

INDEX

S.No.	Program	Date	Signature	Remarks
1	To study the concept of socket programming			
2	To implement echo server and client in java using TCP sockets			
3	To implement date server and client in java using TCP sockets			
4	To implement chat server and client in java using TCP sockets			
5	To implement simple calculator and invoke arithmetic operations from a remote client			
6	To implement sorting of data using a remote client			
7	To implement bubble sort and sort data using a remote client			
8	Write a code simulating ARP protocol			
9	To implement echo server and client in java using UDP sockets			
10	Write a code simulating RARP protocol			
11	Study of TCP/UDP performance			
12	Study of different types of routing			

PROGRAM 1

To study the concept of socket programming

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols –

- **TCP** – TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** – UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

This chapter gives a good understanding on the following two subjects –

- **Socket Programming** – This is the most widely used concept in Networking and it has been explained in very detail.
- **URL Processing** – This would be covered separately. [Click here to learn about URL Processing in Java language.](#)

Socket Programming

A socket is simply an endpoint for communications between the machines.

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets –

- The server instantiates a `ServerSocket` object, denoting which port number communication is to occur on.
- The server invokes the `accept()` method of the `ServerSocket` class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a `Socket` object, specifying the server name and the port number to connect to.
- The constructor of the `Socket` class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a `Socket` object capable of communicating with the server.
- On the server side, the `accept()` method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an `OutputStream` and an `InputStream`. The client's `OutputStream` is connected to the server's `InputStream`, and the client's `InputStream` is connected to the server's `OutputStream`.

TCP is a two-way communication protocol, hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.

The **`java.net.ServerSocket`** class is used by server applications to obtain a port and listen for client requests.

The **`java.net.Socket`** class represents the socket that both the client and the server use to communicate with each other. The client obtains a `Socket` object by instantiating one, whereas the server obtains a `Socket` object from the return value of the `accept()` method.

PROGRAM 2

To implement echo server and client in java using TCP sockets.

// SERVER SIDE CODE

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

public class MyServer
{
    public static void main(String[] args)
    {
        try
        {
            ServerSocket ss = new ServerSocket(6666);
            Socket s = ss.accept();
            DataInputStream dis = new DataInputStream(s.getInputStream());
            String str = (String)dis.readUTF();
            System.out.println("message = " +str);
            ss.close();
        }catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

```
// CLIENT SIDE CODE
```

```
import java.io.*;
import java.net.*;
public class MyClient
{
    public static void main(String[] args)
    {
        try
        {
            Socket s = new Socket("localhost",6666);
            DataOutputStream dout =
newDataOutputStream(s.getOutputStream());
            dout.writeUTF("Hello Server");
            dout.flush();
            dout.close();
            s.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

OUTPUT:

SERVER

```
Command Prompt X + v
C:\Shrey Jain\1221302>javac MyServer.java
C:\Shrey Jain\1221302>java MyServer
message = Hello Server
C:\Shrey Jain\1221302>
```

CLIENT

```
Command Prompt X + v
C:\Shrey Jain\1221302>javac MyClient.java
C:\Shrey Jain\1221302>java MyClient
C:\Shrey Jain\1221302>
```

PROGRAM 3

To implement date server and client in java using TCP sockets

// SERVER SIDE CODE

```
import java.net.*;
import java.io.*;
import java.util.*;
class DateServer
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket s = new ServerSocket(9999);
        while(true)
        {
            System.out.println("Waiting For Connection...");
            Socket soc = s.accept();
            DataOutputStream out = new
DataOutputStream(soc.getOutputStream());
            out.writeBytes("Server Date" + (new Date()).toString() + "\n");
            out.close();
            soc.close();
        }
    }
}
```

// CLIENT SIDE CODE

```
import java.io.*;
import java.net.*;
class DateClient
{
    public static void main(String args[]) throws Exception
    {
        Socket soc = new Socket("localhost", 9999);
        BufferedReader in = new BufferedReader(new
InputStreamReader(soc.getInputStream()));
        System.out.println(in.readLine());
    }
}
```

OUTPUT:

SERVER

CLIENT

SERVER	CLIENT
<pre>C:\Shrey Jain\1221302>javac DateServer.java C:\Shrey Jain\1221302>java DateServer Waiting For Connection... Waiting For Connection... </pre>	<pre>C:\Shrey Jain\1221302>javac DateClient.java C:\Shrey Jain\1221302>java DateClient Server Date : Sat May 11 14:53:17 IST 2024 C:\Shrey Jain\1221302></pre>

PROGRAM 4

To implement chat server and client in java using TCP sockets

// SERVER SIDE CODE

