EXPERIMENT 1. Implementation of Stop and Wait Protocol and Sliding Window Protocol.

STOP AND WAIT PROTOCOL

//SENDER//

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
class stopwaitsender
{
public static void main(String args[]) throws Exception
{
stopwaitsender sws = new stopwaitsender();
sws.run();
}
public void run() throws Exception
{
Scanner sc=new Scanner(System.in);
System.out.println("Enter no of frames to be sent:");
int n=sc.nextInt();
Socket myskt=new Socket("localhost",9999);
PrintStream myps=new PrintStream(myskt.getOutputStream());
for(int i=0;i<=n;)
{
if(i==n)
{
myps.println("exit");
break;
}
System.out.println("Frame no "+i+" is sent");
myps.println(i);
BufferedReader bf=new BufferedReader(new InputStreamReader(myskt.getInputStream()));
String ack=bf.readLine();
if(ack!=null)
{
System.out.println("Acknowledgement was Received from receiver");
i++;
Thread.sleep(4000);
}
else
{
myps.println(i);
}
}
}
}
```

```
//RECEIVER//
import java.io.*;
import java.net.*;
class stopwaitreceiver
{
public static void main(String args[])throws Exception
{
stopwaitreceiver swr = new stopwaitreceiver();
swr.run();
}
public void run() throws Exception
{
String temp=”any message”,str=”exit”;
ServerSocket myss=new ServerSocket(9999);
Socket ss_accept=myss.accept();
BufferedReader ss_bf=new BufferedReader(new
InputStreamReader(ss_accept.getInputStream()));
PrintStream myps=new PrintStream(ss_accept.getOutputStream());
while(temp.compareTo(str)!=0)
{
Thread.sleep(1000);
temp=ss_bf.readLine();
if(temp.compareTo(str)==0)
{ break;}
System.out.println(“Frame “+temp+” was received”);
Thread.sleep(500);
myps.println(“Received”);
}
System.out.println(“ALL FRAMES  WERE RECEIVED SUCCESSFULLY”);
}
}
```

OUTPUT FOR SENDER:

```
C:\javaprog>javac stopwaitsender.java
C:\javaprog>java stopwaitsender
Enter no of frames to be sent:
4
Frame no 0 is sent
Acknowledgement was Received from receiver
Frame no 1 is sent
Acknowledgement was Received from receiver
Frame no 2 is sent
Acknowledgement was Received from receiver
Frame no 3 is sent
Acknowledgement was Received from receiver
```

OUTPUT FOR RECEIVER:

```
C:\javaprog>javac stopwaitreceiver.java
C:\javaprog>java stopwaitreceiver
Frame 0 was received
Frame 1 was received
Frame 2 was received
Frame 3 was received
ALL FRAMES  WERE RECEIVED SUCCESSFULLY
```

SLIDING WINDOW PROTOCOL

```java
//import java.lang.System;
import java.net.*;
import java.io.*;
class bitserver
{
public static void main(String[] args)
{try
{BufferedInputStream in;
ServerSocket Serversocket=new
ServerSocket(500);
System.out.println("waiting for connection");
Socket client=Serversocket.accept();
System.out.println("received request for sending
frames");
in=new
BufferedInputStream(client.getInputStream());
DataOutputStream out=new
DataOutputStream(client.getOutputStream());
int p=in.read();
System.out.println("sending.....");
for(int i=1;i<=p;++i)
{System.out.println("sending frame no"+i);
out.write(i);
out.flush();
System.out.println("waiting for acknowledge");
Thread.sleep(5000);
int a=in.read();
System.out.println("received acknowledge for
frame no:"+i+"as"+a);
}
out.flush();

in.close();
out.close();
client.close();
Serversocket.close();
System.out.println("quiting");
}catch(IOException e)
{System.out.println(e);
}
catch(InterruptedException e)
{}
}
}
```

```java
import java.lang.System;
import java.net.*;
import java.io.*;
import java.math.*;
class bitclient
{
public static void main(String a[])
{
try
{InetAddress
addr=InetAddress.getByName("Localhost");
System.out.println(addr);
Socket connection=new Socket(addr,500);
DataOutputStream out=new
DataOutputStream(connection.getOutputStream()
);
BufferedInputStream in=new
BufferedInputStream(connection.getInputStream()
);
BufferedInputStream inn=new
BufferedInputStream(connection.getInputStream()
);
BufferedReader ki=new BufferedReader(new
InputStreamReader(System.in));
int flag=0;
System.out.println("connect");
System.out.println("enter the no of frames to be
requested to server:");
int c=Integer.parseInt(ki.readLine());
out.write(c);
out.flush();
int i,jj=0;
while(jj<c)
{i=in.read();
System.out.println("received frame no"+i);
System.out.println("sending acknowledgement for
frame no"+i);
out.write(i);
out.flush();

 jj++;
}
out.flush();
in.close();
inn.close();
out.close();
```

	System.out.println("quiting");
	}
	catch(Exception e)
	{System.out.println(e);
	}}}

OUTPUT

BIT CLIENT

```
Localhost/127.0.0.1connect
enter the no of frames to be requested to server:4
received frame no1
sending acknowledgement for frame no1
received frame no2
sending acknowledgement for frame no2
received frame no3
sending acknowledgement for frame no3
received frame no4
sending acknowledgement for frame no4
```

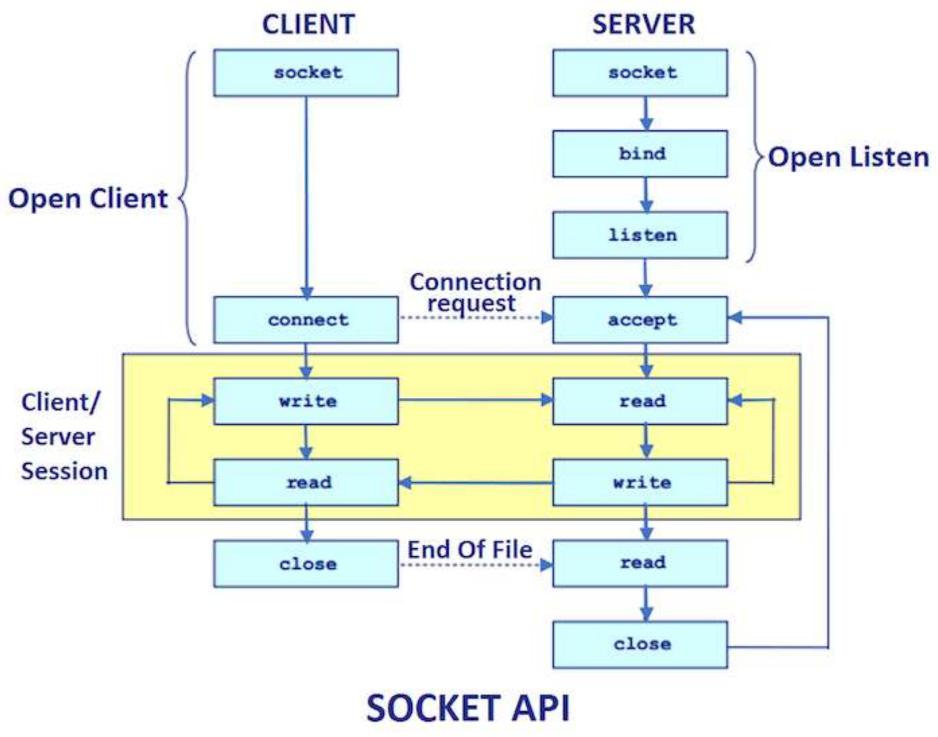
BITSERVER

```
 waiting for connection
received request for sending framessending....
.sending frame no1
waiting for acknowledge
received acknowledge for frame no:1as1
sending frame no2waiting for acknowledge
received acknowledge for frame no:2as2
sending frame no3
waiting for acknowledge
received acknowledge for frame no:3as3
sending frame no4
waiting for acknowledge
received acknowledge for frame no:4as4
```

EXPERIMENT 2. Study of Socket Programming and Client – Server model

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less. The client in socket programming must know two information: IP Address of Server, and Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2)public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example of Java Socket Programming

Creating Server:
To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

```
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection and waits for the client
```

Creating Client:
To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

