## Develop client server-based TCP applications using UNIX socket programming functions.

### AIM:

To run the program of top\_echoserver and top\_echoclient.

### THEORY:

The echo server **receives data from its client and echoes it back**. The EchoClient example creates a socket, thereby getting a connection to the echo server.

### PROGRAM:

**Top\_echoserver.c:**

/\*Required Headers\*/

#include <sys/types.h> #include <sys/socket.h> #include <netdb.h> #include <stdio.h> #include<string.h>

int main()

{

char str[100];

int listen\_fd, comm\_fd;

struct sockaddr\_in servaddr;

listen\_fd = socket(AF\_INET, SOCK\_STREAM, 0); bzero( &servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; servaddr.sin\_addr.s\_addr = htons(INADDR\_ANY); servaddr.sin\_port = htons(22000);

bind(listen\_fd, (struct sockaddr \*) &servaddr, sizeof(servaddr)); listen(listen\_fd, 10);

comm\_fd = accept(listen\_fd, (struct sockaddr\*) NULL, NULL);

while(1)

{

bzero( str, 100); read(comm\_fd,str,100); printf("Echoing back - %s",str); write(comm\_fd, str, strlen(str)+1);

}

}

**Top\_echoclient.c:** #include <sys/types.h> #include <sys/socket.h> #include <netdb.h> #include <stdio.h> #include<string.h>

int main(int argc,char \*\*argv)

{

int sockfd,n;

char sendline[100]; char recvline[100];

struct sockaddr\_in servaddr;

sockfd=socket(AF\_INET,SOCK\_STREAM,0); bzero(&servaddr,sizeof servaddr);

servaddr.sin\_family=AF\_INET; servaddr.sin\_port=htons(22000);

inet\_pton(AF\_INET,"127.0.0.1",&(servaddr.sin\_addr));

connect(sockfd,(struct sockaddr \*)&servaddr,sizeof(servaddr));

while(1)

{

bzero( sendline, 100);

bzero( recvline, 100);

fgets(sendline,100,stdin); /\*stdin = 0 , for standard input \*/

write(sockfd,sendline,strlen(sendline)+1); read(sockfd,recvline,100);

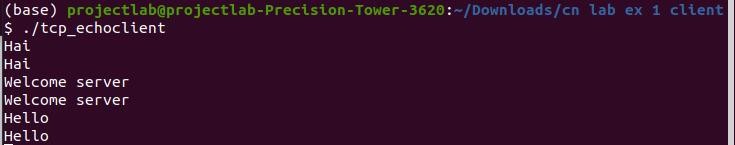
printf("%s",recvline);

}

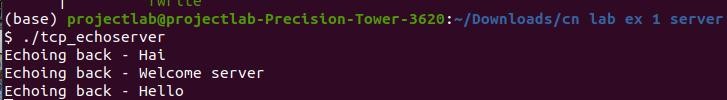
}

**OUTPUT:**

**top\_echoclient.c**



Tcp\_echoserver.c:



1. **Develop client server-based TCP\_FILECLIENT AND TCP FILESERVER**

## AIM:

To run the program of tcp\_fileclient and tcp\_fileserver.

**THEORY:**

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery.

Erroneous packets are retransmitted from the source to the destination.

### PROGRAM:

**Tcp\_fileclient.c** #include <netdb.h> #include <stdio.h> #include <stdlib.h> #include <string.h>

#include <sys/socket.h> #define MAX 80

#define PORT 8080 #define SA struct sockaddr void func(int sockfd)

{

char buff[MAX]; int n;

for (;;) {

bzero(buff, sizeof(buff)); printf("Enter the string : "); n = 0;

while ((buff[n++] = getchar()) != '\n')

;

write(sockfd, buff, sizeof(buff)); bzero(buff, sizeof(buff)); read(sockfd, buff, sizeof(buff)); printf("From Server : %s", buff);

if ((strncmp(buff, "exit", 4)) == 0) {

printf("Client Exit...\n"); break;

