DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

ACADEMIC YEAR: 2015-16 ODD SEMESTER

Programme(UG/PG)	**: UG**
Semester	**: IV**
Course Code	**:CS1034**
Course Title	**: COMPUTER NETWORKS LAB**

Prepared By

T.SENTHIL KUMAR
(Assistant Professor(O.G), Computer Science and Engineering)

FACULTY OF ENGINEERING AND TECHNOLOGY
SRM UNIVERSITY
(Under section 3 of UGC Act, 1956)
SRM Nagar, Kattankulathur- 603203
Kancheepuram District

LIST OF EXPERIMENTS & SCHEDULE

COURSE CODE: CS1034

COURSE TITLE: COMPUTER NETWORKS LAB

Exp. No.	**Title**	**Week No.**
1	Networking commands	1
2	Socket Program for Echo/Ping/Talk commands.	2,3,4,5,
3	File transfer	6
4	Remote Command Execution	7
5	Create a socket (UDP)	8
6	Simulation of ARP	9
7	Web page downloading	10
8	TCP Module Implementation	11,12
9	Implementation of RMI	11,12
10	Implementation of Client in C Server in Java	11,12
11	Case study of routing algorithms	11,12

Course Coordinator **HOD**

HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

INTEL PENTIUM 915 GV

80GB HDD

512MB DDR

SOFTWARE REQUIREMENT

ORACLE 8i,9i.

MY SQL,

DB2.

INTERNAL ASSESSMENT MARK SPLIT UP

Observation	: 20 Marks
Attendance	: 5 Marks
Mini Project with the Report (Max. 8 Pages & 3 Students per Batch)	: 20 Marks
Model Exam	: 15 Marks
TOTAL MARKS	: 60 Marks

EXP NO1
NETWORKING COMMANDS

AIM

To study the basic networking commands.

C:\>arp –a: ARP is short form of address resolution protocol, It will show the IP address of your computer along with the IP address and MAC address of your router.

C:\>hostname: This is the simplest of all TCP/IP commands. It simply displays the name of your computer.

C:\>ipconfig: The ipconfig command displays information about the host (the computer your sitting at)computer TCP/IP configuration.

C:\>ipconfig /all: This command displays detailed configuration information about your TCP/IP connection including Router, Gateway, DNS, DHCP, and type of Ethernet adapter in your system.

C:\>Ipconfig /renew: Using this command will renew all your IP addresses that you are currently (leasing) borrowing from the DHCP server. This command is a quick problem solver if you are having connection issues, but does not work if you have been configured with a static IP address.

C:\>Ipconifg /release: This command allows you to drop the IP lease from the DHCP server.

C:\>ipconfig /flushdns: This command is only needed if you're having trouble with your networks DNS configuration. The best time to use this command is after network configuration frustration sets in, and you really need the computer to reply with flushed.

C:\>nbtstat –a: This command helps solve problems with NetBIOS name resolution. (Nbt stands for NetBIOS over TCP/IP)

C:\>netdiag: Netdiag is a network testing utility that performs a variety of network diagnostic tests, allowing you to pinpoint problems in your network. Netdiag isn't installed by default, but can be installed from the Windows XP CD after saying no to the install. Navigate to the CD ROM drive letter and open the support\tools folder on the XP CD and click the setup.exe icon in the support\tools folder.

C:\>netstat: Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you're having trouble with TCP/IP applications such as HTTP, and FTP.

C:\>nslookup: Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not it's DNS you have a DNS problem.

C:\>pathping: Pathping is unique to Window’s, and is basically a combination of the Ping and Tracert commands. Pathping traces the route to the destination address then launches a 25 second test of each router along the way, gathering statistics on the rate of data loss along each hop.

C:\>ping: Ping is the most basic TCP/IP command, and it’s the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with “Hello” on the other end. Computers make phone calls to each other over a network by using a Ping command. The Ping commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can use to place a phone call to another computer on the network. It can use the computers name or IP address.

C:\>route: The route command displays the computers routing table. A typical computer, with a single network interface, connected to a LAN, with a router is fairly simple and generally doesn’t pose any network problems. But if you’re having trouble accessing other computers on your network, you can use the route command to make sure the entries in the routing table are correct.

C:\>tracert: The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

RESULT

Thus the above list of primitive has been studied.

EXP: 2A

SOCKET PROGRAM FOR ECHO.

AIM

To write a socket program for implementation of echo.

ALGORITHM

CLIENT SIDE

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. After establishing connection send a data to server.
4. Receive and print the same data from server.
5. Close the socket.
6. End the program.

SERVER SIDE

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. After establishing connection receive the data from client.
5. Print and send the same data to client.
6. Close the socket.
7. End the program.

