DATE:	LEARN TO USE COMMANDS LIKE tcpdump,netstat,ifconfig,nslookup AND tracerout.CAPTURE PING AND TRACWROUT PDUs USING A NETWORKK PROTOCOL ANALYSER AND EXAMINE.
EXPT. NO: 1	

`

AIM:

Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine

PROGRAM:

a) Check Network Connectivity Using ping Command

The ping command is one of the most used Linux network commands in network troubleshooting. It is used to check whether or not a specific IP address can be reached.

The ping command works by sending an ICMP echo request to check the network connectivity. $

ping google.com

PING google.com (172.217.160.142) 56(84) bytes of data.

--- google.com ping statistics ---

4 packets transmitted, 0 received, 100% packet loss, time 3022ms jec@jec-WIV59605-

0002:~$ ^C

If you want to limit the number of echo requests made to 3, you can do it like this:

$ ping -c 3 google.com

PING google.com (172.217.26.174) 56(84) bytes of data.

--- google.com ping statistics ---

3 packets transmitted, 0 received, 100% packet loss, time 2016ms

Here ping command stops sending echo requests after 3 cycles.

b) Get DNS Records Using dig and host Commands

Dig stands for (**Domain Information Groper**) is a network administration command-line tool for querying **Domain Name System** (**DNS**) name servers. It is useful for verifying and troubleshooting **DNS** problems and also to perform **DNS** lookups and displays the answers that are returned from the name server that were queried. dig is part of the BIND domain name server software suite. dig command replaces older tool such as **nslookup** and the host. dig tool is available in major Linux distribution

A record

The A record is one of the most commonly used record types in any DNS system. An A record is actually an address record, which means it maps a fully qualified domain name (FQDN) to an IP

address. For example, an A record is used to point a domain name, such as "google.com", to the IP address of Google's hosting server, "74.125.224.147".

CNAMErecord

Canonical name records, or CNAME records, are often called alias records because they map an alias to the canonical name. When a name server finds a CNAME record, it replaces the name with the canonical name and looks up the new name. This allows pointing multiple systems to one IP without assigning an A record to each host name.

MXrecord

The MX record, which stands for "mail exchange", is used to identify mail servers to which mail should be delivered for a domain. MX entries must point to a domain, and never point directly to an IP address. If no MX record exists on a domain to which an SMTP server attempts to deliver mail, the server will attempt to deliver the mail to the matching A record.

NSrecord

An NS record identifies which DNS server is authoritative for a particular zone. The "NS" stands for "name server". NS records that do not exist on the apex of a domain are primarily used for splitting up the management of records on sub-domains.

SOArecord

The SOA or Start of Authority record for a domain stores information about the name of the server that supplies the data for the zone, the administrator of the zone and the current version of the data

TXTrecord

A TXT record allows domain administrators to insert any text into the DNS record. It is usually used to denote facts about the domain.

Dig commands (Equivalent to nslookup)

1.Querying doamin Arecord

$ dig yahoo.com; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.10.rc1.el6_3.2 <<>> yahoo.com

;; global options: +cmd

;; Got answer:

;; ->>HEADER<

Above command causes dig to look up the **"A"** record for the domain name **yahoo.com**. Dig command reads the **/etc/resolv.conf** file and querying the **DNS** servers listed there. The response from the **DNS** server is what dig displays.

1. Query Domain “A” Record with +short

$ dig yahoo.com +short

98.139.183.24

72.30.38.140

98.138.253.109

2. Querying MX Record for Domain

$ **dig yahoo.com MX**

; <> DiG 9.8.2rc1-RedHat-9.8.2-0.10.rc1.el6_3.2 <> yahoo.com MX

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31450 ;; flags: qr rd

ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 24

;; QUESTION SECTION:

;yahoo.com. IN MX

;; ANSWER SECTION: yahoo.com. 33 IN MX 1 mta6.am0.yahoodns.net.

yahoo.com. 33 IN MX 1

mta7.am0.yahoodns.net. yahoo.com. 33 IN MX 1 mta5.am0.yahoodns.net.

You can get all types of records by using ANY query.

3. Querying SOA Record for Domain

$ dig yahoo.com SOA

; <> DiG 9.8.2rc1-RedHat-9.8.2-0.10.rc1.el6_3.2 <> yahoo.com
SOA ;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2197 ;; flags: qr rd

ra; QUERY: 1, ANSWER: 1, AUTHORITY: 7, ADDITIONAL: 7

;; QUESTION SECTION:

;yahoo.com. IN SOA

;; ANSWER SECTION:

yahoo.com. 1800 IN SOA ns1.yahoo.com. hostmaster.yahoo-inc.com. 2012081409 3600 300 1814400 600

4. Querying TTL Record for Domain

$ dig yahoo.com TTL

; <> DiG 9.8.2rc1-RedHat-9.8.2-0.10.rc1.el6_3.2 <> yahoo.com TTL

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56156

;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:

;yahoo.com. IN A

;; ANSWER SECTION:

yahoo.com. 3589 IN A 98.138.253.109

yahoo.com. 3589 IN A 98.139.183.24

yahoo.com. 3589 IN A 72.30.38.140

5. Querying only answer section

$ dig yahoo.com +nocomments +noquestion +noauthority +noadditional +nostats

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.10.rc1.el6 <<>> yahoo.com +nocomments +noquestion +noauthority +noadditional +nostats

;; global options: +cmd yahoo.com.

3442 IN A 72.30.38.140 yahoo.com.

3442 IN A 98.138.253.109

yahoo.com. 3442 IN A 98.139.183.24

6. DNS Reverse Look-up

Querying **DNS** Reverse Look-up. Only display answer section with using **+short**.

 # dig -x 72.30.38.140 +short

ir1.fp.vip.sp2.yahoo.com.

7. Querying Multiple DNS Records

Query multiple website's DNS specific query viz. **MX**, **NS** etc. records.

$ dig yahoo.com mx +noall +answer redhat.com ns +noall +answer

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.10.rc1.el6 <<>> yahoo.com mx +noall +answer redhat.com ns +noall +answer

;; global options: +cmd yahoo.com. 1740 IN MX 1

mta6.am0.yahoodns.net. yahoo.com. 1740 IN MX 1

mta7.am0.yahoodns.net. yahoo.com. 1740 IN MX 1

mta5.am0.yahoodns.net. redhat.com. 132 IN NS

ns1.redhat.com. redhat.com. 132 IN NS

ns4.redhat.com. redhat.com. 132 IN NS

ns3.redhat.com.

redhat.com. 132 IN NS ns2.redhat.com.

c) Traceroute

For most Linux distributions, you'll need to first install the traceroute package.

To install:

1. Open your terminal.

2. Run the following to install in Ubuntu:

$ sudo apt-get install traceroute

$ traceroute google.com traceroute to google.com (172.217.26.174), 30 hops max, 60 byte packets

1 123.63.31.242 (123.63.31.242) 8.919 ms 8.906 ms 8.890 ms

2 182.19.110.48 (182.19.110.48) 52.587 ms 52.579 ms 52.565 ms

3 103.29.44.11 (103.29.44.11) 52.241 ms 51.712 ms 52.196 ms

4 103.29.44.8 (103.29.44.8) 52.496 ms 52.911 ms 52.902 ms

5 72.14.220.66 (72.14.220.66) 58.000 ms 57.992 ms 57.978 ms

6 108.170.251.101 (108.170.251.101) 61.004 ms 74.125.243.98 (74.125.243.98) 55.195 ms108.170.251.118 (108.170.251.118) 62.769 ms

7 172.253.68.97 (172.253.68.97) 51.098 ms 54.552 ms 54.058 ms

8 108.170.253.113 (108.170.253.113) 54.028 ms 216.239.41.85 (216.239.41.85) 68.188 ms

172.253.68.95 (172.253.68.95) 54.967 ms

9 108.170.253.113 (108.170.253.113) 62.035 ms 62.003 ms 108.170.253.97 (108.170.253.97) 70.639 ms

10 209.85.243.49 (209.85.243.49) 67.755 ms 74.125.253.65 (74.125.253.65) 59.933 ms 66.249.94.90 (66.249.94.90) 53.912 ms

11 108.170.253.113 (108.170.253.113) 54.372 ms 209.85.243.49 (209.85.243.49) 51.128 msmaa03s22-in-f174.1e100.net (172.217.26.174) 54.450 ms

You can run the following command to write the results of traceroute to a file, and you can specify any location you wish.

In the following example, the results are written to a file named trace.txt under the user named 'username':

$ traceroute -I example.com > /home/username/trace.tx

To avoid reverse DNS lookup, use the -n option.