}

}

}

int main()

{

int sockfd, connfd;

struct sockaddr\_in servaddr, cli;

// socket create and verification

sockfd = socket(AF\_INET, SOCK\_STREAM, 0); if (sockfd == -1) {

printf("socket creation failed...\n"); exit(0);

}

else

printf("Socket successfully created..\n"); bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); servaddr.sin\_port = htons(PORT);

// connect the client socket to server socket

if (connect(sockfd, (SA\*)&servaddr, sizeof(servaddr)) != 0) { printf("connection with the server failed...\n");

exit(0);

}

else

printf("connected to the server..\n");

// function for chat func(sockfd);

// close the socket close(sockfd);

}

**Tcp\_fileserver.c:** #include <stdio.h> #include <netdb.h> #include <netinet/in.h>

#include <stdlib.h> #include <string.h> #include <sys/socket.h> #include <sys/types.h> #define MAX 80

#define PORT 8080 #define SA struct sockaddr

// Function designed for chat between client and server. void func(int connfd)

{

char buff[MAX]; int n;

// infinite loop for chat for (;;) {

bzero(buff, MAX);

// read the message from client and copy it in buffer read(connfd, buff, sizeof(buff));

// print buffer which contains the client contents printf("From client: %s\t To client : ", buff); bzero(buff, MAX);

n = 0;

// copy server message in the buffer while ((buff[n++] = getchar()) != '\n')

;

// and send that buffer to client write(connfd, buff, sizeof(buff));

// if msg contains "Exit" then server exit and chat ended. if (strncmp("exit", buff, 4) == 0) {

printf("Server Exit...\n"); break;

}

}

}

// Driver function int main()

{

int sockfd, connfd, len;

struct sockaddr\_in servaddr, cli;

// socket create and verification

sockfd = socket(AF\_INET, SOCK\_STREAM, 0); if (sockfd == -1) {

printf("socket creation failed...\n"); exit(0);

}

else

printf("Socket successfully created..\n"); bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); servaddr.sin\_port = htons(PORT);

// Binding newly created socket to given IP and verification if ((bind(sockfd, (SA\*)&servaddr, sizeof(servaddr))) != 0) { printf("socket bind failed...\n");

exit(0);

}

else

printf("Socket successfully binded..\n");

// Now server is ready to listen and verification if ((listen(sockfd, 5)) != 0) {

printf("Listen failed...\n"); exit(0);

}

else

printf("Server listening..\n"); len = sizeof(cli);

// Accept the data packet from client and verification connfd = accept(sockfd, (SA\*)&cli, &len);

if (connfd < 0) {

printf("server accept failed...\n"); exit(0);

}

else

printf("server accept the client...\n");

// Function for chatting between client and server func(connfd);

// After chatting close the socket close(sockfd);

}

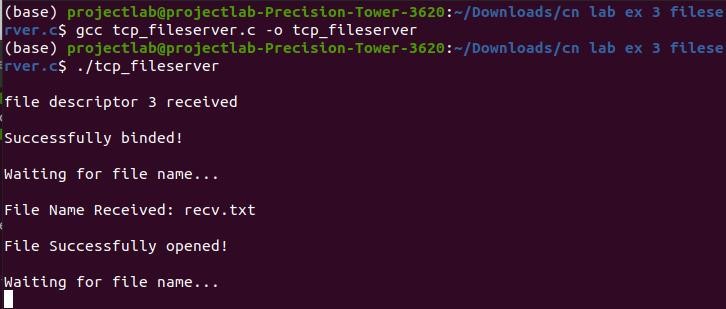
### OUTPUT:

Tcp\_fileclient.c:

Text

Description automatically generated

Tcp\_fileserver.c:



## Develop client server based UDP applications using UNIX socket programming functions.

### AIM:

To run the program of udp\_client and udp\_server.

### THEORY:

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.

