COURSE OBJECTIVES AND OUTCOMES

OBJECTIVES:

- To learn and use network commands.
- To learn socket programming.
- To implement and analyze various network protocols.
- To learn and use simulation tools.
- To use simulation tools to analyze the performance of various network protocols.

LIST OF EXPERIMENTS

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.
2. Write a HTTP web client program to download a web page using TCP sockets.
3. Applications using TCP sockets like:
    - Echo client and echo server
    - Chat
    - File Transfer
4. Simulation of DNS using UDP sockets.
5. Write a code simulating ARP /RARP protocols.
6. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.
7. Study of TCP/UDP performance using Simulation tool.
8. Simulation of Distance Vector/ Link State Routing algorithm.
9. Performance evaluation of Routing protocols using Simulation tool.
10. Simulation of error correction code (like CRC).

OUTCOMES:
Upon Completion of the course, the students will be able to:

- Implement various protocols using TCP and UDP.
- Compare the performance of different transport layer protocols.
- Use simulation tools to analyze the performance of various network protocols.
- Analyze various routing algorithms.
- Implement error correction codes.

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

LABORATORY REQUIREMENT FOR BATCH OF 30 STUDENTS:

HARDWARE:

1.	Standalone desktops	30 Nos

SOFTWARE:

1.	C / C++ / Java / Python / Equivalent Compiler	30
2.	Network simulator like NS2/Glomosim/OPNET/ Packet Tracer / Equivalent	

INDEX

8.a)		Simulation of Distance Vector/ Link State Routing algorithm		
8.b)		Link State Routing		
9)		Performance evaluation of Routing protocols using Simulation tool.		
10)		Simulation of error correction code (like CRC).		

Ex. No: 1	**Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol**
Date:	

1. **Aim:**

To write a program to implement PING and TRACEROUTE Commands

There are a lot of IP commands with short descriptions listed here but you should only need the ones mentioned here at the top of the page to diagnose and configure your network. C:>ping

C:>ipconfig

C:>ipconfig /all

C:>ipconfig /release

C:>ipconfig /renew

C:\>nbtstat –a

Remember when typing from the command prompt you can only type one command per line, and press Enter after each one to execute it.

C:\>arp –a: is short for 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)

Definitions 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.

(1) PING (Packet Internet Groper Command)

If any system (host or router) want to communicate with the other system (host or route)
then it is necessary to check the communication is possible or not? For this, first of all we have to check for destination system is reachable or not. Due to hardware failure or any other reason it is possible the system may on network but not reachable. How can we detect that he destination system is reachable or not? PING command is useful for checking the reachabilty of the system.

AIM:

To write the java program for simulating ping command

ALGORITHM:

1. Start the program.
2. Include necessary package in java.
3. To create a process object p to implement the ping command.
4. Declare one BufferedReader stream class object.
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.

6. Print the results.
7. Stop the program.

PROGRAM:

```
// PingServer.java: Simple Ping Programimport java.io.*;
import java.net.*;

class PingServer
{
    publicstaticvoidmain(String args[])
    {
        try
        {
            String str;
            System.out.print("Enter    IP    address/domain    name:    ");  BufferedReader
            buf1=new BufferedReader(new
                                        InputStreamReader(System.in));   String   ip   =
            buf1.readLine();
            Runtime  rt  =  Runtime.getRuntime();  Process  p  =
            rt.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:

```
$javac PingServer.java
$ java PingServer
Enter IP address/domain name: gmail.com

PINGgmail.com(216.58.199.165) 56(84) bytes of data.

6  byte  fro  bom05s08-in-      (216.58.199.165icmp_req=1   time=34.7
4  s     m    f5.1e100.net      ):              ttl=58       ms
6  byte  fro  bom05s08-in-      (216.58.199.165icmp_req=2   time=36.2
4  s     m    f5.1e100.net      ):              ttl=58       ms
6  byte  fro  bom05s08-in-      (216.58.199.165icmp_req=3   time=32.0
4  s     m    f5.1e100.net      ):              ttl=58       ms
6  byte  fro  bom05s08-in-      (216.58.199.165icmp_req=4   time=32.4
4  s     m    f5.1e100.net      ):              ttl=58       ms
6  byte  fro  bom05s08-in-      (216.58.199.165icmp_req=5   time=31.5
4  s     m    f5.1e100.net      ):              ttl=58       ms
```

```
64 bytes from bom05s08-in-f5.1e100.net (216.58.199.165): icmp_req=6 ttl=58 time=32.8 ms
64 bytes from bom05s08-in-f5.1e100.net (216.58.199.165): icmp_req=7 ttl=58 time=31.1 ms
64 bytes from bom05s08-in-f5.1e100.net (216.58.199.165): icmp_req=8 ttl=58 time=32.7 ms
64 bytes from bom05s08-in-f5.1e100.net (216.58.199.165): icmp_req=9 ttl=58 time=31.8 ms
64 bytes from bom05s08-in-f5.1e100.net (216.58.199.165): icmp_req=10 ttl=58 time=31.4 ms

^C

--- gmail.com ping statistics ---

10 packets transmitted, 10 received, 0% packet loss, time9011ms

rtt min/avg/max/mdev = 31.148/32.724/36.287/1.547 ms
```

RESULT :

Thus using Ping command, connective and communicative status is determined.

(2)TRACEROUTE:

AIM: To write a program to simulate Traceroute command.

ALGORITHM:

1. Start the program and declare the variables.
2. Create the document file path.txt and give the all details
3. Open the document file path.txt and compare the input with the details in the path.txt
4. Trace the route
5. Stop the program

PROGRAM:

```
// TraceServer.java : Traceroute Program import java.io.*;
import java.net.*;

class TraceServer
{
public static void main(String args[])
{
try
{
String str;
System.out.print("Enter domain name : "); BufferedReader buf1=new BufferedReader(new
InputStreamReader(System.in)); String ip = buf1.readLine();
Runtime rt = Runtime.getRuntime(); Process p = rt.exec("tracert " + 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:

```
$javac TraceServer.java
$java TraceServer
Enterdomain name: yahoo.com


Tracing route to yahoo.com [206.190.36.45]over
a maximum of 30 hops:

  1     <1 ms     <1 ms     <1 ms  172.16.4.4
  2     18 ms     17 ms     10 ms  182.19.59.114
  3    154 ms    184 ms    158 ms  182.19.115.233
  4    158 ms    156 ms    155 ms  ae5-xcr2.lsw.cw.net [166.63.217.41]
  5    232 ms    224 ms    230 ms  ae11-xcr1.lns.cw.net [195.2.25.206]
  6    155 ms    155 ms    170 ms  ae1-xcr1.ltw.cw.net [195.2.24.125]
  7    233 ms    234 ms    232 ms  et-9-1-0-xcr2.nyk.cw.net [195.2.8.46]
  8    243 ms    230 ms    228 ms  pat1.nyc.yahoo.com [198.32.118.24]
  9    230 ms    260 ms    231 ms  ae7.pat1.dce.yahoo.com [216.115.104.120]
 10    243 ms    245 ms    244 ms   ae-6.pat1.che.yahoo.com [216.115.96.81]
 11    334 ms    318 ms    294 ms   ae-5.pat1.dnx.yahoo.com [216.115.96.34]
 12    303 ms    313 ms    335 ms  ae-8.pat2.gqb.yahoo.com [216.115.96.204]
 13    314 ms    319 ms    316 ms  et-18-1-0.msr2.gq1.yahoo.com [66.196.67.115]
 14      *       301 ms    304 ms  et-19-1-0.clr2-a-gdc.gq1.yahoo.com [67.195.37.99]
 15    306 ms    311 ms    305 ms  UNKNOWN-67-195-1-X.yahoo.com [67.195.1.251]
 16    307 ms    309 ms    300 ms  po-16.bas2-7-prd.gq1.yahoo.com [206.190.32.43]
 17    303 ms    312 ms    303 ms  ir1.fp.vip.gq1.yahoo.com [206.190.36.45]

Tracecomplete.
```

RESULT:

Thus using traceroute command, path traversed by the packet is determined.

