**Lab Manual**

**CSL 332**

**Network Programming Lab**

## 

## Experiment 1

**Getting started with Basics of Network configurations files and Networking Commands in Linux.**

The important network configuration files in Linux operating systems are

### /etc/hosts

This file is used to resolve hostnames on small networks with no DNS server. This text file contains a mapping of an IP address to the corresponding host name in each line. This file also contains a line specifying the IP address of the loopback device i.e, *127.0.0.1* is mapped to *localhost*.

A typical *hosts* file is as shown

127.0.0.1 localhost

127.0.1.1 anil-300E4Z-300E5Z-300E7Z

### /etc/resolv.conf

This configuration file contains the IP addresses of DNS servers and the search domain.

A sample file is shown

# DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN

nameserver 127.0.1.1

### /etc/sysconfig/network

This configuration file specifies routing and host information for all network interfaces. It contains directives that are global specific. For example if NETWORKING=yes, then /etc/init.d/network activates network devices.

### /etc/nsswitch.conf

This file includes database search entries. The directive specifies which database is to be searched first. The important Linux networking commands are

### ifconfig

This command gives the configuration of all interfaces in the system. It can be run with an interface name to get the details of the interface. ifconfig *wlan0*

Link encap:Ethernet HWaddr b8:03:05:ad:6b:23

inet addr:192.168.43.15 Bcast:192.168.43.255 Mask:255.255.255.0

inet6 addr: 2405:204:d206:d3b1:ba03:5ff:fead:6b23/64 Scope:Global inet6 addr: fe80::ba03:5ff:fead:6b23/64 Scope:Link

inet6 addr: 2405:204:d206:d3b1:21ee:5665:de59:bd4e/64 Scope:Global UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:827087 errors:0 dropped:0 overruns:0 frame:0 TX packets:433391 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000

RX bytes:1117797710 (1.1 GB) TX bytes:53252386 (53.2 MB)

This gives the IP address, subnet mask, and broadcast address of the wireless LAN adapter. Also tells that it can support multicasting.

If eth0 is given as the parameter, the command gives the details of the Ethernet adapter.

### netstat

This command command gives network status information.

Netstat -*i*

Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg

| eth0 | 1500 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | BMU |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lo | 65536 0 | 12166 | 0 | 0 | 0 | 12166 | 0 | 0 | 0 | LRU |
| wlan0 | 1500 0 | 827946 | 0 | 0 | 0 | 434246 | 0 | 0 | 0 | BMRU |

As shown above, the command with -*i* flag provides information on the interfaces. lo stands for loopback interface.

### ping

This is the most commonly used command for checking connectivity.

ping [*www.google.com*](http://www.google.com/)

PING [www.google.com](http://www.google.com/) (172.217.163.36) 56(84) bytes of data.

64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=1 ttl=53 time=51.4 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=2 ttl=53 time=50.3 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=3 ttl=53 time=48.5 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=4 ttl=53 time=59.8 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=5 ttl=53 time=57.8 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=6 ttl=53 time=59.2 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=7 ttl=53 time=68.2 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp\_seq=8 ttl=53 time=58.8 ms

^C

--- [www.google.com](http://www.google.com/) ping statistics ---

8 packets transmitted, 8 received, 0% packet loss, time 7004ms rtt min/avg/max/mdev = 48.533/56.804/68.266/6.030 ms

A healthy connection is determined by a steady stream of replies with consistent times. Packet loss is shown by discontinuity of sequence numbers. Large scale packet loss indicates problem along the

path.

## Experiment 2

**To familiarize and understand the use and functioning of System Calls used for Operating system and network programming in Linux.**

**Some system calls of Linux operating systems**

## ps

This command tells which all processes are running on the system when *ps* runs. ps -ef

UID PID PPID C STIME TTY TIME CMD

| root | 1 | 0 | 0 | 13:55 ? | 00:00:01 | /sbin/init |
| --- | --- | --- | --- | --- | --- | --- |
| root | 2 | 0 | 0 | 13:55 ? | 00:00:00 | [kthreadd] |
| root | 3 | 2 | 0 | 13:55 ? | 00:00:00 | [ksoftirqd/0] |
| root | 4 | 2 | 0 | 13:55 ? | 00:00:01 | [kworker/0:0] |
| root | 5 | 2 | 0 | 13:55 ? | 00:00:00 | [kworker/0:0H] |
| root | 7 | 2 | 0 | 13:55 ? | 00:00:00 | [rcu\_sched] |
| root | 8 | 2 | 0 | 13:55 ? | 00:00:00 | [rcuos/0] |



This command gives processes running on the system, the owners of the processes and the names of the processes. The above result is an abridged version of the output.

## fork

This system call is used to create a new process. When a process makes a *fork* system call, a new process is created which is identical to the process creating it. The process which calls *fork* is called the parent process and the process that is created is called the child process. The child and parent processes are identical, i.e, child gets a copy of the parent's data space, heap and stack, but have different physical address spaces. Both processes start execution from the line next to *fork*. Fork returns the process id of the child in the parent process and returns 0 in the child process.

#include<stdio.h> void main()

{

int pid;

pid = fork(); if(pid > 0)

{

printf (“ Iam parent\n”);

}

else

{

printf(“Iam child\n”);

}

}

The parent process prints the first statement and the child prints the next statement.

### exec

New programs can be run using *exec* system call. When a process calls exec, the process is completely replaced by the new program. The new program starts executing from its main function.

A new process is not created, process id remains the same, and the current process's text, data, heap, and stack segments are replaced by the new program. *exec* has many flavours one of which is *execv.*

*execv* takes two parameters. The first is the pathname of the program that is going to be executed. The second is a pointer to an array of pointers that hold the addresses of arguments. These arguments are the command line arguments for the new program.

## wait

When a process terminates, its parent should receive some information reagarding the process like the process id, the termination status, amount of CPU time taken etc. This is possible only if the parent process waits for the termination of the child process. This waiting is done by calling the *wait* system call. When the child process is running, the parent blocks when *wait* is called. If the child terminates normally or abnormally, *wait* immedaitely returns with the termination status of the child. The wait system call takes a parameter which is a pointer to a location in which the termination status is stored.

## exit

When *exit* function is called, the process undergoes a normal termination.

## open

This system call is used to open a file whose pathname is given as the first parameter of the function. The second parameter gives the options that tell the way in which the file can be used.

open(filepathname , O\_RDWR);

This causes the file to be read or written. The function returns the file descriptor of the file.

## read

This system call is used to read data from an open file. read(fd, buffer, sizeof(buffer));

The above function reads sizeof(buffer) bytes into the array named buffer. If the end of file is encountered, 0 is returned, else the number of bytes read is returned.

## write

Data is written to an open file using *write* function. write(fd, buffer, sizeof(buffer));

### System calls for network programming in Linux

1. **Creating a socket**

**int** *socket* (**int** *domain*, **int** *type*, **int** *protocol*);

This sytem call creates a socket and returns a socket descriptor. The *domain* parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. In this program the AF\_INET family is used. The *type*

parameter indicates the communication semantics. SOCK\_STREAM is used for tcp connection while SOCK\_DGRAM is used for udp connection. The *protocol* parameter specifies the protocol used and is always 0. The header files used are <sys/types.h> and <sys/socket.h>.

## Experiment 3

**Implementation of Client-Server communication using Socket Programming and TCP as transport layer protocol**

**Aim**: Client sends a string to the server using tcp protocol. The server reverses the string and returns it to the client, which then displays the reversed string.

### Description:

*Steps for creating a TCP connection by a client are:*

### Creation of client socket

**int socket(int domain, int type, int protocol);**

This function call creates a socket and returns a socket descriptor. The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. In this program, the domain **AF\_INET** is used. The socket has the indicated type, which specifies the communication semantics. **SOCK\_STREAM** type provides sequenced, reliable, two-way, connection based byte streams. The **protocol** field specifies the protocol used. We always use 0. If the system call is a failure, a -1 is returned. The header files used are **sys/types.h** and **sys/socket.h**.

### Filling the fields of the server address structure.

The socket address structure is of type **struct sockaddr\_in**.

struct sockaddr\_in {

};

u\_short sin\_family; u\_short sin\_port;

struct in\_addr sin\_addr;

char sin\_zero[8]; /\*unused, always zero\*/

struct in\_addr {

};

u\_long s\_addr;

The fields of the socket address structure are

**sin\_family** which in our case is AF\_INET

**sin\_port** which is the port number where socket binds

**sin\_addr** which is the IP address of the server machine The header file that is to be used is **netinet/in.h**

*Example*

### struct sockaddr\_in servaddr; servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(port\_number);

Why htons is used ?. Numbers on different machines may be represented differently ( big-endian machines and little-endian machines). In a little-endian machine the low order byte of an integer appears at the lower address; in a big-endian machine instead the low order byte appears at the higher address. Network order, the order in which numbers are sent on the internet is big-endian. It is necessary to ensure that the right representation is used on each machine. Functions are used to convert from host to network form before transmission- htons for short integers and htonl for long integers.

The value for servaddr.sin\_addr is assigned using the following function

### inet\_pton(AF\_INET, “IP\_Address”, &servaddr.sin\_addr);

The binary value of the dotted decimal IP address is stored in the field when the function returns.

### Binding of the client socket to a local port

This is optional in the case of client and we usually do not use the *bind* function on the client side.

1. Connection of client to the server

A server is identified by an IP address and a port number. The connection operation is used on the client side to identify and start the connection to the server.

### int connect(int sd, struct sockaddr \* addr, int addrlen);

sd – file descriptor of local socket

addr – pointer to protocol address of other socket addrlen – length in bytes of address structure

The header files to be used are sys/types.h and sys/socket.h It returns 0 on sucess and -1 in case of failure.

### Reading from socket

In the case of TCP connection reading from a socket can be done using the *read* system call

### int read(int sd, char \* buf, int length);

1. writing to a socket

In the case of TCP connection writing to a socket can be done using the *write* system call

### int write( int sd, char \* buf, int length);

1. **closing the connection**

The connection can be closed using the close system call

### int close( int sd);

*Steps for TCP Connection for server*

### Creating a listening socket

**int socket( int domain, int type, int protocol);**

This system call creates a socket and returns a socket descriptor. The *domain* field used is **AF\_INET**. The socket type is **SOCK\_STREAM**. The **protocol** field is 0. If the system call is a failure, a -1 is returned. Header files used are **sys/types.h** and **sys/socket.h**.

### Binding to a local port

**int bind(int sd, struct sockaddr \* addr, int addrlen);**

This call is used to specify for a socket the protocol port number where it will wait for messages. A call to *bind* is optional on the client side, but required on the server side. The first field is the *socket* descriptor of local socket. Second is a pointer to protocol address structure of this socket. The third is the length in bytes of the structure referenced by *addr.* This system call returns an integer. It is 0 for success and -1 for failure. The header files are **sys/types.h** and **sys/socket.h**.

### Listening on the port

The listen function is used on the server in the connection oriented communication to prepare a socket to accept messages from clients.

### int listen(int fd, int qlen);

**fd –** file descriptor of a socket that has already been bound

**qlen –** specifies the maximum number of messages that can wait to be processed by the server while the server is busy servicing another request. Usually it is taken as 5. The header files used are *sys/types.h* and *sys/socket.h.* This function returns 0 on success and -1 on failure.

### Accepting a connection from the client

The accept function is used on the server in the case of connection oriented communication to accept a connection request from a client.