### PROGRAM:

**udp\_server.c:** #include<sys/types.h> #include<sys/socket.h> #include<stdio.h> #include<netinet/in.h>

#define MAX 100

#define SERPORT 1090 #define SA struct sockaddr

void str\_echo(FILE\*,int,SA\*,socklen\_t);

int main(int argc,char\*\* argv)

{

int sockfd;

struct sockaddr\_in servaddr,cliaddr; sockfd=socket(AF\_INET,SOCK\_DGRAM,0); bzero(&servaddr,sizeof(servaddr)); servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(0); servaddr.sin\_port=htons(SERPORT); bind(sockfd,(SA\*)&servaddr,sizeof(servaddr)); str\_echo(stdin,sockfd,(SA\*)&cliaddr,sizeof(cliaddr)); exit(0);

}

void str\_echo(FILE\* fp,int sockfd,SA\* cliaddr,socklen\_t clilen)

{

char msg[MAX],send[MAX]; int n;

while(1)

{

if((n=recvfrom(sockfd,msg,MAX,0,cliaddr,&clilen))>0)

{

msg[n]='\0'; printf("Client msg : "); fputs(msg,stdout); printf("SERVER msg : "); fgets(msg,MAX,fp);

sendto(sockfd,msg,strlen(msg),0,cliaddr,clilen);

}

}

}

**Udp\_client.c:** #include<sys/types.h> #include<sys/socket.h> #include<stdio.h> #include<netinet/in.h>

#define MAX 100

#define SERPORT 1090 #define SA struct sockaddr

void str\_cli(FILE\*,int,SA\*,socklen\_t);

int main(int argc,char\*\* argv)

{

int sockfd;

struct sockaddr\_in servaddr; sockfd=socket(AF\_INET,SOCK\_DGRAM,0); bzero(&servaddr,sizeof(servaddr)); servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=inet\_addr("127.0.0.1"); servaddr.sin\_port=htons(SERPORT); printf("Client msg : ");

str\_cli(stdin,sockfd,(SA\*)&servaddr,sizeof(servaddr)); exit(0);

}

void str\_cli(FILE\* fp,int sockfd,SA\* seraddr,socklen\_t len)

{

char msg[MAX],rcv[MAX]; int n;

while(fgets(msg,MAX,fp)!=NULL)

{

sendto(sockfd,msg,MAX,0,seraddr,len); if((n=recvfrom(sockfd,rcv,MAX,0,seraddr,&len))>0)

{

rcv[n]='\0'; fputs(rcv,stdout);

}

printf("Client msg : ");

}

}

### OUTPUT:

Udpclient.c:



Udp\_server.c:

Text

Description automatically generated

1. **Develop a program write a program to find the Physical Address for a given IP address using Simulation.**

### AIM:

To write a program to find the Physical Address for a given IP address using Simulation.

### ALGORITHM:

Step 1: Start the program

Step 2: Generate random numbers to get IP addresses and physical addresses Step 3: Store the IP address and physical addresses in an array

Step 4: Randomly choose an IP address and identify the corresponding Physical address for that IP address with the help of the array.

Step 5: If a match is found then display the IP address and the corresponding physical address otherwise display that no match was found.

Step 6: Similarly choose a physical address randomly and find the corresponding IP address for that Physical address with the help of the array.

Step 7: If a match is found then display the physical address and the corresponding IP address otherwise display that no match was found.

Step 8: Stop the program.

### PROGRAM:

#include<stdlib.h> #include<stdio.h> #include<conio.h> void main()

{

struct

{

char \*ipa; char \*pha;

}ipadd[2]; int i,j,x=10;

char \*temp,\*temp1,\*x1;

char ip[2][15]={"135.237.105.128","225.22.205.221"}; char ph[2][18]={"SYSTEM1","SYSTEM12"};

clrscr(); for(j=0;j<2;j++)

{

srand(x++); for(i=0;i<4;i++)

{

itoa(rand()%256,temp,10); printf("\n%s",temp); strcat(ipadd[j].ipa,temp); if(i<3) strcat(ipadd[j].ipa,".");