$ traceroute -n google.com traceroute to google.com (172.217.26.174),

30 hops max, 60 byte packets

1 123.63.31.242 8.596 ms 8.593 ms 8.582 ms

2 182.19.110.48 49.436 ms 49.871 ms 49.866 ms

3 103.29.44.11 50.205 ms 51.463 ms 51.432 ms

4 103.29.44.8 50.852 ms 50.499 ms 52.612 ms

5 72.14.220.66 56.978 ms 56.954 ms 56.924 ms

6 74.125.244.196 60.639 ms 74.125.243.99 64.776 ms 74.125.244.195 59.104 ms 7

216.239.42.48 59.102 ms 172.253.68.169 51.403 ms 216.239.63.226 56.074 ms 8

108.170.253.113 49.974 ms 172.253.68.95 50.345 ms 52.199 ms

9 108.170.253.113 55.019 ms 209.85.243.49 51.548 ms 172.253.68.95 54.388 ms

10 74.125.253.65 53.963 ms 66.249.94.90 55.773 ms 108.170.253.113 49.812 ms

11 172.217.26.174 55.741 ms 51.685 ms 52.699 ms

If you need to send ICMP packet, you can send it like this:

$ sudo traceroute -I google.com traceroute to google.com

(172.217.26.174), 30 hops max, 60 byte packets

1 * * *

2 * * * 3

* * * 4 *

* * 5 * *

* 6 * * *

7 * * * 8

* * * 9 *

* * 10 *

* *

11 * * *

12 * * *

13 * * *

14 * * *

15 * * *

16 * * *

17 * * *

18 * * *

19 * * *

20 * * *

21 * * *

22 * * *

23 * * *

24 * * *

25 * * *

26 * * *

27 * * *

28 * * *

29 * * *

30 * * *

d) Packet Analysis with tcpdump

One of the most important Linux network commands is The tcpdump command. tcpdump command is used to capture the traffic that is passing through your network interface.

This kind of access to the packets which is the deepest level of the network can be vital when troubleshooting the network.

Installing tcpdump on Ubuntu is very easy. You can install this by running a simple command on terminal.

$ sudo **apt-get install tcpdump**

Capture packets from a specific interface

If you execute the TCPdump command with the "-i" flag you can name an interface and the TCPdump tool will start capture that specific interface packets for you.

$tcpdump -i *eth0*

Capture only specific number of packets

Using "-c" flag will allow you to capture a specific number of packets, for example, with the command below we can capture 20 packets of our eth0 interface:

$tcpdump -i eth0 -c 20

Print captured packets in ASCII

The below TCPdump command with the flag "-A" displays the packages in ASCII format. it's a character-encoding scheme format.

$tcpdump -i eth0 -A

Display available interfaces

To get a list of available interfaces on the system you can run the following command:

$tcpdump -D

Capture and save packets in a file

TCPdump has a feature to capture and save its result in a ".pcap" file, to do this just

execute: $tcpdump -w *eth0*.pcap -i eth0

If you don't use "-c" flag it will start capturing eth0 and write the result to the output file until you break it with "Ctrl+c".

For read and analyze the file that you just created execute:

$tcpdump -r eth0.pcap

Capture IP address packets

If you want to capture your network interface and analyze the IP address you can use the "-n" flag it

will stop translating IP addresses into Hostnames and This can be used to avoid DNS lookups.

$tcpdump -n -i eth0

Capture only TCP packets

To capture packets based on TCP ports, add a "tcp" in your command:

$tcpdump -i eth0 -c 20 -w tcpanalyze.pcap *tcp*

Capture packets from a specific port

Let's assume you want to monitor on a specific port like 80, you can use the following command to do that with TCPdump:

$tcpdump -i eth0 port 80

Filter records with source and destination IP

To Capture packets from a source IP you can use the following command:

$tcpdump -i eth0 *src 192.168.1.1*

You can monitor packets from a destination IP as well with the command below:

$tcpdump -i eth0 *dst 192.168.1.1*

e) ifconfig

Ifconfig is used to configure the kernel-resident network interfaces. It is used at boot time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed.

If no arguments are given, ifconfig displays the status of the currently active interfaces. If a single interface argument is given, it displays the status of the given interface only; if a single -a argument is given, it displays the status of all interfaces, even those that are down. Otherwise, it configures an interface.

$ ifconfig

Options

-a display all interfaces which are currently available, even if down

-s display a short list (like netstat -i)

-v be more verbose for some error conditions

interface

The name of the interface. This is usually a driver name followed by a unit number, for example eth0 for the first Ethernet interface. If your kernel supports alias interfaces, you can specify them with eth0:0 for the first alias of eth0.

f) netstat

netstat (**network statistics**) is a command line tool for monitoring network connections both incoming and outgoing as well as viewing routing tables, interface statistics etc. **netstat** is available on all Unix-like Operating Systems and also available on **Windows OS** as well. It is very useful in terms of network troubleshooting and performance measurement. **netstat** is one of the most basic network service debugging tools, telling you what ports are open and whether any programs are listening on ports.

This tool is very important and much useful for Linux network administrators as well as system administrators to monitor and troubleshoot their network related problems and determine network traffic performance. This article shows usages of **netstat** command with their examples which may be useful in daily operation.

1. Listing all the LISTENING Ports of TCP and UDP connections

Listing all ports (both TCP and UDP) using **netstat** -a option

.$ **netstat -a | more**

2. Listing TCP Ports connections

Listing only **TCP** (**Transmission Control Protocol**) port connections using **netstat -at**.

$ netstat -at

3. Listing UDP Ports connections

Listing only **UDP** (**User Datagram Protocol**) port connections using **netstat -au**.

$ netstat -au

4. Listing all LISTENING Connections

Listing all active listening ports connections with **netstat -l**.

$ netstat -l

5. Listing all TCP Listening Ports

Listing all active listening TCP ports by using option **netstat -lt**.

$ netstat -lt

6. Listing all UDP Listening Ports

Listing all active listening UDP ports by using option **netstat -lu**.

$ netstat -lu

7. Listing all UNIX Listening Ports

Listing all active UNIX listening ports using **netstat -lx**.

$ netstat -lx

`

8. Showing Statistics by Protocol

Displays statistics by protocol. By default, statistics are shown for the TCP, UDP, ICMP, and IP protocols. The -s parameter can be used to specify a set of protocols.

$ netstat -s

9. Showing Statistics by TCP Protocol

Showing statistics of only TCP protocol by using option **netstat -st**.

$ netstat -st

10. Showing Statistics by UDP Protocol

$ netstat -su

11. Displaying Service name with PID

Displaying service name with their PID number, using option **netstat -tp** will display "PID/Program Name".

$ netstat -tp

12. Displaying Promiscuous Mode

Displaying Promiscuous mode with -ac switch, netstat print the selected information or refresh screen every five second. Default screen refresh in every second.

$ netstat -ac 5 | grep tcp

13. Displaying Kernel IP routing

Display Kernel IP routing table with netstat and route command.

$ netstat -r

14. Showing Network Interface Transactions

Showing network interface packet transactions including both transferring and receiving packets with MTU size.

$ netstat -i

15. Showing Kernel Interface Table

Showing Kernel interface table, similar to **ifconfig** command.

$ netstat -ie

16. Displaying IPv4 and IPv6 Information

Displays multicast group membership information for both IPv4 and IPv6.

$ netstat -g

17. Print Netstat Information Continuously

To get netstat information every few second, then use the following command, it will print netstat information continuously, say every few seconds.

$ netstat -c

18. Finding non supportive Address

Finding un-configured address families with some useful information.

$ netstat --verbose

19. Finding Listening Programs

Find out how many listening programs running on a port.

$ netstat -ap | grep http

20. Displaying RAW Network Statistics

$ netstat --statistics --raw

RESULT:

The Sliding Window Protocol using Java Program is Successfully implementation and Output Verified.

`

DATE:	**WRITE A HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEBPAGE USING TCP SOCKETS.**
EXP. NO: 2	

AIM:

To write a java program to create a socket for HTTP for webpage upload and download.

ALGORITHM:

SERVER

1. Create a Server socket and bind it to the port.
2. Listen for new connection and when a connection arrives and accept it.
3. Receive the image from the client
4. Display the image along with its size.
5. Close the input and output streams.
6. Stop.

CLIENT

1. Create a client socket and connect it to the server's port.
2. Open an image from the disk.
3. Send this image to the server.
4. Close the input and output streams.
5. Close the client socket.
6. Stop.

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("Balloons.jpg"));
ByteArrayOutputStream baos=new
ByteArrayOutputStream(); ImageIO.write(img,"jpg",baos);
baos.flush(); byte[] bytes=baos.toByteArray(); baos.close();
System.out.println("Sending image to screen");
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());
45soc.close();
}
soc.close();
}}
```

SERVER:

```
import java.io.*; import java.net.*; 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);
ack();
f.setVisible(true);
}}
```

OUTPUT:

SERVER:

```
thirdyear@rmdl4-29:~$ javac Client.java thirdyear@rmdl4-
29:~$ java Client
Client is running
```

Reading image from disk.

Sending image to screen

Image sent to server.

CLIENT:

thirdyear@rmdl4-29:~$ javac Server.java thirdyear@rmdl4-

29:~$ java Server

Server Waiting for image Client

connected Image Size:6KB

RESULT:

Thus the program to implement the socket for HTTP for web page upload and download has been executed successfully and output verified using various samples

`

DATE:	**APPLICATION ON USING TCP SOCKET LIKE ECHOCLIENT & ECHO SERVER**
EXP. NO: 3(A)	

AIM:

To write a java program to implement Echo Client and server using TCP socket.

ALGORITHM:

SERVER

1. Create a Server socket.
2. Wait for Client to be conne0cted.
3. Read the text back to the Client.

4. Chose the input and output streams.

5. Close the Server sockets.

CLIENT

1. Create a socket and establish connection with the server.

2. Get the input from the receiver.

3. Send the text to the user.

4. Display the text echo by the server

5. Repeat steps 2 to 4.

6. Close input and output streams.

7. Close the client socket.

PROGRAM:

CLIENT:

```
import java.io.*;
import java.net.*;
import java.util.*;
public class echoclient
{
public static void main(String
args[])throws Exception
{
Socket c=null;
DataInputStream usr_inp=null;
DataInputStream din=new
DataInputStream(System.in); DataOutputStream
dout=null; try
{  c=new  Socket("LocalHost",5678);  usr_inp=new
DataInputStream(c.getInputStream());      dout=new
DataOutputStream(c.getOutputStream());
} catch(IOException
e)
{
```

```
}
if(c!=null || usr_inp!=null || dout!=null)
{
String unip;
System.out.println("\n Enter your message");
while((unip=din.readLine())!=null)
{
dout.writeBytes(""+unip);
dout.writeBytes("\n");
System.out.println("\n The echoed message");
System.out.println(usr_inp.readLine());
System.out.println("\n Enter your
message"); }
System.exit(0);
}
din.close(); usr_inp.close(); c.close();
}
}
```

SERVER:

```
import java.io.*; import
java.net.*; public class
echoserver
{
public static void main(String
args[])throws Exception
{
ServerSocket m=null;
Socket c=null;
DataInputStream usr_inp=null;
DataInputStream din=new DataInputStream(System.in);
DataOutputStream dout=null; try { m=new
```

```
ServerSocket(5678);  c=m.accept();  usr_inp=new
DataInputStream(c.getInputStream());  dout=new
DataOutputStream(c.getOutputStream());
} catch(IOException
e) {}
if(c!=null || usr_inp!=null)
{
String unip; while(true)
{
System.out.println("\nMessage from Client...");
String m1=(usr_inp.readLine());
System.out.println(m1);
dout.writeBytes(""+m1); dout.writeBytes("\n");
}

}
dout.close(); usr_inp.close();
lose();
}
}
```

OUTPUT

SERVER:

```
secondyearlab2@rmdl2-34:-/Desktop$ javac echoserver.java secondyearlab2@rmdl2-34:-
/Desktop$ java echoserver
Message from Client...
hai
Message from Client...
how are you?
```

CLIENT:

secondyearlab2@rmdl2-34:-/Desktop$ javac echoclient.java

secondyearlab2@rmdl2-34:-/Desktop$ java echoclient Enter

your message hai

The echoed message

hai

Enter your message how are

you? The echoed message how

are you?

RESULT:

Thus the program to implement the Echo Client and Echo Server using TCP Sockets has been executed successfully and output verified using various samples.

`

DATE:	**APPLICATION ON USING TCP SOCKET LIKE CHAT**
EXP. NO:3(b)	

AIM:

To write a java program for chat application using TCP Sockets.

ALGORITHM:

SERVER

1. Create a server socket.
2. Wait for Client to be connected.
3. Read the messages from the client.
4. Display it on the server machine.
5. Close the input and output streams.

CLIENT

1. Create a socket and establish connection with a server.
2. Enter message and send it to the server.
3. Read message from server and display it on the client.
4. Chose the input and output streams.

5. Chose the client sockets.

PROGRAM:

CLIENT:

```
import java.io.*;

import java.net.*;

public class talkclient
{

public static void main(String

args[])throws Exception

{

Socket c=null;
DataInputStream usr_inp=null;
DataInputStream din=new
DataInputStream(System.in); DataOutputStream
dout=null; try
{

c=new Socket("LocalHost",1234);

usr_inp=new DataInputStream(c.getInputStream());
62dout=new DataOutputStream(c.getOutputStream());
} catch(IOException

e)

{}

if(c!=null || usr_inp!=null || dout!=null)

{

String unip;

System.out.println("\nEnter the message for server:");
while((unip=din.readLine())!=null)
{

dout.writeBytes(""+unip);
dout.writeBytes("\n");
System.out.println("Message from Server:");
System.out.println(usr_inp.readLine());

System.out.println("\n Enter the message for

server:"); }
```

```
System.exit(0);

}din.close(); usr_inp.close(); c.close();
}

}
```

SERVER:

```
import java.io.*;

import java.net.*;

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

{

ServerSocket m=null; Socket c=null;
DataInputStream usr_inp=null;

DataInputStream din=new
DataInputStream(System.in); DataOutputStream
dout=null; try {

m=new ServerSocket(1234); c=m.accept();
usr_inp=new DataInputStream(c.getInputStream());
dout=new DataOutputStream(c.getOutputStream());
} catch(IOException

e) {}

if(c!=null||usr_inp!=null)

{

String unip; while(true)
{

System.out.println("\nMessage from client:");
String m1=usr_inp.readLine();
System.out.println(m1);
System.out.println("Enter the message for
client:"); unip=din.readLine();
dout.writeBytes(""+unip); dout.writeBytes("\n");
}

}

dout.close();
usr_inp.close(); c.close();
```

```
}}
```

OUTPUT:
SERVER:

thirdyear@rmdl2-34:-/Desktop$ javac talkserver.java thirdyear@rmdl2-34:-

/Desktop$ java talkserver

Message from client:

Hai!

Enter the message for client:

Hi!

How are you?

Message

from client:

Fine

CLIENT:

thirdyear@rmdl2-34:-/Desktop$ javac talkclient.java

thirdyear@rmdl2-34:-/Desktop$ java talkclient Enter

the message for server:

Hai!

Message from server:

Hi! How are you?

Enter the message for server:

Fine

RESULT:

Thus the program to implement the Chat application using TCP Sockets has been executed successfully and output verified using various samples.

DATE:	**APPLICATION ON USING TCP SOCKET LIKE FILE TRANSFER**
EXP. NO: 3(c)	

AIM:

To write java program using TCP sockets for File transfer.

ALGORITHM:

SERVER

1. Create a server socket.
2. Wait for Client to be connected.
3. Display content of file transferred on screen.
4. Close the input and output streams.
5. Close the server socket.

CLIENT

1. Create a socket and establish connection with a server.
2. Enter the file name that has to be transferred.
3. Enter the new file name.
4. Contents of the file are displayed on the screen.
5. Chose the input and output streams.
6. Chose the client sockets.

PROGRAM:

CLIENT

```
import java.io.*;
import java.net.*;
import java.util.*;
class fileclient
{ public static void main(String args[])
{
try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
Socket clsct=new
Socket("LocalHost",1391);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the file name:");
String str=in.readLine();
dout.writeBytes(str+'\n');
System.out.println("Enter the new file name:");
```

```
String str2=in.readLine();
String str1,ss;

FileWriter f=new
FileWriter(str2); char buffer[];
while(true)
{ str1=din.readLine();
if(str1.equals("-1")) break;
System.out.println(str1);
buffer=new
char[str1.length()];
str1.getChars(0,str1.length(),buffer,0);
f.write(buffer).f.close(); clsct.close();
} catch (Exception
e)
{
System.out.println(e);
}
}
}
```

SERVER

```
import java.io.*; import java.net.*; import
java.util.*; class fileserver
{ public static void main(String args[])
{
try
{
ServerSocket obj=new ServerSocket(1391);
while(true)
{
Socket obj1=obj.accept();
DataInputStream din
=new DataInputStream(obj1.getInputStream());
DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());
String str=din.readLine();
FileReader f=new FileReader(str);
BufferedReader b=new
BufferedReader(f); String s;
while((s=b.readLine())!=null)
{
System.out.println(s);
dout.writeBytes(s+'\n');
}
lose(); dout.writeBytes("-
1\n");
}
```

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

OUTPUT:

CLIENT:

thirdyear@rmdl2-34:-/Desktop$ javac fileclient.java thirdyear@rmdl2-34:-/Desktop$ java fileclient Enter the file name: abc.txt

Enter the new file name:

xyz.txt

Hi! How are you?

SERVER:

thirdyear@rmdl2-34:-/Desktop$ javac fileserver.java thirdyear@rmdl2-34:-/Desktop$ java fileserver Hi! How are you?

RESULT:

Thus the program to implement the File Transfer application using TCP Sockets has been executed successfully and output verified using various samples.`

DATE:	SIMULATION OF DNS USING UDP SOCKETS
`EXP. NO: 4	

AIM:

To implement a DNS server and client in java using UDP Socket.

ALGORITHM:

SERVER

1. Create a datagram socket.
2. Create a datagram packet to receive client request
3. Read the domain name to be resolved.
4. Lookup the host array for domain name.
5. If found, retrieve the corresponding address.
6. Close the server socket.

CLIENT

1. Create a datagram socket and get domain name.
2. Construct a datagram packet to send domain name to the server.3.

Construct a datagram packet to receive server message. 4. If it contains IP address, then display it.

5. Close the client socket.
6. Stop.

PROGRAM:

CLIENT:

```
import java.io.*; import java.net.*;
public class udpdnsclient
{ 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();
}
}
```

SERVER