PROGRAM

ECHO CLIENT

```
import java.io.*;
import java.net.*;
public class eclient
{
public static void main(String args[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try
{
c=new Socket("localhost",8080);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
```

```
os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do
{
System.out.println("client");
line=is.readLine();
os.println(line);
if(!line.equals("exit"))
System.out.println("server:"+is1.readLine());
}while(!line.equals("exit"));
}
catch(IOException e)
{
System.out.println("socket closed");
}}}
```

Echo Server:

```
import java.io.*;
import java.net.*;
import java.lang.*;
public class eserver
{
public static void main(String args[])throws IOException
{
ServerSocket s=null;
String line;
DataInputStream is;
PrintStream ps;
Socket c=null;
try
{
s=new ServerSocket(8080);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
c=s.accept();
is=new DataInputStream(c.getInputStream());
ps=new PrintStream(c.getOutputStream());
while(true)
{
line=is.readLine();
System.out.println("msg received and sent back to client");
ps.println(line);
}
}
catch(IOException e)
{
System.out.println(e);
}
}
}
```

OUTPUT

CLIEN

```
Enter the IP address 127.0.0.1

CONNECTION ESTABLISHED

Enter the data SRM

Client received SRM
```

SERVER

```
CONNECTION ACCEPTED

Server received SRM
```

RESULT

Thus the program for simulation of echo server was written & executed

EXP: 2B

CLIENT- SERVER APPLICATION FOR CHAT

AIM

To write a client-server application for chat using TCP

ALGORITHM

CLIENT

1. Start the program
2. Include necessary package in java
3. To create a socket in client to server.
4. The client establishes a connection to the server.
5. The client accept the connection and to send the data from client to server.
6. The client communicates the server to send the end of the message
7. Stop the program.

SERVER

1. Start the program
2. Include necessary package in java
3. To create a socket in server to client
4. The server establishes a connection to the client.
5. The server accept the connection and to send the data from server to client and
6. vice versa
7. The server communicate the client to send the end of the message.
8. Stop the program.

PROGRAM

TCPserver1.java

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

public class TCPserver1
{
 public static void main(String arg[])
{
ServerSocket s=null;
String line;
DataInputStream is=null,is1=null;
PrintStream os=null;
Socket c=null;
try
{
s=new ServerSocket(9999);
}
catch(IOException e)
{
System.out.println(e);
 }
try
{
c=s.accept();
is=new DataInputStream(c.getInputStream());
is1=new DataInputStream(System.in);
os=new PrintStream(c.getOutputStream());
do
{
line=is.readLine();
System.out.println("Client:"+line);
System.out.println("Server:");
line=is1.readLine();
os.println(line);
}
while(line.equalsIgnoreCase("quit")==false);
is.close();
os.close();
}
catch(IOException e)
{
System.out.println(e);
}
}
}
```

TCPclient1.java

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

 public class TCPclient1
{
 public static void main(String arg[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try
{
c=new Socket("10.0.200.36",9999);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
 os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do
{
System.out.println("Client:");
line=is.readLine();
os.println(line);
System.out.println("Server:" + is1.readLine());
}
while(line.equalsIgnoreCase("quit")==false);
is1.close();
os.close();
}
catch(IOException e)
{
System.out.println("Socket Closed!Message Passing is over");
}}
```

OUT PUT :

SERVER

```
 C:\Program Files\Java\jdk1.5.0\bin>javac TCPserver1.java
 Note: TCPserver1.java uses or overrides a deprecated API.
 Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java TCPserver1

Client: Hai Server
Server:Hai Client
Client: How are you
Server:Fine
Client: quit
Server:quit
```

CLIENT

```
C:\Program Files\Java\jdk1.5.0\bin>javac TCPclient1.java
 Note: TCPclient1.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java TCPclient1

Client:Hai Server
Server: Hai Client
Client:How are you
Server: Fine
Client:quit
Server: quit
```

RESULT

Thus the above program a client-server application for chat using TCP / IP was executed and successfully.

EXP: 3

FILE TRANSFER IN CLIENT & SERVER

AIM

To Perform File Transfer in Client & Server Using TCP/IP.

ALGORITHM

CLIENT SIDE

1. Start.
2. Establish a connection between the Client and Server.
3. Socket ss=new Socket(InetAddress.getLocalHost(),1100);
4. Implement a client that can send two requests.
   i) To get a file from the server.
   ii) To put or send a file to the server.
5. After getting approval from the server ,the client either get file from the server or send
6. file to the server.

SERVER SIDE

1. Start.
2. Implement a server socket that listens to a particular port number.
3. Server reads the filename and sends the data stored in the file for the‘get’ request.
4. It reads the data from the input stream and writes it to a file in theserver for the ‘put’ instruction.
5. Exit upon client’s request.
6. Stop.

