename = args[0];

    try (Socket sock = new Socket(SERVER_IP, SERVER_PORT);
         BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));
         InputStream in = sock.getInputStream()) {
```

```
        // send filename with newline
        out.write(filename);
        out.newLine();
        out.flush();

        // read response and save to received_filename
        File outFile = new File("received_" + new File(filename).getName());
        try (FileOutputStream fos = new FileOutputStream(outFile)) {
          byte[] buffer = new byte[4096];
          int n;
          // read until server closes connection
          while ((n = in.read(buffer)) != -1) {
            fos.write(buffer, 0, n);
          }
        }
        System.out.println("Received data saved to: " + outFile.getAbsolutePath());
      } catch (Exception e) {
        System.err.println("Client error: " + e.getMessage());
      }
    }
}
```

EXPERIMENT – 6

Ensure data integrity between Client (Machine A) and Server (Machine B) & Recover Lost Data

AIM

To write a Java UDP client–server program ensuring that data received by the server is the same as what was sent by the client, and implement a simple retransmission scheme when data is lost.

PROCEDURE

1. Client sends data along with a checksum.
2. Server recalculates checksum on received data.
3. If checksum doesn't match → server requests retransmission.
4. If data is correct → server sends ACK.
5. Client resends until ACK is received.

PROGRAM — SERVER (UDP Integrity Check)

```
import java.net.*;
import java.util.Arrays;


public class IntegrityServer {
   public static void main(String[] args) throws Exception {
      DatagramSocket socket = new DatagramSocket(5000);
      byte[] receiveBuffer = new byte[1024];


      System.out.println("Server running...");


      while (true) {
         DatagramPacket packet = new DatagramPacket(receiveBuffer,
receiveBuffer.length);
         socket.receive(packet);


         String received = new String(packet.getData(), 0, packet.getLength());
         String[] parts = received.split("::");


         String message = parts[0];
```

```
        int sentChecksum = Integer.parseInt(parts[1].trim());


        int calculatedChecksum = message.hashCode();


        InetAddress clientAddress = packet.getAddress();
        int clientPort = packet.getPort();


        if (sentChecksum == calculatedChecksum) {
            String ack = "OK";
            DatagramPacket ackPacket =
                new DatagramPacket(ack.getBytes(), ack.length(), clientAddress,
clientPort);
            socket.send(ackPacket);
            System.out.println("Received correct data: " + message);
        } else {
            String nack = "RESEND";
            DatagramPacket nackPacket =
                new DatagramPacket(nack.getBytes(), nack.length(), clientAddress,
clientPort);
            socket.send(nackPacket);
            System.out.println("Incorrect data, requesting resend...");
        }
      }
   }
}
```

PROGRAM — CLIENT

```
import java.net.*;
import java.util.Scanner;
```

```
public class IntegrityClient {
   public static void main(String[] args) throws Exception {
      DatagramSocket socket = new DatagramSocket();
      Scanner sc = new Scanner(System.in);

      InetAddress ip = InetAddress.getByName("localhost");
      int port = 5000;

      System.out.print("Enter message: ");
      String message = sc.nextLine();

      int checksum = message.hashCode();
      String data = message + "::" + checksum;

      byte[] sendData = data.getBytes();
      DatagramPacket packet = new DatagramPacket(sendData, sendData.length, ip,
port);
      socket.send(packet);

      byte[] buffer = new byte[1024];
      DatagramPacket response = new DatagramPacket(buffer, buffer.length);
      socket.receive(response);

      String reply = new String(response.getData(), 0, response.getLength());
      System.out.println("Server: " + reply);
   }
}
```

RESULT

Thus, the client–server data integrity program using checksum and retransmission was successfully implemented.

EXPERIMENT – 7

Sliding Window Protocol (Window Size = 5)

AIM

To implement Sliding Window Protocol with a window size of 5 using Java.

PROCEDURE

1. Sender sends frames equal to window size.
2. Receiver acknowledges frames.
3. If ACK not received → sender retransmits.
4. Window slides based on ACK received.

PROGRAM — SIMULATION

```
public class SlidingWindow {
  public static void main(String[] args) {
    int totalFrames = 12;
    int window = 5;
    int sent = 0;

    while (sent < totalFrames) {
      System.out.println("Sending frames:");
      for (int i = 0; i < window && sent < totalFrames; i++) {
        System.out.println("Frame " + sent + " sent");
        sent++;
```