```
import java.net.*;
import java.io.*;
public class MyServer1
{
    public static void main(String[] args)throws Exception
    {
        ServerSocket ss = new ServerSocket(3333);
        Socket s = ss.accept();
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String str = "", str2 = "";
        while (!str.equals("stop"))
        {
            str = din.readUTF();
            System.out.println("client says: " + str);
            str2 = br.readLine();
            dout.writeUTF(str2);
            dout.flush();
        }
        din.close();
        s.close();
        ss.close();
    }
}
```

// CLIENT SIDE CODE

```
import java.net.*;
import java.io.*;
public class MyClient1
{
    public static void main(String[] args)throws Exception
    {
        Socket s = new Socket("localhost", 3333);
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String str = "", str2 = "";
        while (!str.equals("stop"))
        {
            str = br.readLine();
            dout.writeUTF(str);
            dout.flush();
            str2 = din.readUTF();
            System.out.println("Server says: " + str2);
        }
        dout.close();
        s.close();
    }
}
```

OUTPUT:

SERVER

CLIENT

SERVER	CLIENT
<pre>C:\Shrey Jain\1221302>javac MyServer1.java C:\Shrey Jain\1221302>java MyServer1 client says: Hello hii client says: How are you ? I am fine.</pre>	<pre>C:\Shrey Jain\1221302>javac MyClient1.java C:\Shrey Jain\1221302>java MyClient1 Hello Server says: hii How are you ? Server says: I am fine.</pre>

PROGRAM 5

To implement simple calculator and invoke arithmetic operations from a remote client

// SERVER SIDE CODE

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

public class CalcServer
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket ss= new ServerSocket(7777);
        Socket s=ss.accept();

        DataInputStream dis= new DataInputStream(s.getInputStream());
        DataOutputStream dos= new DataOutputStream(s.getOutputStream());

        while(true)
        {
            String input= dis.readUTF();
            if(input.equals("bye"))
                break;

            System.out.println("Equation recieved:-" +input);
            int result;
            StringTokenizer st =new StringTokenizer(input);
            int oprnd1= Integer.parseInt(st.nextToken());
            String operation= st.nextToken();
            int oprnd2= Integer.parseInt(st.nextToken());

            if(operation.equals("+"))
            {
                result=oprnd1+oprnd2;
            }
            else if(operation.equals("-"))
            {
                result=oprnd1-oprnd2;
            }
            else if(operation.equals("*"))
            {
                result=oprnd1*oprnd2;
            }
            else
            {
                result=oprnd1/oprnd2;
            }
        }
    }
}
```

```

    }

    System.out.println("Sending the result");
    dos.writeInt(result);

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

```

// CLIENT SIDE CODE

```

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

public class CalcClient
{
    public static void main(String args[]) throws Exception
    {
        Socket s=new Socket("localhost", 7777);

        DataInputStream dis= new DataInputStream(s.getInputStream());
        DataOutputStream dos= new DataOutputStream(s.getOutputStream());

        while(true)
        {
            System.out.println("Enter the equation in the form: ");
            System.out.println(" 'operand operator operand' ");
            Scanner sc=new Scanner(System.in);
            String inp= sc.nextLine();

            if(inp.equals("bye"))
                break;

            dos.writeUTF(inp);
            int ans= dis.readInt();
            System.out.println("Answer=" +ans);
        }

        s.close();
    }
}

```

OUTPUT:

SERVER

```
Command Prompt - java Cal x + v
C:\Shrey Jain\1221302>javac CalcServer.java
C:\Shrey Jain\1221302>java CalcServer
Equation recieved: 6 + 8
Sending the result
Equation recieved: 4 - 6
Sending the result
Equation recieved: 2 * 8
Sending the result
|
```

CLIENT

```
Command Prompt - java Calc x + v
C:\Shrey Jain\1221302>javac CalcClient.java
C:\Shrey Jain\1221302>java CalcClient
Enter the equation in the form:
'operand operator operand'
6 + 8
Answer=14
Enter the equation in the form:
'operand operator operand'
4 - 6
Answer=-2
Enter the equation in the form:
'operand operator operand'
2 * 8
Answer=16
Enter the equation in the form:
'operand operator operand'
```

PROGRAM 6

To implement sorting of data using a remote client

// SERVER SIDE CODE

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

class Server
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket ss = new ServerSocket(7777);
        Socket s = ss.accept();
        System.out.println("connected...");
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        int r, i = 0;
        int n = din.readInt();
        int a[] = new int[n];
        System.out.println("data");
        int count = 0;
        System.out.println("Receiving Data...");
        for(i=0;i<n;i++)
        {
            a[i] = din.readInt();
        }
        System.out.println("Data Received");
        System.out.println("Sorting Data...");
        Arrays.sort(a);
        for(i=0; i<n; i++)
        {
            dout.writeInt(a[i]);
        }
        System.out.println("\nData Sent Successfully");
        s.close();
        ss.close();
    }
}
```