PROGRAM

CLIENT SIDE

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

public class FileClient{
  public static void main (String [] args ) throws IOException {
    int filesize=6022386; // filesize temporary hardcoded
    long start = System.currentTimeMillis();
    int bytesRead;
    int current = 0;
    // localhost for testing
    Socket sock = new Socket("127.0.0.1",13267);
    System.out.println("Connecting...");
    // receive file
    byte [] mybytearray  = new byte [filesize];
    InputStream is = sock.getInputStream();
    FileOutputStream fos = new FileOutputStream("source-copy.pdf");
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    bytesRead = is.read(mybytearray,0,mybytearray.length);
    current = bytesRead;
    // thanks to A. Cádiz for the bug fix
    do {
       bytesRead =
          is.read(mybytearray, current, (mybytearray.length-current));
       if(bytesRead >= 0) current += bytesRead;
    } while(bytesRead > -1);

    bos.write(mybytearray, 0 , current);
    bos.flush();
    long end = System.currentTimeMillis();
    System.out.println(end-start);
    bos.close();
    sock.close();
 }}
```

SERVER SIDE

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

public class FileServer
{
  public static void main (String [] args ) throws IOException {
    ServerSocket servsock = new ServerSocket(13267);
    while (true) {
      System.out.println("Waiting...");
      Socket sock = servsock.accept();
      System.out.println("Accepted connection : " + sock);
     File myFile = new File ("source.pdf");
      byte [] mybytearray  = new byte [(int)myFile.length()];
      FileInputStream fis = new FileInputStream(myFile);
      BufferedInputStream bis = new BufferedInputStream(fis);
      bis.read(mybytearray,0,mybytearray.length);
      OutputStream os = sock.getOutputStream();
      System.out.println("Sending...");
      os.write(mybytearray,0,mybytearray.length);
      os.flush();
      sock.close();
      }}}
```

OUTPUT

SERVER OUTPUT

```
C:\Program Files\Java\jdk1.6.0\bin>javac FServer.java
C:\Program Files\Java\jdk1.6.0\bin>java FServer

Waiting for clients...

Connection Established
Client wants file:network.txt
```

CLIENT OUTPUT

```
C:\Program Files\Java\jdk1.6.0\bin>javac FClient.java
C:\Program Files\Java\jdk1.6.0\bin>java FClient

Connection request.....Connected
Enter the filename: network.txt
Computer networks: A computer network, often simply referred to as a network, is
acollection of computers and devices connected by communications channels
thatfacilitates communications among users and allows users to shareresources with other
user
```

RESULT

Thus the File transfer Operation is done & executed successfully.

EXP: 4

IMPLEMENTATION OF REMOTE COMMAND EXECUTION

AIM

To implement Remote Command Execution(RCE).

ALGORITHM

CLIENT SIDE

1. Establish a connection between the Client and Server.
   Socket client=new Socket("127.0.0.1",6555);
2. Create instances for input and output streams.
   Print Stream ps=new Print Stream(client.getOutputStream());
3. BufferedReader br=new BufferedReader(newInputStreamReader(System.in));
4. Enter the command in Client Window.
   Send the message to its output
   str=br.readLine();
   ps.println(str);

SERVER SIDE

1. Accept the connection request by the client.
   ServerSocket server=new ServerSocket(6555);
   Socket s=server.accept();
2. Get the IPaddress from its inputstream.
   BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
   ip=br1.readLine();
3. During runtime execute the process
   Runtime r=Runtime.getRuntime();
   Process p=r.exec(str);

CLIENT PROGRAM

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

class clientRCE
{
public static void main(String args[]) throws IOException
{
try
{
String str;Socket client=new  Socket("127.0.0.1",6555);
PrintStream ps=new PrintStream(client.getOutputStream());
BufferedReader br=new BufferedReader(newInputStreamReader(System.in));
System.out.println("\t\t\t\tCLIENT WINDOW\n\n\t\tEnter  TheCommand:");
str=br.readLine();
ps.println(str);
}
catch(IOException e)
{
System.out.println("Error"+e);     }}}
```

SERVER PROGRAM:

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

class serverRCE
{
public static void main(String args[]) throws IOException
{
  try
{
String str;
ServerSocket server=new ServerSocket(6555);
Socket s=server.accept();
BufferedReader br=new BufferedReader(newInputStreamReader(s.getInputStream()));
str=br.readLine();
Runtime r=Runtime.getRuntime();
Process p=r.exec(str);
}
catch(IOException e)
{
System.out.println("Error"+e);
}
}}
```