```
Socket s=new Socket("localhost",6666);
```

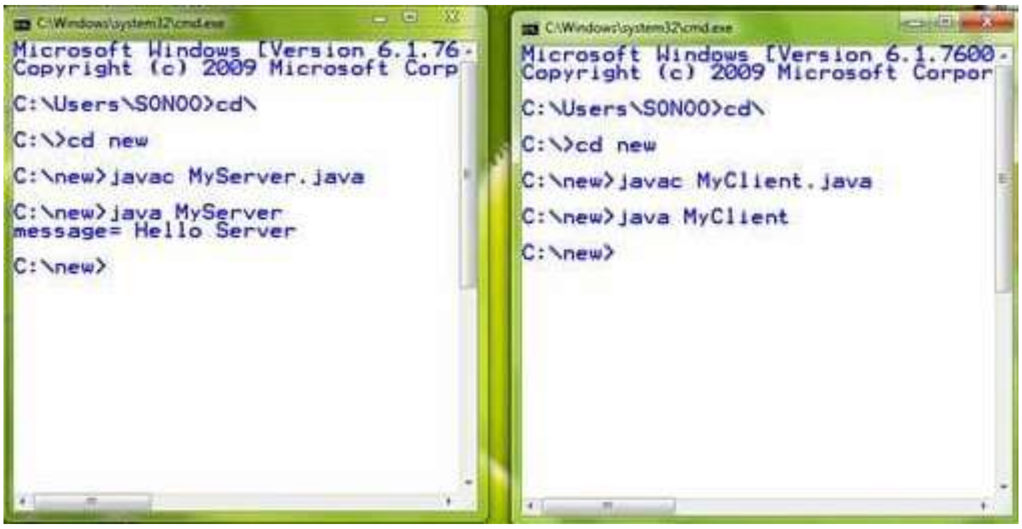
Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

File: MyServer.java

```
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();//establishes connection

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);}
}
}
```

File: MyClient.java

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

public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);

DataOutputStream dout=new
DataOutputStream(s.getOutputStream());

dout.writeUTF("Hello Server");
dout.flush();

dout.close();
s.close();

}catch(Exception e){System.out.println(e);}
}
}
```



```
Microsoft Windows [Version 6.1.76
Copyright (c) 2009 Microsoft Corp

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyServer.java

C:\new>java MyServer
message= Hello Server

C:\new>
```

```
Microsoft Windows [Version 6.1.7600
Copyright (c) 2009 Microsoft Corpor

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyClient.java

C:\new>java MyClient

C:\new>
```

EXPERIMENT 3: Write a code simulating ARP /RARP protocols.

This program automatically creates a file with the IP address of machines, their MAC address and type.

ARP protocol is simulated by reading an IP address and returning the MAC address. RARP protocol is simulated by reading an MAC address and returning the IP address

The program can be extended to read an IP Address / Mac Address and a message and send a packet to the specified machine using TCP / IP or Datagram sockets

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

public class arp_rarp
{
    private static final String Command = "arp -a";

    public static void getARPTable(String cmd) throws Exception
    {
        File fp = new File("ARPTable.txt");
        FileWriter fw = new FileWriter(fp);

        BufferedWriter bw = new BufferedWriter(fw);

        Process P = Runtime.getRuntime().exec(cmd);
        Scanner S = new Scanner(P.getInputStream()).useDelimiter("\\A");

        while(S.hasNext())
           bw.write(S.next());

        bw.close();
        fw.close();
    }



   public static void findMAC(String ip) throws Exception
    {
        File fp = new File("ARPTable.txt");
        FileReader fr = new FileReader(fp);
        BufferedReader br = new BufferedReader(fr);

        String line;

        while ((line = br.readLine()) != null)
        {
            if (line.contains(ip))
           {
               System.out.println("Internet Address      Physical Address      Type");
               System.out.println(line);
               break;
           }
        }
        if((line == null))
```

```
            System.out.println("Not found");

        fr.close();
        br.close();
    }



 public static void findIP(String mac) throws Exception
    {
        File fp = new File("ARPTable.txt");
        FileReader fr = new FileReader(fp);
        BufferedReader br = new BufferedReader(fr);

        String line;

        while ((line = br.readLine()) != null)
        {
            if (line.contains(mac))
            {
                System.out.println("Internet Address      Physical Address      Type");
                System.out.println(line);
                break;
            }
        }

        if((line == null))
            System.out.println("Not found");

        fr.close();
        br.close();
    }



public static void main(String as[]) throws Exception
    {
        getARPTable(Command);

        Scanner S = new Scanner(System.in);

        System.out.println("ARP Protocol.");
        System.out.print("Enter IP Address: ");
        String IP = S.nextLine();
        findMAC(IP);

        System.out.println("RARP Protocol.");
        System.out.print("Enter MAC Address: ");
        String MAC = S.nextLine();
        findIP(MAC);
    }
}
```

OUTPUT:

```
>javac arp_rarp.java
>java arp_rarp
ARP Protocol.
Enter IP Address: 10.0.15.253
Internet Address          Physical Address          Type
  10.0.15.253             00-16-76-bd-41-27         dynamic

RARP Protocol.
Enter MAC Address: 01-00-5e-00-00-fc
Internet Address          Physical Address          Type
  224.0.0.252             01-00-5e-00-00-fc         static

>java arp_rarp
ARP Protocol.
Enter IP Address: 10.0.15.121
Not found

RARP Protocol.
Enter MAC Address: 01-00-5e-00-00-ff
Not found
```

ARPTable.txt

```
Interface: 10.0.15.202 --- 0x3
  Internet Address      Physical Address      Type
  10.0.15.1             54-78-1a-1e-6a-4f     dynamic
  10.0.15.72            00-13-20-b7-49-c9     dynamic
  10.0.15.253           00-16-76-bd-41-27     dynamic
  10.0.15.255           ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
```

EXPERIMENT 4 (a) Write a code simulating PING commands

Algorithm

```
Step 1: start the program.
Step 2: Include necessary package in java.
Step 3: To create a process object p to implement the ping command.
Step 4: declare one BufferedReader stream class object.
Step 5: Get thedetails of the server
        5.1: length of the IP address.
        5.2: time required to get the details.
        5.3: send packets , receive packets and lost packets.
        5.4: minimum ,maximum and average times.
Step 6: print the results.
Step 7:Stop the program.
```

Program:

```
import java.io.*;
import java.net.*;
class pingserver
{
public static void main(String args[])
{
try
{
String str;
System.out.print(" Enter the IP Address to be Ping : ");
BufferedReader buf1=new BufferedReader(new
InputStreamReader(System.in));
String ip=buf1.readLine();
Runtime H=Runtime.getRuntime();
Process p=H.exec("ping " + ip);
InputStream in=p.getInputStream();
BufferedReader buf2=new BufferedReader(new
InputStreamReader(in));
while((str=buf2.readLine())!=null)
{
System.out.println(" " + str);
}
}
catch(Exception e)
{
System.out.println(e.getMessage());
}
}
}
```

Output:

```
Enter the IP address to the ping:192.168.0.1

Pinging 192.168.0.1: with bytes of data =32

Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128

Ping statistics for 192.168.0.1
Packets: sent=4,received=4,lost=0(0% loss),approximate round trip time in milli seconds:
Minimum=1
ms,maximum=4ms,average=2ms
```

EXPERIMENT 4 (b) Write a code simulating Traceroute commands

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class traceroutecmd
{
    public static void runSystemCommand(String command)
    {
        try
        {
           Process p = Runtime.getRuntime().exec(command);
           BufferedReader inputStream = new BufferedReader(
                   new InputStreamReader(p.getInputStream()));

           String s = "";
           while ((s = inputStream.readLine()) != null)
               System.out.println(s);
        }
        catch (Exception e)
        {
        }
    }

    public static void main(String[] args)
    {
        // String ip = "www.google.co.in";
        // String ip = "127.0.0.1";
        String ip = "www.drranurekha.com";
        runSystemCommand("tracert " + ip);
    }
}
```