```
// CLIENT SIDE CODE
```

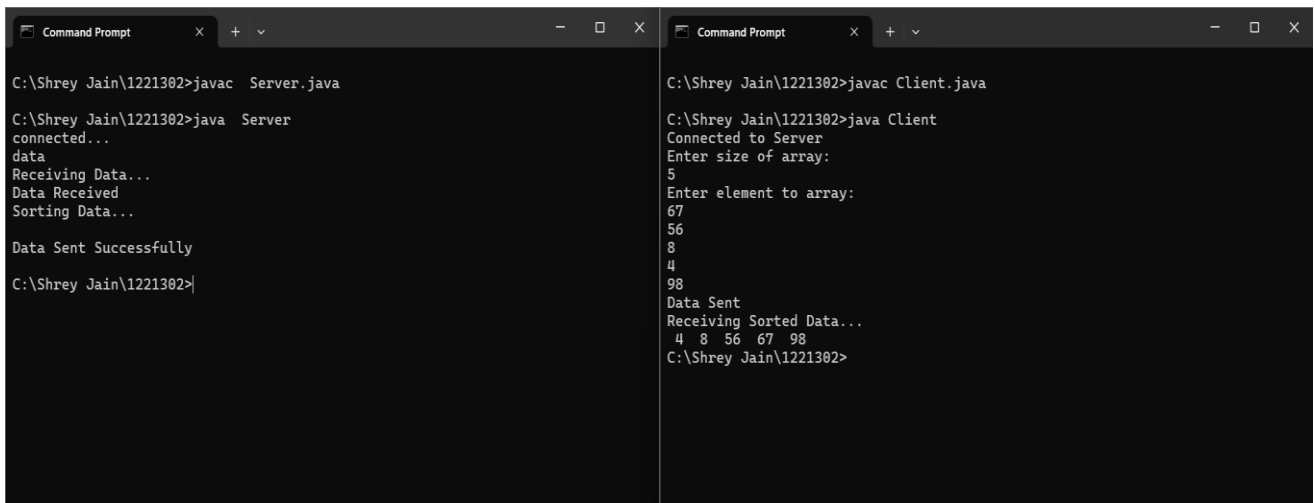
```
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class Client
{
    public static void main(String args[]) throws Exception
    {
        Socket s = new Socket("localhost", 7777);
        if(s.isConnected())
        {
            System.out.println("Connected to Server");
        }
        System.out.println("Enter size of array:");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int a[] = new int[n];
        System.out.println("Enter element to array:");
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        dout.writeInt(n);
        for(int i = 0; i<n; i++)
        {
            a[i] = scanner.nextInt();
            dout.writeInt(a[i]);
        }
        System.out.println("Data Sent");
        DataInputStream din = new DataInputStream(s.getInputStream());
        int r;
        System.out.println("Receiving Sorted Data...");
        for(int i=0; i<n; i++)
        {
            a[i] = din.readInt();
            System.out.print(" " + a[i] + " ");
        }
        s.close();
    }
}
```


OUTPUT:

SERVER

CLIENT



```
C:\Shrey Jain\1221302>javac Server.java
C:\Shrey Jain\1221302>java Server
connected...
data
Receiving Data...
Data Received
Sorting Data...

Data Sent Successfully
C:\Shrey Jain\1221302>
```

```
C:\Shrey Jain\1221302>javac Client.java
C:\Shrey Jain\1221302>java Client
Connected to Server
Enter size of array:
5
Enter element to array:
67
56
8
4
98
Data Sent
Receiving Sorted Data...
4 8 56 67 98
C:\Shrey Jain\1221302>
```

PROGRAM 7

To implement bubble sort and sort data using a remote client

// SERVER SIDE CODE

```
import java.io.*;
import java.net.*;
class ServerBubble
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket ss = new ServerSocket(7777);
        Socket s = ss.accept();
        System.out.println("connected..... ");
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        int r,i=0;
        int n=din.readInt();
        int a[] = new int[n];
        System.out.println("data:");
        int count = 0;
        System.out.println("Receiving data. .");
        for(i=0;i<n;i++)
        {
            a[i]=din.readInt();
        }
        System.out.println("Data Received");
        System.out.println("Sorting Data...");
        int temp=0;
        for(int k=0; k<n;k++)
        {
            for(int j=1;j<(n-k);j++)
            {
                if(a[j-1] > a[j])
                {
                    temp = a[j-1];
                    a[j-1] = a[j];
                    a[j] = temp;
                }
            }
        }