}

printf("%s",itoa(j+1,x1,10)); temp1=strcat("SYSTEM",x1); strcpy(ipadd[j].pha,temp1); printf("\n%s\n",ipadd[j].ipa); printf("\n%s\n",ipadd[j].pha);

}

for(j=0;j<2;j++) printf("\n\n%s\n",ipadd[j].ipa);

for(j=0;j<2;j++) for(i=0;i<2;i++) if(strcmp(ipadd[i].ipa,ip[j])==0)

{

printf("\n the physical address of the given ip address %s is",ip[j]); printf("\n\n%s\n",ph[j]);

}

getch(); for(j=0;j<2;j++) for(i=0;i<2;i++)

if(strcmp(ipadd[i].pha,ph[j])==0)

{

printf("\n the ip address of the given physical address %s is",ph[j]); printf("\n\n%s\n",ip[j]);

}

getch();

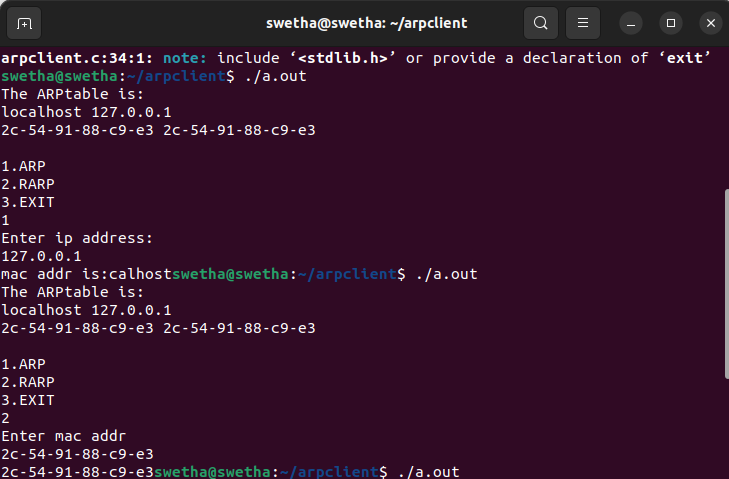
}

### OUTPUT:

**ARP/RARP SERVER**



**ARP/RARP CLIENT**



#### **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 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

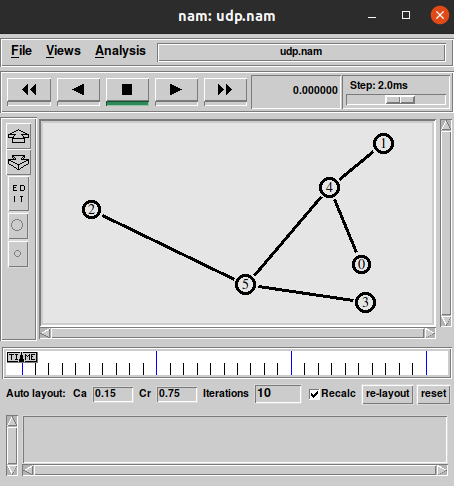
$ns at 0.0 "$ftp start"

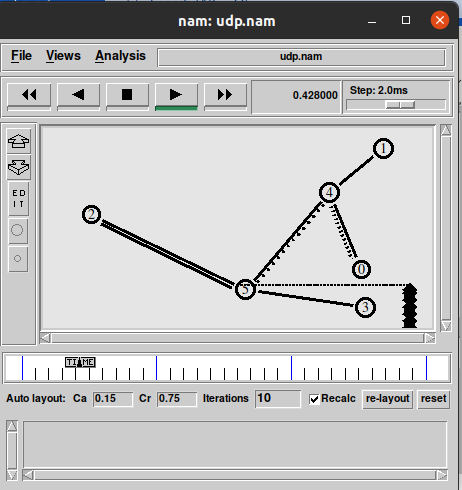
$ns at 2.5 "$ftp stop"