OUTPUT:

```
>javac traceroutecmd.java

>java traceroutecmd

Tracing route to drranurekha.com [160.153.137.167] over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  10.0.15.1
  2    <1 ms    <1 ms    <1 ms  10.0.0.15
  3     1 ms     1 ms     1 ms  210.212.247.209
  4     2 ms     1 ms     1 ms  172.24.75.102
  5     *        *       21 ms  218.248.235.217
  6     *        *       12 ms  218.248.235.218
  7    21 ms    21 ms    21 ms  121.244.37.253.static.chennai.vsnl.net.in [121.244.37.253]
  8     *        *        *     Request timed out.
  9    49 ms    49 ms    49 ms  172.25.81.134
 10    50 ms    50 ms    70 ms  ix-ae-0-4.tcore1.mlv-mumbai.as6453.net [180.87.38.5]
 11   165 ms   165 ms   165 ms  if-ae-9-5.tcore1.wyn-marseille.as6453.net [80.231.217.17]
 12   172 ms   171 ms   171 ms  if-ae-8-1600.tcore1.pye-paris.as6453.net [80.231.217.6]
 13   171 ms   171 ms   171 ms  if-ae-15-2.tcore1.av2-amsterdam.as6453.net
[195.219.194.145]
 14   175 ms   175 ms   175 ms  195.219.194.2
 15   171 ms   170 ms   170 ms  po72.bbsa0201-01.bbn.mgmt.ams1.gdg [188.121.33.74]
 16   170 ms   169 ms   169 ms  10.241.131.203
 17   175 ms   175 ms   175 ms  10.253.1.1
 18   166 ms   166 ms   166 ms  10.253.130.9
 19   173 ms   173 ms   173 ms  10.253.130.3
 20   169 ms   169 ms   169 ms  10.253.130.5
 21   169 ms   169 ms   169 ms  ip-160-153-137-167.ip.secureserver.net [160.153.137.167]

Trace complete.
```

EXPERIMENT 5. Create a socket for HTTP for web page upload and download.

Algorithm

1.Start the program.
2.Get the frame size from the user
3.To create the frame based on the user request.
4.To send frames to server from the client side.
5.If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6.Stop the program

Program :

Client

```
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

```
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);       }         }
```

EXPERIMENT 6 Write a program to implement RPC (Remote Procedure Call)

A remote procedure call is an inter-process communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

The sequence of events in a remote procedure call are given as follows −

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

HelloWorld.java

```
package
rpc_helloworld;
                 import javax.jws.WebMethod;
                 import javax.jws.WebService;
                 import javax.jws.soap.SOAPBinding;
                 import javax.jws.soap.SOAPBinding.Style;
                 @WebService
                 @SOAPBinding(style = Style.RPC)
                 public interface HelloWorld {
                     @WebMethod String getHelloWorld(String name);
                 }
```

HelloWorldImpl.java

```
package
rpc_helloworld;
                 import javax.jws.WebService;
                 @WebService(endpointInterface =
                 "rpc_helloworld.HelloWorld")
                 public class HelloWorldImpl implements HelloWorld{
                     @Override
                     public String getHelloWorld(String name) {
                         return name;
                     }
                 }
```

Publisher.java

```
package
rpc_helloworld;
                 import javax.xml.ws.Endpoint;
                 public class Publisher {
                     public static void main(String[] args) {

                 Endpoint.publish("http://localhost:7779/ws/hello",
                 new HelloWorldImpl());
                     }
                 }
```

rpc_helloworld.java

```
package
rpc_helloworld;
                 import java.net.MalformedURLException;
                 import java.net.URL;
                 import java.util.logging.Level;
                 import java.util.logging.Logger;
                 import javax.xml.namespace.QName;
                 import javax.xml.ws.Service;
                 public class RPC_HelloWorld {

                     public static void main(String[] args) {
                         try {
                             //Refer to wsdl document
                             URL url = new
                 URL("http://localhost:7779/ws/hello?wsdl");

                             //Refer to wsdl document
                             QName qname = new
                 QName("http://rpc_helloworld/",
                 "HelloWorldImplService");
                             Service service = Service.create(url,
                 qname);
                             HelloWorld hello =
                 service.getPort(HelloWorld.class);


                 System.out.println(hello.getHelloWorld("Hello
                 World!"));

                         } catch (MalformedURLException ex) {
                             System.out.println("WSDL document url
                 error: " + ex);
                         }
                     }
                 }
```

After create all these files, you need to run your Publisher.Java and then go to your browser and type the following:

http://localhost:7779/ws/hello?wsdl

Then you will get the response in XML format. After that you need to copy the text that assign for targetNamespace.

Then paste the text in your Client side file, as the QName first parameter. Now run your program and you will get the output.

[https://sivakumar-prasanth.medium.com/java-rpc-remote-procedure-call-99cfaca34c36]

EXPERIMENT 7 Implementation of Subnetting.

```
import java.util.Scanner;
class Subnet{
public static void main(String args[]){
Scanner sc = new Scanner(System.in);
System.out.print(“Enter the ip address: “);
String ip = sc.nextLine();
String split_ip[] = ip.split(“\\.”); //SPlit the string after every .
String split_bip[] = new String[4]; //split binary ip
String bip = “”;
for(int i=0;i<4;i++){
split_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split_ip[i]))); // “18” => 18
=> 10010 => 00010010
bip += split_bip[i];
}
System.out.println(“IP in binary is “+bip);
System.out.print(“Enter the number of addresses: “);
int n = sc.nextInt();

//Calculation of mask
int bits = (int)Math.ceil(Math.log(n)/Math.log(2)); /*eg if address = 120, log 120/log 2 gives
log to the base 2 => 6.9068, ceil gives us upper integer */
System.out.println(“Number of bits required for address = “+bits);
int mask = 32-bits;
System.out.println(“The subnet mask is = “+mask);

//Calculation of first address and last address
int fbip[] = new int[32];
for(int i=0; i<32;i++) fbip[i] = (int)bip.charAt(i)-48; //convert cahracter 0,1 to integer 0,1
for(int i=31;i>31-bits;i–)//Get first address by ANDing last n bits with 0
fbip[i] &= 0;
String fip[] = {“”,””,””,””};
for(int i=0;i<32;i++)
fip[i/8] = new String(fip[i/8]+fbip[i]);
System.out.print(“First address is = “);
for(int i=0;i<4;i++){
System.out.print(Integer.parseInt(fip[i],2));
if(i!=3) System.out.print(“.”);
}
System.out.println();

int lbip[] = new int[32];
for(int i=0; i<32;i++) lbip[i] = (int)bip.charAt(i)-48; //convert cahracter 0,1 to integer 0,1
for(int i=31;i>31-bits;i–)//Get last address by ORing last n bits with 1
lbip[i] |= 1;
String lip[] = {“”,””,””,””};
for(int i=0;i<32;i++)
lip[i/8] = new String(lip[i/8]+lbip[i]);
System.out.print(“Last address is = “);
for(int i=0;i<4;i++){
System.out.print(Integer.parseInt(lip[i],2));
if(i!=3) System.out.print(“.”);
}
System.out.println();
}
static String appendZeros(String s){
String temp = new String(“00000000”);
return temp.substring(s.length())+ s;
}
}
```

/***Output**

```
Enter the ip address: 100.110.150.10
IP in binary is 01100100011011101001011000001010
Enter the number of addresses: 7
Number of bits required for address = 3
The subnet mask is = 29
First address is = 100.110.150.8
Last address is = 100.110.150.15
*/
```

EXPERIMENT 8.a IMPLEMENTATION OF TCP/IP ECHO.

ALGORITHM

Server

```
1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Read the data from client.
4. Echo the data back to the client.
5. Repeat steps 4-5 until „bye‟ or „null‟ is read.
6. Close all streams.
7. Close the server socket.
8. Stop.
```

Client

```
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.
```

OUTPUT

Server:

```
$ javac tcpechoserver.java
$ java tcpechoserver
Server Ready Client Connected Client [ hello ]
Client [ how are you ] Client [ i am fine ] Client [ ok ]
Client Disconnected
```

Client :

```
$ javac tcpechoclient.java
$ java tcpechoclient
Type "bye" to quit
Enter msg to server : hello
Server [ hello ]
Enter msg to server : how are you
Server [ how are you ]
Enter msg to server : i am fine
Server [ i am fine ]
Enter msg to server : ok
Server [ ok ]
Enter msg to server : bye
```