### int accept( int fd, struct sockaddr \* addressp, int \* addrlen);

The first field is the descriptor of server socket that is listening. The second parameter *addressp* points to a socket address structure that will be filled by the address of calling client when the function returns. The third parameter *addrlen* is an integer that will contain the actual length of address structure of client. It returns an integer that is a descriptor of a new socket called the connection socket. Server sockets send data and read data from this socket. The header files used are *sys/types.h* and *sys/socket.h.*

### Algorithm

Client

1. Create socket
2. Connect the socket to the server
3. Read the string to be reversed from the standard input and send it to the server Read the matrices from the standard input and send it to server using socket
4. Read the reversed string from the socket and display it on the standard output Read product matrix from the socket and display it on the standard output
5. Close the socket Server
6. Create listening socket
7. bind IP address and port number to the socket
8. listen for incoming requests on the listening socket
9. accept the incoming request
10. connection socket is created when *accept* returns
11. Read the string using the connection socket from the client
12. Reverse the string
13. Send the string to the client using the connection socket
14. close the connection socket
15. close the listening socket

### Client Program

#include<stdio.h> #include<sys/types.h> #include<sys/socket.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<string.h> #include<stdlib.h> #include<unistd.h>

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

{

struct sockaddr\_in server; int sd ;

char buffer[200];

if((sd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("Socket failed:"); exit(1);

}

// server socket address structure initialisation

bzero(&server, sizeof(server) ); server.sin\_family = AF\_INET; server.sin\_port = htons(atoi(argv[2]));

inet\_pton(AF\_INET, argv[1], &server.sin\_addr);

if(connect(sd, (struct sockaddr \*)&server, sizeof(server))< 0)

{

perror("Connection failed:"); exit(1);

}

fgets(buffer, sizeof(buffer), stdin); buffer[strlen(buffer) - 1] = '\0';

write (sd,buffer, sizeof(buffer)); read(sd,buffer, sizeof(buffer));

printf("%s\n", buffer);

close(fd);

}

### Server Program

#include<stdio.h> #include<sys/types.h> #include<sys/socket.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<string.h> #include<stdlib.h> #include<unistd.h>

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

{

struct sockaddr\_in server, cli; int cli\_len;

int sd, n, i, len; int data, temp;

char buffer[100];

if((sd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("Socket failed:"); exit(1);

}

// server socket address structure initialisation

bzero(&server, sizeof(server) ); server.sin\_family = AF\_INET; server.sin\_port = htons(atoi(argv[1]));

server.sin\_addr.s\_addr = htonl(INADDR\_ANY);

if(bind(sd, (struct sockaddr\*)&server, sizeof(server)) < 0)

{

perror("bind failed:"); exit(1);

}

listen(sd,5);

if((data = accept(sd , (struct sockaddr \*) &cli, &cli\_len)) < 0)

{

perror("accept failed:"); exit(1);

}

read(data,buffer, sizeof(buffer));

len = strlen(buffer);

for( i =0; i<= len/2; i++)

{

temp = buffer[i];

buffer[i] = buffer[len - 1-i]; buffer[len-1-i] = temp;

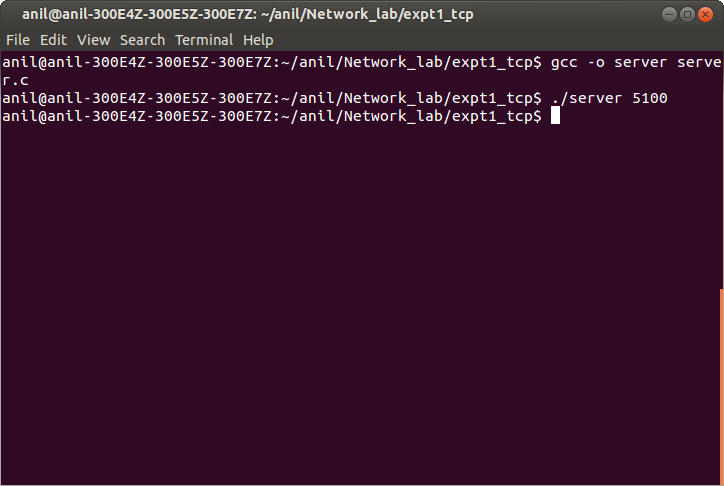
}

write (data,buffer, sizeof(buffer));

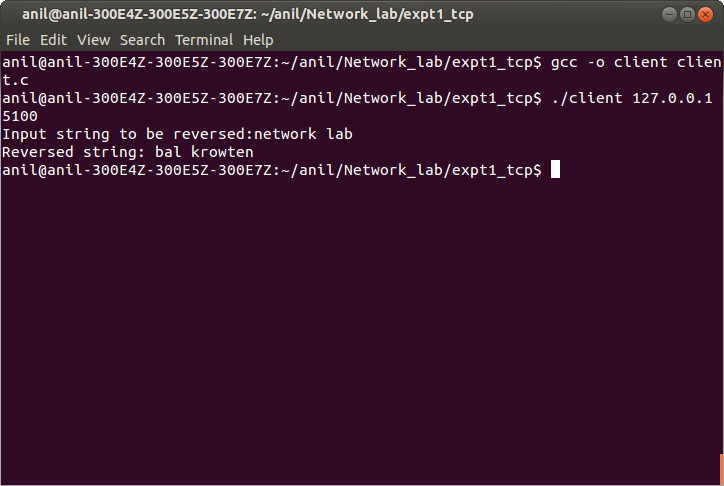
close(data); close(sd);

}

### Output

**Server**

**Client**



## Experiment 4

**Implementation of Client-Server communication using Socket Programming and UDP as transport layer protocol**

**Aim**: Client sends two matrices to the server using udp protocol. The server multiplies the matrices and sends the product to the client, which then displays the product matrix.

### Description:

*Steps for transfer of data using UDP*

### Creation of UDP socket

The function call for creating a UDP socket is

### int socket(int domain, int type, int protocol);

The *domain* parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. In this program, the domain **AF\_INET** is used. The next field *type* has the value **SOCK\_DGRAM.** It supports datagrams (connectionless, unreliable messages of a fixed maximum length). The *protocol* field specifies the protocol used. We always use 0. If the socket function call is successful, a socket descriptor is returned. Otherwise -1 is returned. The header files necessary for this function call are **sys/types.h** and **sys/socket.h**.

### Filling the fields of the server address structure.

The socket address structure is of type **struct sockaddr\_in**.

struct sockaddr\_in {

};

u\_short sin\_family; u\_short sin\_port;

struct in\_addr sin\_addr;

char sin\_zero[8]; /\*unused, always zero\*/

struct in\_addr {

};

u\_long s\_addr;

The fields of the socket address structure are

**sin\_family** which in our case is AF\_INET

**sin\_port** which is the port number where socket binds

**sin\_addr** is used to store the IP address of the server machine and is of type **struct in\_addr**

The header file that is to be used is **netinet/in.h**

The value for **servaddr.sin\_addr** is assigned using the following function

### inet\_pton(AF\_INET, “IP\_Address”, &servaddr.sin\_addr);

The binary value of the dotted decimal IP address is stored in the field when the function returns.

### Binding of a port to the socket in the case of server

This call is used to specify for a socket the protocol port number where it will wait for messages. A call to bind is optional in the case of client and compulsory on the server side.

### int bind(int sd, struct sockaddr\* addr, int addrlen);

The first field is the socket descriptor. The second is a pointer to the address structure of this socket. The third field is the length in bytes of the size of the structure referenced by *addr.* The header files are **sys/types.h** and **sys/socket.h**. This function call returns an integer, which is 0 for success and -1 for failure.

### Receiving data

**ssize\_t recvfrom(int s, void \* buf, size\_t len, int flags, struct sockaddr \* from, socklen\_t \* fromlen);**

The *recvfrom* calls are used to receive messages from a socket, and may be used to receive data on a socket whether or not it is connection oriented. The first parameter *s* is the socket descriptor to read from. The second parameter *buf* is the buffer to read information into. The third parameter *len* is the maximum length of the buffer. The fourth parameter is *flag.* It is set to zero. The fifth parameter *from* is a pointer to **struct sockaddr** variable that will be filled with the IP address and port of the orginating machine. The sixth parameter *fromlen* is a pointer to a local int variable that should be initialized to **sizeof(struct sockaddr)**. When the function returns, the integer variable that *fromlen* points to will contain the actual number of bytes that is contained in the socket address structure. The header files required are **sys/types.h** and **sys/socket.h**. When the function returns, the number of bytes received is returned or -1 if there is an error.

### Sending data

*sendto-* sends a message from a socket

### ssize\_t sendto(int s, const void \* buf, size\_t len, int flags, const struct sockaddr \* to, socklen\_t tolen);

The first parameter *s* is the socket descriptor of the sending socket. The second parameter *buf* is the array which stores data that is to be sent. The third parameter *len* is the length of that data in bytes. The

fourth parameter is the *flag* parameter. It is set to zero. The fifth parameter *to* points to a variable that contains the destination IP address and port. The sixth parameter *tolen* is set to **sizeof(struct sockaddr)**. This function returns the number of bytes actually sent or -1 on error. The header files used are **sys/types.h** and **sys/socket.h**.

### Algorithm

Client

1. Create socket
2. Read the matrices from the standard input and send it to server using socket
3. Read product matrix from the socket and display it on the standard output
4. Close the socket Server
5. Create socket
6. bind IP address and port number to the socket
7. Read the matrices socket from the client using socket
8. Find product of matrices
9. Send the product matrix to the client using socket
10. close the socket

### Client program

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

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

{

int i,j,n;

int sock\_fd;

struct sockaddr\_in servaddr;

int matrix\_1[10][10], matrix\_2[10][10], matrix\_product[10][10]; int size[2][2];

int num\_rows\_1, num\_cols\_1, num\_rows\_2, num\_cols\_2; if(argc != 3)

{

fprintf(stderr, "Usage: ./client IPaddress\_of\_server port\n"); exit(1);

}

printf("Enter the number of rows of first matrix\n"); scanf("%d", &num\_rows\_1);

printf("Enter the number of columns of first matrix\n"); scanf("%d", &num\_cols\_1);

printf("Enter the values row by row one on each line\n" ); for ( i = 0; i < num\_rows\_1; i++)

for( j=0; j<num\_cols\_1; j++)

{

scanf("%d", &matrix\_1[i][j]);

}

size[0][0] = num\_rows\_1; size[0][1] = num\_cols\_1;

printf("Enter the number of rows of second matrix\n"); scanf("%d", &num\_rows\_2);

printf("Enter the number of columns of second matrix\n"); scanf("%d", &num\_cols\_2);

if( num\_cols\_1 != num\_rows\_2)

{

printf("MATRICES CANNOT BE MULTIPLIED\n");

exit(1);

}

printf("Enter the values row by row one on each line\n");

for (i = 0; i < num\_rows\_2; i++) for(j=0; j<num\_cols\_2; j++)

{

scanf("%d", &matrix\_2[i][j]);

}

size[1][0] = num\_rows\_2; size[1][1] = num\_cols\_2;

if((sock\_fd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("Cannot create socket\n"); exit(1);

}

bzero((char\*)&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(atoi(argv[2]));

inet\_pton(AF\_INET, argv[1], &servaddr.sin\_addr);

// SENDING MATRIX WITH SIZES OF MATRICES 1 AND 2

n = sendto(sock\_fd, size, sizeof(size),0, (struct sockaddr\*)&servaddr, sizeof(servaddr)); if( n < 0)

{

perror("error in matrix 1 sending"); exit(1);

}

// SENDING MATRIX 1

n = sendto(sock\_fd, matrix\_1, sizeof(matrix\_1),0, (struct sockaddr\*)&servaddr, sizeof(servaddr));

if( n < 0)

{

perror("error in matrix 1 sending"); exit(1);

}

// SENDING MATRIX 2

n = sendto(sock\_fd, matrix\_2, sizeof(matrix\_2),0, (struct sockaddr\*)&servaddr, sizeof(servaddr)); if( n < 0)

{

perror("error in matrix 2 sending"); exit(1);

}

if((n=recvfrom(sock\_fd, matrix\_product, sizeof(matrix\_product),0, NULL, NULL)) == -1)

{

perror("read error from server:"); exit(1);

}

printf("\n\nTHE PRODUCT OF MATRICES IS \n\n\n");

for( i=0; i < num\_rows\_1; i++)

{

for( j=0; j<num\_cols\_2; j++)

{

printf("%d ",matrix\_product[i][j]);

}

printf("\n");

}

close(sock\_fd);

}

### Server Program

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

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

{

int n;

int sock\_fd; int i,j,k;

int row\_1, row\_2, col\_1, col\_2; struct sockaddr\_in servaddr, cliaddr;

int len = sizeof(cliaddr);

int matrix\_1[10][10], matrix\_2[10][10], matrix\_product[10][10]; int size[2][2];

if(argc != 2)

{

fprintf(stderr, "Usage: ./server port\n"); exit(1);

}

if((sock\_fd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("Cannot create socket\n"); exit(1);

}

bzero((char\*)&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(atoi(argv[1]));

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

if(bind(sock\_fd, (struct sockaddr\*)&servaddr, sizeof(servaddr)) < 0)

{

perror("bind failed:"); exit(1);

}

// MATRICES RECEIVE

if((n = recvfrom(sock\_fd, size, sizeof(size), 0, (struct sockaddr \*)&cliaddr, &len)) == -1)

{

perror("size not received:"); exit(1);

}

// RECEIVE MATRIX 1

if((n = recvfrom(sock\_fd, matrix\_1, sizeof(matrix\_1), 0, (struct sockaddr \*)&cliaddr, &len)) == -1)

{

perror("matrix 1 not received:"); exit(1);

}

// RECEIVE MATRIX 2

if((n = recvfrom(sock\_fd, matrix\_2, sizeof(matrix\_2), 0, (struct sockaddr \*)&cliaddr, &len)) == -1)

{

perror("matrix 2 not received:"); exit(1);

}

row\_1 = size[0][0]; col\_1 = size[0][1]; row\_2 = size[1][0]; col\_2 = size[1][1];

for (i =0; i < row\_1 ; i++) for (j =0; j <col\_2; j++)

{

matrix\_product[i][j] = 0;

}

for(i =0; i< row\_1 ; i++) for(j=0; j< col\_2 ; j++) for (k=0; k < col\_1; k++)

{

matrix\_product[i][j] += matrix\_1[i][k]\*matrix\_2[k][j];

}

n = sendto(sock\_fd, matrix\_product, sizeof(matrix\_product),0, (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

if( n < 0)

{

perror("error in matrix product sending"); exit(1);

}

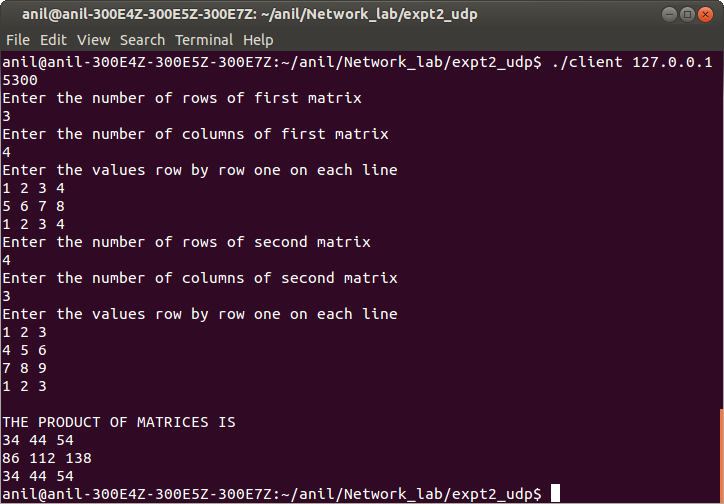
close(sock\_fd);

}

## Output

### Server

**Client**



## Experiment 5

**Implementation and simulation of link state protocol Aim:** To implement and simulate link state protocol

### Description:

Routing algorithms can be classified as global or centralized. A global routing algorithm computes the least cost path between a source and destination using knowledge about the entire network. Global algorithm has complete information about connectivity and link costs. Such algorithms are called link state algorithms.