        System.out.println("Data Sorted");
        System.out.println("Sending Data. ..");
        for(i=0; i<n;i++)
        {
            dout.writeInt(a[i]);
        }
    }
}
```

```

    }
    System.out.println("\nData Sent Successfully");
    s.close();
    ss.close();
}
}

```

// CLIENT SIDE CODE

```

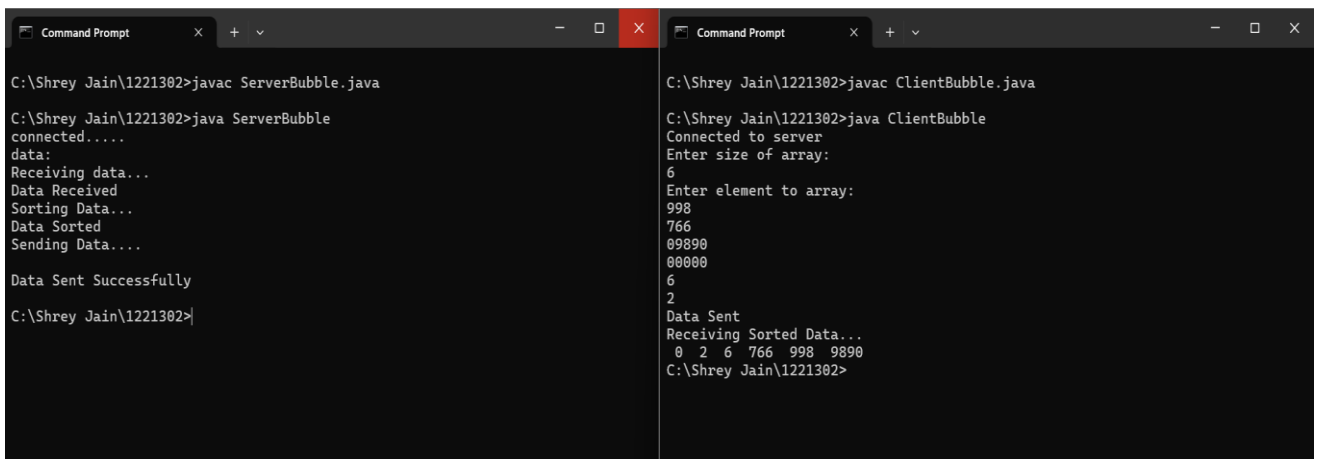
import java.io.*;
import java.net.*;
import java.util.Scanner;
class ClientBubble
{
    public static void main(String args[]) throws Exception
    {
        Socket s = new Socket("localhost", 7777);
        if(s.isConnected())
        {
            System.out.println("Connected to server");
        }
        System.out.println("Enter size of array:");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int a[] = new int[n];
        System.out.println("Enter element to array:");
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        dout.writeInt(n);
        for(int i=0; i<n; i++)
        {
            a[i] = scanner.nextInt();
            dout.writeInt(a[i]);
        }
        System.out.println("Data Sent");
        DataInputStream din = new DataInputStream(s.getInputStream());
        int r;
        System.out.println("Receiving Sorted Data...");
        for(int i=0; i<n; i++)
        {
            a[i] = din.readInt();
            System.out.print(" "+a[i]+" ");
        }
        s.close();
    }
}

```

OUTPUT:

SERVER

CLIENT



```
C:\Shrey Jain\1221302>javac ServerBubble.java
C:\Shrey Jain\1221302>java ServerBubble
connected.....
data:
Receiving data...
Data Received
Sorting Data...
Data Sorted
Sending Data....

Data Sent Successfully
C:\Shrey Jain\1221302>|
```

```
C:\Shrey Jain\1221302>javac ClientBubble.java
C:\Shrey Jain\1221302>java ClientBubble
Connected to server
Enter size of array:
6
Enter element to array:
998
766
09890
00000
6
2
Data Sent
Receiving Sorted Data...
0 2 6 766 998 9890
C:\Shrey Jain\1221302>
```

PROGRAM 8

Write a code simulating ARP protocol

// SERVER SIDE CODE

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverarp
{
    public static void main(String args[])
    {
        try
        {
            ServerSocket obj=new ServerSocket(3333);
            Socket clsct=obj.accept();
            while(true)
            {
                DataInputStream din=new
DataInputStream(clsct.getInputStream());
                DataOutputStream dout=new
DataOutputStream(clsct.getOutputStream());
                String str=din.readLine();
                String ip[]={"165.165.80.80", "165.165.79.1"};
                String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
                for(int i=0; i<ip.length; i++)
                {
                    if(str.equals(ip[i]))
                    {
                        dout.writeBytes(mac[i]+"\\n");
                        break;
                    }
                }
                obj.close();
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
// CLIENT SIDE CODE
```

```

import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp
{
    public static void main(String args[])
    {
        try
        {
            Socket clsct=new Socket("localhost", 3333);
            DataInputStream din=new DataInputStream(clsct.getInputStream());
            DataOutputStream dout=new
DataOutputStream(clsct.getOutputStream());
            BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Enter the Logical address(IP):");
            String str1=in.readLine();
            dout.writeBytes(str1+'\n');
            String str=din.readLine();
            System.out.println("The Physical Address is: "+str);
            clsct.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