```
/ TCP Echo Server--tcpechoserver.java
import java.net.*;
import java.io.*;
public class tcpechoserver
{
public static void main(String[] arg) throws
IOException
{
ServerSocket sock = null;
BufferedReader fromClient = null;
OutputStreamWriter toClient = null;
Socket client = null;
try
{
sock = new ServerSocket(4000);
System.out.println("Server Ready");
client = sock.accept();
System.out.println("Client Connected");
fromClient = new BufferedReader(new
InputStreamReader(client.getInputStream()));
toClient = new
OutputStreamWriter(client.getOutputStream());
String line;
while (true)
{
line = fromClient.readLine();
if ( (line == null) || line.equals("bye"))
break;
System.out.println ("Client [ " + line + " ]");
toClient.write("Server [ "+ line +" ]\n");
toClient.flush();
}
fromClient.close();
toClient.close();
client.close();
sock.close();
System.out.println("Client Disconnected");
}
catch (IOException ioe)
{
System.err.println(ioe);
}
}
}
```

```
//TCP Echo Client--tcpechoclient.java
import java.net.*;
import java.io.*;
public class tcpechoclient
{
public static void main(String[] args) throws
IOException
{
BufferedReader fromServer = null, fromUser =
null;
PrintWriter toServer = null;
Socket sock = null;
try
{
if (args.length == 0)
sock = new
Socket(InetAddress.getLocalHost(),4000);
else
sock = new
Socket(InetAddress.getByName(args[0]),4000);
fromServer = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
fromUser = new BufferedReader(new
InputStreamReader(System.in));
toServer = new
PrintWriter(sock.getOutputStream(),true);
String Usrmsg, Srvmsg;
System.out.println("Type \"bye\" to quit");
while (true)
{
System.out.print("Enter msg to server : ");
Usrmsg = fromUser.readLine();
if (Usrmsg==null || Usrmsg.equals("bye"))
{
toServer.println("bye"); break;
}
else
toServer.println(Usrmsg);
Srvmsg = fromServer.readLine();
System.out.println(Srvmsg);
}
fromUser.close();
fromServer.close();
toServer.close();
sock.close();
}
catch (IOException ioe)
{
System.err.println(ioe);
}
```

EXPERIMENT 8.b. IMPLEMENTATION OF TCP/IP Chat

```
// A Java program for a Client
import java.net.*;
import java.io.*;

public class Client
{
   // initialize socket and input output
streams
   private Socket socket            = null;
   private DataInputStream  input   = null;
   private DataOutputStream out     = null;

   // constructor to put ip address and port
   public Client(String address, int port)
   {
      // establish a connection
      try
      {
         socket = new Socket(address, port);
         System.out.println("Connected");

         // takes input from terminal
         input  = new
DataInputStream(System.in);

         // sends output to the socket
         out    = new
DataOutputStream(socket.getOutputStream()
);
      }
      catch(UnknownHostException u)
      {
         System.out.println(u);
      }
      catch(IOException i)
      {
         System.out.println(i);
      }

      // string to read message from input
      String line = "";

      // keep reading until "Over" is input
      while (!line.equals("Over"))
      {
         try
         {
            line = input.readLine();
            out.writeUTF(line);
         }
         catch(IOException i)
         {
            System.out.println(i);
```

```
// A Java program for a Server
import java.net.*;
import java.io.*;

public class Server
{
   //initialize socket and input stream
   private Socket          socket   = null;
   private ServerSocket    server   = null;
   private DataInputStream in       =  null;

   // constructor with port
   public Server(int port)
   {
      // starts server and waits for a connection
      try
      {
         server = new ServerSocket(port);
         System.out.println("Server started");

         System.out.println("Waiting for a
client ...");

         socket = server.accept();
         System.out.println("Client accepted");

         // takes input from the client socket
         in = new DataInputStream(
            new
BufferedInputStream(socket.getInputStream())
);

         String line = "";

         // reads message from client until
"Over" is sent
         while (!line.equals("Over"))
         {
            try
            {
               line = in.readUTF();
               System.out.println(line);

            }
            catch(IOException i)
            {
               System.out.println(i);
            }
         }
         System.out.println("Closing
connection");

         // close connection
```

```
            }
        }

        // close the connection
        try
        {
            input.close();
            out.close();
            socket.close();
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        Client client = new Client("127.0.0.1",
5000);
    }
}
```

```
$ java Client

Hello

I made my first socket
connection

Over
```

```
            socket.close();
            in.close();
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        Server server = new Server(5000);
    }
}
```

```
$ java Server

Hello

I made my first socket connection

Over

Closing connection
```

EXPERIMENT 8.c. IMPLEMENTATION OF TCP/IP SOCKET FILE TRANSFER

Algorithm Server

Step1: Import java packages and create class file server.
Step2: Create a new server socket and bind it to the port.
Step3: Accept the client connection
Step4: Get the file name and stored into the BufferedReader.
Step5: Create a new object class file and realine.
Step6: If file is exists then FileReader read the content until EOF is reached.
Step7: Stop the program.

Client

Step1: Import java packages and create class file server.
Step2: Create a new server socket and bind it to the port.
Step3: Now connection is established.
Step4: The object of a BufferReader class is used for storing data content which has been retrieved from socket object.
Step5 The connection is closed.
Step6: Stop the program.

Program

File Server :

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;


public class FileServer
{
    public static void main(String[] args) throws Exception
        {
            //Initialize Sockets
            ServerSocket ssock = new ServerSocket(5000);
            Socket socket = ssock.accept();
          //The InetAddress specification
                InetAddress IA = InetAddress.getByName("localhost");

        //Specify the file
            File file = new File("e:\\Bookmarks.html");
            FileInputStream fis = new FileInputStream(file);
                BufferedInputStream bis = new BufferedInputStream(fis);
                //Get socket's output stream
                OutputStream os = socket.getOutputStream();
            //Read File Contents into contents array
                byte[] contents;
               long fileLength = file.length();
                long current = 0;
                long start = System.nanoTime();
                while(current!=fileLength){
                        int size = 10000;
                        if(fileLength - current >= size)
                           current += size;
                        else{
                                    size = (int)(fileLength - current);
                                    current = fileLength;
                         }
                  contents = new byte[size];
                bis.read(contents, 0, size);
                os.write(contents);
                System.out.print("Sending file ... "+(current*100)/fileLength+"% complete!");
                }
          os.flush();
     //File transfer done. Close the socket connection!
        socket.close();
        ssock.close();
        System.out.println("File sent succesfully!");
        } }
```

File Client

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;

public class FileClient {
    public static void main(String[] args) throws Exception{
        //Initialize socket
                Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
                    byte[] contents = new byte[10000];
        //Initialize the FileOutputStream to the output file's full path.
                FileOutputStream fos = new FileOutputStream("e:\\Bookmarks1.html");
                    BufferedOutputStream bos = new BufferedOutputStream(fos);
                    InputStream is = socket.getInputStream();
         //No of bytes read in one read() call
                int bytesRead = 0;
               while((bytesRead=is.read(contents))!=-1)
                         bos.write(contents, 0, bytesRead);
               bos.flush();
               socket.close();
               System.out.println("File saved successfully!");
    }
}
```

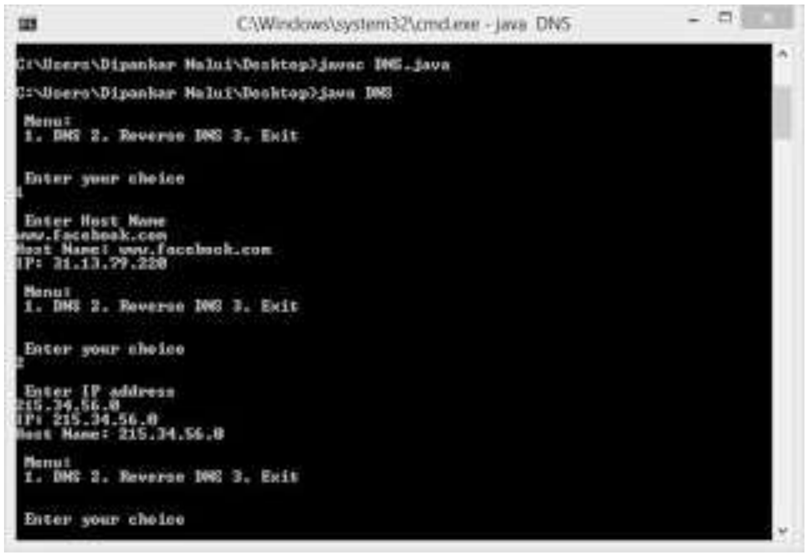
Output

server

```
E:\nwlab>java FileServer
Sending file ... 9% complete!
Sending file ... 19% complete!
Sending file ... 28% complete!
Sending file ... 38% complete!
Sending file ... 47% complete!
Sending file ... 57% complete!
Sending file ... 66% complete!
Sending file ... 76% complete!
Sending file ... 86% complete!
Sending file ... 95% complete!
Sending file ... 100% complete!
File sent successfully!