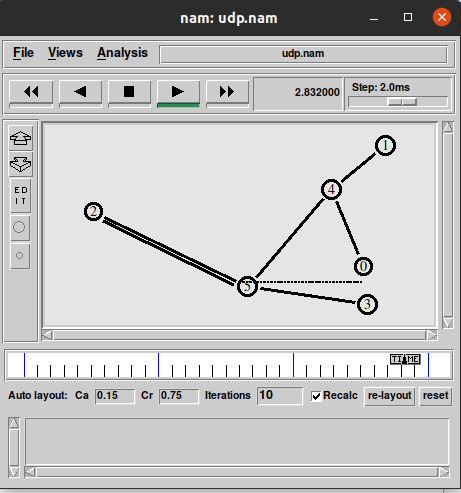
$ns at 3 "finish"

$ns run

**OUTPUT:**







#### RESULT:

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

#### **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

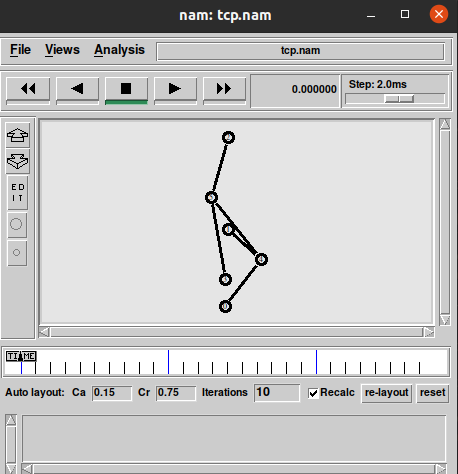
$ns at 0.0 "$ftp start"

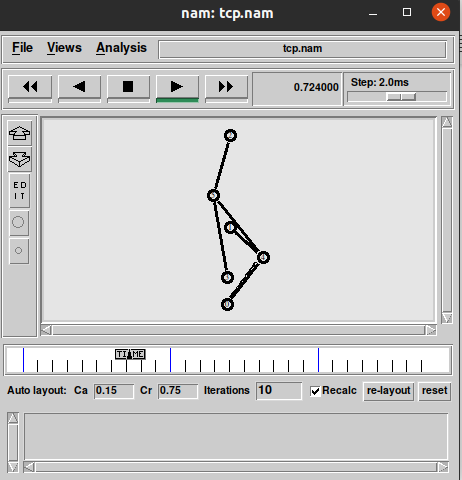
$ns at 2.5 "$ftp stop"

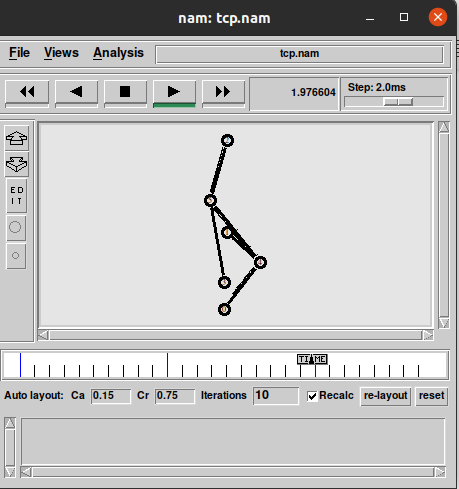
$ns at 3 "finish"

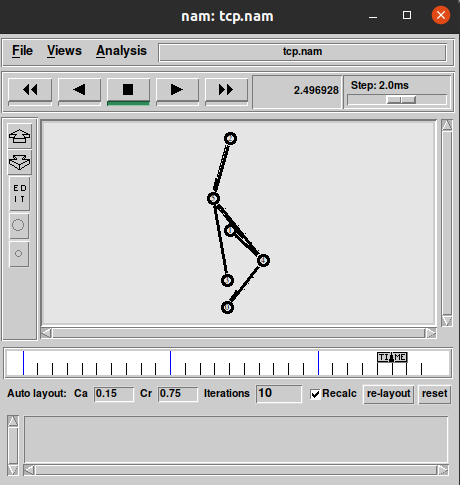
$ns run

**OUTPUT:**









#### RESULT:

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

1. **LINK STATE ROUTING PROTOCOL**

### AIM:

To simulate a link failure and to observe link state routing protocol in action.