```
import java.io.*; import
java.net.*; public class
udpdnsserver
{
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 = {"yahoo.com", "gmail.com","cricinfo.com",
"facebook.com"};
String[] ip = {"68.180.206.184",
"209.85.148.19","80.168.92.140",
"69.63.189.16"}; System.out.println("Press Ctrl +
C to Quit"); while (true)
{
DatagramSocket serversocket=new
DatagramSocket(1362); byte[] senddata = new byte[1021];
byte[] receivedata = new byte[1021]; DatagramPacket
recvpack = new DatagramPacket
(receivedata, receivedata.length);
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();
}
}
}
```

OUTPUT:

CLIENT:

thirdyear@rmdl2-34:-/Desktop$ javac

udpdnsclient.java thirdyear@rmdl2-34:-/Desktop$ java

udpdnsclient Enter the hostname: gmail.com IP

Address: 209.85.148.19

SERVER:

thirdyear@rmdl2-34:-/Desktop$ javac udpdnsserver.java thirdyear@rmdl2-34:-

/Desktop$ java udpdnsserver Press

Ctrl + C to Quit

Request for host gmail.com

RESULT:

Thus the program to implement the DNS application using UDP Sockets has been executed successfully and output verified using various samples.

`

DATE:	**WRITE A CODE SIMULATING ARP/RAPP PROTOCOL**
EXP. NO: 5	

AIM:

To write a java program for simulating ARP protocols using TCP.

ALGORITHM:

CLIENT

1. Start the program
2. Using socket connection is established between client and server.
3. Get the IP address to be converted into MAC address.
4. Send this IP address to server.
5. Server returns the MAC address to client.

SERVER

1. Start the program
2. Accept the socket which is created by the client.
3. Server maintains the table in which IP and corresponding MAC addresses are stored.
4. Read the IP address which is send by the client.
5. Map the IP address with its MAC address and return the MAC address to client.

PROGRAM:

Client:

```
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",139);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the Logical
address(IP):"); Stringstr1=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(1390); 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 implementation of java program for simulating ARP protocols using TCP successfully completed.

EX.NO :6(a)	**STUDY OF NETWORK SIMULATOR (NS)**
DATE:	

` **AIM:**

To Study the Network Simulator (NS2).

INTRODUCTION:

The Network Simulator version 2 (NS-2) is a deterministic discrete event network simulator, initiated at the Lawrence Berkeley National Laboratory (LBNL) through the DARPA funded Virtual Internetwork Test bed (VINT) project.

The VINT project is collaboration between the Information Sciences Institute (ISI) at the University of Southern California (USC), Xerox's Palo Alto Research Centre (Xerox PARC), University of California at Berkeley

(UCB) and LBNL. NS-2 was initially created in 1989 as an alternative to the REAL Network Simulator.

Since then there is significant growth in uses and width of NS project.

Although there are several different network simulators available today, ns-2 is one of the most common.

NS-2 differs from most of the others by being open source software, supplying the source code for free to anyone that wants it.

Whereas most commercial network simulators will offer support and a guarantee but keeping the moneymaking source code for themselves.

THE STRUCTURE OF NS2

NS-2 is made up of hundreds of smaller programs, separated to help the user sort through and find what he or she is looking for.

Every separate protocol, as well as variations of the same, sometimes has separate files. Though some are simple, it still dependent on the parental class.

C++

C++ is the predominant programming language in ns-2. It is the language used for all the small programs that make up the ns-2 hierarchy. C++, being one of the most common programming languages and specially designed for object- oriented coding, was therefore a logical choice what language to be used. This helps when the user wants to either understand the code or do some alterations to the code. There are several books about C++ and hundreds, if not thousands, of pages on the Internet about C++ simplifying the search for help or answers concerning the ns-2 code.

OTcl

Object Tcl (OTcl) is object-oriented version of the command and syntax driven programming language Tool Command Language (Tcl). This is the second of the two programming languages that NS-2 uses. The front-end interpreter in NS-2 is OTcl which link the script type language of Tcl to the C++ backbone of NS-2. Together these two different languages create a script controlled C++ environment. This helps when creating a simulation, simply writing a script that will be carried out when running the simulation.

These scripts will be the formula for a simulation and is needed for setting the specifications of the simulation itself. Without a script properly defining a network topology as well as the data-rows, both type and location, nothing will happen.

NODES

A node is exactly what it sounds like, a node in the network. A node can be either an end connection or an intermediate point in the network. All agents and links must be connected to a node to work. There are also different kinds of nodes based on the kind of network that is to be simulated.

The main types are node and mobile node, where node is used in most wired networks and the mobile node for wireless networks. There are several different commands for setting the node protocols to be used, for instance what kind of routing is to be used or if there is a desire to specify a route that differs from the shortest one. Most of the commands for node and mobile node can be easily found in the ns documentation. Nodes and the closely connected link creating commands, like simplex link and duplex link, could be considered to simulate the behaviour of both the Link Layer.

AGENTS

An agent is the collective name for most of the protocols you can find in the transport layer. In the ns-2 documentation agents are defined as the endpoints where packets are created and consumed. All the agents defined in ns-2, like tcp, udp etc., are all connected to their parent class, simply called Agent. This is where their general behavior is set and the offspring classes are mostly based on some alterations to the inherent functions in the parent class. The modified functions will overwrite the old and thereby change the performance in order to simulate the wanted protocol.

APPLICATIONS:

The applications in ns-2 are related to the Application Layer in the TCP/IP suite. The hierarchy here works in the similar way as in the agent's case. To simulate some of the most important higher functions in network communication, the ns-2 applications are used. Since the purpose of ns-2 is not to simulate software, the applications only represent some different aspects of the higher functions.

Only a few of the higher layer protocols has been implemented, since some are quite similar when it comes to using the lower functions of the TCP/IP stack. For instance there is no use adding both a SMTP and a HTTP application since they both use TCP to transfer small amounts of data in a similar way.

The only applications incorporated in the release version of ns-2 are a number of different traffic generators for use with UDP and telnet and FTP for using TCP. All the applications are script controlled and when concerning the traffic generators, you set the interval and packet-size of the traffic. FTP can be requested to send a data packet whenever the user wants to, or to start a transfer of a file of arbitrary

size. If starting an FTP transmission and not setting a file-size the transmission will go on until someone calls a stop.

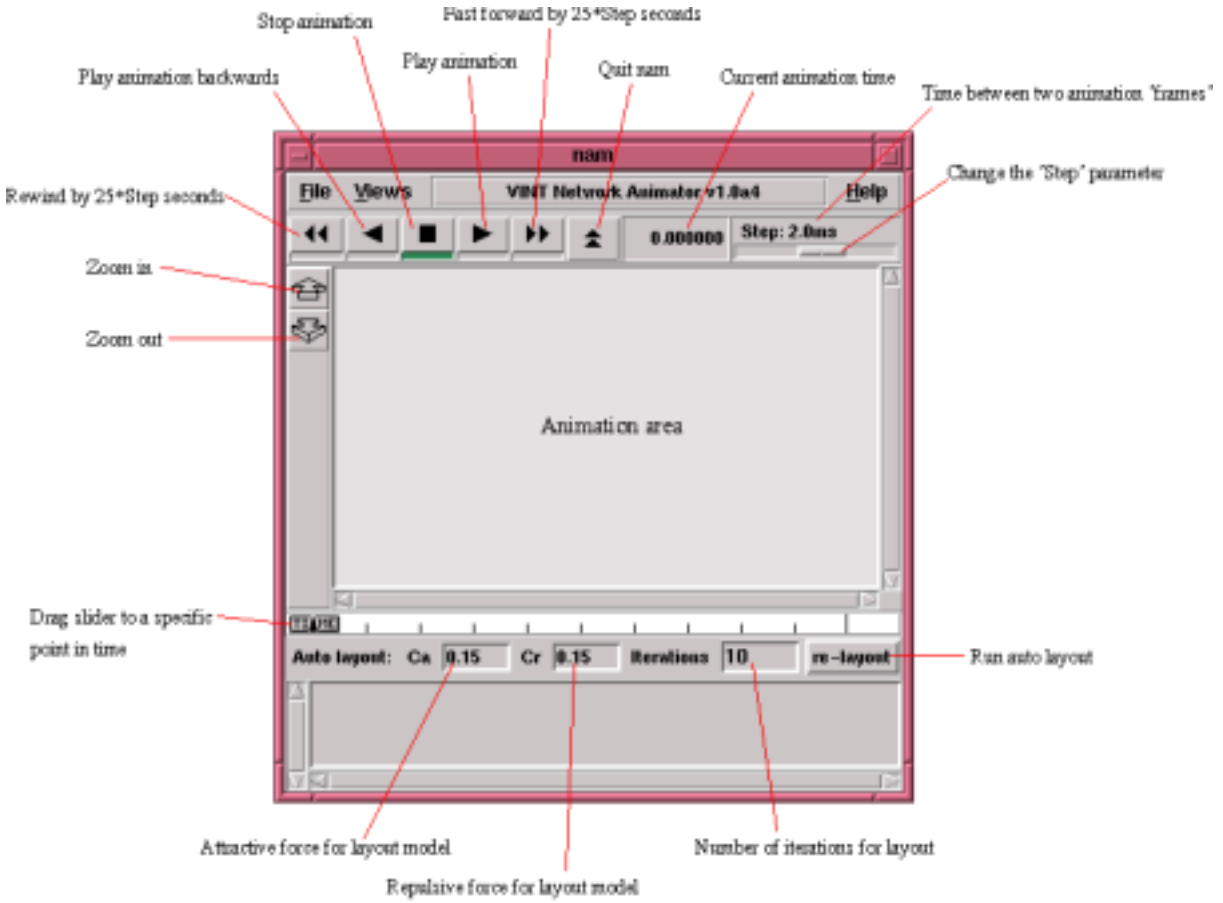
NAM

The Network Animator NAM is a graphic tool to use with ns-2. It requires a nam-tracefile recorded during the simulation and will then show a visual representation of the simulation.

