CS 2307 – NETWORKS LAB

LAB MANUAL

Prepared by

Mr.S.PRAVEEN KUMAR, M.E,MBA.,

Assistant Professor

Department of Computer Science and Engineering

EGSPEC, Nagapattinam

PREFACE

CS2307- Networks laboratory manual is indented to provide a basic knowledge of networking. Networking is developing technology becoming a new emerging trend and developing a variety of programmers and users. This manual will be available in electronic form from College’s official and faculty’s individual websites and for the betterment of students.

E.G.S.PILLAY ENGINEERING COLLEGE – NAGAPATTINAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INFORMATION ON COURSE	**INFORMATION ON INSTRUCTOR**
Course : **CS2307 – NETWORKS LAB** L T P C 0 0 3 2 Course Designed By: Anna University, Chennai. College Name : E.G.S.Pillay Engineering College, Nagapattinam	Course Incharge: **S. PRAVEEN KUMAR, M.E.,** Designation: Assistant Professor Department : Computer Science & Engg Office Hours **:** By appointment Mobile : 9786465881 Email: asv.praveen@gmail.com

REQUIRED TEXT AND SUPPLIES	**Text Book:** 1. James F. Kuross, Keith W. Ross, "Computer Networking, A Top-Down Approach Featuring the Internet", Third Edition, Addison Wesley, 2004. 2. W. Richard Stevens, "Unix Network Programming Vol-I", Second Edition, Pearson Education, 1998. **Supplies:** EGSP Engineering College/IT/CS2307-Networks Lab manual.
OPTIONAL TEXT	1. William Stallings, "Data and Computer Communication", Sixth Edition, Pearson Education, 2000 2. Ferouz.A.Ferouzan, "Computer Communication and Networking", Fifth Edition, Pearson Education, 2001
REFERENCES	*http:// www.homeandlearn.co.uk* *http:// www.w3schools.com*
SOFTWARE	**Operating System :** Linux Distribution **(**A single server could be loaded with Linux and connected from the individual) – Ubuntu / OpenSUSE / Fedora / Red Hat / Debian / Mint OS **Programming Languages:** JDK1.6 and TurboC
PRE LAB REQUISITES	**1.** GE2155 - Computer Practice Laboratory-II **2.** IT2305 - Java Programming Lab
DATES	Ref. Academic Calendar

CS2307– NETWORKS LAB L T P C

0 0 3 2

Syllabus

1. Programs using TCP Sockets (like date and time server & client, echo server & client, etc.)
2. Programs using UDP Sockets (like simple DNS)
3. Programs using Raw sockets (like packet capturing and filtering)
4. Programs using RPC
5. Simulation of sliding window protocols
6. Experiments using simulators (like OPNET)
7. Performance comparison of MAC protocols
8. Performance comparison of Routing protocols
9. Study of TCP/UDP performance

TOTAL: 60 PERIODS

Requirement for a batch of 30 students

SL.No	Description of Equipment	Quantity required	Quantity available	Deficiency %
01	SOFTWARE ➢ C++ Compiler ➢ J2SDK (freeware) ➢ Linux ➢ NS2/Glomosim/OPNET (Freeware)	30	Available	NIL
02	Hardware ➢ PCs	30 Nos	Available	NIL

Aim

The aim of the CS2307 - Networks laboratory is to give students a good understanding of basic concepts of computer networks and the need of developing networks based applications.

Instructional Objective:

The purpose of this course is to be able to explain, configure, verify, and troubleshoot complex computer networks problem.

Instructional Outcomes:

After completing the course, students will be able to:

1. Explain computer networking concepts to both technical peers and non-technical management.
2. Configure network routers and switches so that both LAN and WAN traffic successfully traverses the network.
3. Differentiate true statements from false statements as pertains to computer networking as verified by passing industry standard examinations.

Gap Analysis

To fill the gap in the syllabus content, the students have to study the following additional topics

S.NO	TOPIC	ACTION
1	Comparison study various Network Protocols	Practical Class
2	Study of NS2, Glomosim and Opnet	Practical Class

Internal marks Assessment Method

Observation	Attendance	Record	Total
10	5	5	20

STAFF INCHARGE **HOD** **PRINCIPAL**

List of Experiments

1. Programs using TCP Sockets (like date and time server & client, echo server & client, etc.)
    i. Program Using TCP Sockets Date and Time Server
    ii. Implementation of Client-Server Communication Using TCP.
    iii. Implementation of TCP/IP ECHO

2. Programs using UDP Sockets (like simple DNS)
    i. Program using UDP Socket UDP Chat Server/Client
    ii. DNS Server to Resolve a given Host Name
    iii. UDP DNS server/client

3. Programs using Raw sockets (like packet capturing and filtering)
    i. Packet Capturing and Filtering

4. Programs using RPC
    i. Client – Server Communication using RPC
    ii. Arithmetic Calculator using RPC-RMI

5. Simulation of sliding window protocols
6. Experiments using simulators
    i. Simple Topology Creation using NS - 2
    ii. User Datagram Protocol using NS -2
    iii. Transmission Control Protocol using NS - 2

7. Performance comparison of MAC protocols

8. Performance comparison of Routing protocols

9. Study of TCP/UDP performance
    i. Case Study 1: Study of UDP Performance
    ii. Case Study 2: Study of TCP Performance
    iii. Case Study 3: Study of Performance Comparison of TCP and UDP using NS – 2

Note : Appendix Enclosed for getting fundamental knowledge in Network Programming using Java and Network Simulator - 2

1. Programs using TCP Sockets (like date and time server & client, echo server & client, etc.)

Ex.No: 1. a PROGRAM USING TCP SOCKETS DATE AND TIME SERVER

AIM:

To implement date and time display from client to server using TCP Sockets

DESCRIPTION:

TCP Server gets the system date and time and opens the server socket to read the client details. Client send its address to the server. Then client receives the date and time from server to display. TCP socket server client connection is opened for communication. After the date time is displayed the server client connection is closed with its respective streams to be closed.

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:

//TCP Date Server--tcpdateserver.java

```
import java.net.*;
import java.io.*;
import java.util.*;
class tcpdateserver
{
public static void main(String arg[])
{
ServerSocket ss = null;
Socket cs;
PrintStream ps;
BufferedReader dis;
String inet;
try
{ ss = new ServerSocket(4444);
System.out.println("Press Ctrl+C to quit");
while(true)
{
cs = ss.accept();
ps = new PrintStream(cs.getOutputStream());
Date d = new Date();
ps.println(d);
dis = new BufferedReader(new
InputStreamReader(cs.getInputStream()));
inet = dis.readLine();
System.out.println("Client System/IP address is :"+ inet);
ps.close();
dis.close();
}
}
catch(IOException e)
{
System.out.println("The exception is :" + e);
}
}
}
```

// TCP Date Client--tcpdateclient.java

```
import java.net.*;
import java.io.*;
class tcpdateclient
{
public static void main (String args[])
{
Socket soc;
BufferedReader dis;
String sdate;
PrintStream ps;
try
{
InetAddress ia = InetAddress.getLocalHost();
if (args.length == 0)
soc = new Socket(InetAddress.getLocalHost(),4444);
else
soc = new Socket(InetAddress.getByName(args[0]),4444);
dis = new BufferedReader(new
InputStreamReader(soc.getInputStream()));
sdate=dis.readLine();
System.out.println("The date/time on server is : " +sdate);
ps = new PrintStream(soc.getOutputStream());
ps.println(ia);
ps.close();  catch(IOException e)
{
System.out.println("THE EXCEPTION is :" + e);
} }  }
```

OUTPUT

Server:

```
        $ javac tcpdateserver.java
        $ java tcpdateserver
        Press Ctrl+C to quit
        Client System/IP address is : localhost.localdomain/127.0.0.1
Client System/IP address is : localhost.localdomain/127.0.0.1
```

Client:

```
        $ javac tcpdateclient.java
        $ java tcpdateclient
        The date/time on server is: Wed Jul 06 07:12:03 GMT 2011
```

Every time when a client connects to the server, server's date/time will be returned to the client for synchronization.

RESULT:

Thus the program for implementing to display date and time from client to server using TCP Sockets was executed successfully and output verified using various samples.

Ex.No: 1.b Implementation of Client-Server Communication Using TCP

AIM:

To implement a chat server and client in java using TCP sockets.

DESCRIPTION:

TCP Clients sends request to server and server will receives the request and response with acknowledgement. Every time client communicates with server and receive response from it.