OUTPUT

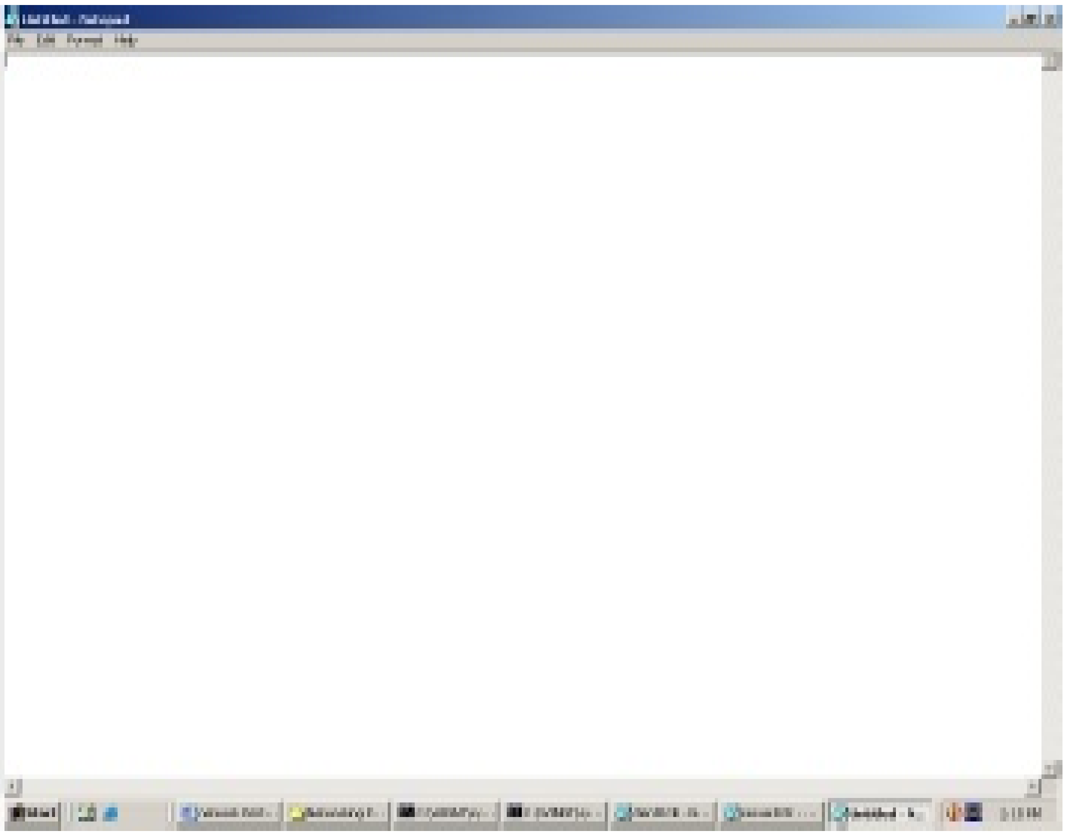
```
C:\Networking Programs>java serverRCE

C:\Networking Programs>java clientRCE
```

CLIENT WINDOW

```
Enter The Command:

Notepad
```



RESULT

Thus the implementation RCE is done & executed successfully.

EXP: 5

CLIENT SERVER APPLICATION IN UDP

AIM

To write a program to implement simple client-server application using UDP.

ALGORITHM

CLIENT SIDE

1. Create a datagram socket with server’s IP address.
2. Create datagram packets with data, data length and the port address.
3. Send the datagram packets to server through datagram sockets
4. Receive the datagram packets from server through datagram sockets
5. Close the socket.

SERVER SIDE

1. Create a datagram socket with port address.
2. Create datagram packets with data, data length and the port address.
3. Send the datagram packets to client through datagram sockets
4. Receive the datagram packets from client through datagram sockets
5. Close the socket.

UDPserver.java

```
import java.io.*;
import java.net.*;
class UDPserver
{
public static DatagramSocket ds;
 public static byte buffer[]=new byte[1024];
 public static int clientport=789,serverport=790;
 public static void main(String args[])throws Exception
{
ds=new DatagramSocket(clientport);
System.out.println("press ctrl+c to quit the program");
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
InetAddress ia=InetAddress.getByName("localhost");
while(true)
{
DatagramPacket p=new DatagramPacket(buffer,buffer.length);
 ds.receive(p);
String psx=new String(p.getData(),0,p.getLength());
System.out.println("Client:" + psx);
System.out.println("Server:");String str=dis.readLine();
if(str.equals("end"))
break;
```

```
 buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,serverport));
}
}
}
```

UDP CLIENT.JAVA

```
import java .io.*;
import java.net.*;
class UDPclient
{
public static DatagramSocket ds;
public static int clientport=789,serverport=790;
public static void main(String args[])throws Exception
{
 byte buffer[]=new byte[1024];
ds=new DatagramSocket(serverport);
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
System.out.println("server waiting");InetAddress ia=InetAddress.getByName("10.0.200.36");
while(true)
{
System.out.println("Client:");
String str=dis.readLine();
if(str.equals("end")) break;
buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,clientport));
DatagramPacket p=new DatagramPacket(buffer,buffer.length);
ds.receive(p);String psx=new String(p.getData(),0,p.getLength());
System.out.println("Server:" + psx);
}
}
}
```

OUTPUT

Server

```
C:\Program Files\Java\jdk1.5.0\bin>javac UDPserver.java
C:\Program Files\Java\jdk1.5.0\bin>java UDPserver
press ctrl+c to quit the program
Client:Hai Server
Server:Hello Client
Client:How are You
Server:I am Fine what about you
```

CLIENT

```
C:\Program Files\Java\jdk1.5.0\bin>javac UDPclient.java
C:\Program Files\Java\jdk1.5.0\bin>java UDPclientserver
Waiting
Client:Hai Server
Server:Hello Clie
Client:How are YouServer:I am Fine
Client:end
```

RESULT

Thus the above program a client-server application for chat using UDP was executed and successfully

EXP: 6
IMPLEMENTATION OF ADDRESS RESOLUTION PROTOCOL

AIM
To implement Address Resolution Protocol .