```

OUTPUT:

SERVER

CLIENT

SERVER	CLIENT
<pre>C:\Shrey Jain\1221302>javac Serverarp.java Note: Serverarp.java uses or overrides a deprecated API. Note: Recompile with -Xlint:deprecation for details. C:\Shrey Jain\1221302>java Serverarp java.lang.NullPointerException: Cannot invoke "String.equals(Object)" because e "<local5>" is null C:\Shrey Jain\1221302></pre>	<pre>C:\Shrey Jain\1221302>javac Clientarp.java Note: Clientarp.java uses or overrides a deprecated API. Note: Recompile with -Xlint:deprecation for details. C:\Shrey Jain\1221302>java Clientarp Enter the Logical address(IP): 165.165.80.80 The Physical Address is: 6A:08:AA:C2 C:\Shrey Jain\1221302></pre>

PROGRAM 9

To implement echo server and client in java using UDP sockets

// SERVER SIDE CODE

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

public class UdpServer
{
    public static void main(String[] args) throws IOException
    {
        DatagramSocket ds = new DatagramSocket(1234);
        byte[] receive = new byte[65535];

        DatagramPacket DpReceive = null;
        while(true)
        {

            DpReceive = new DatagramPacket(receive, receive.length);
            ds.receive(DpReceive);
            System.out.println("Client:-" + data(receive));

            if (data(receive).toString().equals("bye"))
            {
                System.out.println("Client sent bye...EXITING");
                break;
            }

            receive = new byte[65535];
        }
    }

    public static StringBuilder data(byte[] a)
    {
        if (a == null)
            return null;
        StringBuilder ret = new StringBuilder();
        int i = 0;
        while ( a[i] != 0)
        {
            ret.append((char) a[i]);
            i++;
        }
        return ret;
    }
}
```



```
// CLIENT SIDE CODE
```

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

public class UdpClient
{
    public static void main(String args[]) throws IOException
    {
        Scanner sc = new Scanner(System.in);
        DatagramSocket ds = new DatagramSocket();

        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;

        while(true)
        {
            String inp = sc.nextLine();
            buf = inp.getBytes();
            DatagramPacket DpSend = new DatagramPacket(buf, buf.length, ip,
1234);

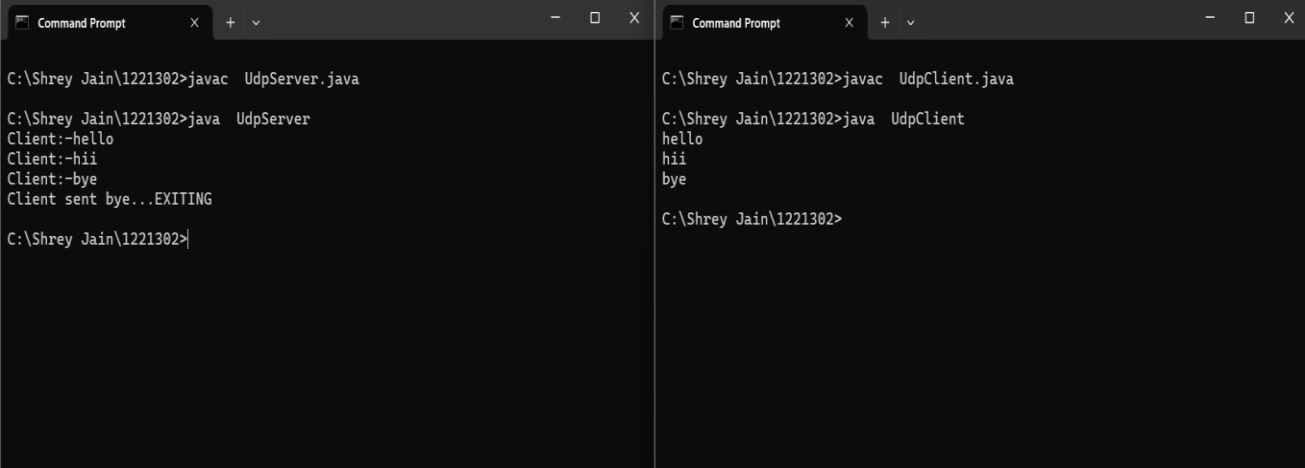
            ds.send(DpSend);

            if(inp.equals("bye"))
                break;
        }
    }
}
```