Ex. No: 2	WRITE A HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS.

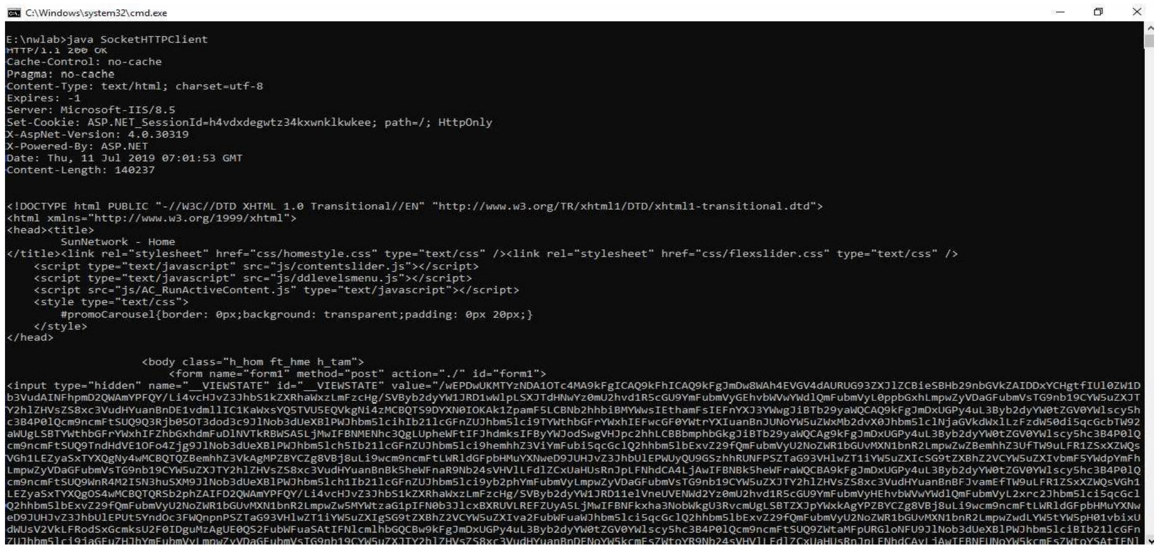
AIM:
To write a java program for socket for HTTP for web page upload and download .

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:
```
import java.io.*;
import java.net.*;
public class SocketHTTPClient
{
public static void main(String[] args)
{
String hostName = "www.sunnetwork.in"; int portNumber = 80;
try
{
Socket socket = new Socket(hostName, portNumber);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true); BufferedReader in =new
BufferedReader(new
InputStreamReader(socket.getInputStream()));       out.println("GET/HTTP/1.1\nHost:
www.sunnetwork.in\n\n"); String inputLine;
while ((inputLine = in.readLine()) != null)
{
System.out.println(inputLine);
}
 }
catch (UnknownHostException e)
{
System.err.println("Don't know about host " + hostName); System.exit(1);
}
catch (IOException e)
{
System.err.println("Couldn't get I/O for the connection to " + hostName); System.exit(1);
}
}
}
```

OUTPUT:



```
C:\Windows\system32\cmd.exe

E:\nwlab>java SocketHTTPClient
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
Set-Cookie: ASP.NET_SessionId=h4vdxdegwtz34kxwnklkwkee; path=/; HttpOnly
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Thu, 11 Jul 2019 07:01:53 GMT
Content-Length: 140237


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
        SunNetwork - Home
</title><link rel="stylesheet" href="css/homestyle.css" type="text/css" /><link rel="stylesheet" href="css/flexslider.css" type="text/css" />
    <script type="text/javascript" src="js/contentslider.js"></script>
    <script type="text/javascript" src="js/ddlevelsmenu.js"></script>
    <script src="js/AC_RunActiveContent.js" type="text/javascript"></script>
    <style type="text/css">
        #promoCarousel{border: 0px;background: transparent;padding: 0px 20px;}
    </style>
</head>

                    <body class="h_hom ft_hme h_tam">
                        <form name="form1" method="post" action="./" id="form1">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTYzNDA1OTc4MA9kFgICAQ9kFhICAQ9kFgJmDw8WAh4EVGV4dAURUG93ZXJlZCBieSBHb29nbGVkZAIDDxYCHgtfIUl0ZW1D
b3VudAINFhpmD2QWAmYPFQY/Li4vcHJvZ3JhbS1kZXRhaWxzLmFzcHg/SVByb2dyYW1JRD1wWlpLSXJTdHNwYz0mU2hvd1R5cGU9YmFubmVyGEhvbWVwYWdlQmFubmVyL0ppbGxhLmpwZyVDaGFubmVsTG9nb19CYW5uZXJT
Y2hlZHVsZS8xc3VudHYuanBnDE1vdmllIC1KaWxsYQ5TVU5EQVkgNi4zMCBQTS9DYXN0IOKAk1ZpamF5LCBNb2hhbiBMYWwsIEthamFsIEFnYXJ3YWwgJiBTb29yaWQCAQ9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5h
c3B4P0lQcm9ncmFtSUQ9Q3Rjb05OT3dod3c9JlNob3dUeXBlPWJhbm5lcihIb21lcGFnZUJhbm5lci9TYWthbGFrYWxhIEFwcGF0YWtrYXIuanBnJUNoYW5uZWxMb2dvX0Jhbm5lclNjaGVkdWxlLzFzdW50di5qcGcbTW92
aWUgLSBTYWthbGFrYWxhIFZhbGxhdmFuDlNVTkRBWSA5LjMwIFBNMENhc3QgLUpheWFtIFJhdmksIFByYWJodSwgVHJpc2hhLCBBbmphbGkgJiBTb29yaWQCAg9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5hc3B4P0lQ
cm9ncmFtSUQ9TndHdVE1OFo4Zjg9JlNob3dUeXBlPWJhbm5lch5Ib21lcGFnZUJhbm5lci9hemhhZ3ViYmFubi5qcGclQ2hhbm5lbExvZ29fQmFubmVyU2NoZWR1bGUvMXN1bnR2LmpwZwZBemhhZ3UfTW9uLFR1ZSxXZWQs
VGh1LEZyaSxTYXQgNy4wMCBQTQZBemhhZ3VkAgMPZBYCZg8VBj8uLi9wcm9ncmFtLWRldGFpbHMuYXNweD9JUHJvZ3JhbUlEPWUyQU9GSzhhRUNFPSZTaG93VHlwZT1iYW5uZXIcSG9tZXBhZ2VCYW5uZXIvbmF5YWdpYmFh
LmpwZyVDaGFubmVsTG9nb19CYW5uZXJTY2hlZHVsZS8xc3VudHYuanBnBk5heWFnaR9Nb24sVHVlLFdlZCxUaHUsRnJpLFNhdCA4LjAwIFBNBk5heWFraWQCBA9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5hc3B4P0lQ
cm9ncmFtSUQ9WnR4M2I5N3huSXM9JlNob3dUeXBlPWJhbm5lch1Ib21lcGFnZUJhbm5lci9yb2phYmFubmVyLmpwZyVDaGFubmVsTG9nb19CYW5uZXJTY2hlZHVsZS8xc3VudHYuanBnBFJvamEfTW9uLFR1ZSxXZWQsVGh1
LEZyaSxTYXQgOS4wMCBQTQRSb2phZAIFD2QWAmYPFQY/Li4vcHJvZ3JhbS1kZXRhaWxzLmFzcHg/SVByb2dyYW1JRD11elVneUVENWd2Yz0mU2hvd1R5cGU9YmFubmVyHEhvbWVwYWdlQmFubmVyL2xrc2Jhbm5lci5qcGcl
Q2hhbm5lbExvZ29fQmFubmVyU2NoZWR1bGUvMXN1bnR2LmpwZw5MYWtzaG1pIFN0b3JlcxBXRUVLREFZUyA5LjMwIFBNFkxha3NobWkgU3RvcmUgLSBTZXJpYWxkAgYPZBYCZg8VBj8uLi9wcm9ncmFtLWRldGFpbHMuYXNw
eD9JUHJvZ3JhbUlEPUt5YndOc3FWQnpnPSZTaG93VHlwZT1iYW5uZXIgSG9tZXBhZ2VCYW5uZXIva2FubWFuaWJhbm5lci5qcGclQ2hhbm5lbExvZ29fQmFubmVyU2NoZWR1bGUvMXN1bnR2LmpwZwdLYW5tYW5pH01vbixU
dWUsV2VkLFRodSxGcmksU2F0IDguMzAgUE0QS2FubWFuaSAtIFNlcmlhbGQCBw9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5hc3B4P0lQcm9ncmFtSUQ9ZWtaMFpURGloNFU9JlNob3dUeXBlPWJhbm5lciBIb21lcGFn
ZUJhbm5lci9jaGFuZHJhYmFubmVyLmpwZyVDaGFubmVsTG9nb19CYW5uZXJTY2hlZHVsZS8xc3VudHYuanBnDFNoYW5kcmFsZWtoYR9Nb24sVHVlLFdlZCxUaHUsRnJpLFNhdCAyLjAwIFBNFUNoYW5kcmFsZWtoYSAtIFNl
```