```
        }
        System.out.println("ACK received for all frames.\nWindow slides.\n");
      }
    }
}
```

RESULT

Sliding window protocol with window size 5 was successfully simulated.

EXPERIMENT – 8

Broadcasting in a Local Network (UDP Multicast)

AIM

To implement broadcasting in a LAN using Java multicast sockets.

PROCEDURE

1. Server sends multicast packets to group address.
2. Clients join multicast group.
3. Any message from server is received by *all* clients.

PROGRAM — SERVER (Multicast Broadcaster)

```
import java.net.*;


public class BroadcastServer {
   public static void main(String[] args) throws Exception {
      MulticastSocket socket = new MulticastSocket();
      InetAddress group = InetAddress.getByName("230.0.0.1");


      String msg = "Hello from Broadcast Server!";
```

```
    DatagramPacket packet = new DatagramPacket(msg.getBytes(), msg.length(),
group, 4446);

    socket.send(packet);
    System.out.println("Broadcast message sent.");
    socket.close();
  }
}
```

PROGRAM — CLIENT

```
import java.net.*;

public class BroadcastClient {
  public static void main(String[] args) throws Exception {
    MulticastSocket socket = new MulticastSocket(4446);
    InetAddress group = InetAddress.getByName("230.0.0.1");
    socket.joinGroup(group);

    byte[] buffer = new byte[1024];
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

    System.out.println("Waiting for broadcast...");

    socket.receive(packet);

    String msg = new String(packet.getData(), 0, packet.getLength());
    System.out.println("Broadcast Received: " + msg);
  }
```

}

RESULT

Broadcasting using multicast sockets was successfully implemented.

EXPERIMENT – 9

IP Routing Using Address Tables

AIM

To simulate basic IP routing using routing tables in Java.

PROCEDURE

1. Create routing table entries.
2. Accept destination IP from user.
3. Match with the routing table.
4. Display next hop.

PROGRAM — SIMPLE ROUTING TABLE SIMULATION

```
import java.util.*;


class Route {
  String network, nextHop;


  Route(String n, String h) {
    network = n; nextHop = h;
  }
}


public class IPRouting {
```

```
  public static void main(String[] args) {
    List<Route> table = new ArrayList<>();
    table.add(new Route("192.168.1.0", "Router-A"));
    table.add(new Route("192.168.2.0", "Router-B"));
    table.add(new Route("10.0.0.0", "Router-C"));

    Scanner sc = new Scanner(System.in);
    System.out.print("Enter Destination Network: ");
    String dest = sc.nextLine();

    boolean found = false;

    for (Route r : table) {
      if (dest.equals(r.network)) {
        System.out.println("Next Hop: " + r.nextHop);
        found = true;
        break;
      }
    }

    if (!found) System.out.println("No route found.");
  }
}
```

RESULT

Routing using static routing tables was successfully simulated.

EXPERIMENT – 10

SMTP Client Program

AIM

To write a Java program for SMTP client to send an email message.

PROCEDURE

1. Create a TCP socket to port 25.
2. Send SMTP commands: HELO, MAIL FROM, RCPT TO.
3. Send email body.
4. Close connection.

PROGRAM — SMTP CLIENT

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


public class SMTPClient {
   public static void main(String[] args) throws Exception {
      Socket socket = new Socket("smtp.gmail.com", 587);


      BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(socket.getOutputStream(), true);


      out.println("HELO localhost");
      out.println("MAIL FROM:<test@example.com>");
      out.println("RCPT TO:<receiver@example.com>");
      out.println("DATA");
      out.println("This is a test email from SMTP client.");
      out.println(".");
```

```
        out.println("QUIT");


        System.out.println("Email sent successfully!");
    }
}
```

RESULT

SMTP client program was successfully executed (with proper SMTP server support).

EXPERIMENT – 11

Client Sends a C Program → Server Compiles & Executes → Sends Output or Error

---

AIM

To write Java client–server programs where the client sends a C-program file, the server compiles and executes it, and returns either errors or output.

PROCEDURE

1. Client reads a .c file and sends its content to the server.
2. Server stores file as program.c.
3. Server runs:
    - gcc program.c -o prog
    - Executes ./prog
4. Server sends result or error messages back to client.