ALGORITHM

CLIENT SIDE

1. Establish a connection between the Client and Server.
   Socket ss=new Socket(InetAddress.getLocalHost(),1100);
2. Create instance output stream writer
   PrintWriter ps=new PrintWriter(s.getOutputStream(),true);
3. Get the IP Address to resolve its physical address.
4. Send the IPAddress to its output Stream.ps.println(ip);
5. Print the Physical Address received from the server.

SERVER SIDE

1. Accept the connection request by the client.
   ServerSocket ss=new ServerSocket(2000);Socket s=ss.accept();
2. Get the IPaddress from its inputstream.
   BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
   ip=br1.readLine();
3. During runtime execute the processRuntime r=Runtime.getRuntime();
   Process p=r.exec("arp -a "+ip);
4. Send the Physical Address to the client.

PROGRAM

ARP CLIENT

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

class ArpClient
{
public static void main(String args[])throws IOException
{
  try
{
Socket ss=new Socket(InetAddress.getLocalHost(),1100);
PrintStream ps=new PrintStream(ss.getOutputStream());
BufferedReader br=new BufferedReader(newInputStreamReader(System.in));
                    String ip;
                    System.out.println("Enter the IPADDRESS:");
                    ip=br.readLine();
ps.println(ip);
String str,data;
BufferedReader br2=new BufferedReader(newInputStreamReader(ss.getInputStream()));
System.out.println("ARP From Server::");
do
{
str=br2.readLine();
System.out.println(str);
}
while(!(str.equalsIgnoreCase("end")));
}
catch(IOException e)
{
System.out.println("Error"+e);

}}}
```

ARP SERVER

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

class ArpServer
{
public static void main(String args[])throws IOException
{
try
{
ServerSocket ss=new ServerSocket(1100);
Socket s=ss.accept();
PrintStream ps=new PrintStream(s.getOutputStream());
BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
String ip;
 ip=br1.readLine();
Runtime r=Runtime.getRuntime();
Process p=r.exec("arp -a "+ip);
BufferedReader br2=new BufferedReader(newInputStreamReader(p.getInputStream()));
String str;
while((str=br2.readLine())!=null)
{
ps.println(str);
}}
catch(IOException e)
{
System.out.println("Error"+e);  }}}
```

OUTPUT

```
C:\Networking Programs>java ArpServer
C:\Networking Programs>java ArpClient
Enter the IPADDRESS:
192.168.11.58
ARP From Server::
Interface: 192.168.11.57 on Interface 0x1000003
Internet Address      Physical Address        Type
192.168.11.58         00-14-85-67-11-84       dynamic
```

RESULT

Thus the implementation of ARP is done & executed successfully.

EXP: 7

WEB PAGE DOWNLOADING

AIM

To download a webpage using Java

ALGORITHM:

CLIENT SIDE:

1) Start the program.
2) Create a socket which binds the Ip address of server and the port address to acquire service.
3) After establishing connection send the url to server.
4) Open a file and store the received data into the file.
5) Close the socket.
6) End the program.

SERVER SIDE

1) Start the program.
2) Create a server socket to activate the port address.
3) Create a socket for the server socket which accepts the connection.
4) After establishing connection receive url from client.
5) Download the content of the url received and send the data to client.
6) Close the socket.
7) End the program.

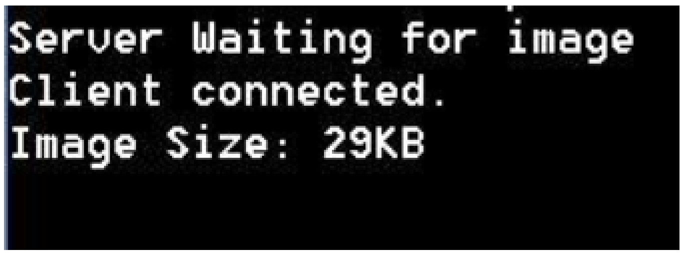
PROGRAM

```
import javax.swing.*;
import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class Client{
public static void main(String args[]) throws Exception{
Socket soc;
BufferedImage img = null;
soc=new Socket("localhost",4000);
System.out.println("Client is running. ");
try {
System.out.println("Reading image from disk. ");
img = ImageIO.read(new File("digital_image_processing.jpg"));
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ImageIO.write(img, "jpg", baos);
baos.flush();
byte[] bytes = baos.toByteArray();
baos.close(); System.out.println("Sending image to server. ");
OutputStream out = soc.getOutputStream();
DataOutputStream dos = new DataOutputStream(out);
dos.writeInt(bytes.length);
dos.write(bytes, 0, bytes.length);
System.out.println("Image sent to server. ");
dos.close();
out.close();
}catch (Exception e) {
System.out.println("Exception: " + e.getMessage());
soc.close();
}
soc.close();
}
}
```