RESULT:

Thus the program for creating sockets for HTTP web page to download was implemented.

Ex. No: 3	**APPLICATION USING TCP SOCKET LIKE ECHO CLIENT AND ECHO SERVER, CHAT, FILE TRANSFER**

A. ECHO CLIENT AND ECHO SERVER

Aim:

To write a program in java to implement TCP Echo Client Server (Iterative model).

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

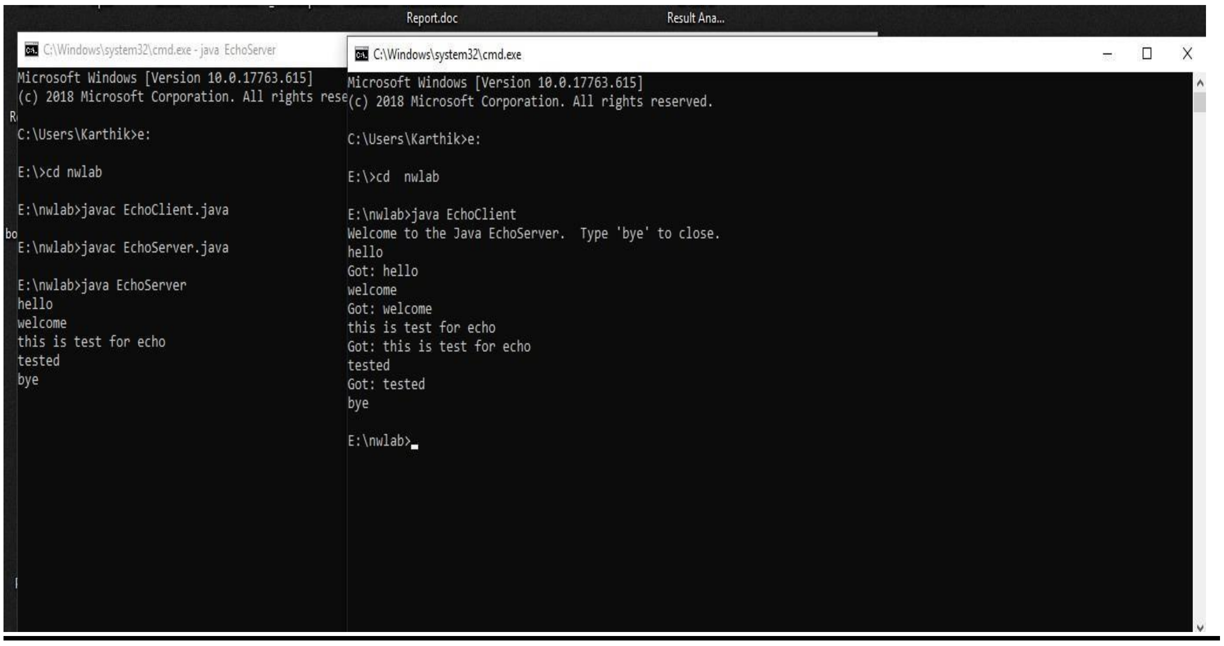
Program :

```
EchoServer.Java: import java.io.*; import java.net.*;
public class EchoServer
{
public EchoServer(int portnum)
{

try
{
server = new ServerSocket(portnum);
}
 catch (Exception err)
{
System.out.println(err);
}
}
public void serve()
{
try
 {
while (true)
{
Socket client = server.accept(); BufferedReader r = new BufferedReader(new
InputStreamReader(client.getInputStream()));
PrintWriter w = new PrintWriter(client.getOutputStream(),
true);
w.println("Welcome to the Java EchoServer. Type 'bye'
to
close.");
String line; do
{
line   =   r.readLine();   if   (   line   !=   null   )
w.println("Got: "+ line); System.out.println (line);
}
while ( !line.trim().equals("bye") ); client.close();
}
}
catch (Exception err)
{
System.err.println(err);
}
}
public static void main(String[] args)
{

EchoServer s = new EchoServer(9999); s.serve();
}
private ServerSocket server;
}
```

```
EchoClient.java : import java.io.*; import java.net.*; public class EchoClient
{
public static void main(String[] args)
{

try
{
 Socket  s  =  new  Socket("127.0.0.1",  9999);  BufferedReader  r  =  new  BufferedReader(new
InputStreamReader(s.getInputStream()));              PrintWriter w         =                        new
PrintWriter(s.getOutputStream(), true);
BufferedReader con = new BufferedReader(new
InputStreamReader(System.in));

String line; do
{
line = r.readLine(); if ( line != null )
System.out.println(line); line = con.readLine(); w.println(line);
}
while ( !line.trim().equals("bye") );

}
catch (Exception err)
{
```

```
System.err.println(err);
}
}
}
```

Output:



Report.doc
Result Ana...
C:\Windows\system32\cmd.exe - java EchoServer
C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.17763.615]
(c) 2018 Microsoft Corporation. All rights rese

C:\Users\Karthik>e:

E:\>cd nwlab

E:\nwlab>javac EchoClient.java

E:\nwlab>javac EchoServer.java

E:\nwlab>java EchoServer
hello
welcome
this is test for echo
tested
bye
```

```
Microsoft Windows [Version 10.0.17763.615]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Karthik>e:

E:\>cd  nwlab

E:\nwlab>java EchoClient
Welcome to the Java EchoServer.  Type 'bye' to close.
hello
Got: hello
welcome
Got: welcome
this is test for echo
Got: this is test for echo
tested
Got: tested
bye

