1. The client-server communication is studied in this experiment using socket programming to understand the UDP protocol. The server program and client program is executed separately. Initially, the UDP socket is created at the server and client sides. The binding is carried with the server address. Ensure that the client initiates the communication. Check the response of the client from the server side. Process the datagram packet and send a reply to the client. Observer that the data transfer between Client and Server occurred.

Aim:

To write a socket programming to establish a connection between client and server using UDP protocol.

Algorithm

Server-Side Algorithm (Java):

1. Initialize the server-side:
   - Create a DatagramSocket to listen for incoming datagrams on a specific port.
2. Set up a DatagramPacket for receiving data:
   - Create a byte array to hold the received data.
   - Create a DatagramPacket with the byte array to receive data.
3. Receive data from the client:
   - Use the DatagramSocket's receive method to receive data into the DatagramPacket.
   - Extract the data from the DatagramPacket's byte array.
4. Process the received data:
   - Process the received data as needed (e.g., modify or analyze it).
5. Prepare a response message:
   - Create a byte array to hold the response message.
   - Pack the response message into the byte array.
6. Send a response to the client:
   - Create a DatagramPacket with the response message and the client's address and port.
   - Use the DatagramSocket's send method to send the DatagramPacket to the client.
7. Close resources (optional):
   - The UDP protocol does not require explicit closing of resources like TCP. You can choose to close the DatagramSocket if necessary.

Client-Side Algorithm (Java):

1. Initialize the client-side:
   - Create a DatagramSocket to send and receive datagrams.
2. Prepare a message to be sent:
   - Create a byte array to hold the message to be sent.
   - Pack the message into the byte array.

3. Send the message to the server:
    - Create a DatagramPacket with the message and the server's address and port.
    - Use the DatagramSocket's send method to send the DatagramPacket to the server.
4. Set up a DatagramPacket for receiving data:
    - Create a byte array to hold the received data.
    - Create a DatagramPacket with the byte array to receive data.
5. Receive a response from the server:
    - Use the DatagramSocket's receive method to receive data into the DatagramPacket.
    - Extract the data from the DatagramPacket's byte array.
6. Process the received data:
    - Process the received data as needed (e.g., modify or analyze it).
7. Close resources (optional):
    - The UDP protocol does not require explicit closing of resources like TCP. You can choose to close the DatagramSocket if necessary.

Program

UDP Server

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

public class UDPServer {
   public static void main(String[] args) {
      DatagramSocket serverSocket = null;

      try {
         // Create a UDP socket
         serverSocket = new DatagramSocket(9876);

         byte[] receiveData = new byte[1024];

         System.out.println("Server is waiting for data...");

         while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);

            String clientMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
            InetAddress clientAddress = receivePacket.getAddress();
            int clientPort = receivePacket.getPort();
```

```
            System.out.println("Received from client at " + clientAddress + ":" + clientPort + ": " + clientMessage);

            // Process the received data (you can add your logic here)

            // Send a reply back to the client
            String serverReply = "Hello from the server!";
            byte[] sendData = serverReply.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress, clientPort);
            serverSocket.send(sendPacket);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (serverSocket != null && !serverSocket.isClosed()) {
            serverSocket.close();
        }
    }
  }
}
```

UDP Client

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

public class UDPClient {
    public static void main(String[] args) {
        DatagramSocket clientSocket = null;

        try {
            // Create a UDP socket
            clientSocket = new DatagramSocket();

            InetAddress serverAddress = InetAddress.getByName("127.0.0.1");
            int serverPort = 9876;

            BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

            while (true) {
                System.out.print("Enter a message to send to the server (or 'exit' to quit): ");
                String message = userInput.readLine();

                if (message.equals("exit")) {
```

```
                break;
            }

            byte[] sendData = message.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData,
    sendData.length, serverAddress, serverPort);
            clientSocket.send(sendPacket);

            byte[] receiveData = new byte[1024];
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
    receiveData.length);
            clientSocket.receive(receivePacket);

            String serverReply = new String(receivePacket.getData(), 0,
    receivePacket.getLength());
            System.out.println("Received from server: " + serverReply);
          }
        } catch (IOException e) {
          e.printStackTrace();
        } finally {
          if (clientSocket != null && !clientSocket.isClosed()) {
            clientSocket.close();
          }
        }
      }
}
```

Result

Thus a socket programming to establish a connection between client and server using UDP protocol was executed successfully.