### Algorithm:

(adapted from

It is the Dijkstra algorithm. This algorithm finds the shortest (least cost paths) from the source *u* to every other node in the network.

D(v) : cost of the least cost path fromthe source node to the destination node *v* as of this iteration of the algorithm

p(v): previous node of *v* along the current least cost path from the source to *v*

N' : subset of nodes. *v* is in N' if the least cost path from the source to *v* is definitely known Iniltialization :

N' = {u}

for all nodes *v*

if v is a neighbour of u then D(v) = c(u,v)

else D(v) = ∞

do {

find *w* not in N' such that D(w) is a minimum add *w* to N'

update D(v) for each neighbour *v* of *w* and not in N'.

D(v) = min{D(v) , D(w) + c(w,v)}

} while ( N' != N)

### Program

#include<stdio.h> int cost[10][10]; int dist[10];

int arr[20];

void djikstra(int); int search(int);

int length\_of( int \* );

void print\_route(int, int);

int prev[20];

int order\_arr[20]; int src;

main()

{

int num\_nodes,i, j, d;

printf("Enter number of nodes:"); scanf("%d", &num\_nodes); printf("Enter the source node:"); scanf("%d", &src);

printf("Enter the cost matrix, For infinity enter 999\n");

for(i=0; i <num\_nodes; i++) for(j=0; j <num\_nodes; j++) scanf("%d", &cost[i][j]);

djikstra( num\_nodes);

}

void djikstra(int \_num\_nodes )

{

int i, min\_val, l; int k =0;

int m =0;

int min =999; int last;

int neighbour[20];

for(i = 0; i <20; i++)

{

arr[i] = -1;

neighbour[i] = -1;

order\_arr[i] = -1;

prev[i] = -1;

}

arr[0] = src; last =0;

for(i = 0; i <\_num\_nodes; i++)

{

if(i != src)

{

if( cost[src][i] < 999)

{

dist[i] = cost[src][i]; prev[i] = src;

}

else

dist[i] = 999;

}

else

{

dist[i] = 0;

}

}

do {

for(i=0; i < \_num\_nodes; i++)

{

if(search(i) == 0)

{

if(dist[i] < min)

{

min = dist[i]; min\_val = i;

}

}

}

last++;

arr[last] = min\_val;

for(i=0; i<\_num\_nodes; i++)

{

if(search(i) == 0)

{

if(cost[min\_val][i] < 999)

{

neighbour[m] = i; m++;

}

}

}

m =0;

while(neighbour[m] != -1)

{

if(dist[min\_val] + cost[min\_val][neighbour[m]] < dist[neighbour[m]])

{

dist[neighbour[m]] = dist[min\_val] + cost[min\_val][neighbour[m]]; prev[neighbour[m]] = min\_val;

}

m++;

}

m=0;

for(i = 0; i <\_num\_nodes; i++) neighbour[i] = -1;

min =999;

min\_val = -1;

}while( length\_of(arr) != \_num\_nodes); i =1;

l=1;

while( i < \_num\_nodes)

{

print\_route(i, l);

printf("[ distance = %d]", dist[i]); printf("\n");

i++; l++;

for(k = 0; k <20; k++)

order\_arr[k] = -1;

}

}

void print\_route( int \_i, int \_l)

{

int begin; int \* ptr;

int h, len, temp; static int inc[20];

if( \_i == src)

{

ptr = order\_arr; while(\*ptr != -1) ptr++;

len = ptr-order\_arr; for(h =0; h < len/2; h++)

{

temp = order\_arr[h];

order\_arr[h] = order\_arr[len - h -1]; order\_arr[len-h-1] = temp;

}

ptr = order\_arr; printf("%d", src);

while(\*ptr != -1)

{

printf("->%d ", \*ptr); ptr++;

}

return;

}

else

{

order\_arr[inc[\_l]] = \_i; inc[\_l]++; print\_route(prev[\_i], \_l);

}

}

int search(\_i)

{

int i =0;

while (arr[i] != -1)

{

if(\_i == arr[i]) break;

else

i++;

}

if(arr[i] == -1)

return 0; else

return 1;

}

int length\_of( int \_arr[])

{

int i=0; while(\_arr[i] != -1)

i++;

return i;

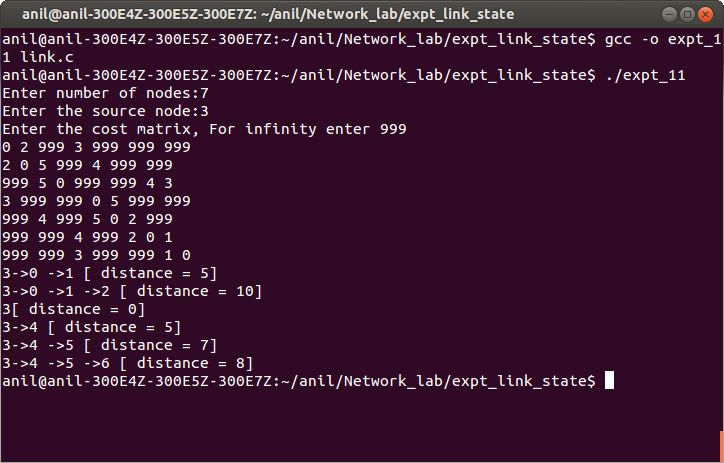
}

### Output

Figure shows a network where the link state algorithm is applied.

### FIGURE

When finding out the cost matrix use 999 in the place of ∞.



## Experiment 6

**Implemention and simulation of algorithm for Distance vector routing protocol. Aim:** To implement and simulate algorithm for Distance vector routing protocol

### Description:

This algorithm is iterative, and distributed. Each node receives information fromits directly attached neighbours, performs some calculations and returns the result the result to its neighbouring nodes. This process of updating the information goes on until there is no exchange of information between neighbours.

### Algorithm:

(adapted from Computer Networking – A top down approach by Kurose and Rose) Bellman Ford algoithm is applied.

Let dx(y) be the cost of the least cost path from node x to node y. Then Bellman Ford equation states that

dx(y) = min{ c(x,v) + dv(y) }

v

where *v* is a neighbour of node *x.* dv(y) is the cost of the least cost path from *v* to *y*. c(x,v) is the cost from *x* to neighbour *v.* The least cost path has a value equal to minimum of c(x,v) + dv(y) over all its neighbours *v*. The solution of Bellman Ford equation provides entries in node x's forwarding table.

Distance vector (DV) algorithm At each node *x*

Initialization:

for all destinations *y* in N:

Dx(y) = c(x,y) /\* if *y* is not a neighbour of *x,* then c(x,y) = ∞ \*/ for each neighbour *w*,

send distance vector Dx = { Dx(y): y in N} to *w*

loop:

for each *y* in *N*:

Dx(y) = min { c(x,v) + Dv(y) }

v

If Dx(y) changed for any destination *y* send distance vector Dx = {Dx(y) : *y* in N} to all neighbours.

forever

### Program

The simulation is done on the principle of a chat server given in Expt 9. Each node is considered as a client that connects to a server. Each node reads the cost matrix and constructs the distance vector matrix which is a 3D matrix (RT), where the first element denotes the number of the node. After entering the cost matrix for each node, each node exchanges its distance vector matrix with its neighbours. This process goes on till there are no changes in the dsitance vector matrix table for each node. In the cost matrix infinite cost between two nodes is represented by 999.

*client.c*

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

struct message {

int to\_node; int from\_node;

int RT[20][20][20];

} ;

struct message mesg, mesg\_from; main(int argc, char \* argv[])

{

int i,j,n, h,k; int num\_nodes;

int sock\_fd, max\_fd, nready, fd[2]; char buffer[100], line[100];

struct sockaddr\_in servaddr; fd\_set rset;

int node\_sel, temp1;

int node, p;

int from\_node\_it; int to\_node\_it, t;

int prev[20];

int cost[10][10];

int N[20][20];

temp1 = 999; h=0;

printf("Enter number of nodes:"); scanf("%d", &num\_nodes);

printf("Enter the cost matrix, For infinity enter 999\n"); for(i=0; i <num\_nodes; i++)

for(j=0; j <num\_nodes; j++)

scanf("%d", &cost[i][j]);

// DISTANCE VECTOR MATRIX FOR EACH NODE

for(i= 0; i < num\_nodes; i++) for(j=0; j < num\_nodes; j++) for(k=0; k < num\_nodes; k++)

{

if(i==j)

mesg.RT[i][j][k] = cost[j][k]; else

mesg.RT[i][j][k] = 999;

}

// NEIGHBOURS

for(i=0; i < 20; i++) for(j=0; j < 20; j++) N[i][j] = -1;

// COST MATRIX

for(i = 0; i < num\_nodes; i++) for(j = 0; j < num\_nodes; j++)

{

if(cost[i][j] < 999)

{

N[i][j] = j;

}

}

if(argc != 3)

{

fprintf(stderr, "Usage: ./client IPaddress\_of\_server port\n"); exit(1);

}

if((sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("Cannot create socket\n"); exit(1);

}

bzero((char\*)&servaddr, sizeof(servaddr)); bzero(line, sizeof(line));

servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(atoi(argv[2]));

inet\_pton(AF\_INET, argv[1], &servaddr.sin\_addr);

if(connect(sock\_fd, (struct sockaddr \*)&servaddr, sizeof(servaddr))< 0)

{

perror("Connection failed:"); exit(1);

}

fd[0] = 0;

fd[1] = sock\_fd;

for(; ; )

{

FD\_ZERO(&rset); FD\_SET(0, &rset); FD\_SET(sock\_fd, &rset); bzero(line, sizeof(line));

max\_fd = sock\_fd;

nready = select(max\_fd + 1, &rset, NULL, NULL, NULL);

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

{

if(FD\_ISSET(fd[j], &rset))

{

if(j==0)

{ // reading from std input printf("Enter node number:"); scanf("%d", &node); printf("Node = %d\n", node);

getchar(); getchar();

// send messages to its neighbours t=0;

while(N[node][t] != -1)

{

if(N[node][t] != node)

{

mesg.from\_node = node; mesg.to\_node = N[node][t];

n = write(fd[j+1], &mesg, sizeof(mesg));

}

t++;

}

}

else

{

read(fd[j], &mesg\_from, sizeof(mesg\_from)); to\_node\_it = mesg\_from.to\_node; from\_node\_it = mesg\_from.from\_node;

for(i =0; i<num\_nodes; i++)

mesg.RT[to\_node\_it][from\_node\_it][i] = mesg\_from.RT[from\_node\_it][from\_node\_it][i];

//DISTANCE VECTOR OF to\_node\_it

mesg.RT[to\_node\_it][to\_node\_it][to\_node\_it] = 0; for(i =0; i<num\_nodes ; i++)

prev[i] = mesg.RT[to\_node\_it][to\_node\_it][i]; i = from\_node\_it;

while(N[to\_node\_it][h] != -1)

{

if(N[to\_node\_it][h] != 0)

{

if(N[to\_node\_it][h] == from\_node\_it)

{

node\_sel = from\_node\_it;

if (temp1 > cost[to\_node\_it][N[to\_node\_it][h]] + mesg\_from.RT[node\_sel][N[to\_node\_it][h]][i])

temp1 = cost[to\_node\_it][N[to\_node\_it][h]] + mesg\_from.RT[node\_sel][N[to\_node\_it][h]][i];

}

else

{

node\_sel = to\_node\_it;

if (temp1 > cost[to\_node\_it][N[to\_node\_it][h]] + mesg.RT[node\_sel][N[to\_node\_it][h]][i])

temp1 = cost[to\_node\_it][N[to\_node\_it][h]] + mesg.RT[node\_sel][N[to\_node\_it][h]][i];

}

} h++;

}

mesg.RT[to\_node\_it][to\_node\_it][i] = temp1; h=0;

for(i =0; i<num\_nodes ; i++)

{

if( prev[i] != mesg.RT[to\_node\_it][to\_node\_it][i])

{

p=0;

while(N[to\_node\_it][p] !=-1 )

{

if(p != to\_node\_it)

{

mesg.to\_node = N[to\_node\_it][p]; mesg.from\_node = to\_node\_it; write(fd[j], &mesg, sizeof(mesg));

}

p++;

}

break;

}

}

temp1 =999;

}

if(--nready == 0) break;

}

}

for(i = 0; i < num\_nodes; i++)

{

for(j = 0; j < num\_nodes; j++)

{

printf("%d ", mesg.RT[node][i][j]);