ALGORITHM:

Server

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Repeat steps 3-4 until the client sends "end"
6. Close all streams
7. Close the server and client socket
8. Stop

Client

1. Create a client socket and connect it to the server's port number
2. Get a message from user and send it to server
3. Read server's response and display it
4. Repeat steps 2-3 until chat is terminated with "end" message
5. Close all input/output streams
6. Close the client socket
7. Stop

PROGRAM:

```
//Server.java
import java.io.*;
import java.net.*;
class Server {
  public static void main(String args[]) {
    String data = "Networks Lab";
    try {
      ServerSocket srvr = new ServerSocket(1234);
      Socket skt = srvr.accept();
      System.out.print("Server has connected!\n");
      PrintWriter out = new PrintWriter(skt.getOutputStream(), true);
      System.out.print("Sending string: '" + data + "'\n");
      out.print(data);
      out.close();
      skt.close();
      srvr.close();
    }
    catch(Exception e) {
      System.out.print("Whoops! It didn't work!\n");
    }
  }
}

//Client.java
import java.io.*;
import java.net.*;
class Client {
  public static void main(String args[]) {
    try {
      Socket skt = new Socket("localhost", 1234);
      BufferedReader in = new BufferedReader(new
        InputStreamReader(skt.getInputStream()));
      System.out.print("Received string: '");
      while (!in.ready()) {}
      System.out.println(in.readLine());
      System.out.print("'\n");
      in.close();
    }
    catch(Exception e) {
      System.out.print("Whoops! It didn't work!\n");
    }}}
```

OUTPUT

Server:

```
$ javac Server.java
$ java Server
        Server started
        Client connected
```

Cilent

```
$ javac Client.java
$ java Client
```

RESULT

Thus both the client and server exchange data using TCP socket programming.

Ex.No: 1. c IMPLEMENTATION OF TCP/IP ECHO

AIM:

To implementation of echo client server using TCP/IP

DESCRIPTION:

TCP Server gets the message and opens the server socket to read the client details. Client send its address to the server. Then client receives the message from server to display.

ALGORITHM

Server

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

Client

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

PROGRAM:

// TCP Echo Server--tcpechoserver.java

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

//TCP Echo Client--tcpechoclient.java

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

OUTPUT

Server:

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

Client :

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

RESULT

Thus data from client to server is echoed back to the client to check reliability/noise level of the channel.

2. Programs using UDP Sockets

Ex.No: 2.a PROGRAM USING UDP SOCKET UDP CHAT SERVER/CLIENT

AIM:

To implement a chat server and client in java using UDP sockets.

DESCRIPTION:

UDP is a connectionless protocol and the socket is created for client and server to transfer the data. Socket connection is achieved using the port number. Domain Name System is the naming convention that divides the Internet into logical domains identified in Internet Protocol version 4 (IPv4) as a 32-bit portion of the total address.

ALGORITHM:

Server

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to client port.
3. Create a datagram packet to receive client message.
4. Wait for client's data and accept it.
5. Read Client's message.
6. Get data from user.
7. Create a datagram packet and send message through server port.
8. Repeat steps 3-7 until the client has something to send.
9. Close the server socket.
10. Stop.

Client

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to server port.
3. Get data from user.
4. Create a datagram packet and send data with server ip address and client port.
5. Create a datagram packet to receive server message.
6. Read server's response and display it.
7. Repeat steps 3-6 until there is some text to send.
8. Close the client socket.
9. Stop.

PROGRAM

// UDP Chat Server--udpchatserver.java

```
import java.io.*;
import java.net.*;
class udpchatserver
{
public static int clientport = 8040,serverport = 8050;
public static void main(String args[]) throws Exception
{
DatagramSocket SrvSoc = new DatagramSocket(clientport);
byte[] SData = new byte[1024];
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Server Ready");
while (true)
{
byte[] RData = new byte[1024];
DatagramPacket RPack = new DatagramPacket(RData,RData.length);
SrvSoc.receive(RPack);
String Text = new String(RPack.getData());
if (Text.trim().length() == 0)
break;
System.out.println("\nFrom Client <<< " + Text );
System.out.print("Msg to Cleint : " );
String srvmsg = br.readLine();
InetAddress IPAddr = RPack.getAddress();
SData = srvmsg.getBytes();
DatagramPacket SPack = new DatagramPacket(SData,SData.length,IPAddr,
serverport);
SrvSoc.send(SPack);
}
System.out.println("\nClient Quits\n");
SrvSoc.close();
}
}
```

// UDP Chat Client--udpchatclient.java

```
import java.io.*;
import java.net.*;
class udpchatclient
{
public static int clientport = 8040,serverport = 8050;
public static void main(String args[]) throws Exception
{
BufferedReader br = new BufferedReader(new InputStreamReader (System.in));
DatagramSocket CliSoc = new DatagramSocket(serverport);
InetAddress IPAddr;
String Text;
if (args.length == 0)
IPAddr = InetAddress.getLocalHost();
else
IPAddr = InetAddress.getByName(args[0]);
byte[] SData = new byte[1024];
System.out.println("Press Enter without text to quit");
while (true)
{
System.out.print("\nEnter text for server : ");
Text = br.readLine();
SData = Text.getBytes();
DatagramPacket SPack = new DatagramPacket(SData,SData.length, IPAddr,
clientport );
CliSoc.send(SPack);
if (Text.trim().length() == 0)
break;
byte[] RData = new byte[1024];
DatagramPacket RPack = new DatagramPacket(RData,RData.length);
CliSoc.receive(RPack);
String Echo = new String(RPack.getData()) ;
Echo = Echo.trim();
System.out.println("From Server <<< " + Echo);
}
CliSoc.close();
}
}
```

OUTPUT

Server

```
$ javac udpchatserver.java
$ java udpchatserver
Server Ready
From Client <<< are u the SERVER
Msg to Cleint : yes
From Client <<< what do u have to serve
Msg to Cleint : no eatables
Client Quits
```

Client

```
$ javac udpchatclient.java$ java udpchatclient
Press Enter without text to quit
Enter text for server : are u the SERVER
From Server <<< yes
Enter text for server : what do u have to serve
From Server <<< no eatables
Enter text for server : Ok
```

RESULT

Thus both the client and server exchange data using UDP sockets.

Ex.No:2.b DNS SERVER TO RESOLVE A GIVEN HOST NAME

AIM:

To develop a client that contacts a given DNS server to resolve a given hostname.

DESCRIPTION:

· Get the host name to be resolve using gethostname()
· Check the host name using nslookup
· Print the IP address, host name, Address length and Address type.
· List the addresses stored in lookup

ALGORITHM

```
Step 1. Find the host name by using gethostbyname()
Step 2. The host name is followed by the list of alias names
Step 3. Pointer points to the array of pointers to the individual address
Step 4. For each address call the inet_ntop() and print the returned string
```

PROGRAM

```
#include<stdio.h>
#include<netdb.h>
#include<arpa/inet.h>
#include<netinet/in.h>
```

```
int main(int argc,char**argv)
{
char h_name;
int h_type;
struct hostent *host;
struct in_addr h_addr;
if(argc!=2)
{
fprintf(stderr,"USAGE:nslookup\n");
}
if((host=gethostbyname(argv[1]))==NULL)
{
fprintf(stderr,"(mini)nslookup failed on %s\n",argv[1]);
}
h_addr.s_addr=*((unsigned long*)host->h_addr_list[0]);
printf("\n IP ADDRESS=%s\n",inet_ntoa(h_addr));
printf("\n HOST NAME=%s\n",host->h_name);
printf("\nADDRESS LENGTH =%d\n",host->h_length);
printf("\nADDRESS TYPE=%d\n",host->h_addrtype);
printf("\nLIST OF ADDRESS=%s\n",inet_ntoa(h_addr_list[0]));
}
```

OUTPUT

```
[it28@localhost ~]$ vi dns.c
[it28@localhost ~]$ cc dns.c
[it28@localhost ~]$ ./a.out 90.0.0.36
IP ADDRESS=90.0.0.36
HOST NAME=90.0.0.36
ADDRESS LENGTH =4
ADDRESS TYPE=2
LIST OF ADDRESS=90.0.0.36
```

Result

Hence the program to develop a client that contacts a given DNS server to resolve a given host name is executed successfully.

EX NO: 2.c UDP DNS SERVER/CLIENT

AIM:

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

DESCRIPTION

DNS stands for domain name system. unique name of the host is identified with its IP address through server client communication.

ALGORITHM:

Server

1. Create an array of hosts and its ip address in another array
2. Create a datagram socket and bind it to a port
3. Create a datagram packet to receive client request
4. Read the domain name from client to be resolved

5. Lookup the host array for the domain name
6. If found then retrieve corresponding address
7. Create a datagram packet and send ip address to client
8. Repeat steps 3-7 to resolve further requests from clients
9. Close the server socket
10. Stop

Client

1. Create a datagram socket
2. Get domain name from user
3. Create a datagram packet and send domain name to the server
4. Create a datagram packet to receive server message5. Read server's response
6. If ip address then display it else display "Domain does not exist"
7. Close the client socket
8. Stop

PROGRAM

// UDP DNS Server -- udpdnsserver.java

```
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();
}
}
}
```

//UDP DNS Client -- udpdnsclient.java

```
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(); }}
```

OUTPUT

Server

```
$ javac udpdnsserver.java
$ java udpdnsserver
Press Ctrl + C to Quit
Request for host yahoo.com
Request for host cricinfo.com
Request for host youtube.com
```

Client

```
$ javac udpdnsclient.java
$ java udpdnsclient
Enter the hostname : yahoo.com
IP Address: 68.180.206.184
$ java udpdnsclient
Enter the hostname : cricinfo.com
IP Address: 80.168.92.140
$ java udpdnsclient
Enter the hostname : youtube.com
IP Address: Host Not Found
```

RESULT

Thus domain name requests by the client are resolved into their respective logical address using lookup method.