E:\nwlab>client
E:\nwlab>java FileClient
File saved successfully!

E:\nwlab>
```

EXPERIMENT 9.a. Program to implement DNS in java

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

public class DNS
{
 public static void main(String[] args)
 {
  int n;
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  do
  {
   System.out.println("\n Menu: \n 1. DNS 2. Reverse DNS 3. Exit \n");
   System.out.println("\n Enter your choice");
   n = Integer.parseInt(System.console().readLine());
   if(n==1)
   {
    try
    {
     System.out.println("\n Enter Host Name ");
     String hname=in.readLine();
     InetAddress address;
     address = InetAddress.getByName(hname);
     System.out.println("Host Name: " + address.getHostName());
     System.out.println("IP: " + address.getHostAddress());
    }
    catch(IOException ioe)
    {
     ioe.printStackTrace();
    }
   }
   if(n==2)
   {
    try
    {
       System.out.println("\n Enter IP address");
       String ipstr = in.readLine();
       InetAddress ia = InetAddress.getByName(ipstr);
       System.out.println("IP: "+ipstr);
       System.out.println("Host Name: " +ia.getHostName());
     }
    catch(IOException ioe)
    {
     ioe.printStackTrace();
    }
   }
  }while(!(n==3));
 }
}
```



C:\Windows\system32\cmd.exe - java DNS

```
C:\Users\Dipankar Malui\Desktop>javac DNS.java

C:\Users\Dipankar Malui\Desktop>java DNS

 Menu:
 1. DNS 2. Reverse DNS 3. Exit


 Enter your choice
1

 Enter Host Name
www.Facebook.com
Host Name: www.facebook.com
IP: 31.13.79.220

 Menu:
 1. DNS 2. Reverse DNS 3. Exit


 Enter your choice
2

 Enter IP address
215.34.56.0
IP: 215.34.56.0
Host Name: 215.34.56.0

 Menu:
 1. DNS 2. Reverse DNS 3. Exit


 Enter your choice
```

EXPERIMENT 10. STUDY OF NETWORK SIMULATOR (NS)

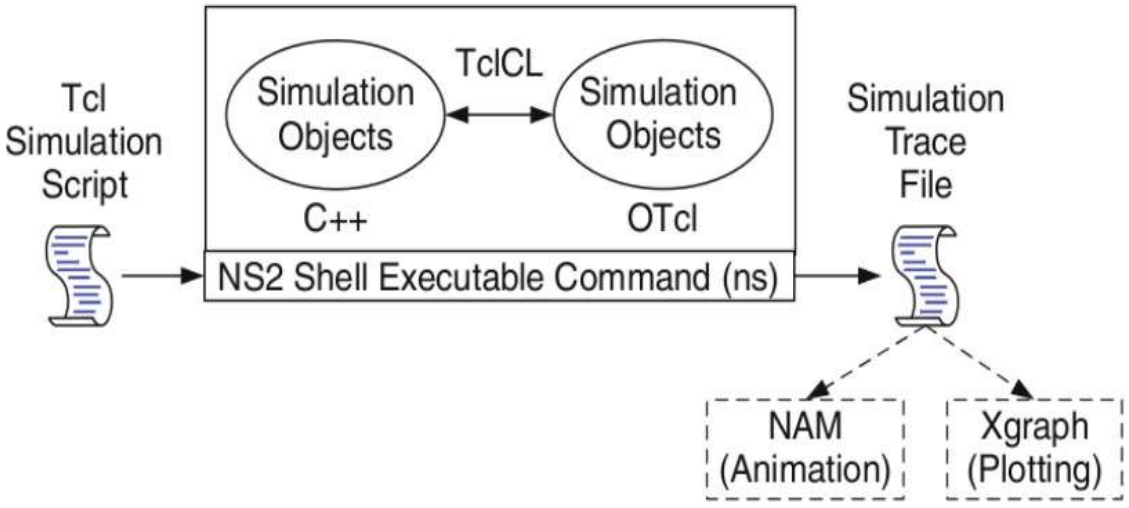
AIM:

To study about NS2 simulator in detail.

THEORY:

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989.

Basic Architecture of NS2:



The above figure shows the basic architecture of NS2. NS2 provides users with an executable command ns which takes on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.

In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

The C++ and the OTcl are linked together using TclCL. Mapped to a C++ object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle (e.g., n as a Node handle) is just a string (e.g.,_o10) in the OTcl domain, and does not contain any functionality. instead, the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and other OTcl objects. It may defines its own procedures and variables to facilitate the interaction. Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively

Tcl scripting:

• Tcl is a general purpose scripting language. [Interpreter]

• Tcl runs on most of the platforms such as Unix, Windows, and Mac.

• The strength of Tcl is its simplicity.

• It is not necessary to declare a data type for variable prior to the usage

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

```
puts stdout{Hello, World!}
```

Hello, World!

Variables

Command Substitution

set a 5 set len [string length foobar]

set b $a set len [expr [string length foobar] + 9]

Simple Arithmetic

expr 7.2 / 4

Procedures

```
proc Diag {a b} {
set c [expr sqrt($a * $a + $b * $b)]
return $c }
puts ―Diagonal of a 3, 4 right triangle is [Diag 3 4]‖
Output: Diagonal of a 3, 4 right triangle is 5.0
```

Loops

while{$i < $n} {
. . .
}
	for {set i 0} {$i < $n} {incr i}
	{
	. . .
	}

NS Simulator Preliminaries.

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

set ns [new Simulator]

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using ―"open" command:

#Open the Trace file

```
            set tracefile1 [open out.tr w]
            $ns trace-all $tracefile1
```

#Open the NAM trace file

```
      set namfile [open out.nam w]
      $ns namtrace-all $namfile
```

The above creates a dta trace file called ―out.tr‖ and a nam visualization trace file called ―out.nam‖.Within the tcl script,these files are not called explicitly by their names, but instead by pointers that are declared above and called ―tracefile1 and ―nam file respectively. Remark that they begins with a # symbol.The second line open the file ―out.tr‖ to be used for writing, declared with the letter ―w‖.The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. In our case, this will be the file pointed at by the pointer ―$namfile ,i.e the file ―out.tr‖.
The termination of the program is done using a ―finish‖ procedure.