This will give the user the possibility to view the traffic packet by packet as they move along the different links in the network. NAM offers the possibility of tracing a single packet during its travel and the possibility to move the nodes around for a user to draw up his network topology according to his own wishes.

Since the simulation has already been performed there is no possibility for the user to change the links or any other aspect of the simulation except the representation.

The existence of an Xserver allows NAM to be able to open a graphical window. Therefore if NAM is to work, there must be a version of X-server running.



RESULT:

The study of Network Simulator 2 was done successfully and the required data was studied.

`

EX.NO :6(b)	SIMULATION OF CONGESTION CONTROL ALGORITHM
DATE:	

AIM:

To simulate a link failure and observe the congestion control algorithm using NS2.

ALGORITHM:

1. Create a simulationobject
2. Set routing protocol torouting
3. Trace packets and all links onto NAM trace and to tracefile
4. Create rightnodes
5. Describe their layout topology asoctagon
6. Add a sink agent tonode
7. Connect source andsink.

PROGRAM:

```
set ns [new Simulator] set
nr [open thro_red.tr w]
$ns trace-all $nr setnf
[open thro.nam w]
$ns namtrace-all $nfproc finish { }{ globalns nrnf
    $nsflush-trace close $nf
    close$nr
    execnamthro.nam&
    exit 0
     }
setn0 [$ns node] set n1 [$ns node] set n2 [$ns node] set
n3 [$ns node] set n4 [$ns node] set n5 [$ns node] set n6
[$ns node] set n7 [$ns node]
$ns duplex-link $n0 $n3 1Mb 10msRED

$ns duplex-link $n1 $n3 1Mb 10msRED
```

```
$ns duplex-link $n2 $n3 1Mb 10msRED
$ns duplex-link $n3 $n4 1Mb 10msRED
$ns duplex-link $n4 $n5 1Mb 10msRED
$ns duplex-link $n4 $n6 1Mb 10msRED
$ns duplex-link $n4 $n7 1Mb 10msRED $ns
   duplex-link-op$n0      $n3      orient
right-up
$ns duplex-link-op $n3 $n4 orient middle $ns duplex-
link-op $n2 $n3 orient right-down $ns duplex-link-op
$n4 $n5 orient right-up $ns duplex-link-op $n4 $n7
orient right-down $ns duplex-link-op $n1 $n3 orient
right $ns duplex-link-op $n6 $n4 orient  left set udp0
[newAgent/UDP]
$ns attach-agent $n2$udp0 set cbr0
[new Application/Traffic/CBR]

$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent
$udp0 set null0 [new
Agent/Null]
$ns attach-agent $n5$null0
$ns connect $udp0
$null0 set udp1
[newAgent/UDP]
$ns attach-agent $n1$udp1

set cbr1 [new
Application/Traffic/CBR] $cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
```

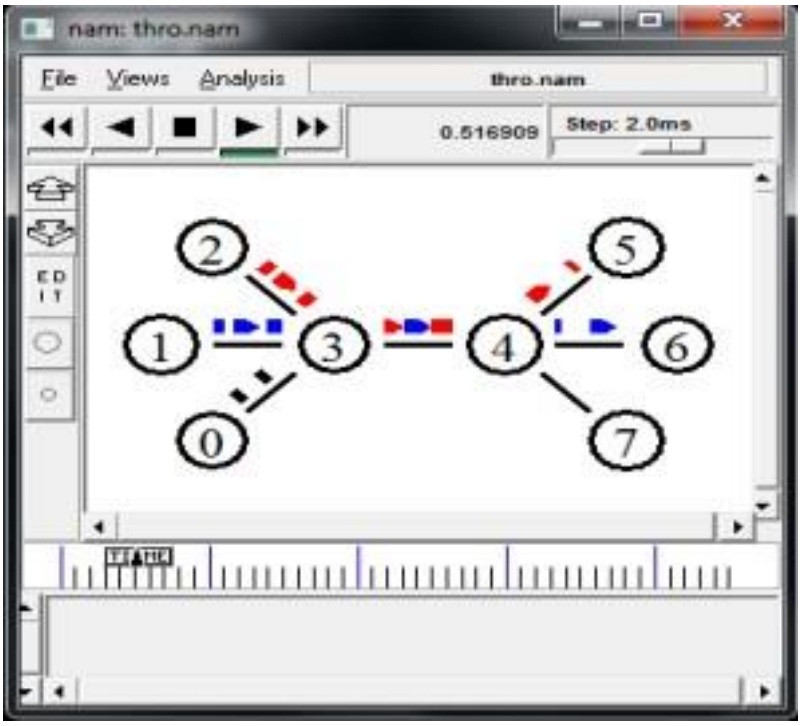
```
$cbr1 attach-agent
$udp1 set null0 [new
Agent/Null]
$ns attach-agent $n6$null0
$ns connect $udp1
$null0 set udp2
[newAgent/UDP]
$ns attach-agent $n0$udp2
set cbr2 [new
Application/Traffic/CBR] $cbr2 set
packet size_ 500
$cbr2 set interval_ 0.005
$cbr2 attach-agent
$udp2 set null0 [new
Agent/Null]
$ns attach-agent $n7 $null0
$ns connect $udp2 $null0
$udp0 set fid_ 1
$udp1 set fid_2
$udp2 set fid_3
$ns color 1Red
$ns color 2 Green
$ns color 2 Blue
 $ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1start" $ns at 0.5 "$cbr2start" $ns at 4.0 "$cbr2stop" $ns at 4.2 "$cbr1stop"
```

NAME:GOWTHAM PRABHAKARAN.R
REGNO:110819104009

$ns at 4.5 "$cbr0stop" $ns at 5.0 "finish"

$ns run

OUTPUT:



RESULT:

Thus the congestion control algorithm is simulated by using NS2 is executed successfully.

EX.NO :7	**SIMULATION OF DISTACNCE VECTOR ALGORITHM.**
DATE:	

AIM:

To simulate and observe traffic route of a network using distance vector routing protocol.

ALGORITHM:

1. Create a simulator object
2. Set routing protocol to Distance vectorrouting
3. Trace packets on all links on to NAM trace and text tracefile.
4. Define finish procedure to close files, flash tracing and runNAM
5. Create 5nodes
6. Specify the link characteristics between thenodes
7. Describer their layout topology as aoctagon
8. Add UDP agent for noden0
9. Create CBR traffic on the top of UDP and set trafficparameters
10. Add NULL agent to noden3
11. Connect source andsink
12. Schedule asfollows
    - Start traffic flow at1.0
    - Down the link n1 – n2 at15.0
    - Up the link n1 – n2 at25.0
    - Call finish procedure at35.0
13. Start thescheduler
14. Observe the traffic route when the link is up anddown
15. View the simulated events and trace file analyzeit
16. Stop.

PROGRAM:

#Distance vector routing protocol –

distvect.tcl #Create a simulator object

set ns [new Simulator]

#Use distance vector routing

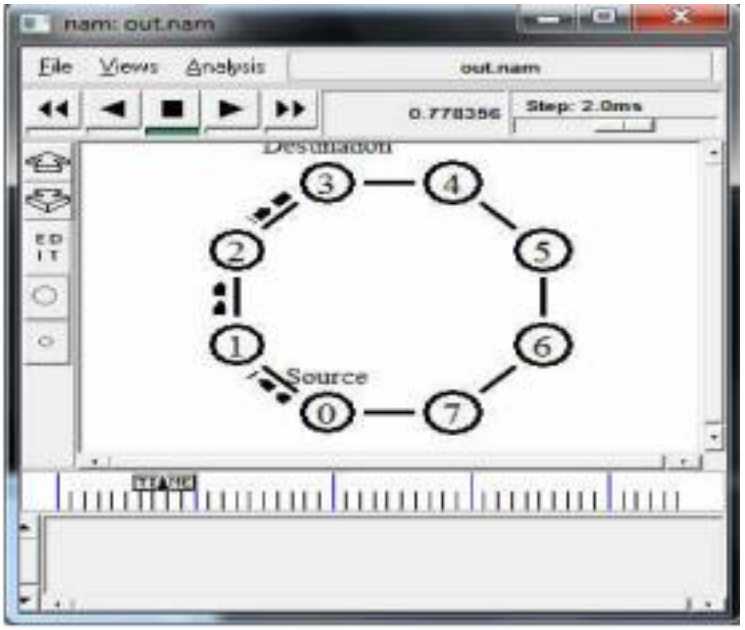
$ns rtproto DV

```
#Open the nam trace

file set nf [open

out.nam w]

$ns namtrace all $nf #

Open tracefile setnt

[open trace.tr w]

$ns trace-all $nt

#Define 'finish'

procedure proc finish

{} { global ns nf

$ns flush-trace #Close the trace file close

$nf #Executenam on the trace file exec

nam -a out.nam& exit 0

 }

# Create 8 nodes set n1 [$ns node] set n2 [$ns

node] set n3 [$ns node] set n4 [$ns node] set n5

[$ns node] set n6 [$ns node] set n7 [$ns node]

set n8 [$ns node]

# Specify link characterestics

$ns duplex-link $n1 $n2 1Mb 10msDropTail

$ns duplex-link $n2 $n3 1Mb 10msDropTail

$ns duplex-link $n3 $n4 1Mb 10msDropTail

$ns duplex-link $n4 $n5 1Mb 10msDropTail

$ns duplex-link $n5 $n6 1Mb 10msDropTail



$ns duplex-link $n6 $n7 1Mb 10msDropTail

$ns duplex-link $n7 $n8 1Mb 10msDropTail
```

```
$ns duplex-link $n8 $n1 1Mb 10ms
DropTail # specify layout as a octagon

$ns duplex-link-op $n1 $n2 orient left-up

$ns duplex-link-op $n2 $n3 orient up

$ns duplex-link-op $n3 $n4 orient right-up

$ns duplex-link-op $n4 $n5 orient right

$ns duplex-link-op $n5 $n6 orient right-down

$ns duplex-link-op $n6 $n7 orient down

$ns duplex-link-op $n7 $n8 orient left-down

$ns duplex-link-op $n8 $n1 orient left

#Create a UDP agent and attach it to

node n1 set udp0 [newAgent/UDP]

$ns attach-agent $n1$udp0

#Create a CBR traffic source and attach it to

udp0 set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 500

$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node

n4 set null0 [new Agent/Null]

$ns attach-agent $n4 $null0

#Connect the traffic source with the traffic sink

$ns connect $udp0 $null0

#Schedule events for the CBR agent and the network dynamics

$ns at 0.0 "$n1 label Source"



$ns at 0.0 "$n4 label Destination"

$ns at 0.5 "$cbr0 start"
```

```
$ns rtmodel-at 1.0 down $n3 $n4
$ns rtmodel-at 2.0 up $n3 $n4

$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time

$ns at 5.0 "finish"

#Run the

simulation

$ns run
```