3. Programs using RAW sockets

Ex.No: 3 PACKET CAPTURING AND FILTERING

AIM:

To implement raw sockets like packet capturing and filtering using java .

DESCRIPTION:

Raw socket is created to define the transmission of packet. Packet is captured for checking error. Error containing packets are filtered during transmission. Raw socket using TCP/IP protocol is created . Packet length is defined along with TCP header attached. Packets with error are checked using CRC mechanism and filtered.

ALGORITHM :

1. Start the program and to include the necessary header files.
2. To define the packet length.
3. To declare the IP header structure using TCP header.
4. Using simple checksum process to check the process.
5. Using TCP \IP communication protocol to execute the program.
6. And using TCP\IP communication to enter the Source IP and port number and Target IP address and port number.
7. The Raw socket () is created and accept the Socket ( ) and Send to ( ), ACK
8. Stop the program

PROGRAM:

```
//---cat rawtcp.c---
// Run as root or SUID 0, just datagram no data/payload
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
// Packet length
#define PCKT_LEN 8192
// May create separate header file (.h) for all
// headers' structures
// IP header's structure
struct ipheader {
unsigned char iph_ihl:5, /* Little-endian */
iph_ver:4;
unsigned char iph_tos;
unsigned short int iph_len;
unsigned short int iph_ident;
unsigned char iph_flags;
unsigned short int iph_offset;
unsigned char iph_ttl;
unsigned char iph_protocol;
unsigned short int iph_chksum;
unsigned int iph_sourceip;
unsigned int iph_destip;
};
/* Structure of a TCP header */
struct tcpheader {
unsigned short int tcph_srcport;
unsigned short int tcph_destport;
unsigned int tcph_seqnum;
unsigned int tcph_acknum;
unsigned char tcph_reserved:4, tcph_offset:4;
// unsigned char tcph_flags;
unsigned int
tcp_res1:4, /*little-endian*/
tcph_hlen:4, /*length of tcp header in 32-bit
words*/
tcph_fin:1, /*Finish flag "fin"*/
tcph_syn:1, /*Synchronize sequence numbers to
start a connection*/
tcph_rst:1, /*Reset flag */
tcph_psh:1, /*Push, sends data to the
application*/
tcph_ack:1, /*acknowledge*/ tcph_urg:1, /*urgent pointer*/
tcph_res2:2;
unsigned short int tcph_win;
unsigned short int tcph_chksum;
unsigned short int tcph_urgptr;
};
// Simple checksum function, may use others such as Cyclic
Redundancy Check, CRC
unsigned short csum(unsigned short *buf, int len)
{
unsigned long sum;
for(sum=0; len>0; len--)
sum += *buf++;
sum = (sum >> 16) + (sum &0xffff);
sum += (sum >> 16);
return (unsigned short)(~sum);
```

```
}
int main(int argc, char *argv[])
{
int sd;
// No data, just datagram
char buffer[PCKT_LEN];
// The size of the headers
struct ipheader *ip = (struct ipheader *) buffer;
struct tcpheader *tcp = (struct tcpheader *) (buffer +
sizeof(struct ipheader));
struct sockaddr_in sin, din;
int one = 1;
const int *val = &one;
memset(buffer, 0, PCKT_LEN);
if(argc != 5)
{
printf("- Invalid parameters!!!\n");
printf("- Usage: %s <source hostname/IP> <source port>
<target hostname/IP> <target port>\n", argv[0]);
exit(-1);
}
sd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
if(sd < 0)
{
perror("socket() error");
exit(-1);
}
else
printf("socket()-SOCK_RAW and tcp protocol is OK.\n");
// The source is redundant, may be used later if needed
// Address family
sin.sin_family = AF_INET;
sin_family = AF_INET;
// Source port, can be any, modify as needed
sin.sin_port = htons(atoi(argv[2]));
din.sin_port = htons(atoi(argv[4]));
// Source IP, can be any, modify as needed
sin.sin_addr.s_addr = inet_addr(argv[1]);
din.sin_addr.s_addr = inet_addr(argv[3]);
// IP structure
ip->iph_ihl = 5;
ip->iph_ver = 4;
ip->iph_tos = 16;
ip->iph_len = sizeof(struct ipheader) + sizeof(struct
tcpheader);
ip->iph_ident = htons(54321);
ip->iph_offset = 0;
ip->iph_ttl = 64;
ip->iph_protocol = 6; // TCP
```

```
ip->iph_chksum = 0; // Done by kernel
// Source IP, modify as needed, spoofed, we accept through
command line argument
ip->iph_sourceip = inet_addr(argv[1]);
// Destination IP, modify as needed, but here we accept
through command line argument
ip->iph_destip = inet_addr(argv[3]);
// The TCP structure. The source port, spoofed, we accept
through the command line
tcp->tcph_srcport = htons(atoi(argv[2]));
// The destination port, we accept through command line
tcp->tcph_destport = htons(atoi(argv[4]));
tcp->tcph_seqnum = htonl(1);
tcp->tcph_acknum = 0;
tcp->tcph_offset = 5;
tcp->tcph_syn = 1;
tcp->tcph_ack = 0;
tcp->tcph_win = htons(32767);
tcp->tcph_chksum = 0; // Done by kernel
tcp->tcph_urgptr = 0;
// IP checksum calculation
ip->iph_chksum = csum((unsigned short *) buffer,
(sizeof(struct ipheader) + sizeof(struct tcpheader)));
// Inform the kernel do not fill up the headers' structure,
we fabricated our own
if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))
< 0)
{
perror("setsockopt() error");
exit(-1);
} else
printf("setsockopt() is OK\n");
printf("Using:::::Source IP: %s port: %u, Target IP: %s
port: %u.\n", argv[1], atoi(argv[2]), argv[3],
atoi(argv[4]));
// sendto() loop, send every 2 second for 50 counts
unsigned int count;
for(count = 0; count < 20; count++)
{
if(sendto(sd, buffer, ip->iph_len, 0, (struct sockaddr
*)&sin, sizeof(sin)) < 0)
// Verify
{
perror("sendto() error");
exit(-1);
}
else
printf("Count #%u - sendto() is OK\n", count);
sleep(2);
```

```
    }
    close(sd);
    return 0;
    }
```

OUTPUT:

```
    Setsockopt() is OK
    Using Source IP : 172.17.1.72
    Port : 3001
    Target IP: 172.17.1.76
    Port : 5001
```

RESULT:

Thus the above programs using raw sockets TCP \IP (like packet capturing and filtering) was executed and successfully.

4. Programs using RPC

Ex.No: 4.a CLIENT – SERVER COMMUNICATION USING RPC

AIM

To write a C-program to implement Client – Server communication using RPC.

DESCRIPTION

A remote procedure call (RPC) is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

- In RPC, the sender makes a request in the form of a procedure, function, or method call. RPC translates these calls into requests sent over the network to the intended destination.
- The RPC recipient then processes the request based on the procedure name and argument list, sending a response to the sender when complete.
- The process is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters.
- The remote server sends a response to the client, and the application continues its process.
- While the server is processing the call, the client is blocked, it waits until the server has finished processing before resuming execution.