#Define a 'finish' procedure

```
      Proc finish { } {
      global ns tracefile1 namfile
      $ns flush-trace
      Close $tracefile1
      Close $namfile
      Exec nam out.nam &
      Exit 0
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method **―flush-trace"** will dump the traces on the respective files. The tcl command

―close" closes the trace files defined before and **exec** executes the nam program for visualization.
The command **exit** will ends the application and return the number 0 as status to the system. Zero
is the default for a clean exit. Other values can be used to say that is a exit because something fails.

At the end of ns program we should call the procedure ―finish‖ and specify at what time the termination should occur. For example,

$ns at 125.0 "finish"

will be used to call **―finish‖** at time 125sec.Indeed,the **at** method of the simulator allows us to
schedule events explicitly.
The simulation can then begin using the command

$ns run

Definition of a network of links and nodes
The way to define a node is

set n0 [$ns node]

The node is created which is printed by the variable n0. When we shall refer to that node in the
script we shall thus write $n0.
Once we define several nodes, we can define the links that connect them. An example of
a definition of a link is:

$ns duplex-link $n0 $n2 10Mb 10ms DropTail

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction. To define a directional link instead of a bi-directional one, we should replace "duplexlink" by "simplex-link".

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to

arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).
In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

```
#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20
```

Agents and Applications

We need to define routing (sources, destinations) the agents (protocols) the application that use them

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received. There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

```
set tcp [new Agent/TCP]
```

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.
The command

```
set sink [new Agent /TCPSink]
```

Defines the behavior of the destination node of TCP and assigns to it a pointer called sink

```
#Setup a UDP connection

set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2
```

#setup a CBR over UDP connection

```
set cbr [new
Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false
```

Above shows the definition of a CBR application using a UDP agent. The command **$ns attach-agent $n4 $sink** defines the destination node. The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command $tcp set packetSize_ 552. When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command $tcp set fid_ 1 that assigns to the TCP connection a flow identification of "1".We shall later give the flow identification of "2" to the UDP connection.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP. Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

```
$cbr set packetSize_ <packet size>
```

Scheduling Events

NS is a discrete event based simulation. The tcp script defines when event should occur. The initializing command set ns [new Simulator] creates an event scheduler, and events are then scheduled using the format:

```
$ns at <time> <event>
```

The scheduler is started when running ns that is through the command $ns run. The beginning and end of the FTP and CBR application can be done through the following command

```
$ns at 0.1 “$cbr start”
$ns at 1.0 “ $ftp start”
$ns at 124.0 “$ftp stop”
$ns at 124.5 “$cbr stop”
```

RESULT:

Thus the Network Simulator 2 is studied in detail.

EXPERIMENT 11. Simulation of Distance Vector/Link State Routing.

.

a) Distance Vector routing protocol ROUTING

Aim:

To simulate and study the link state routing algorithm using simulation using NS2.

Distance Vector routing protocol

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 muchbetter 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.

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.

Program:

```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

set tr [open out.tr w]
$ns trace-all $tr

proc finish {} {
        global nf ns tr
        $ns flush-trace
        close $tr
        exec nam out.nam &
        exit 0
        }

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-down
```

```
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

set udp [new Agent/UDP]
$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

set null [new Agent/Null]
$ns attach-agent $n3 $null

$ns connect $tcp $sink
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto DV

$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"

$ns at 5.0 "finish"

$ns run
```

Result:

Thus the Distance Vector Routing Algorithm was Simulated and studied

b) SIMULATION OF LINK STATE ROUTING ALGORITHM

Aim:
To simulate and study the link state routing algorithm using simulation using NS2.

Link State Routing protocol

In link state routing, each router shares its knowledge of its neighborhood with every other router in the
internet work. (i) Knowledge about Neighborhood: Instead of sending its entire routing table a router sends
info about its neighborhood only. (ii) To all Routers: each router sends this information to every other router
on the internet work not just to its neighbor .It does so by a process called flooding. (iii)Information sharing
when there is a change: Each router sends out information about the neighbors when there is change.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose Link state routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

Program:

```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

set tr [open out.tr w]
$ns trace-all $tr

proc finish {} {
        global nf ns tr
        $ns flush-trace
        close $tr
        exec nam out.nam &
        exit 0
        }

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

```
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

set udp [new Agent/UDP]
$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

set null [new Agent/Null]
$ns attach-agent $n3 $null

$ns connect $tcp $sink
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto LS

$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"

$ns at 5.0 "finish"

$ns run
```

Result:

Thus the program for creating Simulation of Distance Vector/Link State Routing was implemented.

EXPERIMENT-12

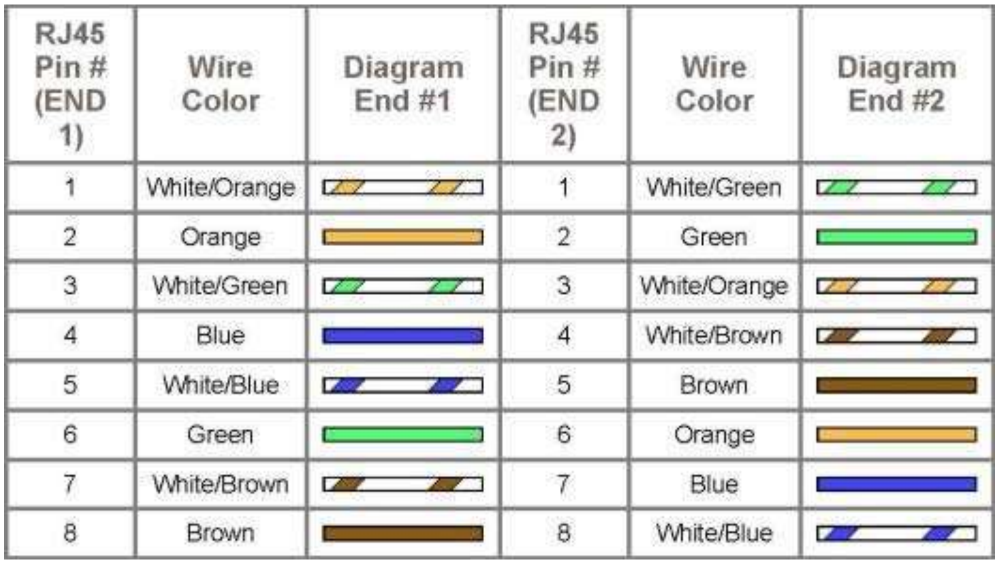
Aim: To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc.

Apparatus (Components): RJ-45 connector, Climping Tool, Twisted pair Cable
Procedure: To do these practical following steps should be done:

1. Start by stripping off about 2 inches of the plastic jacket off the end of the cable. Be very careful at this point, as to not nick or cut into the wires, which are inside. Doing so could alter the characteristics of your cable, or even worse render is useless. Check the wires, **one more time** for nicks or cuts. If there are any, just whack the whole end off, and start over.

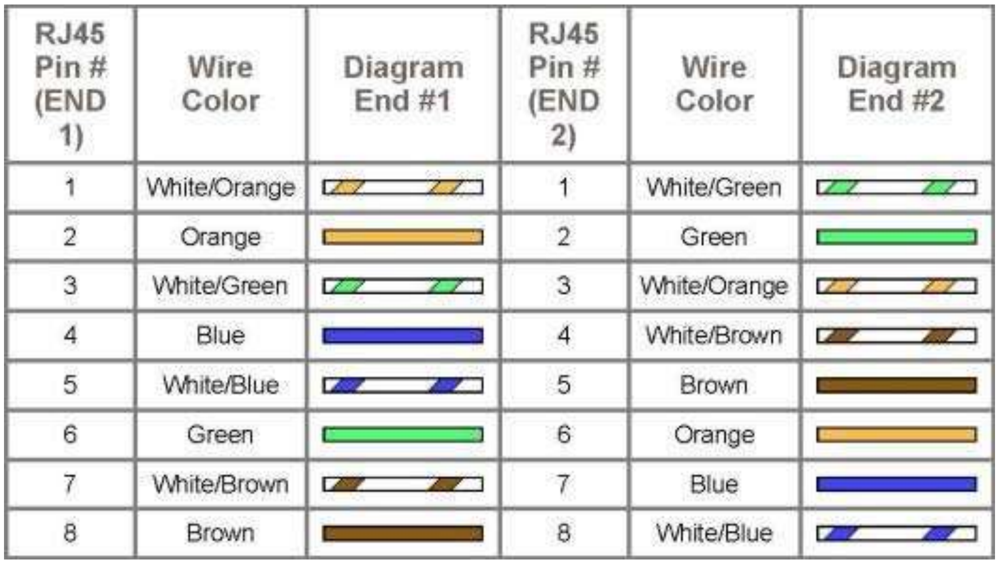
2. Spread the wires apart, but be sure to hold onto the base of the jacket with your other hand. You do not want the wires to become untwisted down inside the jacket. Category 5 cable must only have 1/2 of an inch of 'untwisted' wire at the end; otherwise it will be 'out of spec'. At this point, you obviously have ALOT more than 1/2 of an inch of un-twisted wire.