OUTPUT:



RESULT:

Thus the Distance Vector Routing algorithmwas simulated by using NS2 is executed successfully.

EX.NO :8	**SIMULATION OF LINK STATE ROUTING PROTOCOL**
DATE:	

AIM:

To develop a simulation program and study the link state routing algorithm using Network Simulator.

ALGORITHM:

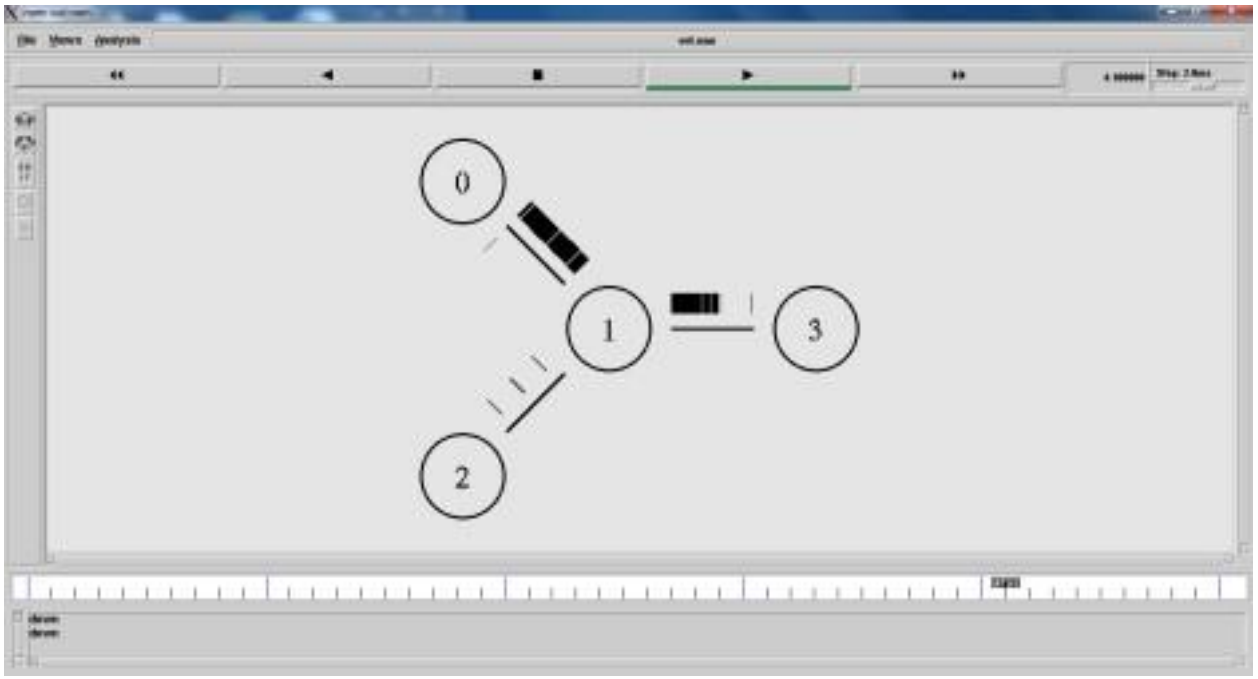
1. Create a simulator object
2. Define different colors for different data flows

3. Open anam trace file anddefine finish procedure then close the trace file,and execute nam on tracefile.
4. Createn number of nodes using for loop
5. Create duplex links between the nodes .
6. Setup UDPConnection between n(0)and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. ApplyCBRTrafficoverbothUDPconnections
9. Choose Link state routing protocol to transmit data from sender to receiver.10. Scheduleeventsand run the program **Program1:**

```
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
$tr
exec namout.nam&
exit 0
} set n0 [$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
```

OUTPUT:



PROGRAM2:

```
#Create a simulator

object set ns [new

Simulator]

#Define different colors for data flows(for NAM)

$ns color 1 Blue

$ns color 2 Red

#Open the NAM trace file

setnf [open out.nam w]

$ns namtrace-all $nf

#Define a 'finish'

procedure proc finish {

} { global

ns nf

$ns flush-trace #close the

NAM trace file


close $nf
```

```
#Execute NAM on the trace file

exec namout.nam&

exit 0

}

for {set i 0} {$i < 5} {incr i 1}

{ set n($i) [$ns node]} for {set i

0} {$i < 4} {incr i} {

$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail}

$ns duplex-link $n(0) $n(1) 1Mb 10msDropTail $ns duplex-

link $n(1) $n(2) 1Mb 10msDropTail $ns duplex-link

$n(2) $n(3) 1Mb 10msDropTail $ns duplex-link $n(3) $n(4)

1Mb 10msDropTail $ns duplex-link $n(4) $n(1) 1Mb

10msDropTail set udp0 [new Agent/UDP]

$ns attach-agent $n(0) $udp0 set cbr0
[new Application/Traffic/CBR]

$cbr0 set packetSize_ 500

$cbr0 set interval_ 0.005

$cbr0 attach-agent

$udp0 set null0 [new

Agent/Null]

$ns attach-agent $n(3) $null0

$ns connect $udp0 $null0

$ns rtproto DV

$ns rtmodel-at 15.0 down $n(1) $n(2)

$ns rtmodel-at 25.0 up $n(1) $n(2)


$ns color 1 Green

$ns at 1.0 "$cbr0 start"
```

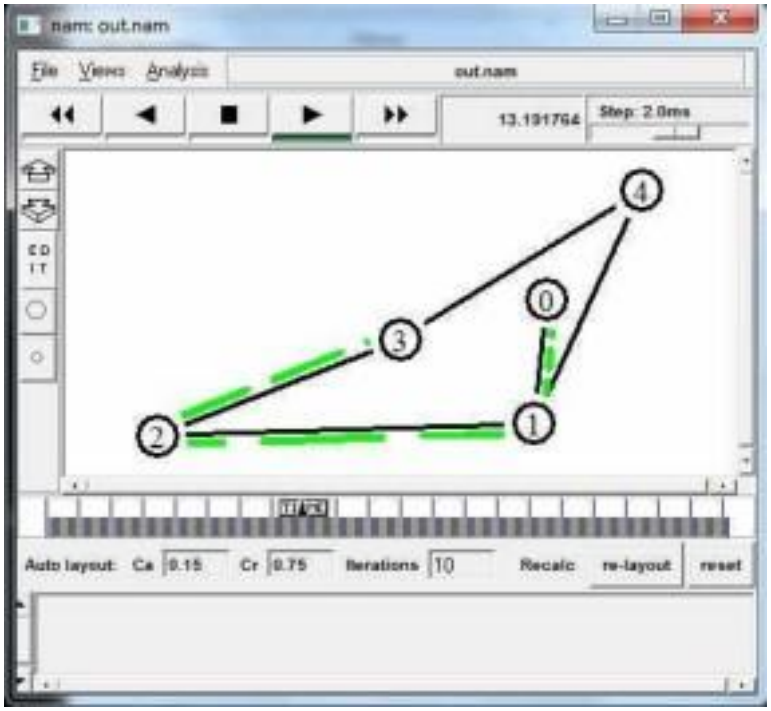
$ns at 35 "finish"

#Run the

simulation

$ns run

OUTPUT:



RESULT:

Thus the program of link state routing protocol program is successfully executed.

EX.NO :9	
DATE:	

AIM:

To write a Program for performance evalution of routing protocols using simulation tool.