}

printf("\n");

}

printf("ITERATION RESULT FOR ROUTING TABLE\n");

}

}

*server.c*

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

struct message {

int to\_node; int from\_node;

int RT[20][20][20];

} ;

struct message mesg;

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

{

int n, i, maxi, max\_fd, k;

int sock\_fd, listen\_fd, connfd, client\_no;

int nready, num\_q, client[100], chat[100], conn[1000]; char line[1000], buffer[1000];

fd\_set rset, allset;

struct sockaddr\_in servaddr, cliaddr; int len = sizeof(cliaddr);

bzero(line, sizeof(line));

client\_no = 0;

if(argc != 2)

{

fprintf(stderr, "Usage: ./server port\n"); exit(1);

}

if((listen\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("Cannot create socket\n"); exit(1);

}

bzero((char\*)&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(atoi(argv[1]));

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

if(bind(listen\_fd, (struct sockaddr\*)&servaddr, sizeof(servaddr)) < 0)

{

perror("bind failed:"); exit(1);

}

listen(listen\_fd, num\_q);

max\_fd = listen\_fd; maxi = -1;

for(i=0; i < 100; i++)

{

client[i] = -1;

chat[i] = -1;

}

FD\_ZERO(&allset); FD\_SET(listen\_fd, &allset);

for(; ;)

{

rset = allset;

nready = select(max\_fd + 1, &rset, NULL, NULL, NULL);

if(FD\_ISSET(listen\_fd, &rset))

{

if((connfd = accept(listen\_fd, (struct sockaddr \*) &cliaddr, &len))<0)

{

perror("accept failed"); exit(1);

}

chat[++client\_no] = connfd; conn[connfd] = client\_no;

k = client\_no;

// sprintf(buffer, "Client %d has joined chat\n", k);

for( i = 0; i < 100; i++)

{

if( client[i] < 0)

{

client[i] = connfd; break;

}

}

FD\_SET(connfd, &allset); if(connfd > max\_fd) max\_fd = connfd;

if( i > maxi)

maxi = i;

if(--nready <= 0) continue;

}

for( i =0; i <=maxi; i++)

{

bzero(line, sizeof(line)); if((sock\_fd = client[i]) <0) continue;

if( FD\_ISSET(sock\_fd, &rset))

{

n= read(sock\_fd, &mesg, sizeof(mesg));

// line contains message from client

// client closing if( n == 0)

{

close(sock\_fd); FD\_CLR(sock\_fd, &allset); client[i] = -1;

}

else

{

// writing to another client

write(chat[mesg.to\_node +1], &mesg, sizeof(mesg));

}

if(--nready <= 0) break;

}

}

}

}

### Output

**Figure for the network**

### server

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ gcc -o server server.c anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./server 5080

### client (Node 0)

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./client 127.0.0.1 5080 Enter number of nodes:3

Enter the cost matrix, For infinity enter 999 0 2 7

2 0 1

7 1 0

Enter node number:0 Node = 0

0 2 7

999 999 999

999 999 999

ITERATION RESULT FOR ROUTING TABLE 0 2 7

2 0 1

999 999 999

ITERATION RESULT FOR ROUTING TABLE 0 2 3

2 0 1

7 1 0

ITERATION RESULT FOR ROUTING TABLE 0 2 3

2 0 1

3 1 0

ITERATION RESULT FOR ROUTING TABLE

### client (Node 1)

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./client 127.0.0.1 5080 Enter number of nodes:3

Enter the cost matrix, For infinity enter 999 0 2 7

2 0 1

7 1 0

Enter node number:1 Node = 1

999 999 999

2 0 1

999 999 999

ITERATION RESULT FOR ROUTING TABLE 0 2 7

2 0 1

999 999 999

ITERATION RESULT FOR ROUTING TABLE 0 2 7

2 0 1

7 1 0

ITERATION RESULT FOR ROUTING TABLE 0 2 3

2 0 1

7 1 0

ITERATION RESULT FOR ROUTING TABLE 0 2 3

2 0 1

3 1 0

ITERATION RESULT FOR ROUTING TABLE

### client (Node 2)

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./client 127.0.0.1 5080 Enter number of nodes:3

Enter the cost matrix, For infinity enter 999 0 2 7

2 0 1

7 1 0

Enter node number:2 Node = 2

999 999 999

999 999 999

7 1 0

ITERATION RESULT FOR ROUTING TABLE 0 2 7

999 999 999

7 1 0

ITERATION RESULT FOR ROUTING TABLE 0 2 7

2 0 1

7 1 0

ITERATION RESULT FOR ROUTING TABLE 0 2 3

2 0 1

3 1 0

ITERATION RESULT FOR ROUTING TABLE

## Experiment 7

**Implementation of Simple Mail Transfer Protocol**

**Aim:** To implement a subset of simple mail transfer protocol (SMTP) using UDP

### Description:

SMTP provides for mail exchanges between users on the same or different computers. The SMTP client and server can be divided into two components: user agent (UA) and mail transfer agent (MTA). The user agent is a program used to send and receive mail. The actual mail transfer is done through mail transfer agents. To send mail, a system must have client MTA, and to receive mail, a system must have a server MTA. SMTP uses commands and responses to transfer messages between an MTA client and MTA server. Commands are sent from the client to the server. It consists of a keyword followed by zero or more arguments. Examples: HELO, MAIL FROM, RCPT TO etc. Responses are sent from the server to the client. It is a three-digit code that may be followed by additional textual information. The process of transferring a mail message occurs in three phases: connection establishmnet, mail transfer, and connection termination.

Although the transport protocol specified for SMTP is TCP, in this experiment, UDP protocol will be used.

### Algorithm:

SMTP Client

1. Create the client UDP socket.
2. Send the message “SMTP REQUEST FROM CLIENT” to the server. This is done so that the server understands the address of the client.
3. Read the first message from the server using client socket and print it.
4. The first command HELO<”Client's mail server address”> is sent by the client
5. Read the second message from the server and print it.
6. The second command MAIL FROM:<”email address of the sender”> is sent by the client.
7. Read the third message from the server and print it.
8. The third command RCPT TO:<”email address of the receiver”> is sent by the client
9. Read the fourth message from the server and print it.
10. The fourth command DATA is sent by the client.
11. Read the fifth message from the server and print it.
12. Write the messages to the server and end with “.”
13. Read the sixth message from the server and print it.
14. The fifth command QUIT is sent by the client.
15. Read the seventh message from the server and print it.

Server

1. Create the server UDP socket
2. Read the message from the client and gets the client's address
3. Send the first command to the client.

*220 “server name”*

1. Read the first message from the client and print it.
2. Send the second command to the client.

*250 Hello “client name”*

1. Read the second message from client and print it.
2. Send the third command to the client.

*250 “client email address “ Sender ok*

1. Read the third message from client and print it
2. Send the fourth command to the client

*250 “server email address” Recepient ok*

1. Read the fourth message from client and print it
2. Send the fifth command to the client

*354 Enter mail, end with “.” on a line by itself*

1. Read the email text from the client till a “.” is reached
2. Send the sixth command to the client

*250 Message accepted for delivery*

1. Read the fifth message from the client and print it.
2. Send the seventh command to the client

*221 “server name” closing connection*

### Program

*Client*

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

#define MAXLINE 100 main(int argc,char \* argv[])

{

int n;

int sock\_fd; int i=0;

struct sockaddr\_in servaddr; char buf[MAXLINE+1];

char address\_buf[MAXLINE],message\_buf[MAXLINE]; char \* str\_ptr, \*buf\_ptr, \*str;

if(argc!=3)

{

fprintf(stderr,"Command is :./client address port\n"); exit(1);

}

if((sock\_fd = socket(AF\_INET, SOCK\_DGRAM, 0))<0)

{

printf("Cannot create socket\n"); exit(1);

}

bzero((char \*) & servaddr,sizeof(servaddr)); servaddr.sin\_family=AF\_INET; servaddr.sin\_port=htons(atoi(argv[2])); inet\_pton(AF\_INET,argv[1],&servaddr.sin\_addr); sprintf(buf,"SMTP REQUEST FROM CLIENT\n");

n=sendto(sock\_fd,buf,strlen(buf),0,(struct sockaddr\*)&servaddr,sizeof(servaddr)); if(n<0)

{

perror("ERROR"); exit(1);

}

if((n=recvfrom(sock\_fd,buf,MAXLINE,0,NULL,NULL))==-1)

{

perror("UDP read error"); exit(1);

}

buf[n]='\0';

printf("S:%s",buf);

sprintf(buf,"HELLO name\_of\_client\_mail\_server\n");

n=sendto(sock\_fd,buf,strlen(buf),0,(struct sockaddr\*)&servaddr,sizeof(servaddr));

if((n=recvfrom(sock\_fd,buf,MAXLINE,0,NULL,NULL))==-1)

{

perror("UDP read error"); exit(1);

}

buf[n]='\0';

printf("S:%s",buf);

printf("please enter the email address of the sender:"); fgets(address\_buf,sizeof(address\_buf),stdin); address\_buf[strlen(address\_buf)-1]='\0';

sprintf(buf,"MAIL FROM :<%s>\n",address\_buf);

sendto(sock\_fd,buf,sizeof(buf),0,(struct sockaddr\*)&servaddr,sizeof(servaddr)); if((n=recvfrom(sock\_fd,buf,MAXLINE,0,NULL,NULL))==-1)

{

perror("UDP read error"); exit(1);

}

buf[n]='\0';

printf("S:%s",buf);

printf("please enter the email address of the receiver:"); fgets(address\_buf,sizeof(address\_buf),stdin); address\_buf[strlen(address\_buf)-1]='\0'; sprintf(buf,"RCPT TO : <%s>\n",address\_buf);

sendto(sock\_fd,buf,strlen(buf),0, (struct sockaddr \*) &servaddr,sizeof(servaddr)); if((n=recvfrom(sock\_fd,buf,MAXLINE,0,NULL,NULL))==-1)

{

perror("UDP read error"); exit(1);

}

buf[n]='\0';

printf("S:%s",buf); sprintf(buf,"DATA\n");

sendto(sock\_fd,buf,strlen(buf),0,(struct sockaddr\*)&servaddr,sizeof(servaddr)); if((n=recvfrom(sock\_fd,buf,MAXLINE,0,NULL,NULL))==-1)

{

perror("UDP read error"); exit(1);

}

buf[n]='\0';

printf("S:%s",buf);

do

{

fgets(message\_buf,sizeof(message\_buf),stdin); sprintf(buf,"%s",message\_buf);

sendto(sock\_fd,buf,strlen(buf),0,(struct sockaddr\*)&servaddr,sizeof(servaddr)); message\_buf[strlen(message\_buf)-1]='\0';

str=message\_buf; while(isspace(\*str++)); if(strcmp(--str,".")==0) break;

} while(1);

if((n=recvfrom(sock\_fd,buf,MAXLINE,0,NULL,NULL))==-1)

{

perror("UDP read error"); exit(1);

}

buf[n]='\0'; sprintf(buf,"QUIT\n"); printf("S:%s",buf);

sendto(sock\_fd,buf,strlen(buf),0,(struct sockaddr\*)&servaddr,sizeof(servaddr)); if((n=recvfrom(sock\_fd,buf,MAXLINE,0,NULL,NULL))==-1)

{

perror("UDP read error"); exit(1);

}

buf[n]='\0';

printf("S:%s",buf);

}

Server

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

#define MAXLINE 100 main(int argc, char \* argv[])

{

int n,sock\_fd;

struct sockaddr\_in servaddr,cliaddr; char mesg[MAXLINE+1]; socklen\_t len;

char \* str\_ptr, \*buf\_ptr, \*str; len=sizeof(cliaddr);

if((sock\_fd=socket(AF\_INET,SOCK\_DGRAM,0))<0)

{

printf("cannot create socket\n"); exit(1);

}

bzero((char\*)&servaddr,sizeof(servaddr)); servaddr.sin\_family=AF\_INET; servaddr.sin\_port=htons(atoi(argv[1])); servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); if(bind(sock\_fd,(struct sockaddr\*)&servaddr,sizeof(servaddr))<0)

{

perror("bind failed:"); exit(1);

}

if((n=recvfrom(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len))==-1)

{

perror("size not received:"); exit(1);

}

mesg[n]='\0'; printf("mesg:%s\n",mesg);

sprintf(mesg,"220 name\_of\_server\_mail\_server\n"); sendto(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,sizeof(cliaddr)); n=recvfrom(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len); mesg[n]='\0';

printf("C:%s\n",mesg); str\_ptr=strdup(mesg); buf\_ptr=strsep(&str\_ptr," "); sprintf(mesg,"250 Hello %s",str\_ptr); free(buf\_ptr);

sendto(sock\_fd, mesg, MAXLINE, 0, (struct sockaddr \*)&cliaddr, sizeof(cliaddr)); n=recvfrom(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len); mesg[n]='\0';

printf("C:%s",mesg); str\_ptr=strdup(mesg); buf\_ptr=strsep(&str\_ptr,":"); str\_ptr[strlen(str\_ptr)-1]='\0';

sprintf(mesg,"250 Hello %s. sender ok\n",str\_ptr);

free(buf\_ptr);

sendto(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,sizeof(cliaddr)); n=recvfrom(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len); mesg[n]='\0';

printf("C:%s",mesg); str\_ptr=strdup(mesg); buf\_ptr=strsep(&str\_ptr,":"); str\_ptr[strlen(str\_ptr)-1]='\0';

sprintf(mesg,"250 Hello %s. Recepient ok\n",str\_ptr);

free(buf\_ptr);

sendto(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,sizeof(cliaddr)); n=recvfrom(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len); mesg[n]='\0';

printf("C:%s\n",mesg);

sprintf(mesg,"354 Enter mail,end with \".\" on a line by itself \n"); sendto(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,sizeof(cliaddr)); while(1)