#### ALGORITHM:

1. Create a simulator object
2. Set routing protocol to link state routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create four nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a quad node.
8. Add TCP agent for node n0
9. Create FTP traffic on top of TCP and set traffic parameters.
10. Add a sink agent to node n3
11. Add UDP agent for node n2
12. Create CBR traffic on top of UDP and set traffic parameters.
13. Connect source and the sink
14. Schedule events as follows:
    1. Start traffic flow at 0.0
    2. Down the link n1-n3 at 1.0
    3. Up the link n1-n3 at 2.0
    4. Call finish procedure at 5.0
15. Start the scheduler
16. Observe the traffic route when link is up and down
17. View the simulated events and trace file analyze it
18. Stop

#### PROGRAM :

set ns [new Simulator] set nf [open out.nam w]

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

$ns trace-all $tr proc finish {} { global nf ns tr

$ns flush-trace close $tr

exec nam out.nam & exit 0

}

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

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

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

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

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

$ns duplex-link-op $n1 $n3 orient right

$ns duplex-link-op $n2 $n1 orient right-up set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp set ftp [new Application/FTP]

$ftp attach-agent $tcp

set sink [new Agent/TCPSink]

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

$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp set null [new Agent/Null]

$ns attach-agent $n3 $null

$ns connect $tcp $sink

$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3

$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto LS

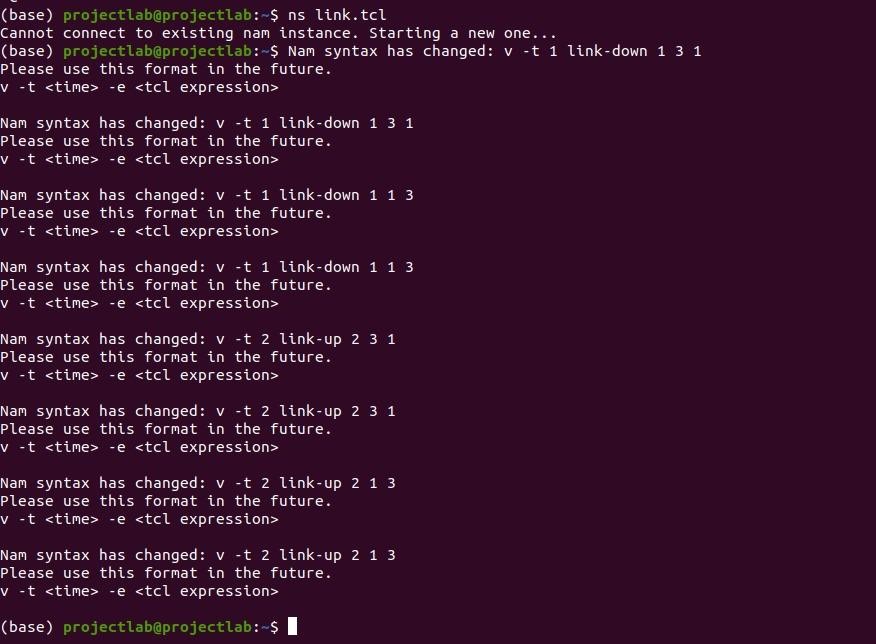
$ns at 0.0 "$ftp start"