PROGRAM:

A 3-node example for ad-hoc simulation with AODV

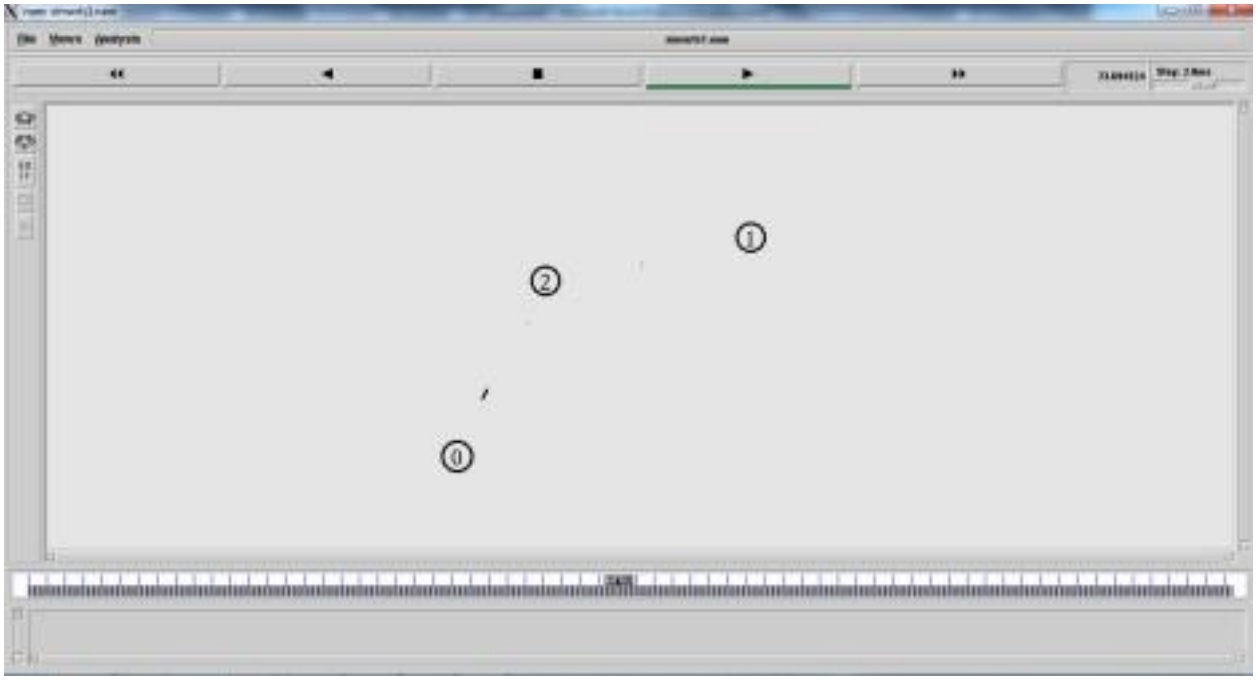
Define options setval(chan) Channel/WirelessChannel ;# channel
type setval(prop) Propagation/TwoRayGround ;# radio-propagation
model

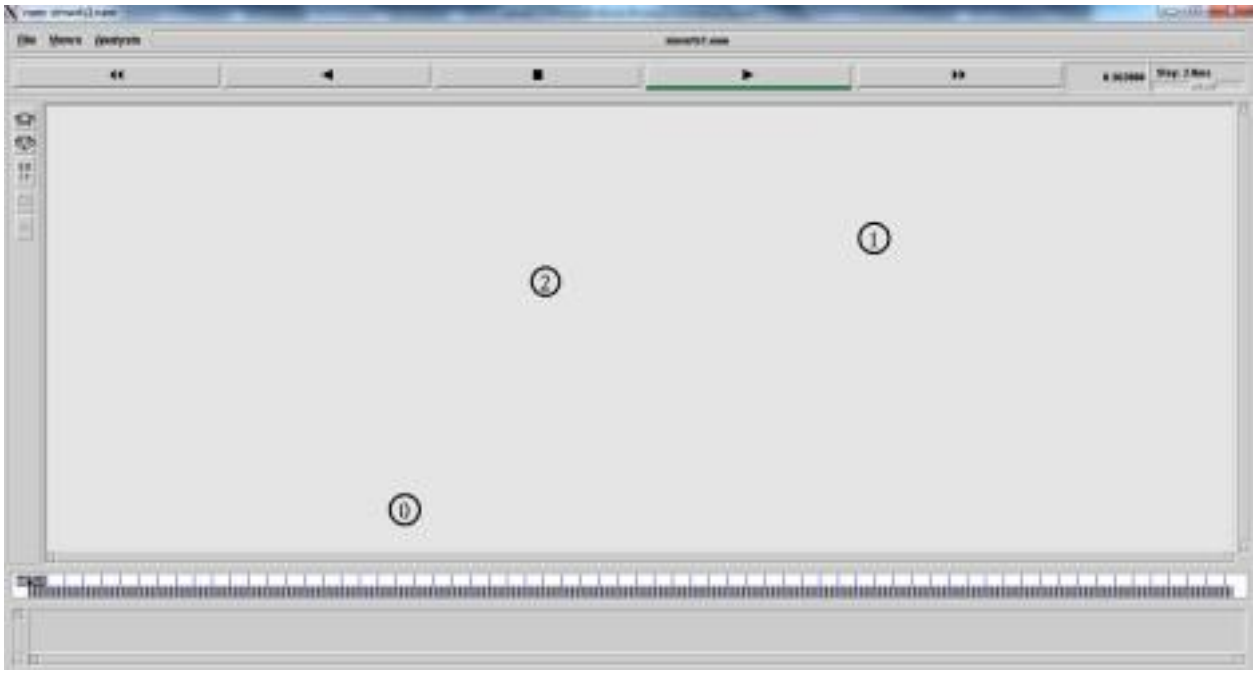
```
setval(netif) Phy/WirelessPhy ;# network interface
type setval(mac) Mac/802_11 ;# MAC type setval(ifq)
Queue/DropTail ;# interface queue type
setval(ll) LL ;# link layer type
setval(ant) Antenna/OmniAntenna ;# antenna model
setval(ifqlen) 50 ;# max packet in ifq
setval(nn) 3 ;# number of mobilenodes

setval(rp) AODV ;# routing protocol setval(x) 500 ;# X dimension of
topography setval(y) 400 ;# Y dimension of topography  setval(stop) 150 ;#
time of simulation end

set ns [new Simulator] settracefd
[open simple-dsdv.tr w] set
windowVsTime2 [open win.tr w]
setnamtrace [open simwrls1.nam w]

$ns trace-all $tracefd
$ns use-newtrace
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object settopo

[new Topography]

$topoload_flatgrid $val(x) $val(y)

create-god $val(nn)

#
# Creatennmobilenodes [$val(nn)] and attach them to the channel.
#

# configure the nodes
 $ns node-config -adhocRouting $val(rp) \
                                 -llType $val(ll) \
                                 -macType $val(mac) \
                                 -ifqType $val(ifq) \
                                 -ifqLen $val(ifqlen) \
                                 -antType $val(ant) \
                                 -propType $val(prop) \
                                 -phyType $val(netif) \
                                 -channelType $val(chan) \
                                 -topoInstance $topo \
                                 -agentTrace ON \
                                 -routerTrace ON \
                                 -macTrace OFF \
                                 -movementTrace ON

        for {set i 0} {$i < $val(nn) } { incr i } {
                set node_($i) [$ns node]
```

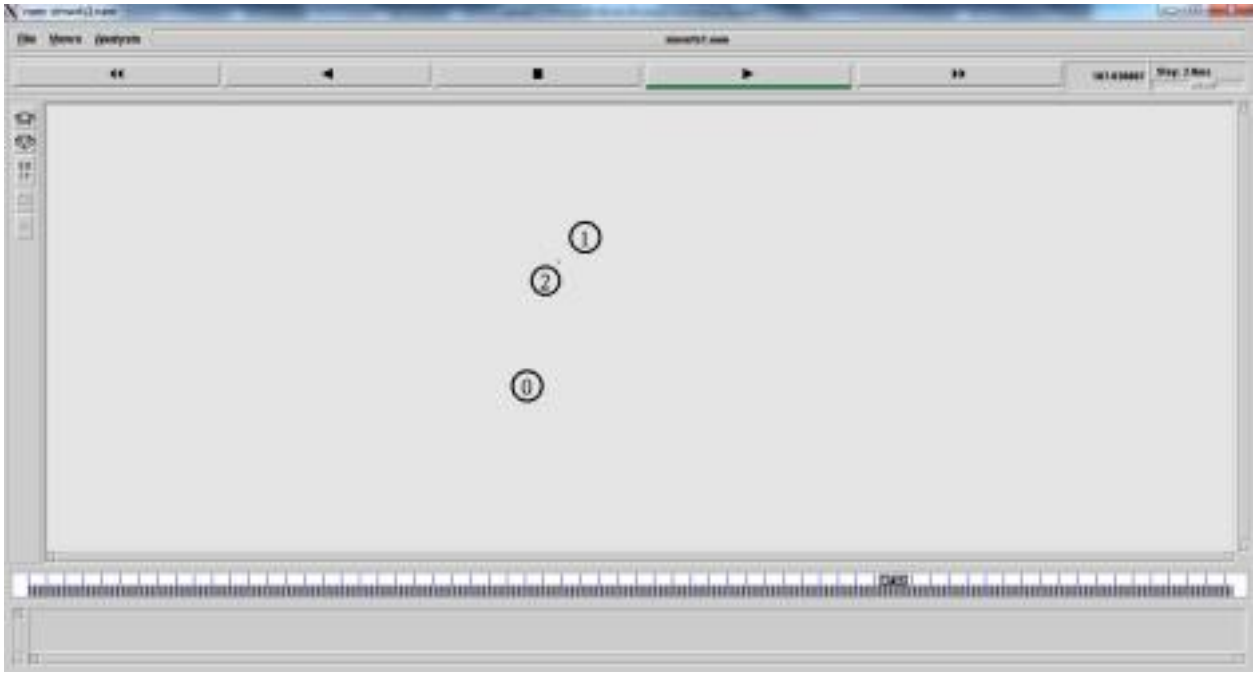
```
}

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0

# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"

# Set a TCP connection between node_(0) and node_(1)
settcp [new Agent/TCP/Newreno]
$tcp set class_ 2 set sink [new
Agent/TCPSink] $ns attach-
agent $node_(0) $tcp
$ns attach-agent $node_(1)
$sink $ns connect $tcp $sink set
ftp [new Application/FTP] $ftp
attach-agent $tcp
$ns at 10.0 "$ftp start"



# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {


# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
```