{

n=recvfrom(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len); mesg[n]='\0';

printf("C:%s\n",mesg);

mesg[strlen(mesg)-1]='\0';

str=mesg; while(isspace(\*str++)); if(strcmp(--str,".")==0) break;

sprintf(mesg,"250 messages accepted for delivery \n"); sendto(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,sizeof(cliaddr)); n=recvfrom(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len); mesg[n]='\0';

printf("C:%s\n",mesg);

sprintf(mesg,"221 servers mail server closing connection\n"); sendto(sock\_fd,mesg,MAXLINE,0,(struct sockaddr\*)&cliaddr,sizeof(cliaddr));

}

}

Server program

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

#define MAXLINE 100 main(int argc, char \* argv[])

{

int n, sock\_fd;

struct sockaddr\_in servaddr, cliaddr;

char mesg[MAXLINE + 1]; socklen\_t len;

char \* str\_ptr, \*buf\_ptr, \*str; len = sizeof(cliaddr);

if((sock\_fd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("Cannot create socket\n"); exit(1);

}

bzero((char\*)&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(atoi(argv[1]));

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

if(bind(sock\_fd, (struct sockaddr\*)&servaddr, sizeof(servaddr)) < 0)

{

perror("bind failed:"); exit(1);

}

if((n = recvfrom(sock\_fd, mesg, MAXLINE, 0, (struct sockaddr \*)&cliaddr, &len)) == -1)

{

perror("size not received:"); exit(1);

}

mesg[n] = '\0'; printf(“mesg:%s\n”, mesg);

sprintf(mesg, “220 name\_of\_server\_mail\_server\n”);

sendto(sock\_fd, mesg, MAXLINE,0, (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

n = recvfrom(sock\_fd, mesg, MAXLINE, 0, (struct sockaddr \*)&cliaddr, &len); mesg[n] = '\0';

printf(“C:%s\n”,mesg);

str\_ptr = strdup(mesg); buf\_ptr = strsep(&str\_ptr, “ “);

sprintf(mesg, “250 Hello %s”, str\_ptr);

free(buf\_ptr);

sendto(sock\_fd, mesg, MAXLINE,0, (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

n = recvfrom(sock\_fd, mesg, MAXLINE, 0, (struct sockaddr \*)&cliaddr, &len); mesg[n] = '\0';

printf(“C:%s\n”,mesg);

str\_ptr = strdup(mesg); buf\_ptr = strsep(&str\_ptr, “:“); str\_ptr[strlen(str\_ptr)-1] = '\0';

sprintf(mesg, “250 Hello %s. Sender ok\n”, str\_ptr);

free(buf\_ptr);

sendto(sock\_fd, mesg, MAXLINE,0, (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

n = recvfrom(sock\_fd, mesg, MAXLINE, 0, (struct sockaddr \*)&cliaddr, &len); mesg[n] = '\0';

printf(“C:%s\n”,mesg)

sprintf(mesg, “354 Enter mail, end with \”.\” on a line by itself\n”); sendto(sock\_fd, mesg, MAXLINE,0, (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

while(1)

{

n = recvfrom(sock\_fd, mesg, MAXLINE, 0, (struct sockaddr \*)&cliaddr, &len); mesg[n] = '\0';

printf(“C:%s\n”,mesg);

mesg[strlen(mesg) – 1] = '\0'; str = mesg; while(isspace(\*str++)); if(strcmp(--str, “.”) == 0) break;

sprintf(mesg, “250 Message accepted for delivery\n”);

sendto(sock\_fd, mesg, MAXLINE,0, (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

n = recvfrom(sock\_fd, mesg, MAXLINE, 0, (struct sockaddr \*)&cliaddr, &len); mesg[n] = '\0';

printf(“C:%s\n”,mesg);

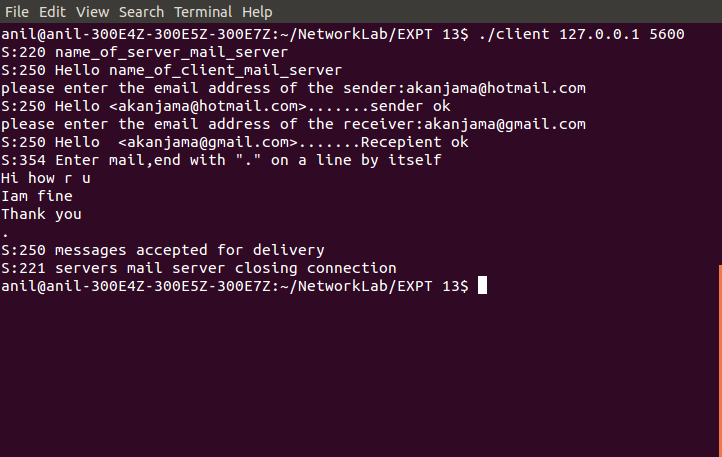
sprintf(mesg, “221 Server's mail server closing connection\n”);

sendto(sock\_fd, mesg, MAXLINE,0, (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

}

**Output**

**Client**



**Server**



## Experiment 8

**Implementation of File Transfer Protocol**

**Aim:** To develop a concurrent file server which will provide the file requested by client if it exists. If not, server sends appropriate message to the client. Server sends its process ID (PID) to clients for display along with file or the message.

### Description:

The file server creates listening sockets on two ports that have consecutive port numbers. One is the listening socket for control connection and the other is the listening socket for data connection. These sockets have descriptors *listen\_control* and *listen\_data* respectively. The client creates a socket and connects to the server using TCP connection. The client socket descriptor is stored in *sock\_ctrl.* The server creates a connection socket when the client makes a TCP connection with the server. This connection socket is stored in *sock\_ctrl.* This connection is for control information to pass between client and server. Next, the server forks a child process. Since the child is a perfect image of its parent, the child process will also have descriptors *listen\_control*, *listen\_data* and *sock\_ctrl*. The control connection will also be duplicated between the client and the child process. The child process closes its listening socket for control connection, i.e; *listen\_control*. The parent server process closes its connection socket for the control connection i.e; *sock\_ctrl*. The client now makes a TCP connection with the server for transferring data is stored in *sock\_data.* The connection socket for the data connection is stored in *sock\_data* in the server child process.

### Algorithm

*Client*

1. Create the client TCP control connection to a port (port\_num) of the server with the client socket descriptor *sock\_ctrl.*
2. `while(1)

{

* 1. Create client socket descriptor *sock\_data* for the data connection with server on another port (portnum + 1). For each file transfer a new data connection is required.
  2. Read the command from the terminal given by the user
  3. Send the command to the server using control socket *sock\_ctrl.*
  4. If command == “close”

close *sock\_data* and *sock\_ctrl*

break

* 1. Enter the filename using the keyboard
  2. Write the filename using control socket *sock\_ctrl* to the server.
  3. Use *while(connect(...) < 0)* to wait for the data connection.
  4. read data using *sock\_data* (file contents) from the server and write to the file
  5. If file does not exist print the process id of server

}

*Server*

1. Initialize variable *file\_present* to 1
2. Create listening socket for the control connection on port (port\_num) and store it in

*listen\_control.*

1. Create a listening socket for the data connection on port (port\_num +1) and store it in *listen\_data*
2. while (1)

{

* 1. accepts the client control connection and returns the connect socket descriptor

*sock\_ctrl*

* 1. fork a child process
     1. if(childprocess)

{

close listening socket *listen\_control*

while(1)

{

read command from the client on the control connection

if command == “close” break

else

}

read the filename from the client using control connection open the file

if (no file)

form a string “@FILE NOT FOUND PROCESS ID = getprocess id” and store it in variable *buffer*

file\_present = 0;

accept the data connection and return the socket descriptor *sock\_data*

if(file\_present)

read file contents and write to *sock\_data*

form a string “FILE *filename* RECEIVED FROM SERVER WITH PROCESS ID = getprocess id” and store it in variable *buffer*

else

file\_present = 1;

write *buffer* using *sock\_data* to the client

close *sock\_data*

close*(sock\_ctrl)* close*(listen\_data)* child process exits

}

close( *sock\_ctrl*)

}

1. close(*listen\_control*);
2. close(*listen\_data*);

### Program

*Client*

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

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

{

int n, fd, i;

int sock\_ctrl, sock\_data;

char buffer[100], line[100], cmd[100]; char name[100];

char \* p;

struct sockaddr\_in servaddr; if(argc != 3)

{

fprintf(stderr, "Usage: ./client IPaddress\_of\_server port\n"); exit(1);

}

if((sock\_ctrl = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("Cannot create control socket\n"); exit(1);

}

bzero((char\*)&servaddr, sizeof(servaddr)); bzero(line, sizeof(line));

bzero(buffer, sizeof(buffer));

servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(atoi(argv[2]));

inet\_pton(AF\_INET, argv[1], &servaddr.sin\_addr);

if(connect(sock\_ctrl, (struct sockaddr \*)&servaddr, sizeof(servaddr))< 0)

{

perror("Connection failed, control socket:"); exit(1);

}

printf(" Enter \"get\" for receiving file from server\n"); printf(" Enter \"close\" for closing connection\n"); while (1)

{

i=0;

if(( sock\_data = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("Creation of client data socket failed”); exit(1);

}

printf("Enter command:"); scanf("%s", cmd);

n = write( sock\_ctrl, cmd, sizeof(cmd));

if(strcmp(cmd, "close") == 0)

{

close(sock\_data); close(sock\_ctrl);

break;

}

printf("Enter filename:"); scanf("%s", name);

printf("name of file = %s\n", name);

if((fd = open(name, O\_WRONLY | O\_CREAT)) < 0)

{

perror("Error in opening file "); exit(1);

}

write(sock\_ctrl, name, sizeof(name)); servaddr.sin\_port= htons(atoi(argv[2]) + 1);

n = connect(sock\_data, (struct sockaddr \*)&servaddr, sizeof(servaddr)); while(n == -1)

{

n = connect(sock\_data, (struct sockaddr \*)&servaddr, sizeof(servaddr)); perror("cannot connect");

}

do

{

n = read(sock\_data, buffer, sizeof(buffer)); write(fd, buffer, n);

i += n;

} while(n>0);

if(buffer[0] =='@')

{

p = buffer; printf("%s\n", p++); remove(name);

}

}

}

*Server*

#include<stdio.h> #include<string.h> #include<sys/socket.h> #include<sys/types.h> #include<netinet/in.h> #include<arpa/inet.h> #include<fcntl.h> #include<stdlib.h>

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

{

int n,m, fd, i ;

int sock\_ctrl, sock\_data, listen\_control, listen\_data; int file\_present =1;

char name[100], buffer[100], cmd[100]; struct sockaddr\_in servaddr, cliaddr;

int cli\_len = sizeof(cliaddr); if(argc != 2)

{

fprintf(stderr, "Usage: ./server port\n"); exit(1);

}

if((listen\_control = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("cannot create listening socket for control connection"); exit(1);

}

bzero(&servaddr, sizeof(servaddr)); servaddr.sin\_family = AF\_INET; servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); servaddr.sin\_port = htons(atoi(argv[1]));

if(bind(listen\_control, (struct sockaddr\*)&servaddr, sizeof(servaddr)) < 0)

{

perror("server bind failed for control listening socket"); exit(1);

}

listen(listen\_control, 5);

if((listen\_data = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("cannot create listening socket for data connection"); exit(1);

}

servaddr.sin\_port = htons(atoi(argv[1]) +1);

if(bind(listen\_data, (struct sockaddr\*)&servaddr, sizeof(servaddr)) < 0)

{

perror("server bind failed for data listening socket"); exit(1);

}

listen(listen\_control, 5);

listen(listen\_data, 5); for(; ;)

{

if((sock\_ctrl = accept(listen\_control, (struct sockaddr\*)&servaddr, & cli\_len)) <0)

{

perror("accept failed"); exit(1);

}

servaddr.sin\_port = htons(atoi(argv[1] + 1));

if(fork() == 0)

{

close(listen\_control); while(1)

{

i=0;

n = read(sock\_ctrl, cmd, 100);

if(strcmp(cmd, "close") == 0)

{

break;

}

else

{

read(sock\_ctrl, name, 100);

if((fd = open(name, O\_RDONLY)) <0)

{

sprintf(buffer, "@FILE NOT FOUND PROCESS ID = %d", getpid()); perror("error in opening file");

file\_present = 0;

}

if((sock\_data = accept(listen\_data, (struct sockaddr\*)&servaddr, & cli\_len)) <0)

{

perror("accept failed"); exit(1);

}

if(file\_present == 1)

{

do

{

n = read(fd, buffer, sizeof(buffer)); write(sock\_data, buffer, n);

i = i+n;

} while(n>0); close(fd);

sprintf(buffer, "FILE %s RECEIVED FROM SERVER WITH PROCESS ID = %d",

name, getpid());

m = write(sock\_data, buffer, strlen(buffer));