ALGORITHM:

Server:

```
Step 1: Start the program
Step 2: Create a socket with address family AF_INET type SOCK_STREAM and
default protocol.
Step 3: Initialize a socket and set its attributes.
Step 4: Bind the server to the socket using bind function.
Step 5: wait for the client request, on request establish a connection using accept
function.
Step 6: Read the number from the client by using read method
Step 7: Display the no.
Step 8: add the digits of a given number.
Step 9: send the result to the client by using write method
Step 10: stop the program.
```

Client:

Step 1: start.
Step 2: create a socket with address family.
Step 3: Initialize the socket and set its attributes set the required port no.
Step 4: Type AF_INET socket with default protocol.
Step 5: Connect to server using connect function to initiate the request.
Step 6: send a no to the server using write method.
Step 7: receive the result using read method.
Step 8: stop

PROGRAM:

//Server.c

```
#include<stdio.h>
#include<stdlib.h>
#include<netdb.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<string.h>
#include<errno.h>
#include<math.h>
int main(int argc,char **argv)
{
struct sockaddr_in servaddr,cliaddr;
char buff[1024],str[1024];
int len,len1,listenfd,connfd,n,i;
int count=0;
int no,sum=0;
listenfd=socket(AF_INET,SOCK_STREAM,0);
if(listenfd<0)
{
printf("Socket Error");
exit(0);
}
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(20003);
servaddr.sin_addr.s_addr=inet_addr("192.168.0.250");
bind(listenfd,&servaddr,sizeof(servaddr));
listen(listenfd,5);
len=sizeof(cliaddr);
connfd=accept(listenfd,(struct sockaddr*)&cliaddr,&len);
n=read(connfd,buff,1024);
printf("Received no is \t %s\n",buff);
no=atoi(buff);
printf("no=%d\n",no4);
while(no>0)
{
sum+=no%10;
no=no/10;
}
sprintf(buff,"%d",sum);
n=write(connfd,buff,1024);
close(listenfd);
return 0;
}
```

//Client.c

```
#include<stdio.h>
#include<stdlib.h>
#include<netdb.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<string.h>
#include<errno.h>
int main(int argc,char **argv)
{
struct sockaddr_in servaddr,cliaddr;
char buff[1024],str[1000];
int len,sockfd,n;
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd<0)
printf("Socket Error");
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(20003);
servaddr.sin_addr.s_addr=inet_addr("192.168.0.250");
connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
printf("\nenter a number\t");
scanf("%s",buff);
n=write(sockfd,buff,1024);
n=read(sockfd,buff,1024);
printf("\nResult is \t%s",buff);
close(sockfd);
return 0;
}
```

OUTPUT

Server

```
[admin@localhost admin]$ cc Server.c
rpcserv.c: In function `main':
rpcserv.c:30: warning: passing arg 2 of `bind' from incompatible pointer type
[admin@localhost admin]$
[admin@localhost admin]$ ./a.out
Received no is   879
no4=879
```

Client

```
[admin@localhost admin]$ cc Client.c
[admin@localhost admin]$ ./a.out
```

```
enter a no      879
Recived msg is  24
```

RESULT:

Thus the C-Program to implement Client - Server Communication using RPC was executed and output verified using various samples.

Ex.No: .4.b ARITHMETIC CALCULATOR USING RPC-RMI

AIM:

To create RMI to perform arithmetic operations using RPC.

DESCRIPTION:

A **remote procedure call (RPC)** is an IPC that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

Declare server's remote interface for all calculator operation by extending Remote interface. Define basic calculator operations such as summation, difference, product, quotient and remainder by extending **UnicastRemoteObjec**t.

An **interface** is the point of interaction with software or computer hardware, or with peripheral devices such as a computer monitor or a keyboard computing. Some computer interfaces such as a touch screen can send and receive data, while others such as a mouse or microphone, can only send data.

ALGORITHM:

1. Create a remote interface naming Display Int. Each remote method in interface throws a java.rmi.remote exception.
2. Define the class that implements the interface of the remote object extending the class Java.rmi.unicast Remote object.
3. Create the server program, which contains the method used to bind the name to the remote server object.
4. Write the client code. The client program usually consists of simple Java code. Within the client program we have to use lookup method available in java.rmi.
5. Name class to locate a remote object.
6. Compile all java files using javac.
7. Create a stub and skeleton using the compiler rmic Impl filename.
8. Start the RMI registry enabling us to register the server object. RMI registry is started using the command.
9. Run the server program.
10. Run the client program in separate command prompt using command.

PROGRAM:

//interface.java

```
import java.rmi.*;
public interface OperationInterface extends Remote
```

```
{
        public int add(int a,int b)throws RemoteException;
        public int sub(int a,int b)throws RemoteException;
        public int mul(int a,int b)throws RemoteException;
        public int div(int a,int b)throws RemoteException;
}
```

//OperationImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class OperationImpl extends UnicastRemoteObject implements OperationInterface
{
        public int add(int a,int b)throws RemoteException
        {
                return(a+b);
        }
        public int sub(int a,int b)throws RemoteException
        {
                return(a-b);
        }
        public int mul(int a,int b)throws RemoteException
        {
                return(a*b);
        }
        public int div(int a,int b)throws RemoteException
        {
                return(a/b);
        }
        public OperationImpl()throws RemoteException
        {
                super();
        }}
```

//OperationServer.java

```
import java.rmi.*;
import java.rmi.server.*;
public class OperationServer
{
        public static void main(String args[])
        {       try
                 {
                        OperationImpl impl=new OperationImpl();
                        Naming.rebind("HelloService",impl);
                }       catch(Exception e)
                {       System.out.println(e.toString());       }
         }
   }
```

//OperationClient.java

```
import java.io.*;
import java.rmi.*;
public class OperationClient
{
        public static void main(String arg[])throws RemoteException
        {       try
                {
                int a,b,ch;
                String str,str1;
                DataInputStream in=new DataInputStream(System.in);
                OperationInterface                    obj                    =
                (OperationInterface)Naming.lookup("HelloService");
                do
                {
                        System.out.println("ARITHMATIC OPERATION");
                        System.out.println("1.ADDITION");
                        System.out.println("2.SUBTRACTION");
                        System.out.println("3.MULTIPLICATION");
                        System.out.println("4.DIVITION");
                        System.out.println("5.EXIT");
                        System.out.println("ENTER YOUR OPTION :");
                        str1=in.readLine();
                        ch=Integer.parseInt(str1);
                       switch(ch)
                       {
                       case 1:
                                System.out.println("Enter the value for A:");
                        str=in.readLine();
                              a=Integer.parseInt(str);
                              System.out.println("Enter the value for B:");
                                str=in.readLine();
                                b=Integer.parseInt(str);
                                System.out.println("ADD="+obj.add(a,b));
                                break;
                        case 2:
                              System.out.println("Enter the value for A:");
                              str=in.readLine();
                                a=Integer.parseInt(str);
                              System.out.println("Enter the value for B:");
                              str=in.readLine();
                                b=Integer.parseInt(str);
                              System.out.println("SUB="+obj.sub(a,b));
                             break;
                        case 3:
                                System.out.println("Enter the value for A:");
                                str=in.readLine();
                                a=Integer.parseInt(str);
                                System.out.println("Enter the value for B:");
                                str=in.readLine();
```

```
                                        b=Integer.parseInt(str);
                                        System.out.println("MUL="+obj.mul(a,b));
                                        break;
                                case 4:
                                        System.out.println("Enter the value for A:");
                                        str=in.readLine();
                                        a=Integer.parseInt(str);
                                        System.out.println("Enter the value for B:");
                                        str=in.readLine();
                                        b=Integer.parseInt(str);
                                        System.out.println("DIV="+obj.div(a,b));
                                        break;
                                        }
                                 }while(ch!=5);
                        }       catch(Exception e)
                        {       System.out.println(e.toString());       }
                }
        }
```

OUTPUT

```
        C:\>cd Arithmetic op
        C:\Arithmetic op>javac *.java
        C:\Arithmetic op>rmic OperationImpl
        C:\Arithmetic op>start rmiregistry
        C:\Arithmetic op>start java OperationServer
        C:\Arithmetic op>java OperationClient
ARITHMATIC OPERATION
                1.ADDITION
                2.SUBTRACTION
                3.MULTIPLICATION
                4.DIVITION
                5.EXIT
                ENTER YOUR OPTION :
                1
                Enter the value for A:
                3
                Enter the value for B:
                5
```

RESULT:

Thus the RMI program for performing arithmetic operations such as add, sub, mul and div is created and the output is verified.

5. Simulation of sliding window protocols

Ex.No: 5 SIMULATION OF SLIDING WINDOW PROTOCOLS

AIM:

To write a C program to perform sliding window.

DESCRIPTION

A **sliding window frame** of data is sent from server to client. If that frame is received correctly at the client then acknowledgement signal is sent to server. Otherwise negative acknowledgement is sent. Frame size for the sliding window is provided by the user. Then the frame of data is sent with ACK signal or NACK signal.

A **sliding window protocol** is a feature of packet-based data transmission protocol. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the data link layer of OSI model as well as in the TCP.

A **data frame** is an aggregate of numerous, partly overlapping collections of data and metadata that have been derived from massive amounts of network activity such as content production, consumption, and other user behavior.

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:

```
// Sliding window protocol Client:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mymsgbuf
{
long mtype;
char mtext[25];
};
FILE *fp;
int main()
{
struct mymsgbuf buf;
int msgid;
int i=0,s;
int count=0,frmsz;
int a[100];
char d;
if((msgid=msgget(89,IPC_CREAT|0666))==-1)
{
printf("\n ERROR IN MSGGET");
exit(0);
}
printf("\n Enter the frame size:"); scanf("%d",&frmsz);
if((fp=fopen("check","r"))==NULL) printf("\n FILE NOT OPENED");
else
printf("\n FILE OPENED");
while(!feof(fp))
{ d=getc(fp);
a[i]=d;
i++;
}
s=i;
for(i=0;i<frmsz;i++)
//print from the check file printf("\t %c",a[i]);
for(i=0;i<frmsz;i++)
{ if((msgrcv(msgid,&buf,sizeof(buf),0,1))==-1)
{
printf("\n ERROR IN MSGRCV");
exit(0);
}
printf("\n RECEIVED FRAMES ARE:%c",buf.mtext[i]);
}
for(i=0;i<frmsz;i++)
{ if(a[i]==buf.mtext[i])
count++;
} if(count==0)
{
printf("\n FRAMES WERE NOT RECEIVED IN CORRECT SEQ");
exit(0);
} if(count==frmsz)
{
printf("\n FRAMES WERE RECEIVED IN CORRECT SEQ");
} else
{
printf("\n FRAMES WERE NOT RECEIVED IN CORRECT SEQ");
}}
```

//Sliding Window Protocol - Server

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mymsgbuf
{ long mtype;
char mtext[25];
};
FILE *fp;
int main()
{s
truct mymsgbuf buf;
int si,ei,sz; int msgid; int i=0,s; int a[100]; char d;
if((fp=fopen("send","r"))==NULL) printf("\n FILE NOT OPENED"); else
printf("\n FILE OPENED");
printf("\n Enter starting and ending index of frame array:");
scanf("%d%d",&si,&ei); sz=ei-si; if((msgid=msgget(89,IPC_CREAT|0666))==-1)
{
printf("\n ERROR IN MSGGET");
exit(0);
}
while(!feof(fp))
{ d=getc(fp); a[i]=d;
i++;
}s
=i; buf.mtype=1; for(i=si;i<=ei;i++)
{
buf.mtext[i]=a[i];
}
for(i=si;i<=ei;i++) //the frames to be sent
printf("\t %c",buf.mtext[i]); for(i=0;i<=sz;i++)
{ if((msgsnd(msgid,&buf,sizeof(buf),0))==-1)
{
printf("\n ERROR IN MSGSND");
exit(0);
}}
printf("\n FRAMES SENT");
return 0;
}
```

OUTPUT:

```
Enter the frame size : 5
File opened
Enter starting & ending index of frame array : 0 9
Frames sent
Received frames are: 0 3 6 7 9
```

RESULT:

Thus the C-program for performing client server based sliding window protocol was executed and output verified using various samples.

6. Experiments using simulators – Network Simulator 2

Ex.No: 6.a SIMPLE TOPOLOGY CREATION USING NS - 2

AIM:

To create simple topology using Network Simulator – 2.

ALGORITHM:

Step 1: Start network simulator OTCL editor.
Step 2: Create new simulator using **set ns [new Simulator]** syntax
Step 3: Create Trace route to Network Animator

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

Step 4: Create procedure to trace all path

```
proc create_testnet {}  {
        global s1 s2 r1 k1
        set s1 [$ns node]
        set s2 [$ns node]
        set r1 [$ns node]
        set k1 [$ns node]
 }
```

Step 5: Create full duplex connection using

```
$ns duplex-link $s1 $r1 8Mb 5ms drop-tail
$ns duplex-link $s2 $r1 8Mb 5ms drop-tail
set L [ns_duplex $r1 $k1 800Kb 100ms drop-tail]
```

Step 4: Connect with TCP and SINK command.

```
$ns connect $tcp $sink
```

Step 5: Run and Execute the program.

```
$ns run
```

PROGRAM:

```
set ns [new Simulator]
set nf [open udp.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf
proc create_testnet {}  {
        global s1 s2 r1 k1
        set s1 [$ns node]
        set s2 [$ns node]
        set r1 [$ns node]
        set k1 [$ns node]
 }
        $ns duplex-link $s1 $r1 8Mb 5ms drop-tail
        $ns duplex-link $s2 $r1 8Mb 5ms drop-tail
        set L [ns_duplex $r1 $k1 800Kb 100ms drop-tail]
        [lindex $L 0] set queue-limit 6
        [lindex $L 1] set queue-limit 6
$ns run
```

OUTPUT:

RESULT:

Thus the program for implementing UDP was executed using NS-2 and output verified using Network Animator.

Ex.No: 6.b USER DATAGRAM PROTOCOL USING NS-2

AIM:

To implement User Datagram Protocol (UDP) using NS-2

ALGORITHM:

Step 1: Start network simulator OTCL editor.

Step 2: Create new simulator using **set ns [new Simulator]** syntax

Step 3: Create procedure to trace all path

```
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam udp.nam &
              exit 0 }
```

Step 4: Connect with TCP and SINK command.

```
$ns connect $tcp $sink
```

Step 5: Run and Execute the program.

```
$ns run
```

PROGRAM:

```
set ns [new Simulator]
set nf [open udp.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam udp.nam &
              exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns duplex-link $n1 $n4 1Mb 50ms DropTail
$ns duplex-link $n2 $n5 0.1Mb 1ms DropTail
$ns duplex-link $n3 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 50ms DropTail
$ns duplex-link-op $n2 $n5 queuePos 1
set tcp [new Agent/UDP]
$ns attach-agent $n0 $tcp
set sink [new Agent/Null]
```

```
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
set ftp [new Application/Traffic/CBR]
$ftp attach-agent $tcp
$ns at 0.0 "$ftp start"
$ns at 2.5 "$ftp stop"
$ns at 3 "finish"
$ns run
```

OUTPUT:

RESULT:

Thus the program for implementing UDP was executed using NS-2 and output verified using Network Animator.

Ex.No: 6.c TRANSMISSION CONTROL PROTOCOL USING NS-2

AIM:

To implement Transmission Control Protocol (TCP) using NS-2

ALGORITHM:

Step 1: Start network simulator OTCL editor.

Step 2: Create new simulator using **set ns [new Simulator]** syntax

Step 3: Create procedure to trace all path

```
proc finish {} {
    global ns nf tf
    $ns flush-trace
    close $nf
    close $tf
    exec nam tcp.nam &
        exit 0}
```

Step 4: Connect with TCP and SINK command.

```
$ns connect $tcp $sink
```

Step 5: Run and Execute the program.

```
$ns run
```

PROGRAM:

```
set ns [new Simulator]
set nf [open tcp.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf
proc finish {} {
    global ns nf tf
    $ns flush-trace
    close $nf
```

```
          close $tf
          exec nam tcp.nam &
               exit 0
    }
    set n0 [$ns node]
    set n1 [$ns node]
    set n2 [$ns node]
    set n3 [$ns node]
    set n4 [$ns node]
    set n5 [$ns node]
    $ns duplex-link $n0 $n4 1Mb 50ms DropTail
    $ns duplex-link $n1 $n4 1Mb 50ms DropTail
    $ns duplex-link $n2 $n5 1Mb 1ms DropTail
    $ns duplex-link $n3 $n5 1Mb 1ms DropTail
    $ns duplex-link $n4 $n5 1Mb 50ms DropTail
    $ns duplex-link-op $n4 $n5 queuePos 0.5
    set tcp [new Agent/TCP]
    $ns attach-agent $n0 $tcp
    set sink [new Agent/TCPSink]
    $ns attach-agent $n2 $sink
    $ns connect $tcp $sink
    set ftp [new Application/FTP]
    $ftp attach-agent $tcp
    $ns at 0.0 "$ftp start"
    $ns at 2.5 "$ftp stop"
    $ns at 3 "finish"
    $ns run
```

OUTPUT:

RESULT:

Thus the program for implementing TCP was executed using NS-2 and output verified using Network Animator.

7. Performance comparison of MAC protocols

Ex.No: 7 PERFORMANCE COMPARISONS OF MAC PROTOCOLS

AIM:

To compare various MAC Protocols performance using NS-2

ALGORITHM:

Step 1: Start network simulator OTCL editor.
Step 2: Create new simulator using **set ns [new Simulator]** syntax
Step 3: Create Trace route to Network Animator

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

Step 4: Create procedure to trace all path

```
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
```

Step 4: Connect with TCP and SINK command.

```
$ns connect $tcp $sink
```

Step 5: Setup a FTP over TCP connection

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

Step 6: Setup a CBR over UDP connection

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

Step 7: Run and Execute the program.

```
$ns run
```

PROGRAM:

```
#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
set nf [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 $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $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 type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"
```

```
#Run the simulation
$ns run
```

OUTPUT:

RESULT:

Thus the MAC Protocols performance compared by using NS-2 and output verified by using Network Animator.

8. Performance comparison of Routing protocols

Ex.No: 8 PERFORMANCE COMPARISON OF ROUTING PROTOCOLS

AIM:

To compare various Routing Protocols performance using NS-2

ALGORITHM:

Step 1: Start network simulator OTCL editor.