SERVER PROGRAM

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;
class Server {
public static void main(String args[]) throws Exception{
ServerSocket server=null;
Socket socket;
server=new ServerSocket(4000);
System.out.println("Server Waiting for image");
socket=server.accept();
System.out.println("Client connected.");
InputStream in = socket.getInputStream();
DataInputStream dis = new DataInputStream(in);
int len = dis.readInt();
System.out.println("Image Size: " + len/1024 + "KB");
byte[] data = new byte[len];
dis.readFully(data);
dis.close();
in.close();
InputStream ian = new ByteArrayInputStream(data);
BufferedImage bImage = ImageIO.read(ian);
JFrame f = new JFrame("Server");
ImageIcon icon = new ImageIcon(bImage);
JLabel l = new JLabel();
        l.setIcon(icon);
        f.add(l);
        f.pack();
        f.setVisible(true); }}
```

OUTPUT



```
Server Waiting for image
Client connected.
Image Size: 29KB
```

RESULT

The webpage is successfully downloaded and the contents are displayed and verified.

EXP: 8

TCP MODULE IMPLEMENTATION

AIM

To write a socket program for implementation of TCP module.

ALGORITHM

CLIENT SIDE

1) Start the program.
2) Create a socket which binds the Ip address of server and the port address to acquire service.
3) After establishing connection send a data to server.
4) Close the socket.
5) End the program.

SERVER SIDE

1) Start the program.
2) Create a server socket to activate the port address.
3) Create a socket for the server socket which accepts the connection.
4) After establishing connection receive the data from client.
5) Print the data.
6) Close the socket.
7) End the program.

SERVER PROGRAM

```
 import java.io.*;
import java.net.*;
public class server1
{
public static void main(String args[])throws IOException
{
ServerSocket s=new ServerSocket(8080);
System.out.println("socket is created");
System.out.println("waiting for client");
Socket s1=s.accept();
DataOutputStream d1=new DataOutputStream(s1.getOutputStream());
BufferedReader c=new BufferedReader(new InputStreamReader(System.in));
String e=c.readLine();
d1.writeUTF(e);
d1.close();
s1.close();
}}
```

CLIENT PROGRAM

```
import java.io.*;
import java.net.*;
public class client2
{
static String a;
static String b;
public static void main(String args[])throws IOException
{

Socket s=new Socket("127.0.0.1",8080);
BufferedReader c=new BufferedReader(new InputStreamReader(System.in));
DataOutputStream d1=new DataOutputStream(s.getOutputStream());
DataInputStream d2=new DataInputStream(s.getInputStream());
do
{
String st=new String(d2.readUTF());
System.out.println("s::"+st);
System.out.println("c::");
a=c.readLine();
d1.writeUTF(a);
}while(!a.equals("exit"));
d2.close();
d1.close();
s.close();
}
}
```

OUTPUT

CLIENT

```
Enter the IP address 127.0.0.1
CONNECTION ESTABLISHED
Enter the data SRM
```

SERVER

```
CONNECTION ACCEPTED
Server received SRM
```

RESULT

Thus the program for tcp module implementation was written & executed.

EXP: 9
IMPLEMENTATION OF REMOTE METHOD INVOCATION

AIM

To implement Remote Method Invocation.

ALGORITHM

1. Start.
2. Create the rmi interface where the following methods are declared.
   public int add(float a,float b)throws RemoteException;
   public int multiply(int a,int b)throws RemoteException;
   public float divide(float a,float b)throws RemoteException;
3. This application uses four source files-Interface, implementation, server and client programs.
4. Implement remote objects at server side which are declared in the rmiinterface.
5. Generate stubs and skeletons using a tool called the RMI compiler which is invoked from the command line using 'rmic'.
6. Start the RMI registry from the command line as shown here:
   start rmiregistry
7. Start the server and client. Client can be started by passing address of the local machine. At client side the remote objects are called and their process is executed.
8. Stop.

DESCRIPTION

RMI allows a java object that executes on one machine to invoke a method of a Java object that executes on another machine. It allows to build distributed applications. All remote interfaces must extend Remote interface which is a part of java.rmi.
All remote methods can throw Remote Exception.
All remote objects must extend UnicastRemoteObject, which provides functionality that is needed to make objects available from remote machine.

PROGRAM

RMI INTERFACE

```
import java.rmi.*;
public interface rmiInterface extends Remote
{
public int add(float a,float b)throws RemoteException;
public int multiply(int a,int b)throws RemoteException;
public float divide(float a,float b)throws RemoteException;
}
```

RMI SERVER

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;
public class rmiServer extends UnicastRemoteObject implementsrmiInterface
{
```

```
public rmiServer()throws RemoteException
{}
public int add(float a,float b)throws RemoteException
{
int c;c=(int)(a+b);
return(c);
}
public int multiply(int a,int b)throws RemoteException
{
int c;
c=(int)a*b;
return(c);
}
public float divide(float a,float b)throws RemoteException
{
c=(float)a/b;return(c);
}
public static void main(String args[])throws Exception
 {
try{
rmiServer rs=new rmiServer();
Naming.rebind(args[0],rs);
}
catch(Exception e)
{
System.out.println("Error"+e);}}}
```