E:\nwlab>
```

RESULT :

Thus the java program to concurrently communicate using TCP Sockets was executed successfully

B.CHAT

Aim:

To perform the full duplex chat by sending and receiving the message from the client to server and vice versa using TCP sockets.

Algorithm

Server:

1. Establish socket connection to the client
2. After successful connection with the client, acknowledgement will be sent
3. An Internet socket address structure is filled in with the wildcard address (INADDR_ANY) and the server’s well-known port (PORT).
4. The socket is converted into a listening socket by calling the listen function.
5. The server blocks in the call to accept, waiting for the client connection to complete.
6. When the connection is established, the server reads the line from the client using connected socket and display the message in the standard output using fputs.
7. Then again the server reads a line of text from the standard input and writes it back to the client through the connected socket.
8. The server went through the steps (5) and (6) until it receives 'bye' either from the standard input or client.
9. Finally, the server closes the connected socket.

Client:

1. Establish socket connection with the server.
2. After successful connection with the server, acknowledgement will be received
3. An Internet socket address structure is filled in with the server’s IP address and the same port number.
4. The connect function establishes the connection with the server.
5. When the connection is established, the client reads the line from the standard input using fgets and send the message to the server through the socket.
6. Then again the client reads a line of text from the server through the connected socket and writes it back to the standard output using fputs.
7. The client went through the steps (4) and (5) until it receives 'bye' either from the standard input or server.
8. Finally, the client closes the connected socket.

Program:

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

public class  chatserver

{
public static void main(String args[]) throws Exception
{

ServerSocket ss=new ServerSocket(2000);Socket sk=ss.accept();
BufferedReader cin=new BufferedReader(newInputStreamReader(sk.getInputStream()));
PrintStream  cout=new  PrintStream(sk.getOutputStream());
BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in)); String s;
while (  true )

{
s=cin.readLine();
if (s.equalsIgnoreCase("END"))
{
cout.println("BYE");
break;

}
System. out.print("Client : "+s+"\n");System.out.print("Server : "); s=stdin.readLine();
cout.println(s);

}
ss.close();
sk.close();

cin.close();
cout.close();stdin.close();
}

}
```

<u>Chatclient.java</u>

```
import java.net.*;
 import java.io.*;
 public class  chatclient
{
public static void main(String args[]) throws Exception
{
Socket sk=new Socket("127.0.0.1",2000); BufferedReader sin=new BufferedReader(new
InputStreamReader(sk.getInputStream()));
PrintStream  sout=new  PrintStream(sk.getOutputStream());

BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in)); String s;
while (  true )
{
```

```
System.out.print("Client : ");s=stdin.readLine();
sout.println(s); s=sin.readLine();
System.out.print("Server : "+s+"\n");if ( s.equalsIgnoreCase("BYE") )
break;

}
sk.close();
sin.close();
sout.close();stdin.close();
}

}
```

Output:

Server:

```
E:\nwlab>javac *.java E:\nwlab>java chatserverClient : hi
Server : hi
```

Client:

```
E:\nwlab>java chatclientClient : hi
Server : hiClient :
```

Result :

CHAT application using TCP Socket is implemented successfully.

C. FILE TRANSFER:

Aim:

To write a program for File Transfer Application using TCP socket.

Algorithm:

Server:

1. Establish the socket connection to the client
2. After successful connection with the client, acknowledgement will be sent.
3. Create an unnamed socket for the server using parameters AF_INET as domain and SOCK_STREAM as type.
4. Get the server port number.
5. Register the host address to the system by using bind() system call in server side.
6. Create a connection queue and wait for clients using listen() system call with the number of clients requests as parameter.
7. Create a Child process using fork( ) system call.
8. If the process identification number is equal to zero accept the connection using accept( ) system call when the client request for connection.
9. If pid is not equal to zero then exit the process.

Client:

1. Establish a socket connection with the server
2. After successful connection with the server, acknowledgement will be received.
3. Create an unnamed socket for the client using parameters AF_INET as a domain and SOCK_STREAM as type.
4. Get the client port number.
5. Now connect the socket to server using connect ( ) system call.
6. Enter the file name that need to be transferred
7. The file is transferred from client to server using send ( ) function.
8. Print the contents of the file in a new file and display it.

Program:

File Client

```
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 arraybyte[] 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:

```
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 FileClientFile saved successfully!

E:\nwlab>
```

Result:

Thus the java program file transfer application using TCP Sockets was executed

Ex. No: 4	APPLICATION USING UDP USING DNS
Date:	

A. DNS (Domain Name System)

Aim:

To write a C program for the simulation of Domain Name System

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

UDP DNS Server

```
import java.io.*;
import java.net.*;
public class dnsserver
{
private static int indexOf(String[] array, String str)
{
str = str.trim();
for (int i=0; i < array.length; i++)
{
if (array[i].equals(str)) return i;
}
return -1;
}
public static void main(String arg[])throws IOException
{
String[] hosts = {"zoho.com", "gmail.com","google.com", "facebook.com"};
String[] ip = {"172.28.251.59", "172.217.11.5","172.217.11.14",
"31.13.71.36"}; System.out.println("Press Ctrl + C to Quit");
while (true)
{
 receivedata.length);
 DatagramSocket serversocket=new DatagramSocket(1362);
byte[] senddata = new byte[1021];
byte[] receivedata = new byte[1021];
DatagramPacket recvpack = new DatagramPacket(receivedata,
serversocket.receive(recvpack);
String sen = new String(recvpack.getData());
InetAddress ipaddress = recvpack.getAddress(); int port = recvpack.getPort();
String capsent;
System.out.println("Request for host " + sen);
if(indexOf (hosts, sen) != -1)
capsent = ip[indexOf (hosts, sen)];
 else
capsent = "Host Not Found";
senddata = capsent.getBytes();
DatagramPacket pack = new DatagramPacket (senddata, senddata.length,ipaddress,port);
serversocket.send(pack); serversocket.close();
}
}}
```

//UDP DNS Client –

```
import java.io.*;
import java.net.*;
public class dnsclient
{
public static void main(String args[])throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
DatagramSocket clientsocket = new DatagramSocket(); InetAddress ipaddress;
if (args.length == 0)
ipaddress = InetAddress.getLocalHost();
else
ipaddress = InetAddress.getByName(args[0]);
byte[] senddata = new byte[1024]; byte[] receivedata = new byte[1024]; int portaddr = 1362;
System.out.print("Enter the hostname : "); String sentence = br.readLine();
senddata = sentence.getBytes();
DatagramPacket pack = new DatagramPacket(senddata,senddata.length, ipaddress,portaddr);
clientsocket.send(pack); DatagramPacket recvpack =new
DatagramPacket(receivedata,receivedata.length);
clientsocket.receive(recvpack);
String modified = new String(recvpack.getData()); System.out.println("IP Address: " +
modified); clientsocket.close();
}
}
```

OUTPUT

Server

```
 E:\nwlab>java dnsserverPress Ctrl + C to Quit
Request for host google.comRequest for host flipkart.com
```

Client

```
E:\nwlab>java dnsclient
Enter the hostname : google.comIP Address: 172.217.11.14
E:\nwlab>java dnsclient
Enter the hostname : flipkart.comIP Address: Host Not Found
E:\nwlab>
```

Result:

Thus the DNS application program was executed.

Ex. No: 5	WRITE A CODE SIMULATING ARP/RARP PROTOCOLS
Date:	

Aim:

To write a program to implement Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP) ..

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. Send server‟s date and time to the client.
4. Read client‟s IP address sent by the client.
5. Display the client details.
6. Repeat steps 2-5 until the server is terminated.
7. Close all streams.
8. Close the server socket.
9. Stop.

Client

1. Create a client socket and connect it to the server‟s port number.
2. Retrieve its own IP address using built-in function.
3. Send its address to the server.
4. Display the date & time sent by the server.
5. Close the input and output streams.
6. Close the client socket.
7. Stop.

Program

Program for Address Resolutuion Protocol (ARP) using TCP

Clientarp

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp
{
public static void main(String args[])
{
try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
Socket clsct=new Socket("127.0.0.1",5604);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the Logical address(IP):");
String str1=in.readLine();
dout.writeBytes(str1+'\n'); String str=din.readLine();
System.out.println("The Physical Address is: "+str);

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

Server:

```
import java.io.*; import java.net.*; import java.util.*;
class Serverarp
{
public static void main(String args[])
{
try
{
ServerSocket obj=new ServerSocket(5604); Socket obj1=obj.accept(); while(true)
{
DataInputStream din=new DataInputStream(obj1.getInputStream()); DataOutputStream
dout=new DataOutputStream(obj1.getOutputStream()); String str=din.readLine();
String ip[]={"165.165.80.80","165.165.79.1"};
String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
for(int i=0;i<ip.length;i++)
{

if(str.equals(ip[i]))
{
dout.writeBytes(mac[i]+'\n'); break;
}
}
obj.close();
}

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

Output:

```
E:\networks>java Serverarp
E:\networks>java Clientarp Enter the Logical address(IP):
165.165.80.80
The Physical Address is: 6A:08:AA:C2
```

Result:

Thus the ARP protocol using TCP Sockets program was executed.

5B. RARP Client/Server

Aim
To know the logical address of a host when its physical address is known using RARP protocol.

Algorithm

Target/Server
Create a server socket.
Accept client connection.
Read MAC address from the client request
Check its configuration file and compare with its physical address.
If there is a match, send the host logical address.
Stop

Client
Create a socket.
Send physical address to the target machine
Receive target's response
If it is a IP address then display it and go to step 6
Display "Host not found"
Stop

Program

```
// RARP Server -- rarpserver.javaimport java.io.*;
import java.net.*;
classrarpserver
{
public static void main(String args[])throws IOException
{
try
{
ServerSocket soc = new ServerSocket(2500); System.out.println("Server started"); Socket
client = null;
client=soc.accept();String str;
PrintStream ps = new
PrintStream(client.getOutputStream());BufferedReader br = new BufferedReader(new
InputStreamReader(client.getInputStream())); Runtime r = Runtime.getRuntime();
Process p = r.exec("ifconfig eth0"); BufferedReader pin =new BufferedReader(new
InputStreamReader(p.getInputStream())); String ipaddr = "";
Stringhaddr = br.readLine();int flag = 0;
while((str = pin.readLine())!=null)
{
System.out.println(str);
if ((str.indexOf(haddr)) != -1)
{
flag = 1;
}
else if((str.indexOf("inet addr")) !=-1)
{
int pos = str.indexOf("inet addr:") + 10;int offset = pos + 13;
ipaddr = str.substring(pos,offset);
}
}
if (flag == 1) ps.println(ipaddr);
ps.close();
br.close();
pin.close(); client.close();soc.close();
}
catch(IOException io)
{
System.err.println("Exception : " + io.toString());
}}}
```

//RARP CLIENT-- RARP-CLIENT.

```
javaimport java.io.*;
import java.net.*;
classrarpclient
{
publicstaticvoidmain(String args[])
{
try
{
Socket client =new Socket("localhost", 2500);BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
PrintStream ps = new
PrintStream(client.getOutputStream());String haddr,ipaddr = null;
BufferedReader sin = new BufferedReader(new
InputStreamReader(client.getInputStream()));
System.out.print("Enterthephysicaladdress : ");haddr = br.readLine();
ps.println(haddr); ipaddr = sin.readLine();if (ipaddr == null)
System.out.println("Host does not exist");else
System.out.println("Logical Address " + ipaddr);ps.close();
br.close(); client.close();
}
catch(IOException io)
{
System.err.println(io.toString());
}
}}
```

Output

Server

```
$javac rarpserver.java
$java rarpserverServer started
eth0     Link encap:Ethernet  HWaddr B8:AC:6F:1B:AB:06
inet addr:172.16.12.251  Bcast:172.255.255.255   Mask:255.0.0.0
inet6 addr: fe80::baac:6fff:fe1b:ab06/64 Scope:Link UP BROADCAST RUNNING
MULTICAST  MTU:1500  Metric:1
RX packets:450 errors:0 dropped:0 overruns:0 frame:0 TX packets:127 errors:0 dropped:0
overruns:0 carrier:0collisions:0 txqueuelen:1000
RX bytes:48118 (46.9 KiB)  TX bytes:21025 (20.5 KiB)
Interrupt:16
```

Client

```
$javac rarpclient.java
$ java rarpclient
Enterthephysicaladdress: B8:AC:6F:1B:AB:06Logical Address 172.16.12.251
```

Result

Thus using RARP protocol, IP address of the server is obtained.

Ex. No: 6	STUDY OF NETWORK SIMULATOR (NS). TO SIMULATE THE PERFORMANCE STOP WAIT PROTOCOL UNDER CONGESTION

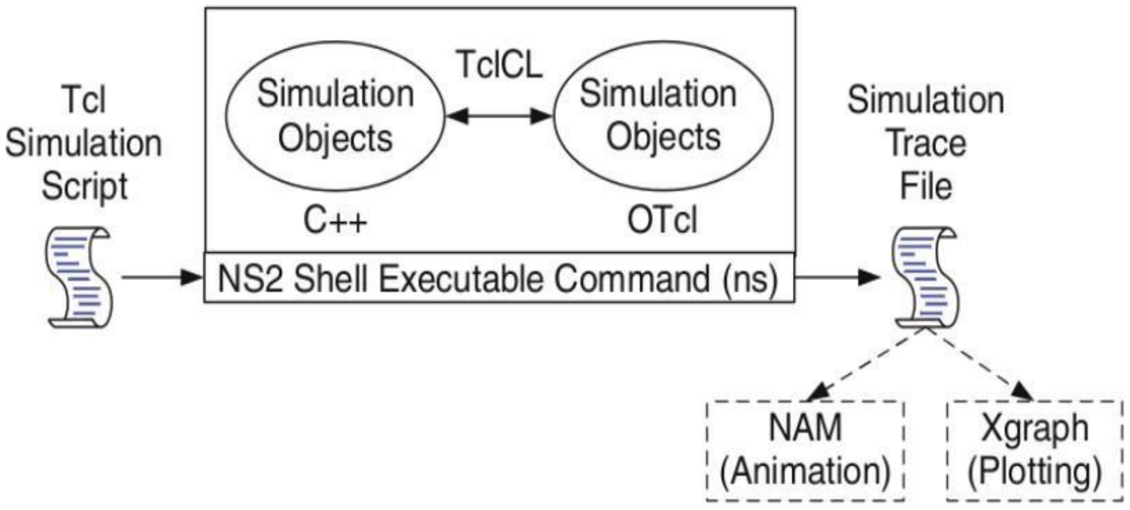
AIM:

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 andvariables 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 becauseit 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 ofa 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 examplewould 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 applicationthat 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.

Ex. No: 7	STUDY OF TCP/UDP PERFORMANCE USING SIMULATION TOOL.
Date:	

Aim: To Study the Performance of TCP and UDP protocols.