Step 2: Create new simulator using **set ns [new Simulator]** syntax

Step 3: Create Trace route to Network Animator

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

Step 4: Create procedure to trace all path

```
proc finish {} {
        global ns
        $ns flush-trace
        puts "running nam..."
        exec nam out.nam &
        exit 0
        }
```

Step 4: Connect with UDP and CBR command.

```
set cbr1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
$cbr1 attach-agent $udp1
$udp1 set dst_ 0x8002
$udp1 set class_ 1
$ns attach-agent $n3 $udp1
```

Step 5: Setup a new agent for performing Multicast Routing procedure

```
set rcvr [new Agent/LossMonitor]
#$ns attach-agent $n3 $rcvr
```

Step 6: Create a group and start services

```
$ns at 1.2 "$n2 join-group $rcvr 0x8002"
```

```
$ns at 1.25 "$n2 leave-group $rcvr 0x8002"
$ns at 1.3 "$n2 join-group $rcvr 0x8002"
$ns at 1.35 "$n2 join-group $rcvr 0x8001"
$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"
$ns at 2.0 "finish"
```

Step 7: Run and Execute the program.

```
$ns run
```

PROGRAM:

```
set ns [new Simulator]
$ns multicast
set f [open out.tr w]
$ns trace-all $f
$ns namtrace-all [open out.nam w]
$ns color 1 red
# prune/graft packets
$ns color 30 purple
$ns color 31 green
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
# Use automatic layout
$ns duplex-link $n0 $n1 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n1 $n3 orient right-down
$ns duplex-link-op $n0 $n1 queuePos 0.5
set mrthandle [$ns mrtproto DM {}]
set cbr0 [new Application/Traffic/CBR]
set udp0 [new Agent/UDP]
$cbr0 attach-agent $udp0
$ns attach-agent $n1 $udp0
$udp0 set dst_ 0x8001
set cbr1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
$cbr1 attach-agent $udp1
$udp1 set dst_ 0x8002
$udp1 set class_ 1
$ns attach-agent $n3 $udp1
set rcvr [new Agent/LossMonitor]
#$ns attach-agent $n3 $rcvr
$ns at 1.2 "$n2 join-group $rcvr 0x8002"
$ns at 1.25 "$n2 leave-group $rcvr 0x8002"
$ns at 1.3 "$n2 join-group $rcvr 0x8002"
$ns at 1.35 "$n2 join-group $rcvr 0x8001"
$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"
```

```
$ns at 2.0 "finish"
proc finish {} {
        global ns
        $ns flush-trace
        puts "running nam..."
        exec nam out.nam &
        exit 0
        }
$ns run
```

OUTPUT:

RESULT:

Thus the Routing Protocols performance compared by using NS-2 and output verified by using Network Animator.

9. Study of TCP/UDP performance

CASE STUDY – 1

STUDY OF TCP PERFORMANCE

Topics:

1. TCP Protocol – Introduction
2. TCP Header
3. TCP Connection Establishment Protocol
4. TCP Socket Options
5. NS2 – Code for TCP Operations.
6. Conclusion

CASE STUDY – 2

STUDY OF UDP PERFORMANCE

Topics:

1. UDP Protocol – Introduction
2. UDP Header
3. UDP Socket Options
4. NS2 – Code for UDP Operations
5. Conclusion

CASE STUDY – 3

TCP/UDP PERFORMANCE COMPARISON

Topics:

1. TCP/UDP Difference
2. NS2 – Code for TCP/UDP Performance Comparison
3. Conclusion

APPENDIX

Network Programming

1. Looking up Internet Addresses.

```
import java.util.*;
import java.lang.*;
import java.net.*;
public class GetOwnIP
{
public static void main(String args[]) {
try{
InetAddress ownIP=InetAddress.getLocalHost();
System.out.println("IP of my system is := "+ownIP.getHostAddress());
}catch (Exception e){
System.out.println("Exception caught ="+e.getMessage());
                    }
          }
}
```

2. Pinging Command

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

public class ping
{
public static void main(String args[])
{
System.out.println("pinging status");
String ipAddress="GFL-344";
try
{
InetAddress inet=InetAddress.getByName("GFL-344");
System.out.println("sending ping request to"+ipAddress);
boolean status=inet.isReachable(5000);
if(status)
{
System.out.println("status:Host is reachable");
}
else
{
System.out.println("Status:Host is not reachable");
}
}
catch(IOException e)
{
System.out.println("host does not exist");
}
}
}
```

3. Implementation Of Peer to Peer connection using UDP

```
import java.io.*;
import java.net.*;
class UDPServer
{
  public static void main(String args[]) throws Exception
    {
      DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while(true)
          {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String( receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
          }
    }
}
import java.io.*;
import java.net.*;
class UDPClient
{
  public static void main(String args[]) throws Exception
  {
    BufferedReader inFromUser =
      new BufferedReader(new InputStreamReader(System.in));
    DatagramSocket clientSocket = new DatagramSocket();
    InetAddress IPAddress = InetAddress.getByName("localhost");
    byte[] sendData = new byte[1024];
    byte[] receiveData = new byte[1024];
    String sentence = inFromUser.readLine();
    sendData = sentence.getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
IPAddress, 9876);
    clientSocket.send(sendPacket);
```

```
     DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
     clientSocket.receive(receivePacket);
     String modifiedSentence = new String(receivePacket.getData());
     System.out.println("FROM SERVER:" + modifiedSentence);
     clientSocket.close();}}
```

4. Mulitcast Programming

//SimpleMulticastSource.java

```
import java.io.*;
import java.net.*;
public class SimpleMulticastSource
{
	public static void main(String[] args)
	{
		try
		{
			DatagramSocket s = new DatagramSocket();     // Create socket
			byte[] line = new byte[100];
			System.out.print("Enter text to send: ");
			int len = System.in.read(line);
			InetAddress dest = InetAddress.getByName("224.0.0.1");
			DatagramPacket pkt = new DatagramPacket(line, len, dest, 16900);
			s.send(pkt);
			s.close();
		}
		catch (Exception err)
		{
			System.err.println(err);}}}
```

//SimpleMulticastDestination.java

```
import java.io.*;
import java.net.*;
public class SimpleMulticastDestination
{
	public static void main(String[] args)
	{
		try
		{
			MulticastSocket ms = new MulticastSocket(16900);  // Create socket
			ms.joinGroup(InetAddress.getByName("224.0.0.1"));
			String msg;
			do
			{
				byte[] line = new byte[100];
				DatagramPacket pkt = new DatagramPacket(line, line.length);
				ms.receive(pkt);
				msg = new String(pkt.getData());
```

```
                        System.out.println("From "+pkt.getAddress()+":"+msg.trim());
                    }
                    while ( !msg.trim().equals("close") );
                    ms.close();                     // Close connection
                }
                catch (Exception err)
                {
                    System.err.println(err);
                }
        }}
```

5. Domain Name System

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char domain[20];
int i,j,n,pos[10],no;
clrscr();
printf("enter the domain name");
scanf("%s",domain);
n=strlen(domain);
no=0;
pos[no]=0;
for(i=0;i<n;i++)
{
if(domain[i]=='.')
{
no++;
pos[no]=i;
}
}
no++;
pos[no]=n;
printf("\nthe sub domains are:\n");
for(i=0;i<n;i++)
{
if(domain[i]!='.')
putch(domain[i]);
else
printf("\n");
}
printf("\n");
for(i=no;i>=1;i--)
{
printf("\nconducting sub domain:");
for(j=pos[i-1];j<=pos[i];j++)
{
if(domain[j]=='.')
{
continue;
```

```
	}
	putch(domain[j]);
	}
	delay(200);
	printf("\nredirecting\7.\7.\7.");
	}
	printf("\nobtained ip address is %d.%d.%d.%d",domain[4],domain[5],domain[7],domain[8]);
	getch();
	}
```

6. TCP Client – Server Communication using GUI

//ServerApp.java

```
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class ServerApp extends Frame implements ActionListener,Runnable
{
	ServerSocket s,s2;
	Socket s1,s4;
	BufferedReader br,br1;
	BufferedWriter bw,bw1;
	TextField text;
	Button button1,button2;
	List list;
	public void run()
	{
		try
		{
			s1.setSoTimeout(1);
			s4.setSoTimeout(1);
		}
		catch(Exception e)
		{}
		while(true)
		{
			try{
				list.addItem(br.readLine());
			}
			catch(Exception h){}
		}
		}
		public static void main(String args[])
		{
			new ServerApp("Server Application:");
		}
		public ServerApp(String m)
		{
			super(m);
			setSize(200,300);
```

```
                        setLocation(0,0);
                        this.setLayout(new BorderLayout());
                        button1 = new Button("Send");
                        button2 = new Button("Exit");
                        button1.addActionListener(this);
                        button2.addActionListener(this);
                        list = new List();
                        text = new TextField();
                        add(list,"Center");
                        add(button1,"West");
                        add(button2,"East");
                        add(text,"South");
                        setVisible(true);
                        try{
                                s=new ServerSocket(100);
                                s2=new ServerSocket(1000);
                                s1=s.accept();
                                s4=s2.accept();
                        br = new BufferedReader(new InputStreamReader(s1.getInputStream()));
                bw = new BufferedWriter(new OutputStreamWriter(s1.getOutputStream()));
                br1 = new BufferedReader(new InputStreamReader(s4.getInputStream()));
                bw1 = new BufferedWriter(new OutputStreamWriter(s4.getOutputStream()));
                                Thread th;
                                th = new Thread(this);
                                th.start();
                        }catch(Exception e){}
                }
        public void actionPerformed (ActionEvent e)
        {
                if (e.getSource().equals(button2))
                        System.exit(0);
                else
                {
                        try{
                                bw.write(text.getText());
                                bw.newLine();bw.flush();
                                text.setText("");
                        }
                        catch(Exception m){}            }           }}
```