RMI CLIENT

```
import java.io.*;
import java.rmi.*;
public class rmiClient
{
public static void main(String sdfs[])throws Exception
{
try{
float a,b,c,d ;
int e,f;
String rr;
rmiInterface ri=(rmiInterface)Naming.lookup(sdfs[0]);
BufferedReader br = new BufferedReader(newInputStreamReader(System.in));
do{
 System.out.println("1.Add()\n2.Multiply()\n3.Divide()\n4.Exit()\nEnterU'R choice:");
int sw=Integer.parseInt(br.readLine());
switch(sw)
{
case 1:
System.out.println("Enter the First Value");
a=Float.parseFloat(br.readLine());
System.out.println("Enter the Second Value");
b=Float.parseFloat(br.readLine());
System.out.println("The Added Value Is:"+ri.add(a,b));
break;
case 2:
System.out.println("Enter the First Value");
e=Integer.parseInt(br.readLine());
System.out.println("Enter the Second Value");
f=Integer.parseInt(br.readLine());
System.out.println("The Added Value Is:"+ri.multiply(e,f));
break;
case 3:
System.out.println("Enter the First Value");
c=Float.parseFloat(br.readLine());
System.out.println("Enter the Second Value");
d=Float.parseFloat(br.readLine());
System.out.println("The Added Value Is:"+ri.divide(c,d));
break;
case 4:
System.exit(0);
break;
}
System.out.println("Do u Want to Continue 1/0");
rr=br.readLine();
}
while(rr.equalsIgnoreCase("1"));
}
catch(Exception e)
{
System.out.println("Error"+e);
}}}
```

OUTPUT

```
C:\Networking Programs>rmic rmiServer
C:\Networking Programs>start rmiregistry
C:\Networking Programs>java rmiServer 127.0.0.1
C:\Networking Programs>java rmiClient 127.0.0.11.
1.Add()
2.Multiply()
3.Divide()
4.Exit()
Enter U'R choice:   1
Enter the First Value   12
Enter the Second Value   12
The Added Value Is:   24
Do u Want to Continue 1/0

1.
1.Add()
2.Multiply()
```

```
3.Divide()
4..Exit()
Enter U'R choice: 2
Enter the First Value      12
Enter the Second Value  12
The Added Value Is:      144
Do u Want to Continue   1/0
0
```

RESULT

Thus the implementation RMI is done & executed successfully.

EXP: 10

IMPLEMENTATION OF SERVER IN C/CLIENT IN JAVA

AIM

To write a socket program for implementation of client program in c language and server program in java language.

ALGORITHM

CLIENT SIDE

1. Create a client socket and connect it to the server's port number.
2. Get input from user.
3. If equal to bye or null, then go to step 7.
4. Send user data to the server.
5. Display the data echoed by the server.
6. Repeat steps 2-4.
7. Close the input and output streams.
8. Close the client socket.
9. Stop.

SERVER SIDE

1 Start the program.
2 Create a server socket to activate the port address.
3 Create a socket for the server socket which accepts the connection.
4 After establishing connection receive the data from client.
5 Print and send the data to client till client terminates.
6 Close the socket.
7 End the program.

SERVER PROGRAM IN C

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <netinet/in.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

int main()
{
   int server_sockfd, client_sockfd;
   int server_len, client_len;
   struct sockaddr_in server_address;
   struct sockaddr_in client_address;
```

```
    server_sockfd = socket( AF_INET, SOCK_STREAM, 0 );
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr( "127.0.0.1" );
    server_address.sin_port = htons( 10000 );

    server_len = sizeof( server_address );

       if( bind( server_sockfd, ( struct sockaddr *)&server_address, server_len ) != 0 )
       {
             perror("oops: server-tcp-single");
             exit( 1 );
       }

    listen( server_sockfd, 5 );

    signal( SIGCHLD, SIG_IGN );

    while( 1 )
    {
       char ch;
       printf( "server wait...\n" );

       client_len = sizeof( client_address );
       client_sockfd = accept( server_sockfd, ( struct sockaddr *)&client_address, &client_len
);

       printf( "Client connected \n" );

       if( fork() == 0 )
       {
          read( client_sockfd, &ch, 1 );
          printf( "Client send = %c\n", ch );

          ch++;

          sleep( 5 );

          printf( "Server send = %c\n", ch );
          write( client_sockfd, &ch, 1 );
          close( client_sockfd );
          exit (0 );
       }
       else
          close( client_sockfd );

    }}
```

CLIENT PROGRAM IN JAVA

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

class clientTCP
{
 public static void main(String argv[]) throws Exception
 {
  String sentence;
  String modifiedSentence;
  Socket socket = new Socket("127.0.0.1", 10000);
  InetAddress adresa = socket.getInetAddress(); //address
  System.out.print("Connecting on : "+adresa.getHostAddress()+" with hostname : 
"+adresa.getHostName()+"\n" );
  ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
                        oos.writeObject("HalloXXXX");
  ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
                        String message = (String) ois.readObject();
                        System.out.println("Message Received: " + message);
                        ois.close();
                        oos.close();
                        socket.close();
 }}
```

OUTPUT

CLIENT

```
Enter the IP address 127.0.0.1
CONNECTION ESTABLISHED
Enter the data SRM
Server said: University
Client: Have a good day, BYE!
```

SERVER

```
CONNECTION ACCEPTED
Client Said: SRM
Server: University
Client Said: Have a good day, BYE!
```

RESULT

Thus the programs for implementing client in c language and server program in java language were written & executed.