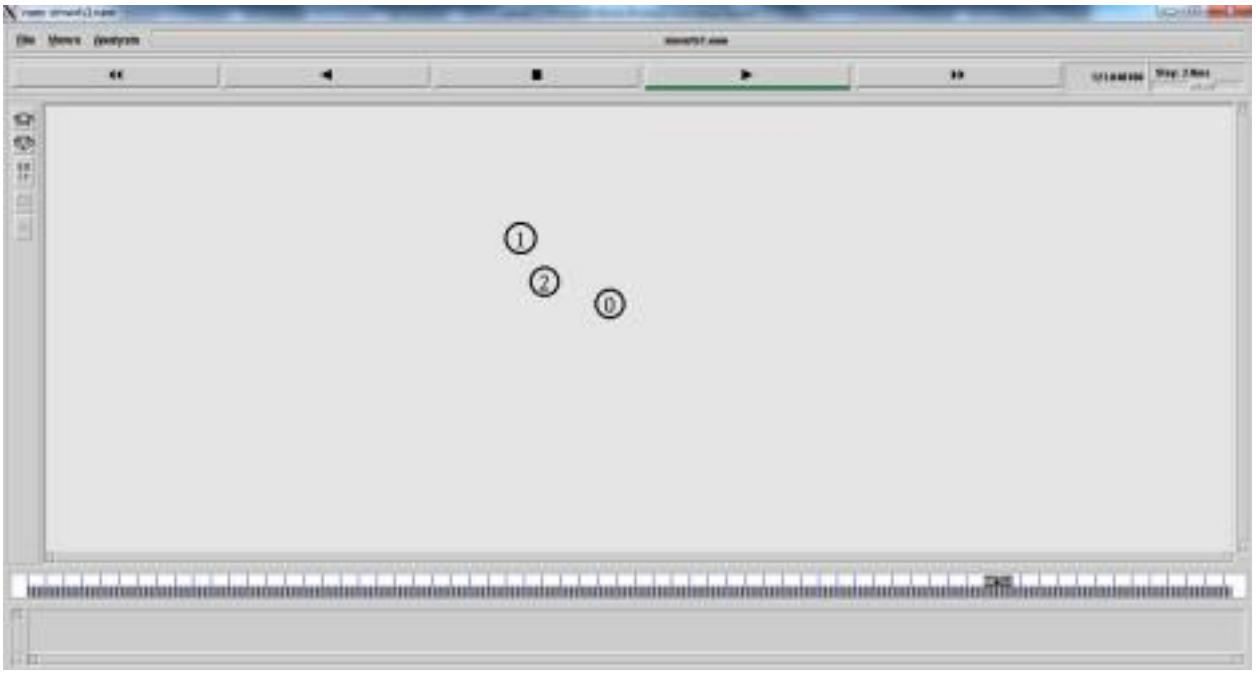
```
$ns at 150.01 "puts \"end simulation\" ; $ns
halt" proc stop {} { global ns tracefdnamtrace
 $ns flush-trace close
$tracefd close $namtrace
execnam simwrls1.nam &
}

$ns run
```

OUTPUT:




1
2
0


1
2
0



RESULT:

Thus the Program for performance evalution of routing protocols using simulation tool is executed successfully.

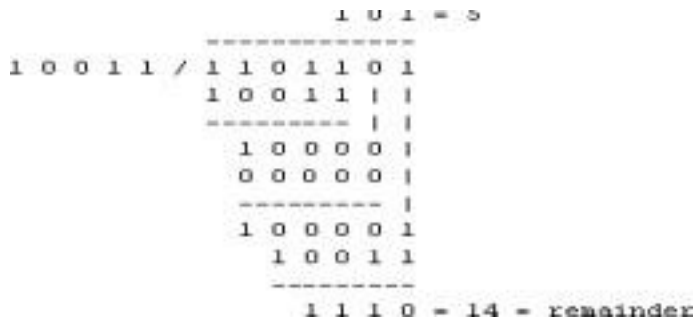
EX.NO :10	**SIMULATION OF ERROR CORRECTION CODE(CRC)**
DATE:	

AIM:

Write a Program for ERROR detecting code using CRC-CCITT (16bit).

THEORY:

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The **CRC** calculation or *cyclic redundancy check* was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips areused.



The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation—by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you wereyoung. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bitextra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "*Lammert*"—and the result can be calculated with 56 binary

comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an *exclusive or* operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always thesame:

• The message bits are appended with *c* zero bits; this *augmented message* is the dividend • A predetermined *c+1*-bit binary sequence, called the *generator polynomial*, is the divisor • The checksum is the *c*-bit remainder that results from the division operation Table 1 lists some of the most commonly used generator polynomials for 16- and32-bit

CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

Table 1: International Standard CRC Polynomials

	CRC-CCITT	**CRC-16**	**CRC-32**
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	10000010011000001000111011011011

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x)

This is used to generate a CRC = C(x) to be appended to M(x). Note this G(x) is prime.

Steps:

1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros toM(x).

2. Divide the result by G(x). The remainder =C(x).

Special case: This won't work if bitstring =all zeros. We don't allow such an M(x).But M(x) bitstring = 1 will work, for example. Can divide 1101 into 1000.

3. If: x div y gives remainder c that means: x = n y + c Hence (x-c) = ny

(x-c) div y gives remainder 0 Here (x-c) = (x+c)

Hence (x+c) div y gives remainder 0

4. Transmit: T(x) = M(x) +C(x)

5. Receiver end: Receive T(x). Divide by G(x), should have remainder0.

Note if G(x) has order n - highest power is xn, then G(x) will cover (n+1) bits and the *remainder* will cover n bits.

i.e. Add n bits (Zeros) to message.

Some CRC polynomials that are actually used

Some CRC polynomials

- CRC-8: x8+x2+x+1 ○ Used in: 802.16

(along with error *correction*).

- CRC-CCITT: x16+x12+x5+1 ○ Used

in: HDLC, SDLC, PPP default

- IBM-CRC-16 (ANSI): x16+x15+x2+1

- 802.3:

x32+x26+x23+x22 +x16+x12+x11+x10 +x8+x7+x5+x4+x2+x+1

○ Used in: Ethernet, PPP rootion

PROGRAM:

Source Code:

```
importjava.util.*;
classcrc
{ void div(int a[],int k)
{ intgp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};
int count=0; for(int i=0;i<k;i++)
{
if(a[i]==gp[0])
{
for(int j=i;j<17+i;j++)
{
a[j]=a[j]^gp[count++];
}


count=0;
}
}
}
public static void main(String args[])
{
int a[]=new int[100]; int b[]=new int[100]; intlen,k; crcob=new
crc();
System.out.println("Enter the length of Data Frame:"); Scanner sc=new Scanner(System.in); len=sc.nextInt();
int flag=0;
System.out.println("Enter the Message:"); for(int i=0;i<len;i++)
{ a[i]=sc.nextInt();
}
for(int i=0;i<16;i++)
{ a[len++]=0; }
k=len-16; for(int
i=0;i<len;i++)
{ b[i]=a[i];
}
```

```
ob.div(a,k);
for(int i=0;i<len;i++) a[i]=a[i]^b[i];
System.out.println("Data to be transmitted: "); for(int i=0;i<len;i++)
{
System.out.print(a[i]+" ");
}
System.out.println();
System.out.println("Enter the Reveived Data: "); for(int i=0;i<len;i++)
{
a[i]=sc.nextInt();
}
ob.div(a, k);

for(int i=0;i<len;i++)
{
if(a[i]!=0)
{
System.out.println("ERROR in Received data"); return;
}
System.out.println("no error");
}

}
}
```

Output:

Enter the length of Data Frame: 4 Enter the Message: 1 0 1 1
Data to be transmitted: 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Received Data: 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 ERROR in Received
Data **********

RESULT:

Thus the Program for ERROR detecting code using CRC-CCITT (16bit) is executed successfully.