STUDY – 1 TCP

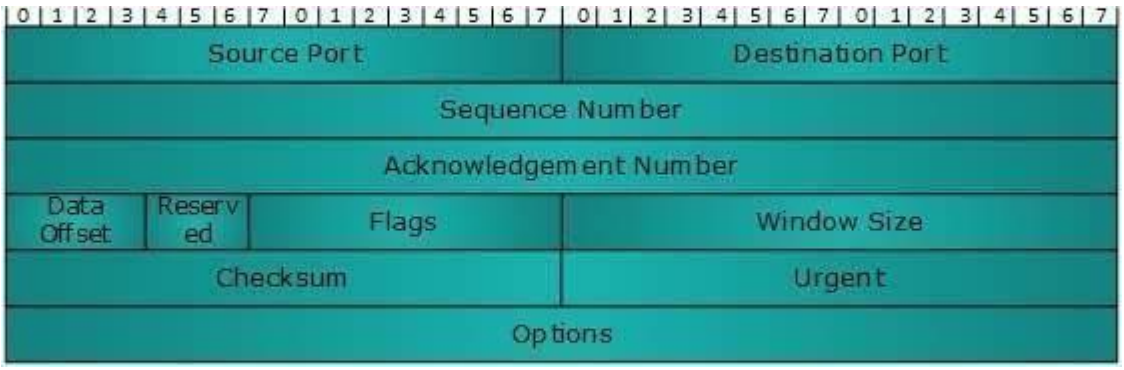
The transmission Control Protocol (TCP) is one of the most important protocols of Internet Protocols suite. It is most widely used protocol for data transmission in communication network such as internet.

Features

- TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.
- TCP ensures that the data reaches intended destination in the same order it was sent.
- TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
- TCP provides error-checking and recovery mechanism.
- TCP provides end-to-end communication.
- TCP provides flow control and quality of service.
- TCP operates in Client/Server point-to-point mode.
- TCP provides full duplex server, i.e. it can perform roles of both receiver and sender.

Header

The length of TCP header is minimum 20 bytes long and maximum 60 bytes.



- **Source Port (16-bits)** - It identifies source port of the application process on the sending device.
- **Destination Port (16-bits)** - It identifies destination port of the application process on the receiving device.
- **Sequence Number (32-bits)** - Sequence number of data bytes of a segment in a session.
- **Acknowledgement Number (32-bits)** - When ACK flag is set, this number contains

the next sequence number of the data byte expected and works as acknowledgement of the previous data received.

- **Data Offset (4-bits)** - This field implies both, the size of TCP header (32-bit words) and the offset of data in current packet in the whole TCP segment.
- **Reserved (3-bits)** - Reserved for future use and all are set zero by default.
- **Flags (1-bit each)**
  - **NS** - Nonce Sum bit is used by Explicit Congestion Notification signaling process.
  - **CWR** - When a host receives packet with ECE bit set, it sets Congestion Windows Reduced to acknowledge that ECE received.
  - **ECE** -It has two meanings:
    - If SYN bit is clear to 0, then ECE means that the IP packet has its CE (congestion experience) bit set.
    - If SYN bit is set to 1, ECE means that the device is ECT capable.
  - **URG** - It indicates that Urgent Pointer field has significant data and should be processed.
  - **ACK** - It indicates that Acknowledgement field has significance. If ACK is cleared to 0, it indicates that packet does not contain any acknowledgement.
  - **PSH** - When set, it is a request to the receiving station to PUSH data (as soon as it comes) to the receiving application without buffering it.
  - **RST** - Reset flag has the following features:
    - It is used to refuse an incoming connection.
    - It is used to reject a segment.
    - It is used to restart a connection.
  - **SYN** - This flag is used to set up a connection between hosts.
  - **FIN** - This flag is used to release a connection and no more data is exchanged thereafter. Because packets with SYN and FIN flags have sequence numbers, they are processed in correct order.
- **Windows Size** - This field is used for flow control between two stations and indicates the amount of buffer (in bytes) the receiver has allocated for a segment, i.e. how much data is the receiver expecting.
- **Checksum** - This field contains the checksum of Header, Data and Pseudo Headers.
- **Urgent Pointer** - It points to the urgent data byte if URG flag is set to 1.
- **Options** - It facilitates additional options which are not covered by the regular header. Option field is always described in 32-bit words. If this field contains data less than 32-bit, padding is used to cover the remaining bits to reach 32-bit boundary.

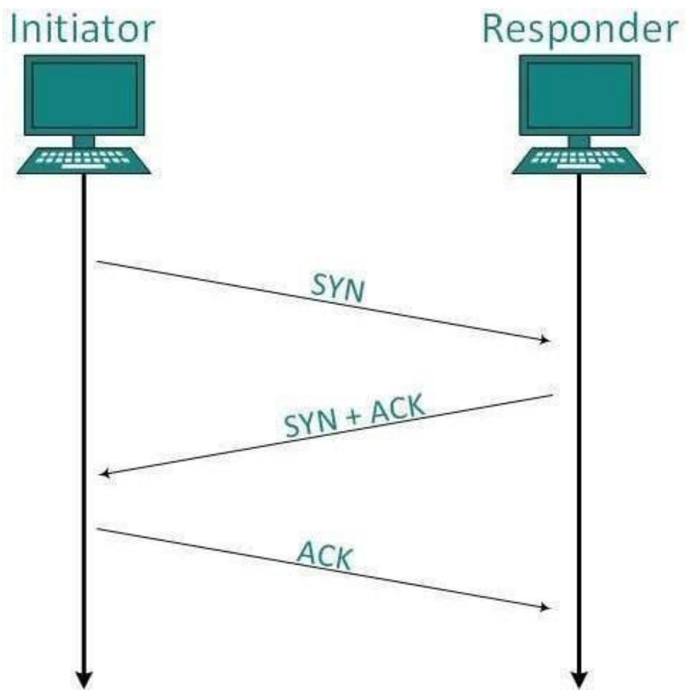
Addressing

TCP communication between two remote hosts is done by means of port numbers (TSAPs). Ports numbers can range from 0 – 65535 which are divided as:

- System Ports (0 – 1023)
- User Ports (1024 – 49151)
- Private/Dynamic Ports (49152 – 65535)

Connection Management

TCP communication works in Server/Client model. The client initiates the connectionand the server either accepts or rejects it. Three-way handshaking is used for connection management.



Establishment
Client initiates the connection and sends the segment with a Sequence number. Server acknowledges it back with its own Sequence number and ACK of client’s segment which is one more than client’s Sequence number. Client after receiving ACK of its segment sends an acknowledgement of Server’s response.

Release
Either of server and client can send TCP segment with FIN flag set to 1. When the receiving end responds it back by Acknowledging FIN, that direction of TCP communication is closed and connection is released.

Bandwidth Management

TCP uses the concept of window size to accommodate the need of Bandwidth management. Window size tells the sender at the remote end, the number of data byte segmentsthe receiver at this end can receive. TCP uses slow start phase by using window size 1 and increases the window size exponentially after each successful communication.

For example, the client uses windows size 2 and sends 2 bytes of data. When the acknowledgement of this segment received the windows size is doubled to 4 and next sent the segment sent will be 4 data bytes long. When the acknowledgement of 4-byte data segment is received, the client sets windows size to 8 and so on.

If an acknowledgement is missed, i.e. data lost in transit network or it received NACK,then the window size is reduced to half and slow start phase starts again.

Error Control &and Flow Control

TCP uses port numbers to know what application process it needs to handover the data segment. Along with that, it uses sequence numbers to synchronize itself with the remote host. All data segments are sent and received with sequence numbers. The Sender knows which last data segment was received by the Receiver when it gets ACK. The Receiver knows about the last segment sent by the Sender by referring to the sequence number of recently received packet.

If the sequence number of a segment recently received does not match with the sequence number the receiver was expecting, then it is discarded and NACK is sent back. If two segments arrive with the same sequence number, the TCP timestamp value is compared tomake a decision.