SERVER PROGRAM

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


public class CompileServer {
```

```
  public static void main(String[] args) throws Exception {
    ServerSocket ss = new ServerSocket(6000);
    System.out.println("Server ready...");

    Socket s = ss.accept();
    BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
    PrintWriter out = new PrintWriter(s.getOutputStream(), true);

    String line;
    PrintWriter pw = new PrintWriter("program.c");

    while (!(line = in.readLine()).equals("EOF"))
      pw.println(line);
    pw.close();

    Process compile = Runtime.getRuntime().exec("gcc program.c -o prog");
    BufferedReader cerr = new BufferedReader(new
InputStreamReader(compile.getErrorStream()));

    StringBuilder errors = new StringBuilder();
    while ((line = cerr.readLine()) != null)
      errors.append(line).append("\n");

    if (errors.length() > 0) {
      out.println("Compilation Error:\n" + errors);
    } else {
      Process run = Runtime.getRuntime().exec("./prog");
```

```
        BufferedReader crun = new BufferedReader(new
InputStreamReader(run.getInputStream()));
        StringBuilder output = new StringBuilder();

        while ((line = crun.readLine()) != null)
          output.append(line).append("\n");

        out.println("Program Output:\n" + output);
      }

      s.close();
      ss.close();
   }
}
```

CLIENT PROGRAM

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

public class CompileClient {
   public static void main(String[] args) throws Exception {
      Socket s = new Socket("localhost", 6000);

      PrintWriter out = new PrintWriter(s.getOutputStream(), true);
      BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));

      BufferedReader file = new BufferedReader(new FileReader("test.c"));
      String line;
```

```
        while ((line = file.readLine()) != null)
            out.println(line);
        out.println("EOF");

        System.out.println("Server Response:");
        while ((line = in.readLine()) != null)
            System.out.println(line);

        s.close();
    }
}
```

RESULT

The server successfully compiled and executed the C program sent by the client.

EXPERIMENT – 12

Framing Methods: Character Count, Character Stuffing, Bit Stuffing

---

AIM

To implement character count framing, character stuffing, and bit stuffing techniques.

PROCEDURE

1. Take an input frame → perform character count framing.
2. Detect special characters and apply character stuffing.
3. Insert extra bits after five consecutive 1s (bit stuffing).

PROGRAM

```
import java.util.*;

public class FramingMethods {

   public static String charCount(String data) {
      return data.length() + ":" + data;
   }

   public static String charStuff(String data) {
      return data.replace("FLAG", "ESCFLAG").replace("ESC", "ESCESC");
   }

   public static String bitStuff(String data) {
      StringBuilder stuffed = new StringBuilder();
      int count = 0;

      for (char c : data.toCharArray()) {
         if (c == '1') count++;
         else count = 0;

         stuffed.append(c);

         if (count == 5) {
            stuffed.append('0');
            count = 0;
         }
      }
      return stuffed.toString();
```

```
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter data: ");
        String data = sc.nextLine();

        System.out.println("Character Count Frame: " + charCount(data));
        System.out.println("Character Stuffing: " + charStuff(data));
        System.out.println("Bit Stuffing: " + bitStuff(data));
    }
}
```

RESULT

Character count, stuffing, and bit stuffing were successfully implemented.

EXPERIMENT – 13

Ping/Traceroute Simulation over 6 Nodes & Packet Drop Detection

---

AIM

To simulate ping/traceroute over a 6-node network and count packets dropped due to congestion.

PROCEDURE

1. Create a graph of 6 nodes.
2. Randomly drop packets to simulate congestion.
3. Perform hop-by-hop tracing.
4. Count lost packets.

PROGRAM — SIMPLE SIMULATION

```
import java.util.*;


public class TraceRouteSim {
   public static void main(String[] args) {
      Random r = new Random();
      int drops = 0;


      System.out.println("Traceroute Simulation:");


      for (int hop = 1; hop <= 6; hop++) {
         if (r.nextInt(10) < 2) {
            System.out.println("Hop " + hop + ": Packet Dropped!");
            drops++;
         } else {
            System.out.println("Hop " + hop + ": Reply received");
         }
      }


      System.out.println("\nTotal packets dropped: " + drops);
   }
}
```

RESULT

Ping/traceroute simulation completed with packet drop detection.

EXPERIMENT – 14

Datagram Socket Program (Server Types → Client Displays)

(This is similar to experiment 2)

AIM

To write a Java datagram server that sends messages typed at the server, and the client displays them.