}

else

{

m = write(sock\_data, buffer, sizeof(buffer)); file\_present =1;

bzero(buffer, sizeof(buffer));

}

close(sock\_data);

}

}

close(sock\_ctrl); close(listen\_data); exit(0);

}

close(sock\_ctrl);

}

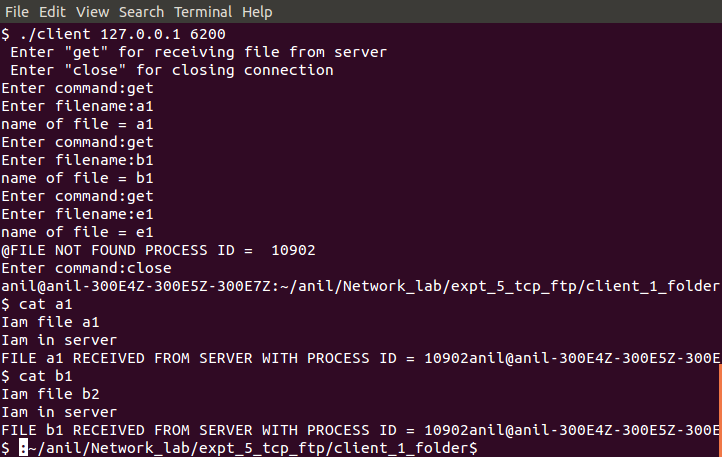
close(listen\_control); close(listen\_data);

}

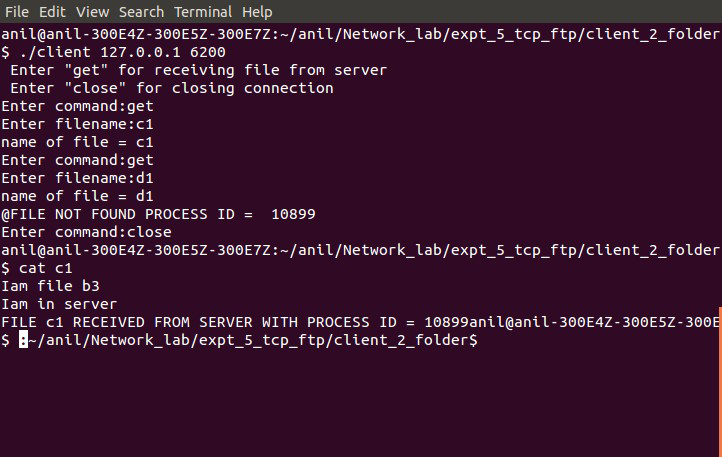
### Output

The files stored in the server are *a1, b1,* and *c1.* Client 1 connects to the server and gets files a1 and b1. Client 2 connects to server and receives *c1.* The server process id is stored in the files received.

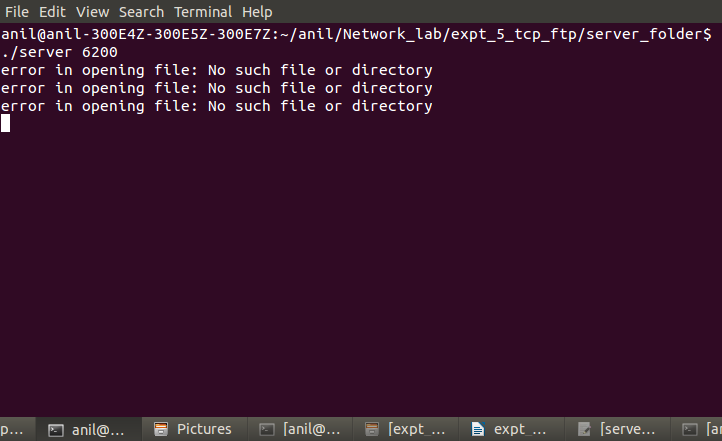
### Client 1



**Client 2**



**Server**



## Experiment 9

**UDP datagram in client server communication using Wireshark**

**Aim:** To observe data transferred using UDP in client server communication and to study UDP datagram.

### Description:

Wireshark is a free network protocol analyzer that runs on Windows, Mac, and Linux/Unix computer. It operates in computers using Ethernet, serial (PPP and SLIP), 802.11 wireless LANs, and many other link-layer technologies. It is also a packet sniffer since it can be used to observe messages exchanged between protocol entities. It will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer receives a copy of packets that are sent/received from/by application and protocols executing on your machine. Packet sniffer software consists of two components. One is packet capture library that receives a copy of every link-layer frame that is sent from or received by your computer. The second component of a packet sniffer is the packet analyzer, which displays the contents of all fields within a protocol message. Wireshark has a rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface.

Wireshark is installed in the computer. The wireshark interface has 5 major components.

1. Command menus

The two noteworthy menus are File menu that allows you to save captured packet data and exit the Wireshark application. The Capture menu allows you to begin packet capture and stop capturing when needed.

1. Packet listing window

Displays a one-line summary for each packet captured, including the packet number assigned by Wireshark, the time at which the packet was captured, the packet’s source and destination addresses, the protocol type, and protocol-specific information contained in the packet.

1. Packet-header details window

Provides details about the packet selected in the packet-listing window.

1. Packet-contents window

Displays the entire contents of the captured frame in both ASCII and hexadecimal format.

### Output

*UDP datagrams*

Packet capturing is started. Skype application is also started. This generates UDP datagrams.

From the trace, the details of frame 47 can be seen. It is an UDP datagram as specified by column 5.

UDP header has the following fields

source port = 64163 destination port = 3478

Length field = Length of UDP header + Length of data field = 713 bytes checksum =0x9beb

UDP header has a length of 8bytes. The payload of the UDP datagram is therefore 705 bytes whose hexadecimal values are given in the trace.

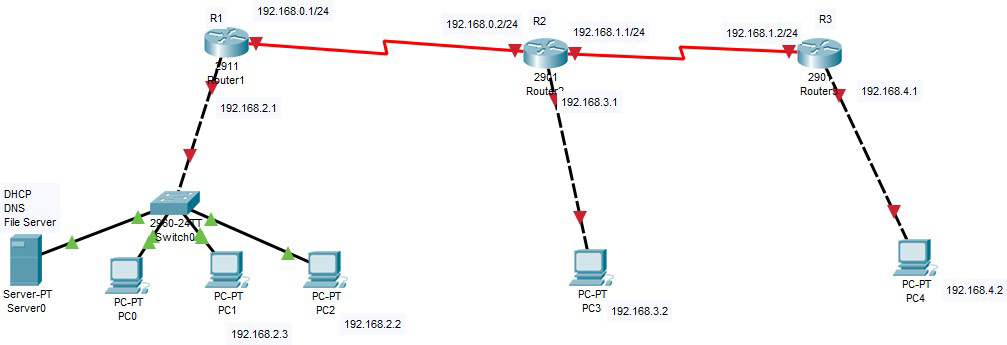
## Outputudp_screen_1.jpg

**Experiment 10**

# Design and Configuration of network and services

**Aim:** To design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure the following services in the network- TELNET, SSH, FTP server, Webserver, File server, DHCP server and DNS server.\*

### Description:



#### Design

The above topology shows 3 subnets .The network is designed with server, hosts, switch and Routers.

#### Configuration

Subnet 1 contains 3 hosts.

Hosts are configured with IP addresses 192.168.2.2, 192.168.2.3, 192.168.2.4 and the server with 192.168.2.100 . These hosts are connected to the fast Ethernet port of the router through the switch

Server is configured with DHCP, DNS etc. Subnet 2 contains one host with IP address 192.168.3.2 Subnet 3 contains one host with IP address 192.168.4.2. Routers R1, R2, R3 are connected through serial cable (R1,R2,R3)

We need to configure static routing in the routers to enable communication of hosts in different subnets

*Router 1*

Configure the router serial interface with IP address 192.168.0.1 255.255.255.0 Configure the fast Ethernet port with IP address 192.168.2.1

Configure the static route -ip route 192.168.1.0 255.255.255.0 s0/0 ip route 192.168.3.0 255.255.255.0 s0/0

ip route 192.168.4.0 255.255.255.0 s0/0

*Router 2*

Configure the router serial interface 0 with IP address 192.168.0.2 255.255.255.0 Configure the router serial interface 1 with IP address 192.168.1.1 255.255.255.0 Configure the fast Ethernet port with IP address 192.168.3.1

Configure the static route -ip route 192.168.2.0 255.255.255.0 s0/0 ip route 192.168.4.0 255.255.255.0 s0/1

*Router 3*

Configure the router serial interface with IP address 192.168.1.2 255.255.255.0 Configure the fast Ethernet port with IP address 192.168.4.1

Configure the static route -ip route 192.168.0.0 255.255.255.0 s0/0 ip route 192.168.1.0 255.255.255.0 s0/0

ip route 192.168.2.0 255.255.255.0 s0/0

**Output**

Files and services can be accessed across subnets.

## Experiment 11

**Simulation using NS2 simulator**

**Aim:** To Install network simulator NS-2 in any of the Linux operating system and simulate wired and wireless scenarios.

### Description

Basics of Computer Network Simulation:

A simulation can be thought of as a flow process of network entities (e.g., nodes, packets). As these entities move through the system, they interact with other entities, join certain activities, trigger events, cause some changes to the state of the system, and leave the process. From time to time, they contend or wait for some type of resources. This implies that there must be a logical execution sequence to cause all these actions to happen in a comprehensible and manageable way.

Introduction to Network Simulator 2 (NS2)

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

Basic Architecture

NS2 provides users with an executable command *ns* which takes on input argument, the name of a Tcl simulation scripting file. In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

#### Ns2 Installation On Ubuntu 16.04

1. Download **'ns-allinone-2.35'** from :

<http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/ns-allinone-2.35.tar.gz/download>

1. Extract the downloaded zip file 'ns-allinone-2.35.tar.gz file' to desktop.
2. Now you need to download some essential packages for ns2.Type the below lines one by one on the terminal window

"sudo apt-get update" "sudo apt-get dist-upgrade" "sudo apt-get update" "sudo apt-get gcc"

"sudo apt-get install build-essential autoconf automake" "sudo apt-get install tcl8.5-dev tk8.5-dev"

"sudo apt-get install perl xgraph libxt-dev libx11-dev libxmu-dev"

1. Now change your directory(here i have already extracted the downloaded files to desktop,so my location is desktop) type the following codes in the command window to install NS2.

cd Desktop

cd ns-allinone-2.35

./install

### The installation procedure will take a few minutes..........

1. After compleating the installation type the following command in the command window gedit ~/.bashrc
2. Now an editor window appears,please copy and paste the follwing codes in the end of the text file (note that '/home/abhiram/Desktop/ns-allinone-2.35/octl-1.14' in each line in the below code should be replaced with your location where the 'ns-allinone-2.35.tar.gz'file is extracted)

# LD\_LIBRARY\_PATH

OTCL\_LIB=/home/abhiram/Desktop/ns-allinone-2.35/otcl-1.14 NS2\_LIB=/home/abhiram/Desktop/ns-allinone-2.35/lib X11\_LIB=/usr/X11R6/lib

USR\_LOCAL\_LIB=/usr/local/lib export

LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:$OTCL\_LIB:$NS2\_LIB:$X11\_LIB:$USR\_LOCAL

\_LIB

# TCL\_LIBRARY

TCL\_LIB=/home/abhiram/Desktop/ns-allinone-2.35/tcl8.5.10/library

USR\_LIB=/usr/lib

export TCL\_LIBRARY=$TCL\_LIB:$USR\_LIB

# PATH

XGRAPH=/home/abhiram/Desktop/ns-allinone-2.35/bin:/home/abhiram/Desktop/ns-allinone- 2.35/tcl8.5.10/unix:/home/abhiram/Desktop/ns-allinone-2.35/tk8.5.10/unix NS=/home/abhiram/Desktop/ns-allinone-2.35/ns-2.35/

NAM=/home/abhiram/Desktop/ns-allinone-2.35/nam-1.15/ PATH=$PATH:$XGRAPH:$NS:$NAM

1. Save and close the text editor and then type the following command on the terminal source ~/.bashrc
2. Close the terminal window and start a new terminal window and now change the directory to ns-

2.35 and validate ns-2.35 by exicuting the following command ( it takes 30 to 45 minutes)

cd ns-2.35

./validate

1. If the installation is successful, then you will be able to see % at the command prompt while typing the following command

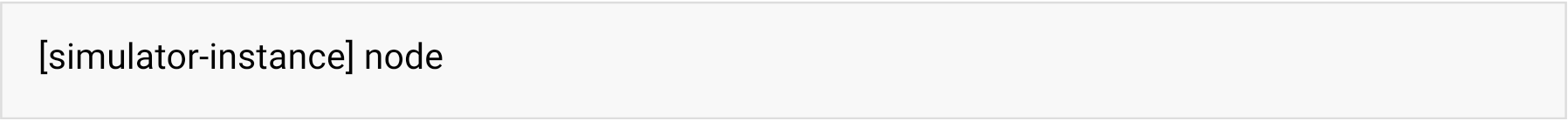
ns

1. Now type exit

#### Ns2 Commands

*Creating wired node*

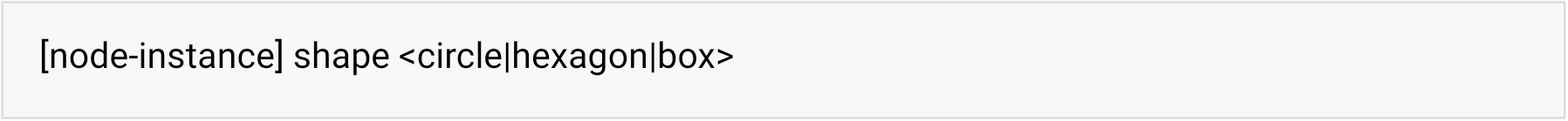
Simulator class has a procedure node which return a node.command



*Changing shape of Node*

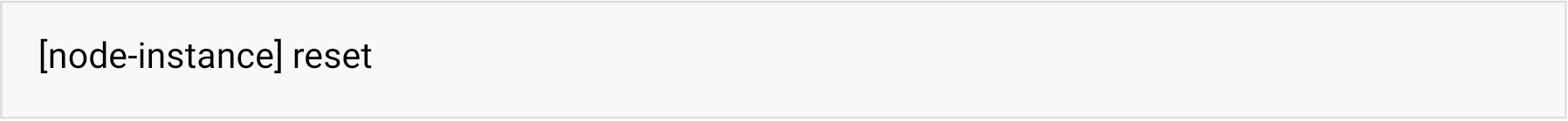
Shape of node can be changed using shape procedure of node class. Shapes in ns2 available are box, hexagon and circle. Default shape is circle. Once a shape is defined then it can't be changed after simulation starts.