//Client.java

```
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class Client extends Frame implements ActionListener, Runnable
{
        Socket s;
        BufferedReader br;
        BufferedWriter bw;
```

```
TextField text;
Button button1, button2;
List list;
public static void main(String args[])
{
          new Client("Client Application :");
}
public void run()
{
          try
          {
                    s.setSoTimeout(1);
          }
          catch (Exception e)
          {
          }
          while (true)
          {
                    try
                    {
                              list.addItem(br.readLine());
                    }
                    catch (Exception h)
                    {}}}
public Client(String m)
{
          super(m);
          setSize(200,300);
          setLocation(300,0);
          this.setLayout(new BorderLayout());
          button1 = new Button("Send");
          button2 = new Button("Exit");
          button1.addActionListener(this);
          button2.addActionListener(this);
          list=new List();
          text=new TextField();
          add(list,"Center");
          add(button1,"West");
          add(button2,"East");
          add(text, "South");
          setVisible(true);
          try
          {
                    s=new Socket("localhost",100);
                    br=new BufferedReader(new InputStreamReader(s.getInputStream()));
                    bw=new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
                    Thread th;
                    th=new Thread(this);
                    th.start();
```

```
                }
                catch(Exception e)
                {}}
        public void actionPerformed(ActionEvent e)
        {
                if (e.getSource().equals(button2))
                        System.exit(0);
                else
                {
                        try
                        {
                                bw.write(text.getText());
                                bw.newLine();
                                bw.flush();
                        }
                        catch (Exception m)
                        {}}}}
```

7. Cyclic Redundancy Check

```
                import java.io.*;
                import java.util.*;
                class crc
                {
                public static void main(String args[]) throws IOException
                {
                DataInputStream in=new DataInputStream(System.in);
                System.out.println("enter the data bits to be transmitted=");
                String m=in.readLine();
                String md=m;
                System.out.println("Enter the generator code");
                String g=in.readLine();
                for(int x=0;x<g.length()-1;x++)
                {
                m=m+"0";
                }
                char am[]=m.toCharArray();
                char ag[]=g.toCharArray();
                int lm=am.length;
                int lg=ag.length;
                int i=0;
                for(;i<  ++)
                {
                while(i<(lm-1)&&am[i]=='0')
                {
                i++;
                }
                if(i<(lm-lg+1))
                {
                for(int j=0;j<lg;i++,j++)
                {
```

```
if(am[i]==ag[j])
{
am[i]='0';
}
else
{
am[i]='1';
}
}
i=0;
}
else
i=lm;
}
int y=0;
while(y<(lm-1)&&am[y]=='0')
{
y++;
}
for(;y<lm;y++)
md=md+am[y];

System.out.println("The transmitted frame is:");
System.out.println(md);
System.out.println("The frame received:"+md);
String r=md;
if(r.equals(md))
System.out.println("The transmitted frame is correct");
else
System.out.println("the transmitted frame is incorrect");
}
}
```

8. Sliding Window Protocol

```
#include<stdio.h>
#include<conio.h>
void showwindow(int n)
{
int i,j;
cprintf("\n\r");
for(i=0;i<=n*3;i++)
cprintf("-");
cprintf("\n\r");
for(i=0;i<n*3;i++)
if(i%3==0)
cprintf("|");
else
cprintf(" ");
cprintf("|\n\r");
for(i=0;i<n*3;i++)
cprintf("-");
cprintf("\n\r");
}
void main()
{
int i,sn,rn,scr,rcr;
clrscr();
printf("enter the size of sender window");
scanf("%d",&sn);
printf("enter the size of the reciver window");
scanf("%d",&rn);
scr=sn;
rcr=rn;
for(i=0;i<rn;i++)
{
if(scr==0)
break;
cprintf("\r sending one frame");
scr--;
cprintf("\n\rsender window");
showwindow(scr);
delay(1000);
rcr--;
cprintf("\n\rreceved one frame");
rcr=rn;
cprintf("\n\rrecever window");
showwindow(rcr);
delay(1000);
cprintf("\n\rrecevied ack%d",rcr+1);
scr+=rcr;
cprintf("\n\rsender window");
showwindow(rcr);
delay(1000);
getch();
cprintf("sending ack for received frame%d",(rn+3)%rn);
rcr=rn;
cprintf("\n\rrecived window");
showwindow(rcr);
cprintf("received ack%d",rcr);
scr+=rcr;
showwindow(rcr);
getch();
}}
```

9. Trace Route Program

```
#include <stdlib.h>
#include <stdio.h>
#include<stdio.h>
#include<conio.h>
void traceroute(char *string)
{
  char *trace="tracert ";
  printf("string is %s\n",string);
  strcat(trace,string);
  printf("trace command is %s\n",trace);
   printf("traceroute begins\n");
   system(trace);
}
void main()
{
   char dest[50];
   clrscr();
   printf("enter the destination to find the traceroute\n");
   scanf("%s",dest);
   traceroute(dest);
   printf("\nthe path has found successfully\n");
   getch();
}
```

10. ARP and RARP

//ARP SERVER

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/shm.h>
#include<string.h>
main()
{
int shmid, a, i;
char *ptr, *shmptr;
shmid=shmget(3000,10,IPC_CREAT | 0666);
shmptr=shmat(shmid,NULL,0);
ptr=shmptr;
for(i=0;i<3;i++)
{
puts("enter the mac");
scanf("%s",ptr);
a=strlen(ptr);
printf("string length:%d",a);
ptr[a]= ' ' ;
puts("enter ip");
ptr=ptr+a+1;
scanf("%s",ptr);
ptr[a]='\n' ;
ptr= ptr+a+1;
}
ptr[strlen(ptr)]= '\0';
printf("\n ARP table at serverside is=\n%s", shmptr);
shmdt(shmptr);
}
```

//ARP CLIENT

```
#include<stdio.h>
#include<string.h>
```

```
#include<sys/types.h>
#include<sys/shm.h>
main()
{
int shmid,a;
char *ptr, *shmptr;
char ptr2[51], ip[12], mac[26];
shmid=shmget(3000,10,0666);
shmptr=shmat(shmid,NULL,0);
puts("the arp table is");
printf("%s",shmptr);
printf("\n1.ARP\n 2.RARP\n 3.EXIT\n");
scanf("%d",&a);
switch(a)
{
case 1:
puts("enter ip address");
scanf("%s",ip);
ptr=strstr(shmptr, ip);
ptr-=8;
sscanf(ptr,"%s%*s",ptr2);
printf("mac addr is %s",ptr2);
break;
case 2:
puts("enter mac addr");
scanf("%s",mac);
ptr=strstr(shmptr, mac);
sscanf(ptr,"%*s%s",ptr2);
printf("%s",ptr2);
break;
case 3:
exit(1); } }
```

11. Border Gateway Protocol

```
#include <stdio.h>
#include<conio.h>
int main()
{
int n;
int i,j,k;
int a[10][10],b[10][10];
printf("\n Enter the number of nodes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("\n Enter the distance between the host %d - %d:",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(k=0;k<n;k++)
{
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(a[i][j]>a[i][k]+a[k][j])
{
a[i][j]=a[i][k]+a[k][j];
}
}
}
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
b[i][j]=a[i][j];
if(i==j)
{
b[i][j]=0;
}
}}
printf("\n The output matrix:\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
getch();
}
```

12. Open Shortest Path First Routing Protocol