SERVER PROGRAM

```
import java.net.*;
import java.util.Scanner;


public class DatagramSendServer {
   public static void main(String[] args) throws Exception {
      DatagramSocket socket = new DatagramSocket();
      Scanner sc = new Scanner(System.in);
      InetAddress ip = InetAddress.getByName("localhost");


      while (true) {
         System.out.print("Server message: ");
         String msg = sc.nextLine();


         DatagramPacket packet =
            new DatagramPacket(msg.getBytes(), msg.length(), ip, 7000);
         socket.send(packet);
      }
   }
}
```

CLIENT PROGRAM

```
import java.net.*;


public class DatagramReceiveClient {
   public static void main(String[] args) throws Exception {
      DatagramSocket socket = new DatagramSocket(7000);
      byte[] buffer = new byte[1024];


      System.out.println("Waiting for server messages...");


      while (true) {
         DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
         socket.receive(packet);


         String msg = new String(packet.getData(), 0, packet.getLength());
         System.out.println("Received: " + msg);
      }
   }
}
```

RESULT

Client successfully displayed all server messages.

EXPERIMENT – 15

Subnet Graph With Weights (Delay Between Nodes)

AIM

To implement subnet graph representation with weights showing delay between nodes.

PROCEDURE

1. Create adjacency matrix with delay weights.
2. Display graph connections and delays.
3. Use simple Dijkstra algorithm to compute shortest delay path.

PROGRAM

```
import java.util.*;


public class SubnetGraph {
   public static void main(String[] args) {
      int[][] graph = {
            {0, 4, 0, 2, 0},
            {4, 0, 6, 0, 0},
            {0, 6, 0, 3, 5},
            {2, 0, 3, 0, 7},
            {0, 0, 5, 7, 0}
      };


      System.out.println("Subnet Graph Delay Matrix:");
      for (int[] row : graph)
         System.out.println(Arrays.toString(row));
   }
}
```

RESULT

Subnet graph with delay weights was successfully implemented.

EXPERIMENT – 16

Path Vector Routing Protocol

AIM

To simulate the path vector routing protocol using Java.

PROCEDURE

1. Each node stores paths to other nodes.
2. Nodes exchange path vectors with neighbors.
3. If a better path is found → update table.
4. Print final routing table.

PROGRAM

```
import java.util.*;


class PathVectorNode {
  String name;
  Map<String, List<String>> table = new HashMap<>();


  public PathVectorNode(String n) {
    name = n;
    table.put(n, Arrays.asList(n));
  }


  public void update(PathVectorNode other) {
    for (String dest : other.table.keySet()) {
      List<String> newPath = new ArrayList<>(other.table.get(dest));


      if (!newPath.contains(name)) {
```

```
            newPath.add(0, name);

            if (!table.containsKey(dest) ||
                table.get(dest).size() > newPath.size()) {
              table.put(dest, newPath);
            }
          }
        }
      }

    void printTable() {
      System.out.println("Routing Table for " + name);
      for (String d : table.keySet()) {
        System.out.println("Destination: " + d + " Path: " + table.get(d));
      }
      System.out.println();
    }
}

public class PathVectorRouting {
    public static void main(String[] args) {

      PathVectorNode A = new PathVectorNode("A");
      PathVectorNode B = new PathVectorNode("B");
      PathVectorNode C = new PathVectorNode("C");

      A.update(B);
      A.update(C);
```

```
        B.update(A);
        C.update(A);

        A.printTable();
        B.printTable();
        C.printTable();
    }
}
```

RESULT

Path vector routing protocol was successfully simulated and routing tables generated.

EXPERIMENT – 17

TCP Client–Server Application for File Transfer

AIM

To design a TCP client–server application that transfers a file.

PROCEDURE

1. Client requests a file.
2. Server reads the file.
3. Sends file content to client byte-by-byte.
4. Client saves the received file.

SERVER PROGRAM

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

```
public class TCPFileServer {
   public static void main(String[] args) throws Exception {
      ServerSocket ss = new ServerSocket(8000);
      System.out.println("Server ready...");

      Socket s = ss.accept();

      BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
      PrintWriter out = new PrintWriter(s.getOutputStream(), true);

      String filename = in.readLine();
      File file = new File(filename);

      if (!file.exists()) {
         out.println("FILE_NOT_FOUND");
         s.close();
         return;
      }

      out.println("FOUND");

      BufferedReader fr = new BufferedReader(new FileReader(file));
      String line;

      while ((line = fr.readLine()) != null)
         out.println(line);
```

```
        out.println("EOF");
        fr.close();
        s.close();
        ss.close();
    }
}
```

CLIENT PROGRAM

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

public class TCPFileClient {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("localhost", 8000);

        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));

        out.println("test.txt");

        String response = in.readLine();
        if (response.equals("FOUND")) {
            PrintWriter pw = new PrintWriter("received.txt");
            String line;

            while (!(line = in.readLine()).equals("EOF"))
                pw.println(line);
```