OUTPUT:

SERVER

CLIENT



```
C:\Shrey Jain\1221302>javac  UdpServer.java
C:\Shrey Jain\1221302>java  UdpServer
Client:-hello
Client:-hii
Client:-bye
Client sent bye...EXITING
C:\Shrey Jain\1221302>|

C:\Shrey Jain\1221302>javac  UdpClient.java
C:\Shrey Jain\1221302>java  UdpClient
hello
hii
bye
C:\Shrey Jain\1221302>
```

PROGRAM 10

Write a code simulating RARP protocol

// SERVER SIDE CODE

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverrarp
{
    public static void main(String args[])
    {
        try
        {
            DatagramSocket server = new DatagramSocket(1309);
            while(true)
            {
                byte[] sendbyte = new byte[1024];
                byte[] receivebyte = new byte[1024];
                DatagramPacket receiver = new DatagramPacket(receivebyte,
receivebyte.length);

                server.receive(receiver);
                String str = new String(receiver.getData());
                String s = str.trim();
                InetAddress addr = receiver.getAddress();
                int port = receiver.getPort();
                String ip[] = {"165.165.80.80", "165.165.79.1"};
                String mac[] = {"6A:08:AA:C2", "8A:BC:E3:FA"};
                for(int i=0;i<mac.length;i++)
                {
                    if(s.equals(mac[i]))
                    {
                        sendbyte = ip[i].getBytes();
                        DatagramPacket sender = new
DatagramPacket(sendbyte, sendbyte.length, addr, port);
                        server.send(sender);
                        break;
                    }
                }
                break;
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

```
// CLIENT SIDE CODE
```

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientrarp
{
    public static void main(String args[])
    {
        try
        {
            DatagramSocket client = new DatagramSocket();
            InetAddress addr = InetAddress.getByName("127.0.0.1");
            byte[] sendbyte=new byte[1024];
            byte[] receivebyte = new byte[1024];
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Enter the Physical address (MAC):");
            String str = in.readLine();
            sendbyte = str.getBytes();
            DatagramPacket sender = new DatagramPacket(sendbyte,
sendbyte.length, addr, 1309);
            client.send(sender);
            DatagramPacket receiver = new DatagramPacket(receivebyte,
receivebyte.length);
            client.receive(receiver);
            String s = new String(receiver.getData());
            System.out.println("The Logical Address is(IP): " + s.trim());
            client.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

OUTPUT:

SERVER

CLIENT

SERVER	CLIENT
<pre>C:\Shrey Jain\1221302>javac Serrerrarp.java C:\Shrey Jain\1221302>java Serrerrarp C:\Shrey Jain\1221302></pre>	<pre>C:\Shrey Jain\1221302>javac Clientrarp.java C:\Shrey Jain\1221302>java Clientrarp Enter the Physical address (MAC): 8A:BC:E3:FA The Logical Address is(IP): 165.165.79.1 C:\Shrey Jain\1221302></pre>

PROGRAM 11

Study of TCP/UDP performance

TCP

TCP stands for **Transmission Control Protocol**. It is a transport layer protocol that facilitates the transmission of packets from source to destination. It is a connection-oriented protocol that means it establishes the connection prior to the communication that occurs between the computing devices in a network. This protocol is used with an IP protocol, so together, they are referred to as a TCP/IP.

The main functionality of the TCP is to take the data from the application layer. Then it divides the data into several packets, provides numbering to these packets, and finally transmits these packets to the destination. The TCP, on the other side, will reassemble the packets and transmits them to the application layer. As we know that TCP is a connection-oriented protocol, so the connection will remain established until the communication is not completed between the sender and the receiver.

Features of TCP protocol:

The following are the features of a TCP protocol:

1. Transport Layer Protocol

TCP is a transport layer protocol as it is used in transmitting the data from the sender to the receiver.

2. Reliable

TCP is a reliable protocol as it follows the flow and error control mechanism. It also supports the acknowledgment mechanism, which checks the state and sound arrival of the data. In the acknowledgment mechanism, the receiver sends either positive or negative acknowledgment to the sender so that the sender can get to know whether the data packet has been received or needs to resend.

3. Order of the data is maintained

This protocol ensures that the data reaches the intended receiver in the same order in which it is sent. It orders and numbers each segment so that the TCP layer on the destination side can reassemble them based on their ordering.

4. Connection-oriented

It is a connection-oriented service that means the data exchange occurs only after the connection establishment. When the data transfer is completed, then the connection will get terminated.