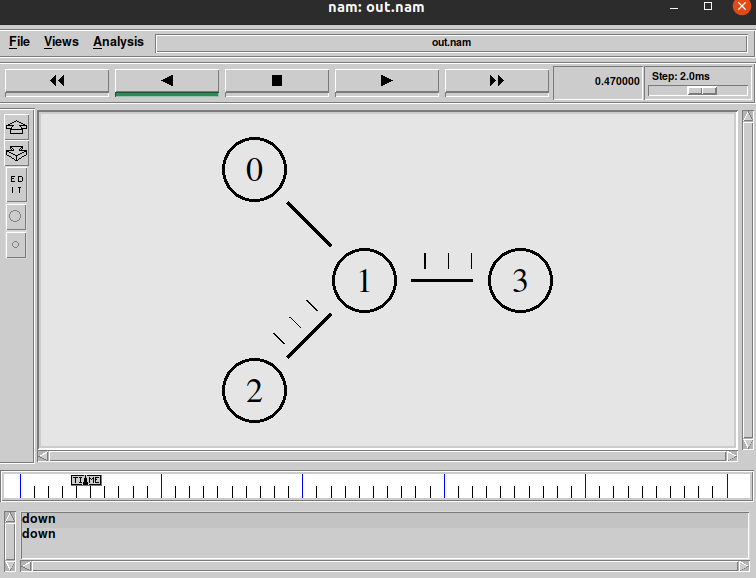
$ns at 0.0 "$cbr start"

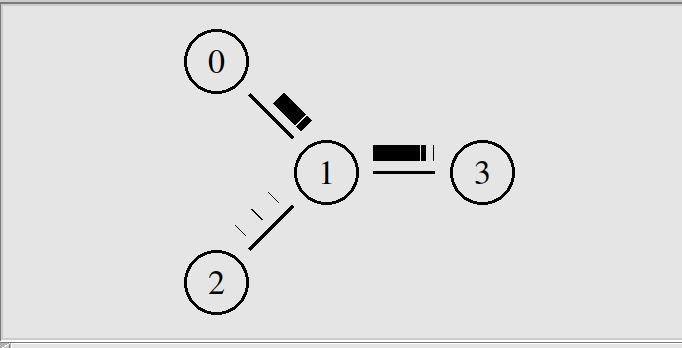
$ns at 5.0 "finish"

$ns run

#### OUTPUT:







1. **DISTANCE VECTOR ROUTING PROTOCOL**

**AIM:**

To simulate a link failure and to observe distance vector routing protocol in action.

#### ALGORITHM:

1. Create a simulator object
2. Set routing protocol to Distance Vector routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create eight nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a octagon
8. Add UDP agent for node n1
9. Create CBR traffic on top of UDP and set traffic parameters.
10. Add a sink agent to node n4
11. Connect source and the sink
12. Schedule events as follows:
    1. Start traffic flow at 0.5
    2. Down the link n3-n4 at 1.0
    3. Up the link n3-n4 at 2.0
    4. Stop traffic at 3.0
    5. Call finish procedure at 5.0
13. Start the scheduler
14. Observe the traffic route when link is up and down
15. View the simulated events and trace file analyze it
16. Stop the program.

#### PROGRAM:

set ns [new Simulator]

$ns rtproto DV

set nf [open out.nam w]

$ns namtrace-all $nf set nt [open trace.tr w]

$ns trace-all $nt proc finish {} {

global ns nf

$ns flush-trace close $nf

exec nam -a out.nam & exit 0

}

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]

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

$ns duplex-link $n2 $n3 1Mb 10ms DropTail

$ns duplex-link $n3 $n4 1Mb 10ms DropTail

$ns duplex-link $n4 $n5 1Mb 10ms DropTail

$ns duplex-link $n5 $n6 1Mb 10ms DropTail

$ns duplex-link $n6 $n7 1Mb 10ms DropTail

$ns duplex-link $n7 $n8 1Mb 10ms DropTail

$ns duplex-link $n8 $n1 1Mb 10ms DropTail

$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 set udp0 [new Agent/UDP]

$ns attach-agent $n1 $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 $n4 $null0

$ns connect $udp0 $null0

$ns at 0.0 "$n1 label Source"

$ns at 0.0 "$n4 label Destination"

$ns at 0.5 "$cbr0 start"