3. You have 2 end jacks, which must be installed on your cable. If you are using a pre-made cable, with one of the ends whacked off, you only have one end to install - the crossed over end. Below are two diagrams, which show how you need to arrange the cables for each type of cable end. Decide at this point which end you are making and examine the associated picture below.
Diagram shows you how to prepare Cross wired connection



RJ45 Pin # (END 1)	Wire Color	Diagram End #1	RJ45 Pin # (END 2)	Wire Color	Diagram End #2
1	White/Orange	▨	1	White/Green	▨
2	Orange	🟧	2	Green	🟩
3	White/Green	▨	3	White/Orange	▨
4	Blue	🟦	4	White/Brown	▨
5	White/Blue	▨	5	Brown	🟫
6	Green	🟩	6	Orange	🟧
7	White/Brown	▨	7	Blue	🟦
8	Brown	🟫	8	White/Blue	▨

Diagram shows you how to prepare straight through wired connection



RJ45 Pin # (END 1)	Wire Color	Diagram End #1	RJ45 Pin # (END 2)	Wire Color	Diagram End #2
1	White/Orange	▨	1	White/Green	▨
2	Orange	🟧	2	Green	🟩
3	White/Green	▨	3	White/Orange	▨
4	Blue	🟦	4	White/Brown	▨
5	White/Blue	▨	5	Brown	🟫
6	Green	🟩	6	Orange	🟧
7	White/Brown	▨	7	Blue	🟦
8	Brown	🟫	8	White/Blue	▨

EXPERIMENT-13

Aim: Study of following Network Devices in Detail

- Repeater
- Hub
- Switch
- Bridge
- Router
- Gate Way

Apparatus (Software): No software or hardware needed.

Procedure: Following should be done to understand this practical.

1. **Repeater:**Functioning at Physical Layer.A **repeater** is an electronic device that receives a signal and retransmits it at a higher level and/or higher power, or onto the other side of an obstruction, so that the signal can cover longer distances. Repeater have two ports ,so cannot be use to connect for more than two devices

2. Hub: An **Ethernet hub**, **active hub**, **network hub**, **repeater hub**, **hub** or **concentrator** is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

3. **Switch:**A **network switch** or **switching hub** is a computer networking device that connects network segments.The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.

4. **Bridge:** A **network bridge** connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term *bridge* formally means a device that behaves according to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports. *Switch* or *Layer 2 switch* is often used interchangeably with *bridge*.Bridges can analyze incoming data packets to determine if the bridge is able to send the given packet to another segment of the network.

5. **Router:** A **router** is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

6. **Gate Way:** In a communications network, a network node equipped for interfacing with

another network that uses different protocols.

- A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks.
- A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

EXPERIMENT- 14

Aim: Study of basic network command and Network configuration commands.

Apparatus (Software): Command Prompt And Packet Tracer.

Procedure: To do this EXPERIMENT- follows these steps:

In this EXPERIMENT- students have to understand basic networking commands e.g ping, tracert etc.

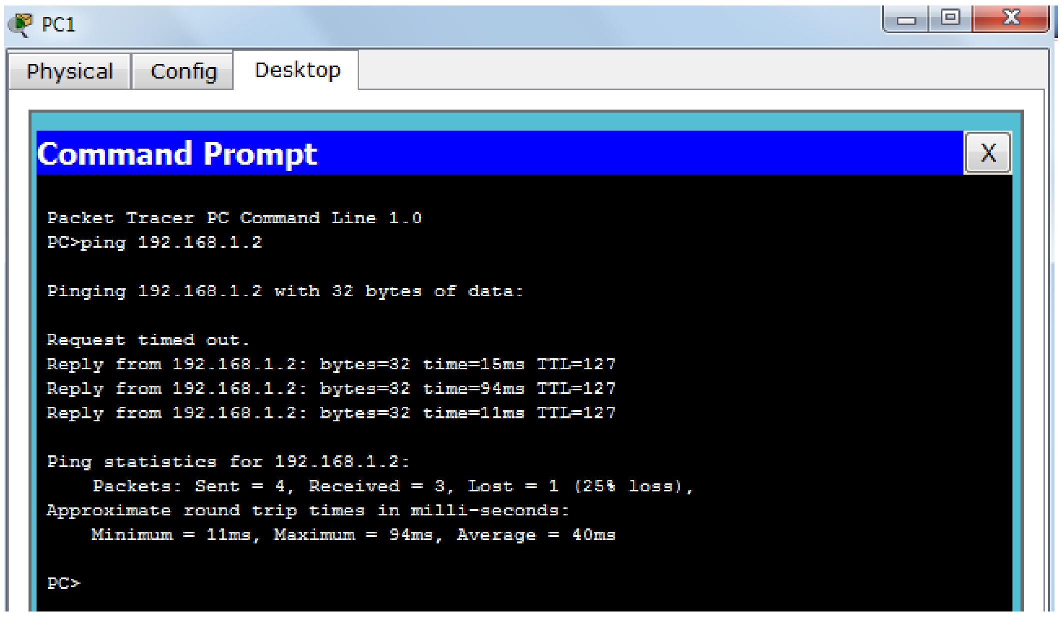
All commands related to Network configuration which includes how to switch to privilege mode and normal mode and how to configure router interface and how to save this configuration to flash memory or permanent memory.

This commands includes

- Configuring the Router commands
- General Commands to configure network
- Privileged Mode commands of a router
- Router Processes & Statistics
- IP Commands
- Other IP Commands e.g. show ip route etc.

ping:

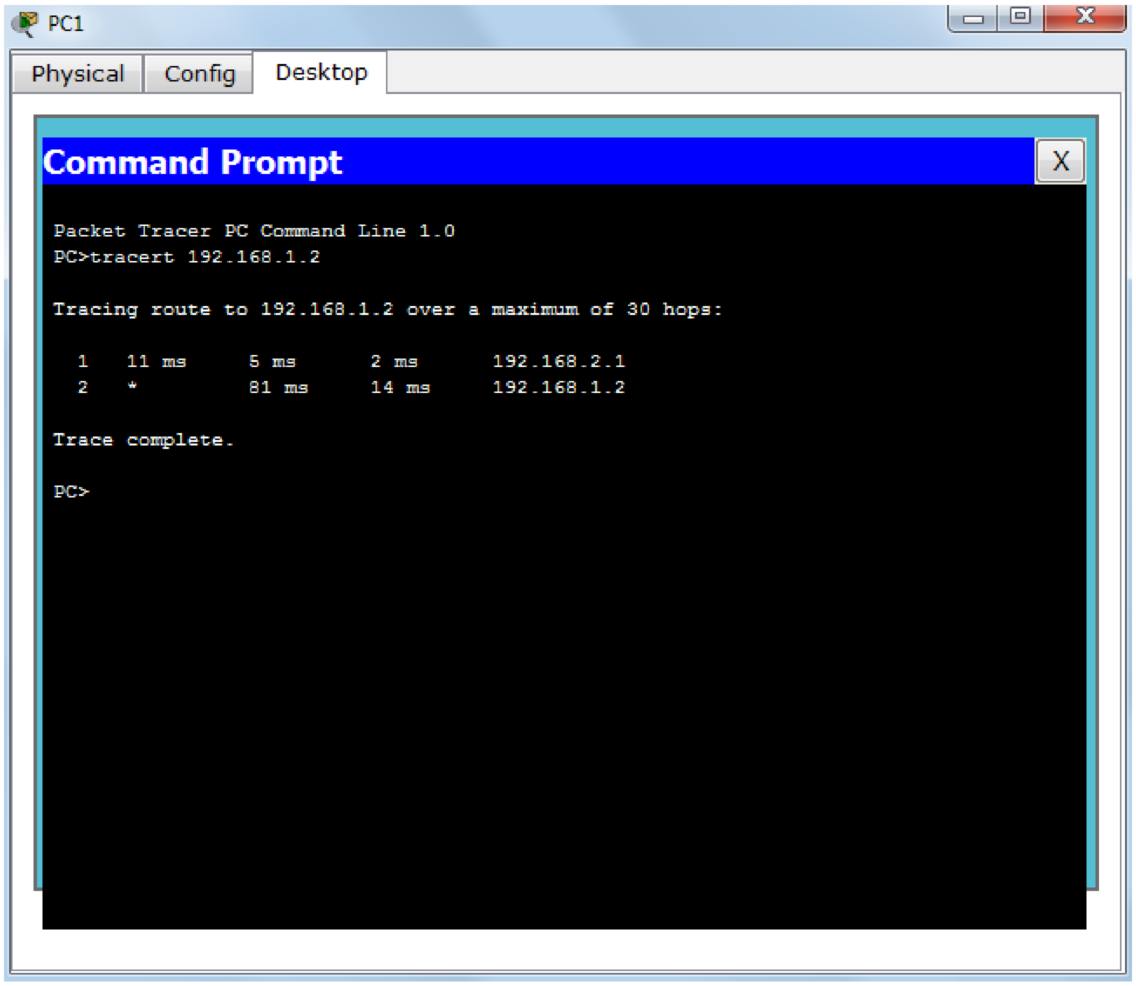
ping(8) sends an ICMP ECHO_REQUEST packet to the specified host. If the host responds, you get an ICMP packet back. Sound strange? Well, you can "ping" an IP address to see if a machine is alive. If there is no response, you know something is wrong.



```
Packet Tracer PC Command Line 1.0
PC>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Request timed out.
Reply from 192.168.1.2: bytes=32 time=15ms TTL=127
Reply from 192.168.1.2: bytes=32 time=94ms TTL=127
Reply from 192.168.1.2: bytes=32 time=11ms TTL=127

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 11ms, Maximum = 94ms, Average = 40ms

PC>
```