5. Full duplex

It is a full-duplex means that the data can transfer in both directions at the same time.

6. Stream-oriented

TCP is a stream-oriented protocol as it allows the sender to send the data in the form of a stream of bytes and also allows the receiver to accept the data in the form of a stream of bytes. TCP creates an environment in which both the sender and receiver are connected by an imaginary tube known as a virtual circuit. This virtual circuit carries the stream of bytes across the internet.

UDP

The UDP stands for **User Datagram Protocol**. Its working is similar to the TCP as it is also used for sending and receiving the message. The main difference is that UDP is a connectionless protocol. Here, connectionless means that no connection establishes prior to communication. It also does not guarantee the delivery of data packets. It does not even care whether the data has been received on the receiver's end or not, so it is also known as the "fire-and-forget" protocol. It is also known as the "**fire-and-forget**" protocol as it sends the data and does not care whether the data is received or not. UDP is faster than TCP as it does not provide the assurance for the delivery of the packets.

	TCP	UDP
Full form	It stands for Transmission Control Protocol .	It stands for User Datagram Protocol .
Type of connection	It is a connection-oriented protocol, which means that the connection needs to be established before the data is transmitted over the network.	It is a connectionless protocol, which means that it sends the data without checking whether the system is ready to receive or not.
Reliable	TCP is a reliable protocol as it provides assurance for the delivery of data packets.	UDP is an unreliable protocol as it does not take the guarantee for the delivery of packets.
Speed	TCP is slower than UDP as it performs error checking, flow control, and provides assurance for the delivery of	UDP is faster than TCP as it does not guarantee the delivery of data packets.
Header size	The size of TCP is 20 bytes.	The size of the UDP is 8 bytes.
Acknowledgment	TCP uses the three-way-handshake concept. In this concept, if the sender receives the ACK, then the sender will send the data. TCP also has the ability to resend the lost data.	UDP does not wait for any acknowledgment; it just sends the data.
Flow control mechanism	It follows the flow control mechanism in which too many packets cannot be sent to the receiver at the same time.	This protocol follows no such mechanism.
Error checking	TCP performs error checking by using a checksum. When the data is corrected, then the data is retransmitted to the receiver.	It does not perform any error checking, and also does not resend the lost data packets.
Applications	This protocol is mainly used where a secure and reliable communication process is required, like military services, web browsing, and e-mail.	This protocol is used where fast communication is required and does not care about the reliability like VoIP, game streaming, video and music streaming, etc.

PROGRAM 12

Study of different types of routing

Routing

- A Router is a process of selecting path along which the data can be transferred from source to the destination. Routing is performed by a special device known as a router.
- A Router works at the network layer in the OSI model and internet layer in TCP/IP model
- A router is a networking device that forwards the packet based on the information available in the packet header and forwarding table.
- The routing algorithms are used for routing the packets. The routing algorithm is nothing but a software responsible for deciding the optimal path through which packet can be transmitted.
- The routing protocols use the metric to determine the best path for the packet delivery. The metric is the standard of measurement such as hop count, bandwidth, delay, current load on the path, etc. used by the routing algorithm to determine the optimal path to the destination.
- The routing algorithm initializes and maintains the routing table for the process of path determination.

Types of Routing

Routing can be classified into three categories:

- Static Routing
- Default Routing
- Dynamic Routing

Static Routing

- Static Routing is also known as Nonadaptive Routing.
- It is a technique in which the administrator manually adds the routes in a routing table.
- A Router can send the packets for the destination along the route defined by the administrator.
- In this technique, routing decisions are not made based on the condition or topology of the networks

Default Routing

- Default Routing is a technique in which a router is configured to send all the packets to the same hop device, and it doesn't matter whether it belongs to a particular network or not. A Packet is transmitted to the device for which it is configured in default routing.
- Default Routing is used when networks deal with the single exit point.

- It is also useful when the bulk of transmission networks have to transmit the data to the same hp device.
- When a specific route is mentioned in the routing table, the router will choose the specific route rather than the default route. The default route is chosen only when a specific route is not mentioned in the routing table.

Dynamic Routing

- It is also known as Adaptive Routing.
- It is a technique in which a router adds a new route in the routing table for each packet in response to the changes in the condition or topology of the network.
- Dynamic protocols are used to discover the new routes to reach the destination.
- In Dynamic Routing, RIP and OSPF are the protocols used to discover the new routes.
- If any route goes down, then the automatic adjustment will be made to reach the destination.

Features of Dynamic protocol:

- All the routers must have the same dynamic routing protocol in order to exchange the routes.
- If the router discovers any change in the condition or topology, then router broadcast this information to all other routers.