$ns rtmodel-at 1.0 down $n3 $n4

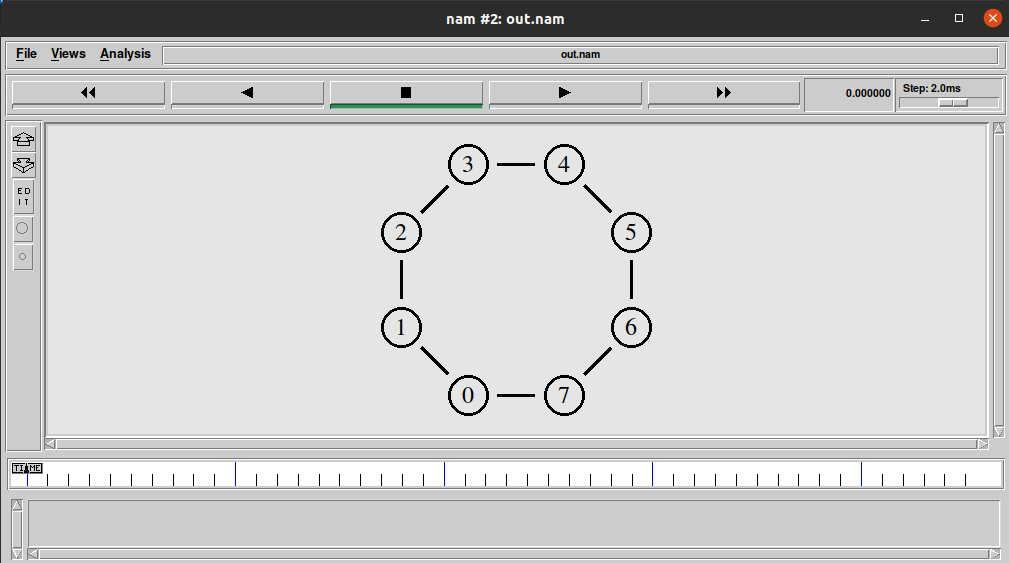
$ns rtmodel-at 2.0 up $n3 $n4

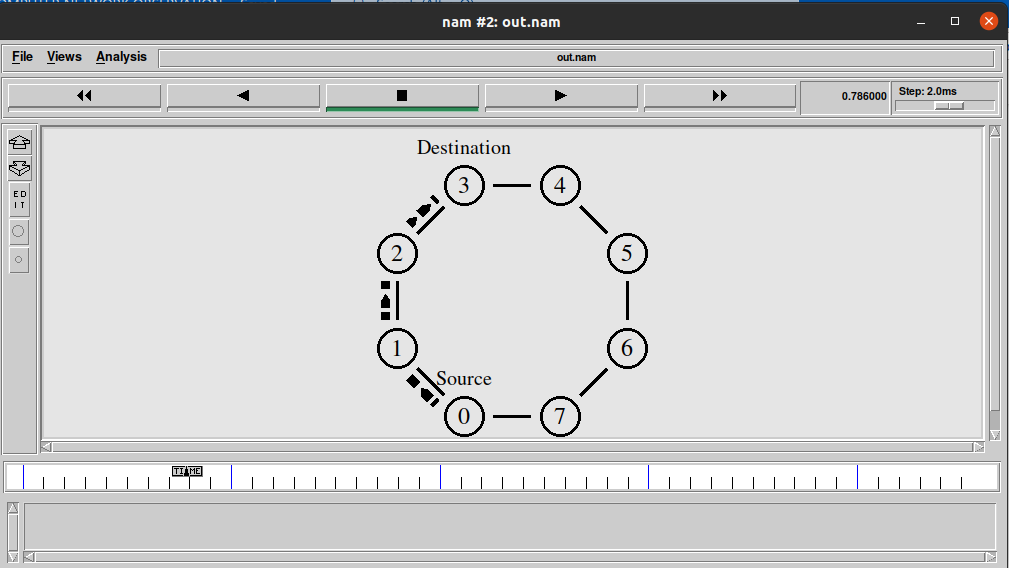
$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"

$ns run

## OUTPUT:





Graphical user interface, application

Description automatically generated