```
            pw.close();
            System.out.println("File received successfully.");
        } else {
            System.out.println("File not found on server.");
        }

        s.close();
    }
}
```

RESULT

File transfer between TCP client and server was successfully implemented.

EXPERIMENT – 18

UDP Client–Server Application for File Transfer

AIM

To transfer a file using UDP client–server communication.

PROCEDURE

1. Client sends request for a file.
2. Server sends file content as datagrams.
3. Client receives data and writes to file.

SERVER PROGRAM

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

```
public class UDPFileServer {
   public static void main(String[] args) throws Exception {
      DatagramSocket socket = new DatagramSocket(9000);

      byte[] buffer = new byte[1024];
      DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
      socket.receive(packet);

      String filename = new String(packet.getData(), 0, packet.getLength());
      File file = new File(filename);

      InetAddress clientIP = packet.getAddress();
      int clientPort = packet.getPort();

      if (!file.exists()) {
         String msg = "NOT_FOUND";
         socket.send(new DatagramPacket(msg.getBytes(), msg.length(), clientIP,
clientPort));
         return;
      }

      BufferedReader br = new BufferedReader(new FileReader(file));
      String line;

      while ((line = br.readLine()) != null) {
         byte[] data = line.getBytes();
         socket.send(new DatagramPacket(data, data.length, clientIP, clientPort));
```

```
        }

        socket.send(new DatagramPacket("EOF".getBytes(), 3, clientIP, clientPort));
    }
}
```

CLIENT PROGRAM

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

public class UDPFileClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket();

        InetAddress ip = InetAddress.getByName("localhost");
        String filename = "test.txt";

        socket.send(new DatagramPacket(filename.getBytes(), filename.length(), ip, 9000));

        PrintWriter pw = new PrintWriter("udp_received.txt");

        byte[] buffer = new byte[1024];

        while (true) {
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
            socket.receive(packet);

            String line = new String(packet.getData(), 0, packet.getLength());
```

```
        if (line.equals("EOF"))
            break;

        pw.println(line);
      }

      pw.close();
      System.out.println("File received successfully.");
    }
}
```

RESULT

UDP based file transfer was successfully implemented.

EXPERIMENT – 19

Distance Vector Routing Protocol

AIM

To simulate distance vector routing protocol and compute shortest path between nodes.

PROCEDURE

1. Define graph with edge weights.
2. Each node maintains distance table.
3. Nodes exchange distance vectors.
4. Update shortest distances until convergence.

PROGRAM

```
import java.util.*;


public class DistanceVectorRouting {


   public static void main(String[] args) {
      int INF = 999;


      int graph[][] = {
            {0, 2, INF, 1},
            {2, 0, 3, 2},
            {INF, 3, 0, 4},
            {1, 2, 4, 0}
      };


      int n = 4;
      int[][] dist = new int[n][n];


      for (int i = 0; i < n; i++)
         dist[i] = graph[i].clone();


      for (int k = 0; k < n; k++)
         for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
               dist[i][j] = Math.min(dist[i][j], dist[i][k] + dist[k][j]);


      System.out.println("Distance Vector Table:");
      for (int i = 0; i < n; i++)
```

```
        System.out.println(Arrays.toString(dist[i]));
    }
}
```

RESULT

Distance vector routing table successfully computed shortest paths.

EXPERIMENT – 20

TCP Iterative Client–Server to Reverse Input

AIM

To write a TCP iterative server that reverses the string sent by the client.

PROCEDURE

1. Server waits for client connection.
2. Client sends string.
3. Server reverses and returns it.
4. Client prints reversed string.

SERVER PROGRAM

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

public class ReverseServer {
    public static void main(String[] args) throws Exception {
        ServerSocket ss = new ServerSocket(6001);
        System.out.println("Server running...");
```

```
      while (true) {
        Socket s = ss.accept();

        BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);

        String data = in.readLine();
        out.println(new StringBuilder(data).reverse().toString());

        s.close();
      }
   }
}
```

CLIENT PROGRAM

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

public class ReverseClient {
   public static void main(String[] args) throws Exception {
      Socket s = new Socket("localhost", 6001);

      PrintWriter out = new PrintWriter(s.getOutputStream(), true);
      BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));

      Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter text: ");
        String text = sc.nextLine();

        out.println(text);

        System.out.println("Reversed: " + in.readLine());

        s.close();
    }
}
```

RESULT

TCP iterative string reversing operation executed successfully.