Routing algorithm

- In order to transfer the packets from source to the destination, the network layer must determine the best route through which packets can be transmitted.
- Whether the network layer provides datagram service or virtual circuit service, the main job of the network layer is to provide the best route. The routing protocol provides this job.
- The routing protocol is a routing algorithm that provides the best path from the source to the destination. The best path is the path that has the "least-cost path" from source to the destination.
- Routing is the process of forwarding the packets from source to the destination but the best route to send the packets is determined by the routing algorithm.

Classification of a Routing algorithm

The Routing algorithm is divided into two categories:

- Adaptive Routing algorithm
- Non-adaptive Routing algorithm

Adaptive Routing algorithm

- An adaptive routing algorithm is also known as dynamic routing algorithm.
- This algorithm makes the routing decisions based on the topology and network traffic.

- The main parameters related to this algorithm are hop count, distance and estimated transit time.

An adaptive routing algorithm can be classified into three parts:

- Centralized algorithm: It is also known as global routing algorithm as it computes the least-cost path between source and destination by using complete and global knowledge about the network. This algorithm takes the connectivity between the nodes and link cost as input, and this information is obtained before actually performing any calculation. Link state algorithm is referred to as a centralized algorithm since it is aware of the cost of each link in the network.
- Isolation algorithm: It is an algorithm that obtains the routing information by using local information rather than gathering information from other nodes.
- Distributed algorithm: It is also known as decentralized algorithm as it computes the least-cost path between source and destination in an iterative and distributed manner. In the decentralized algorithm, no node has the knowledge about the cost of all the network links. In the beginning, a node contains the information only about its own directly attached links and through an iterative process of calculation computes the least-cost path to the destination. A Distance vector algorithm is a decentralized algorithm as it never knows the complete path from source to the destination, instead it knows the direction through which the packet is to be forwarded along with the least cost path.

Non-Adaptive Routing algorithm

- Non Adaptive routing algorithm is also known as a static routing algorithm.
- When booting up the network, the routing information stores to the routers.
- Non Adaptive routing algorithms do not take the routing decision based on the network topology or network traffic.

The Non-Adaptive Routing algorithm is of two types:

Flooding: In case of flooding, every incoming packet is sent to all the outgoing links except the one from it has been reached. The disadvantage of flooding is that node may contain several copies of a particular packet.

Random walks: In case of random walks, a packet sent by the node to one of its neighbors randomly. An advantage of using random walks is that it uses the alternative routes very efficiently.

Distance Vector Routing Algorithm

- The Distance vector algorithm is iterative, asynchronous and distributed.
 - Distributed: It is distributed in that each node receives information from one or more of its directly attached neighbors, performs calculation and then distributes the result back to its neighbors.
 - Iterative: It is iterative in that its process continues until no more information is available to be exchanged between neighbors.

- Asynchronous: It does not require that all of its nodes operate in the lock step with each other.
- The Distance vector algorithm is a dynamic algorithm.
- It is mainly used in ARPANET, and RIP.
- Each router maintains a distance table known as Vector.

Three Keys to understand the working of Distance Vector Routing Algorithm:

- Knowledge about the whole network: Each router shares its knowledge through the entire network. The Router sends its collected knowledge about the network to its neighbors.
- Routing only to neighbors: The router sends its knowledge about the network to only those routers which have direct links. The router sends whatever it has about the network through the ports. The information is received by the router and uses the information to update its own routing table.
- Information sharing at regular intervals: Within 30 seconds, the router sends the information to the neighboring routers.

Link State Routing

Link state routing is a technique in which each router shares the knowledge of its neighborhood with every other router in the internetwork.

The three keys to understand the Link State Routing algorithm:

- Knowledge about the neighborhood: Instead of sending its routing table, a router sends the information about its neighborhood only. A router broadcast its identities and cost of the directly attached links to other routers.
- Flooding: Each router sends the information to every other router on the internetwork except its neighbors. This process is known as Flooding. Every router that receives the packet sends the copies to all its neighbors. Finally, each and every router receives a copy of the same information.
- Informationsharing: A router sends the information to every other router only when the change occurs in the information.

Link State Routing has two phases:

1. Reliable Flooding

- **Initial state**: Each node knows the cost of its neighbors.
- **Final state**: Each node knows the entire graph.

2. Route Calculation

Each node uses Dijkstra's algorithm on the graph to calculate the optimal routes to all nodes.

- The Link state routing algorithm is also known as Dijkstra's algorithm which is used to find the shortest path from one node to every other node in the network.
- The Dijkstra's algorithm is an iterative, and it has the property that after k^{th} iteration of the algorithm, the least cost paths are well known for k destination nodes.