Traceroute:

Tracert is a command which can show you the path a packet of information takes from your computer to one you specify. It will list all the routers it passes through until it reaches its destination, or fails to and is discarded. In addition to this, it will tell you how long each 'hop' from router to router takes.



```
Packet Tracer PC Command Line 1.0
PC>tracert 192.168.1.2

Tracing route to 192.168.1.2 over a maximum of 30 hops:

  1   11 ms     5 ms      2 ms      192.168.2.1
  2   *         81 ms     14 ms     192.168.1.2

Trace complete.

PC>
```

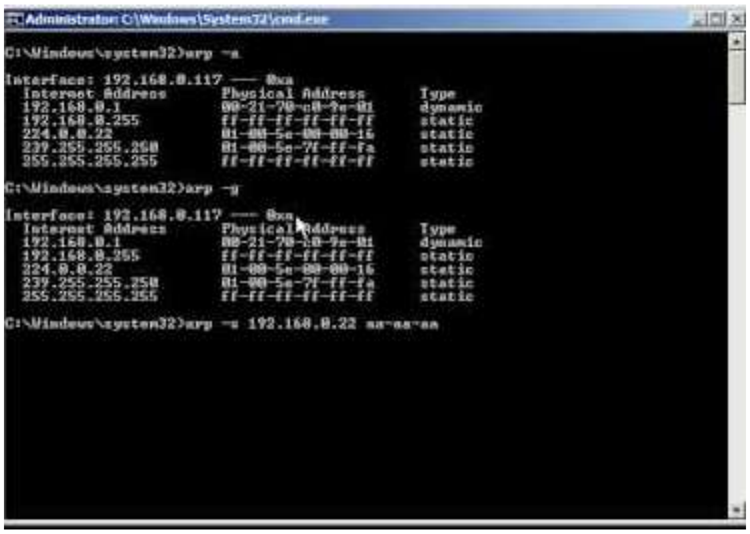
nslookup:

Displays information from Domain Name System (DNS) name servers.
NOTE :If you write the command as above it shows as default your pc's server name firstly.

ARP:

The ARP commands to view, display, or modify the details/information in an ARP table/cache.

The ARP cache or table has the dynamic list of IP and MAC addresses of those devices to which your computer has communicated recently in a local network. The purpose of maintaining an ARP table is that when you want to communicate with another device, your device does not need to send the ARP request for the MAC address of that device.



```
C:\Windows\system32>arp -a

Interface: 192.168.0.117 --- 0xa
  Internet Address      Physical Address      Type
  192.168.0.1           00-21-70-c0-9e-01     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Windows\system32>arp -g

Interface: 192.168.0.117 --- 0xa
  Internet Address      Physical Address      Type
  192.168.0.1           00-21-70-c0-9e-01     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Windows\system32>arp -s 192.168.0.22 aa-aa-aa
```

TELNET:

Telnet is an application that is used to connect to a remote host's command line terminal interface. Network and system administrators use this application to configure and administer network devices such as servers, routers, switches, etc. This application is based on the connection-oriented Transmission Control Protocol (TCP).

FTP:

FTP is the simplest file transfer protocol to exchange files to and from a remote computer or network. Similar to Windows, Linux and UNIX operating systems also have built-in command-line prompts that can be used as FTP clients to make an FTP connection.

In any command mode, you can get a list of available commands by entering a question mark (?).

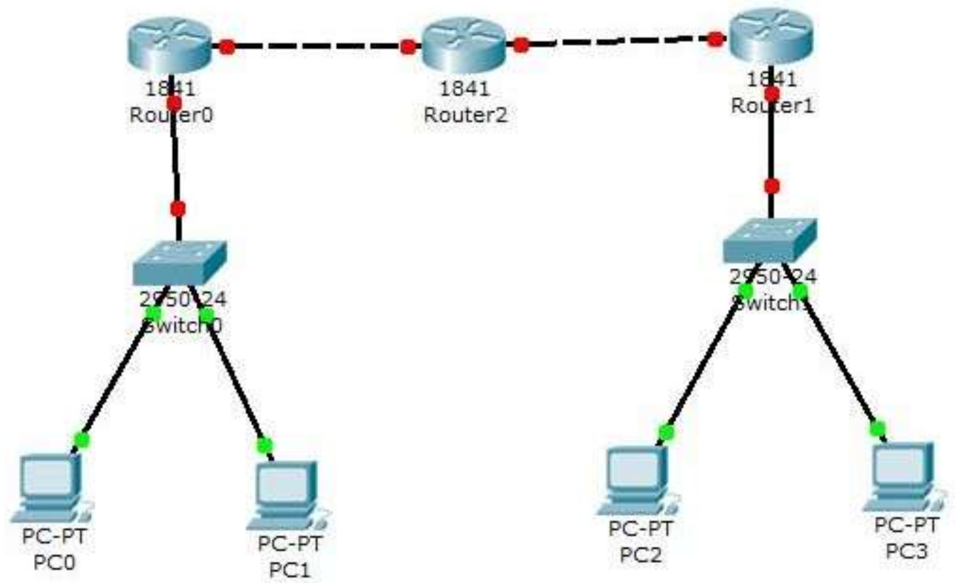
EXPERIMENT-15

Aim: Configure a Network topology using packet tracer software.

Apparatus (Software): Packet tracer Software

Procedure: To implement this practical following network topology is required to be configured using the commands learned in previous practical.

After configuring the given network a packet should be ping from any one machine to another.



Router0 Configuration Command :.........

Continue with configuration dialog? [yes/no]: no

Press RETURN to get started!

Router>

Router>Enable

Router#config t

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#hostname router0

router0(config)#interface fastethernet 0/0

```
router0(config-if)#ip address 192.168.1.1 255.255.255.0

router0(config-if)#description router0 fastethernet 0/0

router0(config-if)#no shutdown


%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

router0(config-if)#exit

router0(config)#interface fastethernet 0/1

router0(config-if)#description router0 fastethernet 0/1

router0(config-if)#no shutdown


%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

router0(config-if)#exit


router0(config)#exit

%SYS-5-CONFIG_I: Configured from console by console

router0#show running-config

Building configuration...


Current configuration : 437 bytes
!

version 12.4

no service password-encryption

!

hostname router0

!

!

!

!

!
```

```
ip ssh version 1
!
!
interface FastEthernet0/0
 description router0 fastethernet 0/0
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet0/1
 description router0 fastethernet 0/1
 no ip address
 duplex auto
 speed auto
!
interface Vlan1
 no ip address
 shutdown
!
ip classless
!
!
!
!
!
line con 0
line vty 0 4
 login
!
```

```
!

end
router0#

router0#

router0#copy running-config startup-config

Destination filename [startup-config]?

Building configuration...

[OK]

router0#
```