Syntax:



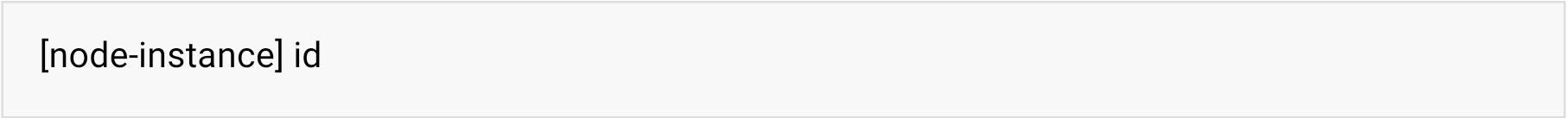
*Reset all agents of Node*

*reset* command is used to reset all agents attached to node.

Syntax:

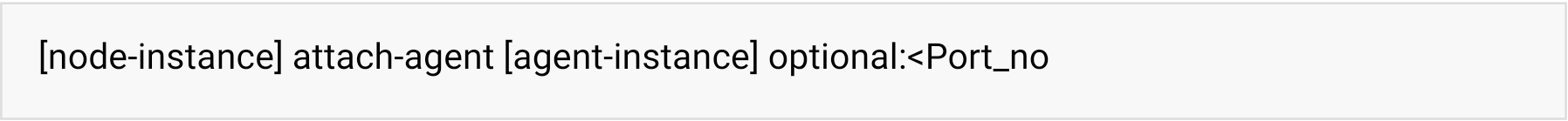
*Fetch id of Node*

Node id can be fetched in ns2. Sytanx:



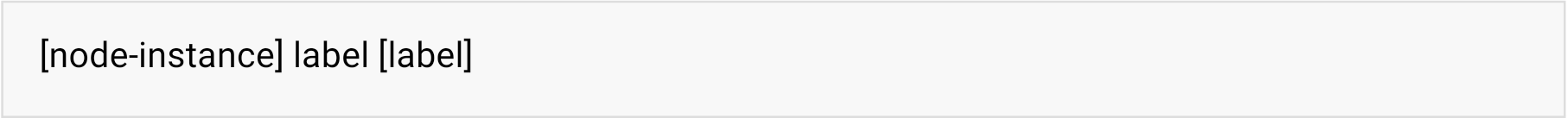
*Attach agent to node on a specific port*

Agent can be attached to node on a specified port number.

Sytanx:

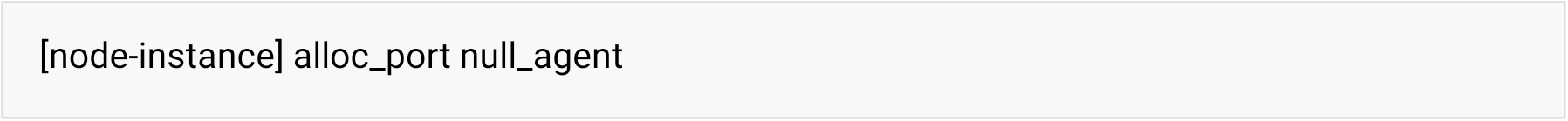
*Attach label to node*

Label can be attached to node in NAM for better understanding. Sytanx:



Fetch next available port number

alloc-port is used to access available port number on a node for new agents. Sytanx:



*Fetch list of neighbors*

Every node maintains list of neighbors which are connected to that node. Syntax:

[*node-instance] neighbor*

#### Link Commands

In *ns2*, nodes can be connected in two ways, simplex and duplex. Simplex connection allows one-way communication and duplex connection allows two-way communication. Each type requires bandwidth, delay and type of queue for configuration.

Bandwidth is specified in Mbps(Mb) and delay is specified in milli seconds (ms).

Type of Queue available in ns2: DropTail, RED, CBQ, FQ, SFQ, DRR.

Syntax:

*$ns simplex-link/duplex-link [node-instance1] [node-instance2] bandwidth delay Q-Type*

*Setting orientation of links in NAM*

Position of links can be defined for better representation of network in NAM(Network AniMator).

Syntax : *$ns simplex-link-op/duplex-link-op [node-instance1] [node-instance2] orient [orientation]*

Orientation can be right, left, up, down, up-right, up-left etc.

*Agent Class*

For every node transport mechanism need to be defined to send data. These transport mechanism in ns2 defined using agent. For example FTP application requires TCP transport protocol, that's why TCP agent need to be associated with sending node. All agents are subclass of Agent class

*Attach agent to node*

Before attaching a agent to node, instance of agent need to created. Instance of above given agent can be instantiated as

*$ [simulator-instance] attach-agent [node-instance] [agent-instance]*

*Fetch port number to which agent is attached*

Every agent store port number assigned to it. This port number can be accessed by using following command

Syntax:

*$ [agent-instance] port Connect agent to another agent*

For communication agent need to be connected to another agent to which it wants to send data.

*Syntax : $ [simulator-instance] connect [agent-instance] [agent-instance]*

### Analysing trace file

In *ns2* simulation result stored in trace files. These file formats need to be understood for analysing result. Here we will discuss general trace file and nam trace file.

*$set trace [open result.tr w]*

*$ns trace-all $trace*

#### LAN simulation on NS2(CSMA)

* Carrier Sense Multiple Access (CSMA) is a network protocol that listens to or senses network signals on the carrier/medium before transmitting any data.
* CSMA is implemented in Ethernet networks with more than one computer or network device attached to it.

*Type of CSMA Protocols*

* p-persistent CSMA
* I-persistent CSMA
* Non- Persistent CSMA

*Features of CSMA routing protocol:*

* Low packet delays
* To provide QoS
* Low packet loss
* Collision avoidance etc.

*A research topic in CSMA is:*

* For channel sharing resource management.
* For collision avoidance.
* Collision detection problem.
* Discuss energy related issues.
* To manage medium access.
* Achieve scalability.
* Considered mobility .

*Simulation of CSMA*

set ns [new Simulator] #define color for data flows

$ns color 1 Blue

$ns color 2 Red #open tracefiles

set tracefile1 [open out.tr w] set winfile [open winfile w]

$ns trace-all $tracefile1 #open nam file

set namfile [open out.nam w]

$ns namtrace-all $namfile #define the finish procedure proc finish {} {

global ns tracefile1 namfile

$ns flush-trace close $tracefile1 close $namfile

exec nam out.nam & exit 0

}

#create six nodes set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node] set n4 [$ns node] set n5 [$ns node]

$n1 color Red

$n1 shape box

#create links between the nodes

$ns duplex-link $n0 $n2 2Mb 10ms DropTail

$ns duplex-link $n1 $n2 2Mb 10ms DropTail

$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail

$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]

#Give node position

$ns duplex-link-op $n0 $n2 orient right-down

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

$ns simplex-link-op $n2 $n3 orient right

$ns simplex-link-op $n3 $n2 orient left

#set queue size of link(n2-n3) to 20

$ns queue-limit $n2 $n3 20

#setup TCP connection

set tcp [new Agent/TCP/Newreno]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink/DelAck]

$ns attach-agent $n4 $sink

$ns connect $tcp $sink

$tcp set fid\_ 1

$tcp set packet\_size\_ 552

#set ftp over tcp connection set ftp [new Application/FTP]

$ftp attach-agent $tcp

#setup a UDP connection set udp [new Agent/UDP]

$ns attach-agent $n1 $udp set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid\_ 2

#setup a CBR over UDP connection set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set type\_ CBR

$cbr set packet\_size\_ 1000

$cbr set rate\_ 0.01Mb

$cbr set random\_ false #scheduling the events

$ns at 0.1 "$cbr start"

$ns at 1.0 "$ftp start"

$ns at 124.0 "$ftp stop"

$ns at 125.5 "$cbr stop"

proc plotWindow {tcpSource file} { global ns

set time 0.1

set now [$ns now]

set cwnd [$tcpSource set cwnd\_] puts $file "$now $cwnd"

$ns at [expr $now+$time] "plotWindow $tcpSource $file"