```
#include<stdio.h>
#include<conio.h>
int a[5][5],n,i,j;
void main()
{
void getdata();
void shortest();
void display();
clrscr();
printf("\n\n PROGRAM TO FIND SHORTEST PATH BETWEEN TWO
NODES\n");
getdata();
shortest();
display();
getch();
}
void getdata()
{
clrscr();
printf("\n\nENTER THE NUMBER OF HOST IN THE GRAPH\n");
scanf("%d",&n);
printf("\n\nIF THERE IS NO DIRECT PATH \n");
printf(" \n\nASSIGN THE HIGHEST DISTANCE VALUE 1000 \n");
for(i=0;i<n;i++)
{
a[i][j]=0;
for(j=0;j<n;j++)
{
if(i!=j)
{
printf("\n\nENTER THE DISTANCE BETWENN (%d,
%d): ",i+1,j+1);
scanf("%d",&a[i][j]);
if(a[i][j]==0)
a[i][j]=1000;
}
} } }
void shortest()
{
int i,j,k;
for(k=0;k<n;k++)
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(a[i][k]+a[k][j]<a[i][j])
a[i][j]=a[i][k]+a[k][j];
} }
void display()
{
int i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(i!=j)
{
printf("\n SHORTEST PATH IS : (%d,%d)--%d\n",i+1,j+1,a[i][j]);
}
getch();
}
```

13. Network Simulator – 2 (NS-2)

ABSTRACT:

Network simulation is used widely in network research to test new protocols, modifications to existing protocols and new ideas. The tools used in many cases is ns-2.The nature of the ns-2 protocols means that they are often based on presents the network simulation cradle which allows real world network stacks to be simulator. The network stacks from the open source operating systems Linux, designed for embedded systems, lwIP.Our results show that ns-2's TCP implementations do not match observed.

Simulator: A device that enables the operator to reproduce or represent under test conditions phenomena likely to occur in actual performance.

Simulation: is an imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviors of a selected physical or abstract system.

NS: The Network Simulator ns-2 is a discrete event simulator, which means it simulates such events as sending, receiving, forwarding and dropping packets. The events are sorted by time (seconds) in ascending order.

NS-2 AN OVERVIEW:

NS-2 is an event driven network simulator that simulates variety of networks. It implements network protocols such as TCP and UDP, traffic source mechanism such as FTP, telnet, web, CBR and VBR, router queue management mechanism such as drop tail, RED and CBQ, routing algorithms such as Dijkstra and more.NS also implements multicasting and some of the MAC layer protocols of the LAN simulations.

DESIGN:

NS is object oriented Tcl (OTcl) script interpreter that has a simulation event schedule and network component object libraries and network setup (PLUMBING) module libraries. To setup and run a simulation network, user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tell the traffic sources when to start and stop the transmitting packets through the event scheduler. The term "plumbing" is used for network setup, because setting up network is plumbing data paths among objects by setting the 'neighbors' pointer of an object to the address of an appropriate object.

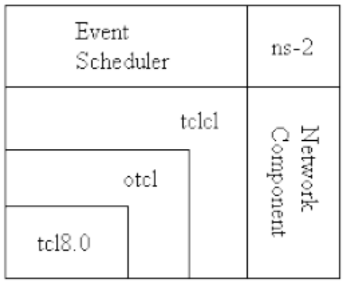

Figure 2. Architectural View of NS

WHAT IS NS-2?

- ✓ A discrete event simulator targeted at networking behaviors from real machines in some respects and using the network simulation cradle produces results closer to real world network stacks.
- ✓ Simulation of protocols (TCP, routing multicast….) over networks (wireless, wireline, satellite)
- ✓ Currently development is support through DARPA with SAMAN and through NSF with CONSER, in collaboration with ACIRI.

NS-2 COMPONENTS:

The tclobject is considered as the ancestor class of the ns-2components, ns object is the super class of all basic network components such as nodes and links. The basic network component such as nodes and links. The basic networks components are further divided into two subclasses, connector and classifier, based on the possible output data paths. The basic network objects that have only one output data path are under the connector class, and switching objects that have possible multiple output data paths are under the classifier class.

- **EVENTS SCHEDULER**:

In NS, an event scheduler keeps track of simulation time and fires all the events in the scheduled for the current time by invoking appropriate network components. Network components communicate with one another passing packet; however this does not consume actual simulation time.All the network components that need to spend some simulation time handling a packets i.e., need a delay use the events scheduler by issuing an event for the packet and waiting for the events to be forced to itself before doing further action handling a packet.

- **NODE AND ROUTING**:

Node is a compound object composed of a node entry object and classifiers. There are types of nodes in NS.A unicast node has an address classifier that does explicitly notified in the input of the OTcl script, right after creating a scheduler object, that all the nodes that will be created are multicast nodes. After specifying the node the user an also select a specific routing protocol other than using a default one.

- **LINK:**

A link is another major compound object in NS.when a user creates a link using a duplex-link member function of the simulator object; simplex links in both directions are created. One thing to note is that output queue of a node is actually implemented as a part of simplex link object.packets

- **TRACING:**

In TS, network activities traced around simplex links if the simulator is directed to trace network activities, the links created after the command will have the following trace objects inserted. When each trace object receives a packet, it writes to the specified trace file without consuming any simulation time, and passes the packet to the next network object.

- **QUEUE MONITOR:**

  Basically, tracing objects are designed to record packet arrival time at which they are located although user gets enough information from the trace, they might be interested in what is going on inside a specific output queue. When a packet arrives, snoop queue object notifies the queue monitor object of this event the queue monitor using this information monitors the queue.

ADVANTAGES:

- ✓ The TCP module in NS-2 is originally based on the source code of the BSD kernel.
- ✓ The main use of an event scheduler us to schedule simulation events,
- ✓ When to start an FTP application, when to finish a simulation, or for simulation scenario generation prior to a simulation run.

DRAWBACKS OF NS-2

- ✓ Extensibility
- ✓ Validity of NS-2 results
- ✓ Simulation speed

CONCLUSION:

Ns-2 provides a very convenient platform for testing different TCP congestion control protocols in many scenarios. It is a good foundation towards an implementation for TCP congestion control algorithms.

19. NS-2 Code to Printing Message

```
set sim [new Simulator]
$sim at 1 “puts \“Hello World!\””
$sim at 1.5 “exit”
$sim run
```

20. NS-2 OTCL Code

- Variables and Expressions
  - set a 5
  - set b $a
  - set c [expr $a + $b]
  - set d [expr [expr $a - $b] * $c]
  - incr x
  - incr x -1
  - set y [pow x 2]
  - set y [expr x*x]
  - int: set c [expr 1/6]
  - float: set c [expr 1.0 / 6.0]
  - Display: puts "Must enter atleast one Arguments"
  - Control structures:
    ```
    if {$x > 0} {
    return $x
    } else { return [expr -$x] }
    ```

• Open file for NS tracing

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

- Open file for nam tracing

  set nf [open out.nam w]
  $ns namtrace-all $nf
- Open your own trace file

  set my_f [open my_out.tr w]
  puts $my_f "[$ns now] [expr $x(1) + $y(1)]"
- Creating nodes

  set node_(h1) [$ns node]
  set node_(h2) [$ns node]
  set node_(r1) [$ns node]
  set node_(r2) [$ns node]
  set node_(h3) [$ns node]
  set node_(h4) [$ns node]
- Creating Link and Queue

  $ns duplex-link $node_(h1) $node_(r1) 10Mb 2ms DropTail
  $ns duplex-link $node_(h2) $node_(r2)10Mb 3ms DropTail
  $ns duplex-link $node_(r1) $node_(r2)1.5Mb 20ms DropTail
- Creating a TCP connection

  $set tcp0 [$ns create-connection TCP/Reno $node_(h1) TCPSink/DelAck $node_(h4) 0]
- Attaching FTP traffic on the top of TCP

  $set ftp0 [$tcp0 attach-app FTP]
- Schedule an event to start traffic at time 1.0

  $ns at 1.0 "$ftp0 start"
- Schedule an event to stop ns at time 17.0

  $ns at 17.0 "$ftp0 stop"
- Start ns

  $ns run
- Stop ns

  exit 0

21. NS-2 Code for opening Network Animator

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

22. Sum of n numbers (sum of series)

```
set n 5
set s 1
for {set i 1} { $i <= $n } {incr i} {
    set s [expr $s * $i]
          } puts $s
```

23. Factorial Calculation

```
proc fac { x y } {
#global argv
set fact 1
set n $y
#incr n
for {set i $x} {$i <= $n} {incr i} {
set fact [expr $fact * $i]
}
puts "fatorial is $fact"
}
```

24. UDP Distance Vector

```
set ns [new Simulator]
set nf [open udpdv.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam udpdv.nam &
            exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns duplex-link $n1 $n4 1Mb 50ms DropTail
$ns duplex-link $n2 $n5 1Mb 1ms DropTail
$ns duplex-link $n3 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 50ms DropTail
$ns duplex-link $n2 $n3 1Mb 50ms DropTail
$ns duplex-link-op $n4 $n5 queuePos 0.5
set tcp [new Agent/UDP]
$ns attach-agent $n0 $tcp
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
set ftp [new Application/Traffic/CBR]
$ftp attach-agent $tcp
$ns rtproto DV
$ns rtmodel-at 2.0 down $n5 $n2
$ns rtmodel-at 3.0 up $n5 $n2
$ns at 0.0 "$ftp start" $ns at 4.5 "$ftp stop" $ns at 5 "finish" $ns run
```