Multiplexing

The technique to combine two or more data streams in one session is called Multiplexing. When a TCP client initializes a connection with Server, it always refers to a well- defined port number which indicates the application process. The client itself uses a randomly generated port number from private port number pools.

Using TCP Multiplexing, a client can communicate with a number of different application process in a single session. For example, a client requests a web page which in turn contains different types of data (HTTP, SMTP, FTP etc.) the TCP session timeout is increasedand the session is kept open for longer time so that the three-way handshake overhead can be avoided.

This enables the client system to receive multiple connection over single virtual connection. These virtual connections are not good for Servers if the timeout is too long.

Congestion Control

When large amount of data is fed to system which is not capable of handling it, congestion occurs. TCP controls congestion by means of Window mechanism. TCP sets a window size telling the other end how much data segment to send. TCP may use three algorithms for congestion control:

- Additive increase, Multiplicative Decrease
- Slow Start
- Timeout React

Timer Management

TCP uses different types of timer to control and management various tasks:

Keep-alive timer:

- This timer is used to check the integrity and validity of a connection.
- When keep-alive time expires, the host sends a probe to check if the connection still exists.

Retransmission timer:

- This timer maintains stateful session of data sent.
- If the acknowledgement of sent data does not receive within the Retransmission time, the data segment is sent again.

Persist timer:

- TCP session can be paused by either host by sending Window Size 0.
- To resume the session a host needs to send Window Size with some larger value.
- If this segment never reaches the other end, both ends may wait for each other for infinite time.
- When the Persist timer expires, the host re-sends its window size to let the other end know.
- Persist Timer helps avoid deadlocks in communication.

Timed-Wait:

- After releasing a connection, either of the hosts waits for a Timed-Wait time to terminate the connection completely.
- This is in order to make sure that the other end has received the acknowledgement of its connection termination request.
- Timed-out can be a maximum of 240 seconds (4 minutes).

Crash Recovery

TCP is very reliable protocol. It provides sequence number to each of byte sent in segment. It provides the feedback mechanism i.e. when a host receives a packet, it is bound to ACK that packet having the next sequence number expected (if it is not the last segment).

When a TCP Server crashes mid-way communication and re-starts its process it sends TPDU broadcast to all its hosts. The hosts can then send the last data segment which was never unacknowledged and carry onwards.

STUDY – 2 UDP

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

Requirement of UDP

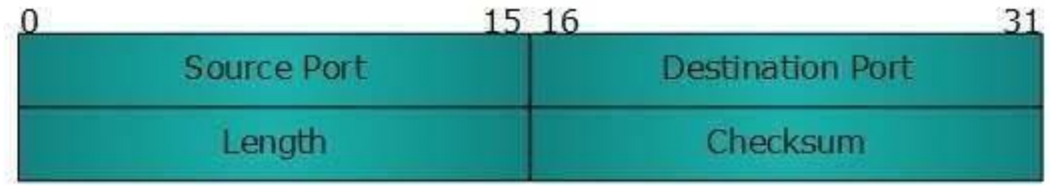
A question may arise, why do we need an unreliable protocol to transport the data? Wedeploy UDP where the acknowledgement packets share significant amount of bandwidth alongwith the actual data. For example, in case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amount of bandwidth wastage. The best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost,the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features

- UDP is used when acknowledgement of data does not hold any significance.
- UDP is good protocol for data flowing in one direction.
- UDP is simple and suitable for query based communications.
- UDP is not connection oriented.
- UDP does not provide congestion control mechanism.
- UDP does not guarantee ordered delivery of data.
- UDP is stateless.
- UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

UDP Header

UDP header is as simple as its function.



UDP header contains four main parameters:

- **Source Port** - This 16 bits information is used to identify the source port of the packet.
- **Destination Port** - This 16 bits information, is used identify application level service on destination machine.
- **Length** - Length field specifies the entire length of UDP packet (including header). It is 16-bits field and minimum value is 8-byte, i.e. the size of UDP header itself.
- **Checksum** - This field stores the checksum value generated by the sender before sending. IPv4 has this field as optional so when checksum field does not contain any value it is made 0 and all its bits are set to zero.

UDP application

Here are few applications where UDP is used to transmit data:

- Domain Name Services
- Simple Network Management Protocol
- Trivial File Transfer Protocol
- Routing Information Protocol
- Kerberos

Result:

Thus, we studied in detail about the Performance of TCP and UDP protocols.

Ex. No: 8	SIMULATION OF DISTANCE VECTOR/ LINK STATE ROUTING ALGORITHM
Date:	

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 tablea router sends
info about its neighborhood only. (ii) To all Routers: each router sends this information toevery 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 ischange.

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 executenam 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.

Ex. No: 9	PERFORMANCE EVALUATION OF ROUTING PROTOCOLS USING SIMULATION TOOL.

AIM:

To Study of performance evaluation of routing protocols using simulation tool

ROUTING PROTOCOLS

There are many routing protocols available. Among them all we are working with AODV and DSR for performance analysis.

A. Ad-hoc On demand Distance Vector (AODV)

It is purely On-Demand route acquisition routing protocol. It is better protocol than DSDV network as the size of network may increase depending on the number of vehicle nodes.

1) Path Discovery Process: In order to discover the path between source and destination, a Route Request message (RREQ) is broadcasted to all the neighbours in radio range who again continue to send the same to their neighboursin there radio range, until the destination is reached. Every node maintains two counters: sequence number and broadcast-id in order to maintain loop-free and most recent route information. The broadcast-id is incremented for every RREQthe source node initiates. If an intermediate node receives the same copy of request, it discards it without routing it further. When a node forwards the RREQ message, it records the address of the neighbour from which it received the first copy of the broadcast packet, in order to maintain a reverse path to the source node. The RREQ packet contains: the source sequence number and the last destination sequence number know to the source. The source sequence number isused to maintain information about reverse route and destination sequence number tells about the actual distance to the final node.

2) Route Maintenance: A source node sends a new moving request packet RREQ to find a new route to the destination. But, if an intermediate node moves from its place, its upstream neighbor noticed the move and sends a message notification failure of the link to each of its active upstream neighbors to inform them about the move to source nodes is achieved. After the detection process is again initiated.

B. Dynamic Source Routing (DSR)

It is an On-Demand routing protocol in which the sequence of nodes through which a packet needs to travel is calculated and maintained as an information in packet header. Every mobile node in the network needs to maintaina route cache where it caches source routes that it has learned. When a packet is sent, the route-cache inside the node is compared with the actual route needs to be covered.

1) Route Discovery: The source node broadcasts request-packets to all theneighbours in the network containing the address of the destination node, and a reply is sent back to the source node with the list of network-nodes through which it should propagate in the process. Sender initiates the route record as a list with a single element containing itself followed by the linking of its neighbour in that route. A request packet also contains an identification number called request-id, which is counter increased only when a new route request packet is being sent bythe source node. To make sure that no loops occur during broadcast, the request is processed in the given order. A route reply is obtained in DSR by two ways: Symmetric-links (bidirectional), in which the backward route is followed again to catch the source node. Asymmetric-links (unidirectional) needs to discover the route up to the source node in the same manner as the forward route is discovered.

2) Route Maintenance: In the hop by hop acknowledgement at data link layer allows the early detection and retransmission of lost or corrupt packets in the data-link layer. If a transmission error occurs, a route error packet containingthe address of node detecting the error and the host address is sent back to the sender. Whenever a node receives a route error packet, the hop in error is removed from the route cache and all routes containing this hop are truncated atthat point. When the wireless transmission between two nodes does not work equally well in both directions, and then end-to-end replies on the application ortransport layer may be used to indicate the status of the route from one host to the other

RESULT

Thus the performance evaluation of routing protocols was studied.