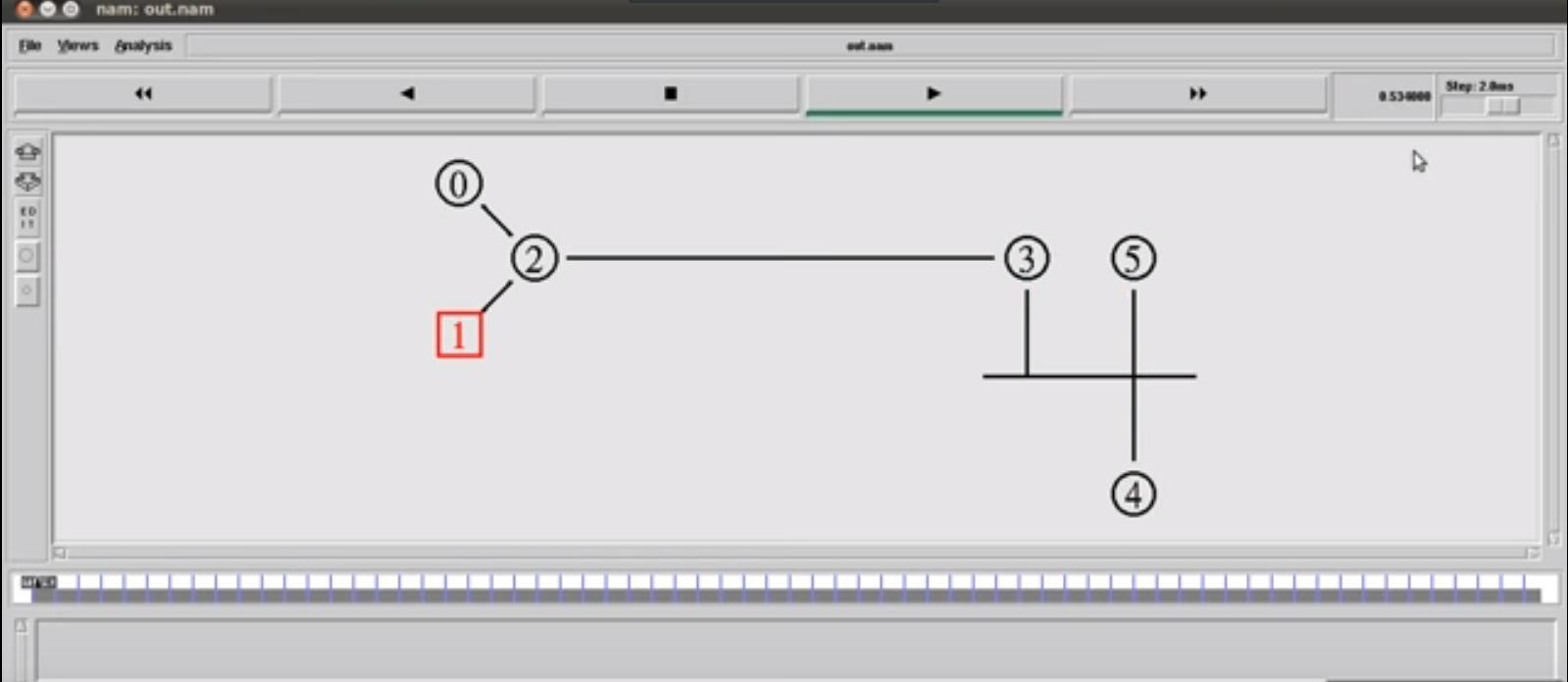
}

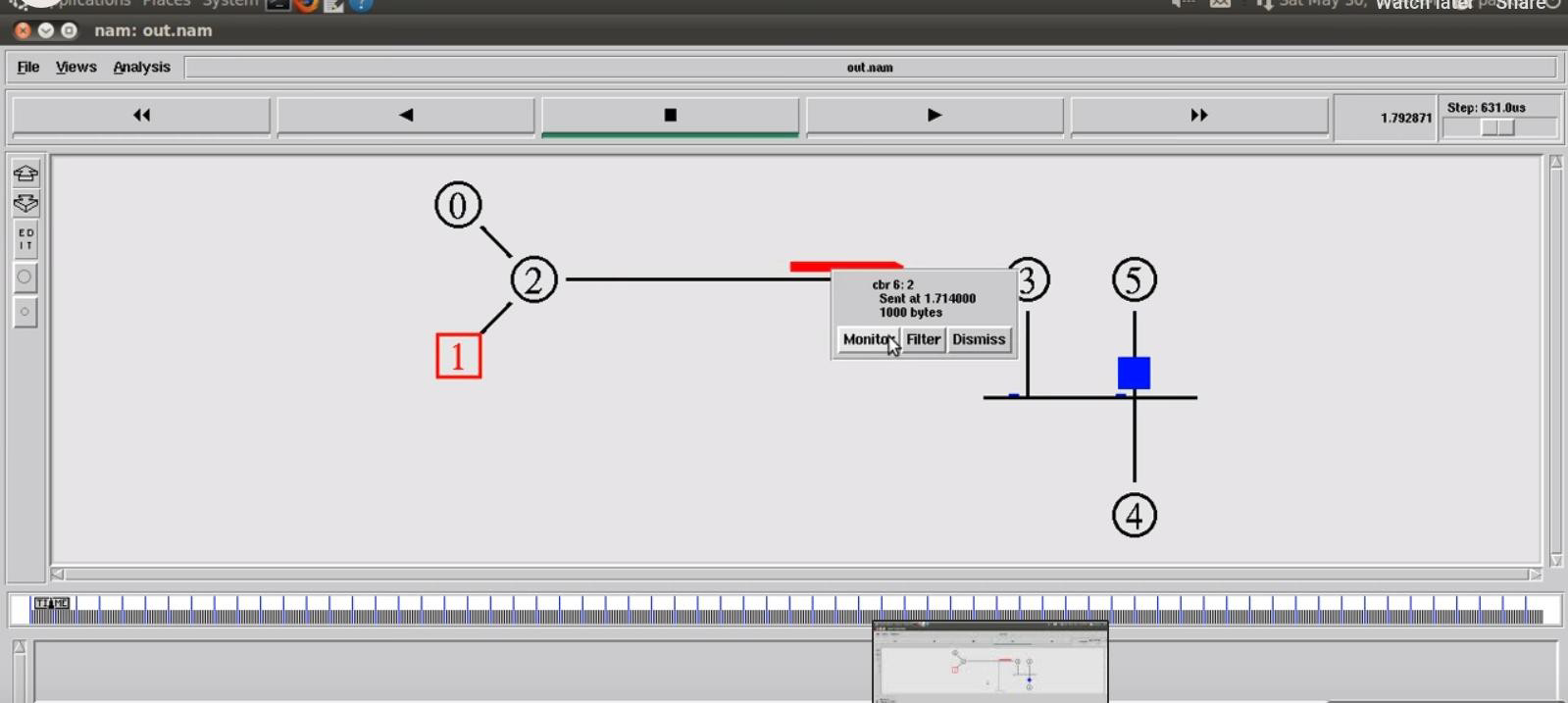
$ns at 0.1 "plotWindow $tcp $winfile"

$ns at 125.0 "finish"

$ns run

### Output





*AODV*

An Ad Hoc On-Demand Distance Vector (AODV) is a routing protocol designed for wireless and mobile ad hoc networks. This protocol establishes routes to destinations on demand and supports both unicast and multicast routing.The AODV protocol builds routes between nodes only if they are requested by source nodes. AODV is therefore considered an on-demand algorithm and does not create any extra traffic for communication along links. The routes are maintained as long as they are required by the sources.

In AODV, networks are silent until connections are established. Network nodes that need connections

broadcast a request for connection. The remaining AODV nodes forward the message and record the node that requested a connection. Thus, they create a series of temporary routes back to the requesting node.

A node that receives such messages and holds a route to a desired node sends a backward message through temporary routes to the requesting node. The node that initiated the request uses the route containing the least number of hops through other nodes. The entries that are not used in routing tables are recycled after some time. If a link fails, the routing error is passed back to the transmitting node and the process is repeated.

*Simulation of AODV*

# A 3-node example for ad-hoc simulation with DSDV # Define options

set val(chan) Channel/WirelessChannel ;# channel type

set val(prop) Propagation/TwoRayGround ;# radio-propagation model set val(netif) Phy/WirelessPhy ;# network interface type

set val(mac) Mac/802\_11 ;# MAC type

set val(ifq) Queue/DropTail ;# interface queue type set val(ll) LL ;# link layer type

set val(ant) Antenna/OmniAntenna ;# antenna model set val(ifqlen) 50 ;# max packet in ifq

set val(nn) 3 ;# number of mobilenodes set val(rp) AODV ;# routing protocol

set val(x) 500 ;# X dimension of topography

set val(y) 400 ;# Y dimension of topography

set val(stop) 150 ;# time of simulation end

set ns [new Simulator]

set tracefd [open simple-dsdv.tr w] set windowVsTime2 [open win.tr w] set namtrace [open simwrls1.nam w]

$ns trace-all $tracefd

$ns use-newtrace

$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object

set topo [new Topography]

$topo load\_flatgrid $val(x) $val(y) create-god $val(nn)

#

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

# configure the nodes

$ns node-config -adhocRouting $val(rp) \

-llType $val(ll) \

-macType $val(mac) \

-ifqType $val(ifq) \

-ifqLen $val(ifqlen) \

-antType $val(ant) \

-propType $val(prop) \

-phyType $val(netif) \

-channelType $val(chan) \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace OFF \

-movementTrace ON

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

}

# Provide initial location of mobilenodes

$node\_(0) set X\_ 5.0

$node\_(0) set Y\_ 5.0

$node\_(0) set Z\_ 0.0

$node\_(1) set X\_ 490.0

$node\_(1) set Y\_ 285.0

$node\_(1) set Z\_ 0.0

$node\_(2) set X\_ 150.0

$node\_(2) set Y\_ 240.0

$node\_(2) set Z\_ 0.0

# Generation of movements

$ns at 10.0 "$node\_(0) setdest 250.0 250.0 3.0"

$ns at 15.0 "$node\_(1) setdest 45.0 285.0 5.0"

$ns at 110.0 "$node\_(0) setdest 480.0 300.0 5.0"

# Set a TCP connection between node\_(0) and node\_(1) set tcp [new Agent/TCP/Newreno]

$tcp set class\_ 2

set sink [new Agent/TCPSink]

$ns attach-agent $node\_(0) $tcp

$ns attach-agent $node\_(1) $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ns at 10.0 "$ftp start"

# Define node initial position in nam for {set i 0} {$i < $val(nn)} { incr i } { # 30 defines the node size for nam

$ns initial\_node\_pos $node\_($i) 30

}

# Telling nodes when the simulation ends for {set i 0} {$i < $val(nn) } { incr i } {

$ns at $val(stop) "$node\_($i) reset";

}

# ending nam and the simulation

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "stop"

$ns at 150.01 "puts \"end simulation\" ; $ns halt" proc stop {} {

global ns tracefd namtrace

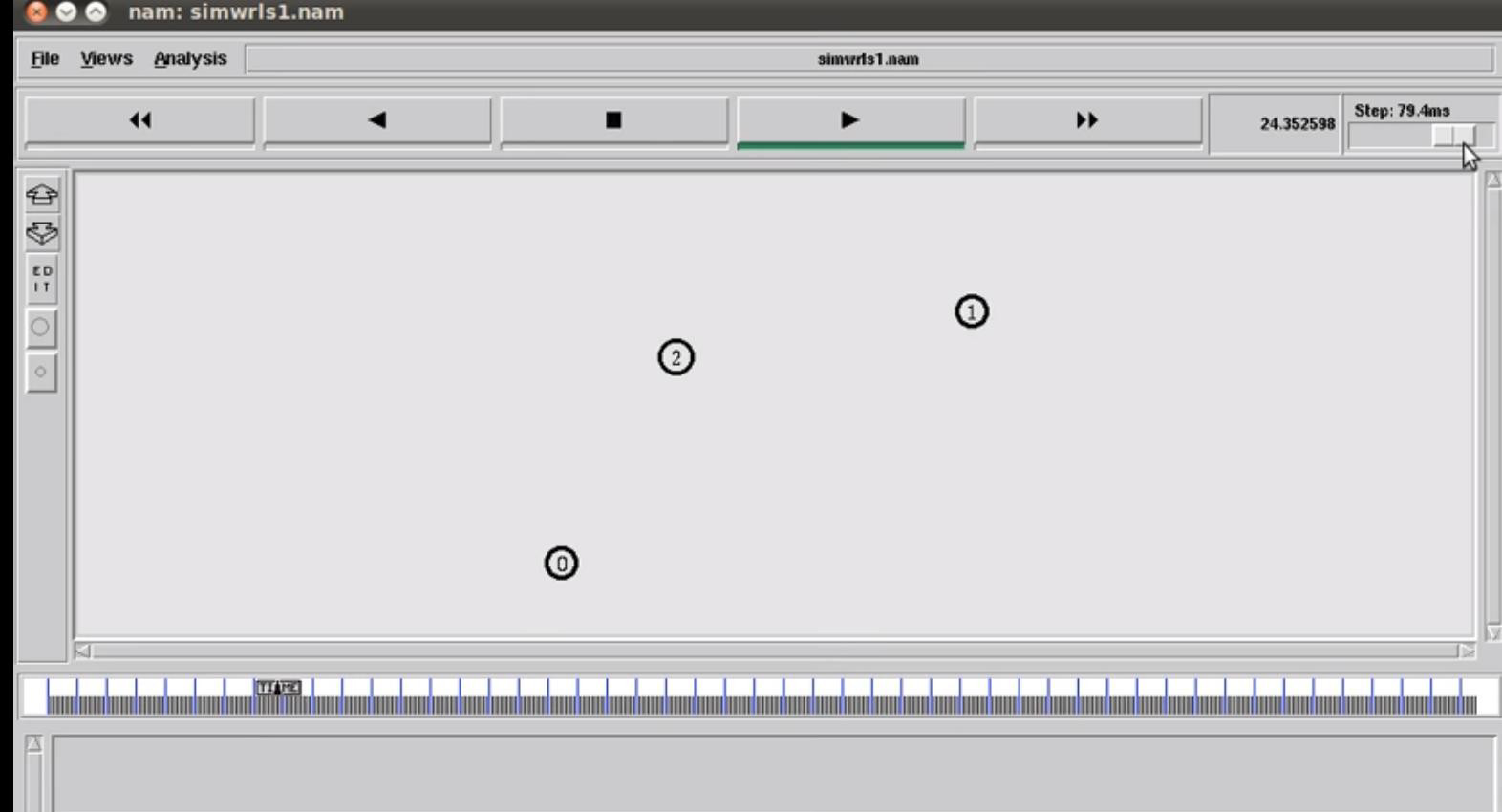
$ns flush-trace close $tracefd close $namtrace

exec nam simwrls1.nam &

}

$ns run

### Output



## Experiment 12

### Write a program for congestion control using Leaky bucket algorithm.

### Description

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

### Algorithm:

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#define NOF\_PACKETS 10

int rand(int a)

{

int rn = (random() % 10) % a;

return rn == 0 ? 1 : rn;

}

int main()

{

int packet\_sz[NOF\_PACKETS], i, clk, b\_size, o\_rate, p\_sz\_rm=0, p\_sz, p\_time, op;

for(i = 0; i<NOF\_PACKETS; ++i)

packet\_sz[i] = rand(6) \* 10;

for(i = 0; i<NOF\_PACKETS; ++i)

printf("\npacket[%d]:%d bytes\t", i, packet\_sz[i]);

printf("\nEnter the Output rate:");

scanf("%d", &o\_rate);

printf("Enter the Bucket Size:");

scanf("%d", &b\_size);

for(i = 0; i<NOF\_PACKETS; ++i)

{

if( (packet\_sz[i] + p\_sz\_rm) > b\_size)

if(packet\_sz[i] > b\_size)/\*compare the packet siz with bucket size\*/

printf("\n\nIncoming packet size (%dbytes) is Greater than bucket capacity (%dbytes)-PACKET REJECTED", packet\_sz[i], b\_size);

else

printf("\n\nBucket capacity exceeded-PACKETS REJECTED!!");

else

{

p\_sz\_rm += packet\_sz[i];

printf("\n\nIncoming Packet size: %d", packet\_sz[i]);

printf("\nBytes remaining to Transmit: %d", p\_sz\_rm);

p\_time = rand(4) \* 10;

printf("\nTime left for transmission: %d units", p\_time);

for(clk = 10; clk <= p\_time; clk += 10)

{

sleep(1);

if(p\_sz\_rm)

{

if(p\_sz\_rm <= o\_rate)/\*packet size remaining comparing with output rate\*/

op = p\_sz\_rm, p\_sz\_rm = 0;

else

op = o\_rate, p\_sz\_rm -= o\_rate;

printf("\nPacket of size %d Transmitted", op);

printf("----Bytes Remaining to Transmit: %d", p\_sz\_rm);

}

else

{

printf("\nTime left for transmission: %d units", p\_time-clk);

printf("\nNo packets to transmit!!");

}

}

}

}

}

*Screenshots:-* 