EXP: 11

CASE STUDY ON ROUTING ALGORITHMS

AIM

To study the various routing algorithms

DESCRIPTION

1. Link state routing algorithm
2. Flooding
3. Distance vector routing algorithm

1.LINK STATE ROUTING

AIM

To study the link state routing

LINK STATE ROUTING

Routing is the process of selecting best paths in a network. In the past, the term routing was also used to mean forwarding network traffic among networks. However this latter function is much better described as simply forwarding. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as the Internet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

In case of overlapping/equal routes, the following elements are considered in order to decide which routes get installed into the routing table (sorted by priority):

1. *Prefix-Length*: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol)
2. *Metric*: where a lower metric/cost is preferred (only valid within one and the same routing protocol)
3. *Administrative distance*: where a lower distance is preferred (only valid between different routing protocols)

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Structured addresses allow a single routing table entry to represent the route to a group of devices. In large networks, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging). Routing has become the dominant form of addressing on the Internet. Bridging is still widely used within localized environments.

2. FLOODING

Flooding s a simple routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on.Flooding is used in bridging and in systems such as Usenet and peer-to-peer file sharing and as part of some routing protocols, including OSPF, DVMRP, and those used in ad-hoc wireless networks.There are generally two types of flooding available, Uncontrolled Flooding and Controlled Flooding.Uncontrolled Flooding is the fatal law of flooding. All nodes have neighbours and route packets indefinitely. More than two neighbours creates a broadcast storm.

Controlled Flooding has its own two algorithms to make it reliable, SNCF (Sequence Number Controlled Flooding) and RPF (Reverse Path Flooding). In SNCF, the node attaches its own address and sequence number to the packet, since every node has a memory of addresses and sequence numbers. If it receives a packet in memory, it drops it immediately while in RPF, the node will only send the packet forward. If it is received from the next node, it sends it back to the sender.

ALGORITHM

There are several variants of flooding algorithm. Most work roughly as follows:

1. Each node acts as both a transmitter and a receiver.
2. Each node tries to forward every message to every one of its neighbours except the source node.

This results in every message eventually being delivered to all reachable parts of the network.

Algorithms may need to be more complex than this, since, in some case, precautions have to be taken to avoid wasted duplicate deliveries and infinite loops, and to allow messages to eventually expire from the system. A variant of flooding called *selective flooding* partially addresses these issues by only sending packets to routers in the same direction. In selective flooding the routers don't send every incoming packet on every line but only on those lines which are going approximately in the right direction.

3. DISTANCE VECTOR

In computer communication theory relating to packet-switched networks, a distance-vector routing protocol is one of the two major classes of routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems's protocols) to calculate paths.

A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The term *distance vector* refers to the fact that the protocol manipulates *vectors* (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of RIP (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and RIPv2 and IGRP.

Method

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.
2. Distance from its destination

Distance-vector protocols are based on calculating the direction and distance to any link in a network. "Direction" usually means the next hop address and the exit interface. "Distance" is a measure of the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics. RIP uses the hop count of the destination whereas IGRP takes into account other information such as node delay and available bandwidth.
Updates are performed periodically in a distance-vector protocol where all or part of a router's routing table is sent to all its neighbors that are configured to use the same distance-vector routing protocol. RIP supports cross-platform distance vector routing whereas IGRP is a Cisco Systems proprietary distance vector routing protocol. Once a router has this information it is able to amend its own routing table to reflect the changes and then inform its neighbors of the changes. This process has been described as ‗routing by rumor' because routers are relying on the information they receive from other routers and cannot determine if the information is actually valid and true. There are a number of features which can be used to help with instability and inaccurate routing information.
EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.
Count-to-infinity problem

The Bellman–Ford algorithm does not prevent routing loops from happening and suffers from the count-to-infinity problem. The core of the count-to-infinity problem is that if A tells B that it has a path somewhere, there is no way for B to know if the path has B as a part of it. To see the problem clearly, imagine a subnet connected like A–B–C–D–E–F, and let the metric between the routers be "number of jumps". Now suppose that A is taken offline. In the vector-update-process B notices that the route to A, which was distance 1, is down – B does not receive the vector update from A. The problem is, B also gets an update from C, and C is still not aware of the fact that A is down – so it tells B that A is only two jumps from C (C to B to A), which is false. This slowly propagates through the network until it reaches infinity (in which case the algorithm corrects itself, due to the relaxation property of Bellman–Ford).

.

RESULT

Thus the study about the various routing algorithms has been completed successfully.