Ex. No: 10	**SIMULATION OF ERROR CORRECTION CODE (LIKE CRC).**
Date:	

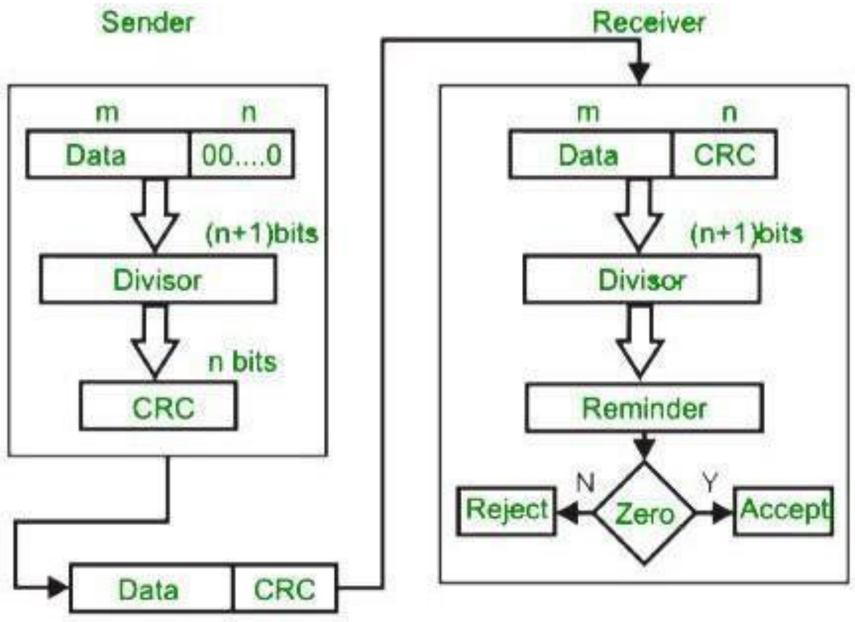
AIM :

To Simulation of Error Correction Code using Java.

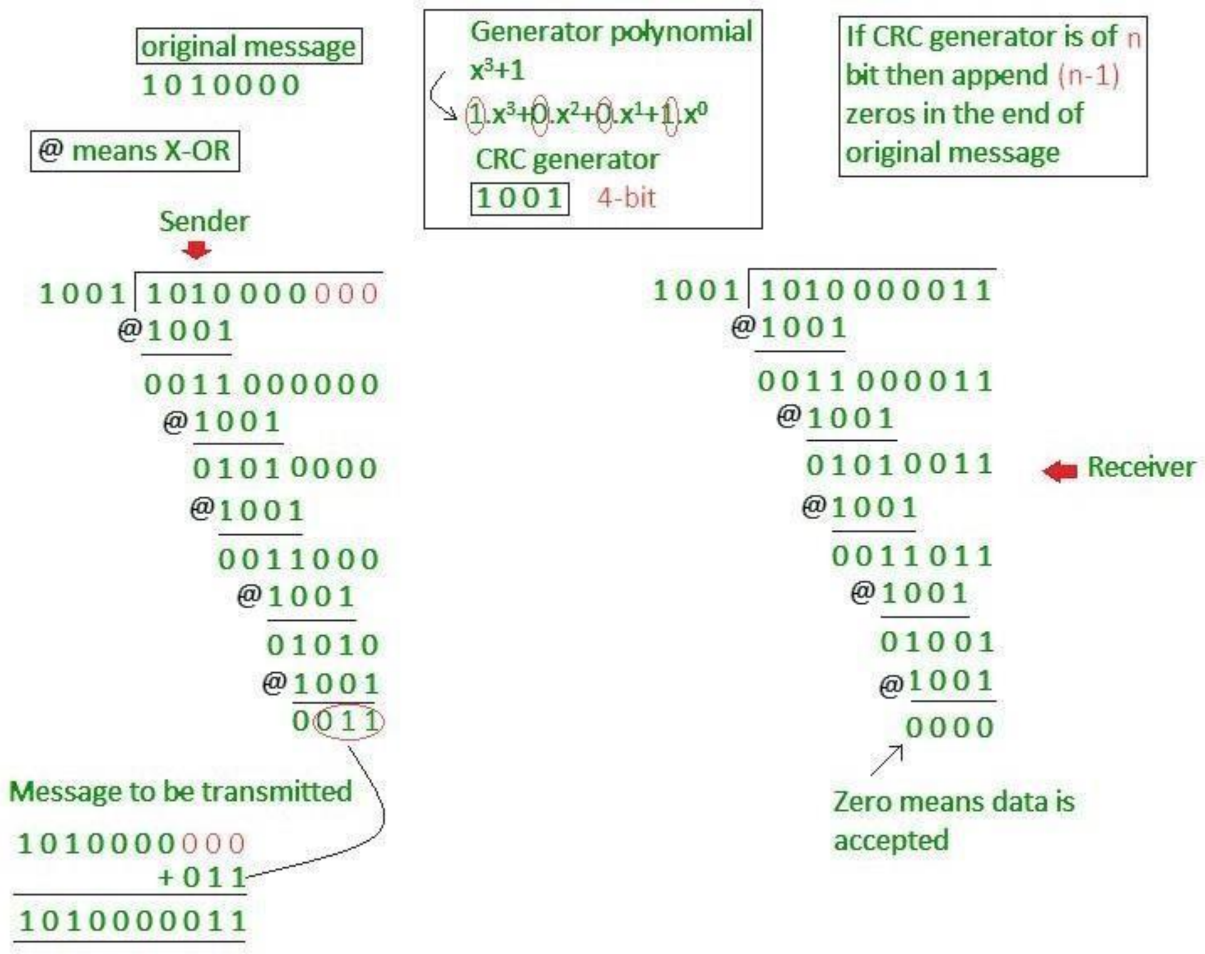
Theory:

Cyclic redundancy check (CRC)

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.



Example :



:

Procedure:

1. Open the editor and type the program for error detection
2. Get the input in the form of bits.
3. Append the redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise there is some error in the received bits
9. Run the program.

Program:

```
import java.io.*;

class CRC

{

 public static void main(String args[]) throws IOException
```

```
{
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter Generator:");
 String gen = br.readLine();
 System.out.println("Enter Data:");
 String data  = br.readLine();
 String code = data;
 while(code.length() < (data.length() + gen.length() - 1))
  code = code + "0";
 code = data + div(code,gen);
 System.out.println("The transmitted Code Word is: " + code);
 System.out.println("Please enter the received Code Word: ");
 String rec = br.readLine();
 if(Integer.parseInt(div(rec,gen)) == 0)
  System.out.println("The received code word contains no errors.");
 else
  System.out.println("The received code word contains errors.");
}

static String div(String num1,String num2)
{
 int pointer = num2.length();
 String result = num1.substring(0, pointer);
 String remainder = "";
 for(int i = 0; i < num2.length(); i++)
 {
  if(result.charAt(i) == num2.charAt(i))
   remainder += "0";
  else
   remainder += "1";
  }
  while(pointer < num1.length())
  {
   if(remainder.charAt(0) == '0')
   {
    remainder = remainder.substring(1, remainder.length());
    remainder = remainder + String.valueOf(num1.charAt(pointer));
    pointer++;
   }
   result = remainder;
   remainder = "";
   for(int i = 0; i < num2.length(); i++)
   {
    if(result.charAt(i) == num2.charAt(i))
     remainder += "0";
    else
     remainder += "1";
   }
  }
  return remainder.substring(1,remainder.length());
 }
}
```

Result:

Thus the error detection and error correction is implemented successfully.