

**ALBERTIAN INSTITUTE OF SCIENCE AND TECHNOLOGY**  
**AISAT-TECHNICAL CAMPUS**  
Archbishop Angel Mary Nagar  
Cochin University P. O., Kochi – 6820 22

**DEPARTMENT OF  
COMPUTER SCIENCE AND ENGINEERING**



**CSL:332 NETWORKING LAB MANUAL**

**S6 CSE**

**Published by:**  
Department of Computer Science & Engineering, AISAT

## **Dept. of Computer Science & Engineering**

### **Vision**

To create highly competent computer science and engineering professionals who are technically sound, employable and socially committed towards nation.

### **Mission**

- Groom competent computer science engineers by providing high quality professional training and research ambience.
- Enhance innovations, leadership qualities, team-spirit and ethical value based living by collaborating in multidisciplinary activities.
- Promote industry institute interaction and to encourage entrepreneurship skills among students.

### **Program Educational Objectives PEOs**

Graduates of Computer Science and Engineering shall

**PEO1:** Evolve as globally competent professionals, researchers and entrepreneurs.

**PEO2:** Develop innovative computing solutions for real world problems.

**PEO3:** Demonstrate communication skills, leadership skills, ethical values and team work in their profession.

### **Program Specific Outcomes (PSOs)**

Graduates of Computer Science and Engineering will be able to

**PSO1:** Design and develop computer programs and computer based systems of moderate complexity in the areas relevant to the current computational demands of the global society.

**PSO2:** Gather knowledge in the various domains of Computer Science and identify a suitable career path to succeed in life.

**PSO3:** Apply the fundamentals of computer science in order to evaluate potential risks and provide creative solutions to meet the needs of society.

**INDEX**

<b>SL. NO.</b>	<b>CONTENTS</b>	<b>PAGE NO.</b>
1.	SYLLABUS	5
2.	FAMILARIZATION OF THE NETWORK CONFIGURATION FILES AND NETWORKING COMMANDS IN LINUX.	6
3.	FAMILIARIZE AND UNDERSTAND THE USE AND FUNCTIONING OF SYSTEM CALLS USED FOR NETWORK PROGRAMMING IN LINUX	9
4.	IMPLEMENT CLIENT-SERVER COMMUNICATION USING SOCKET PROGRAMMING AND TCP AS TRANSPORT LAYER PROTOCOL	12
5.	IMPLEMENT CLIENT-SERVER COMMUNICATION USING SOCKET PROGRAMMING AND UDP AS TRANSPORT LAYER PROTOCOL	21
5.	SIMULATE SLIDING WINDOW FLOW CONTROL PROTOCOLS	31
6.	IMPLEMENT AND SIMULATE ALGORITHM FOR DISTANCE VECTOR ROUTING PROTOCOL	59
7.	IMPLEMENT AND SIMULATE ALGORITHM FOR LINK STATE ROUTING PROTOCOL	72
8.	IMPLEMENT SIMPLE MAIL TRANSFER PROTOCOL	78
9.	IMPLEMENT FILE TRANSFER PROTOCOL	94
10.	IMPLEMENT CONGESTION CONTROL USING A LEAKY BUCKET ALGORITHM	109
11.	UDP DATAGRAM IN CLIENT SERVER COMMUNICATION USING WIRESHARK	114
12.	3 WAY HANDSHAKE DURING TCP CONNECTION ESTABLISHMENT USING WIRESHARK	117
13.	PACKET CAPTURING AND FILTERING APPLICATION USING RAW SOCKETS	119

14.	DESIGN AND CONFIGURATION OF NETWORK AND SERVICES	123
15.	SIMULATION USING NS2 SIMULATOR	125
16.	VIVA QUESTIONS	138

**SYLLABUS**  
**CSL 332: NETWORKING LAB**

**Teaching scheme**

3 hours practical per week

**Credits: 2**

**Objectives**

- *To get hands-on experience in network programming using Linux System calls and network monitoring tools.*
- *To develop, implement protocols and evaluate its performance for real world networks.*

**\*Mandatory**

(Note: At least one program from each topic in the syllabus should be completed in the Lab)

1. Getting started with the basics of network configuration files and networking commands in Linux.\*
2. To familiarize and understand the use and functioning of system calls used for network programming in Linux.\*
3. Implement client-server communication using socket programming and TCP as transport layer protocol\*
4. Implement client-server communication using socket programming and UDP as transport layer protocol\*
5. Simulate sliding window flow control protocols.\* (Stop and Wait, Go back N, Selective Repeat ARQ protocols)
6. Implement and simulate algorithm for Distance Vector Routing protocol or Link State Routing protocol.\*
7. Implement Simple Mail Transfer Protocol.
8. Implement File Transfer Protocol.\*
9. Implement congestion control using a leaky bucket algorithm.\*
10. Understanding the Wireshark tool.\*
11. Design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure commonly used services in the network.\*
12. Study of NS2 simulator\*

## **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

#### **1. /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
```

#### **2. /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
```

#### **3. /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.

#### **4. /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

## 1. 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.

## 2. netstat

This 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.

### 3. ping

This is the most commonly used command for checking connectivity.

```
ping www.google.com
```

```
PING 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 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**

### 1. **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.

### 2. **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.

### 3. 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.

### 4. wait

When a process terminates, its parent should receive some information regarding 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* immediately 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.

## 5. exit

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

## 6. 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.

## 7. 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.

## 8. 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 system 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:*

#### **1. 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**.

#### **2. 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.

### **3. 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.

#### **4. 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 success and -1 in case of failure.

### 5. 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);
```

### 6. 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);
```

### 7. closing the connection

The connection can be closed using the *close* system call

```
int close( int sd);
```

*Steps for TCP Connection for server*

#### 1. 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**.

#### 2. 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**.

#### 3. 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.

#### 4. 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

1. Create listening socket
2. bind IP address and port number to the socket
3. listen for incoming requests on the listening socket
4. accept the incoming request

5. connection socket is created when *accept* returns
6. Read the string using the connection socket from the client
7. Reverse the string
8. Send the string to the client using the connection socket
9. close the connection socket
10. 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));
```

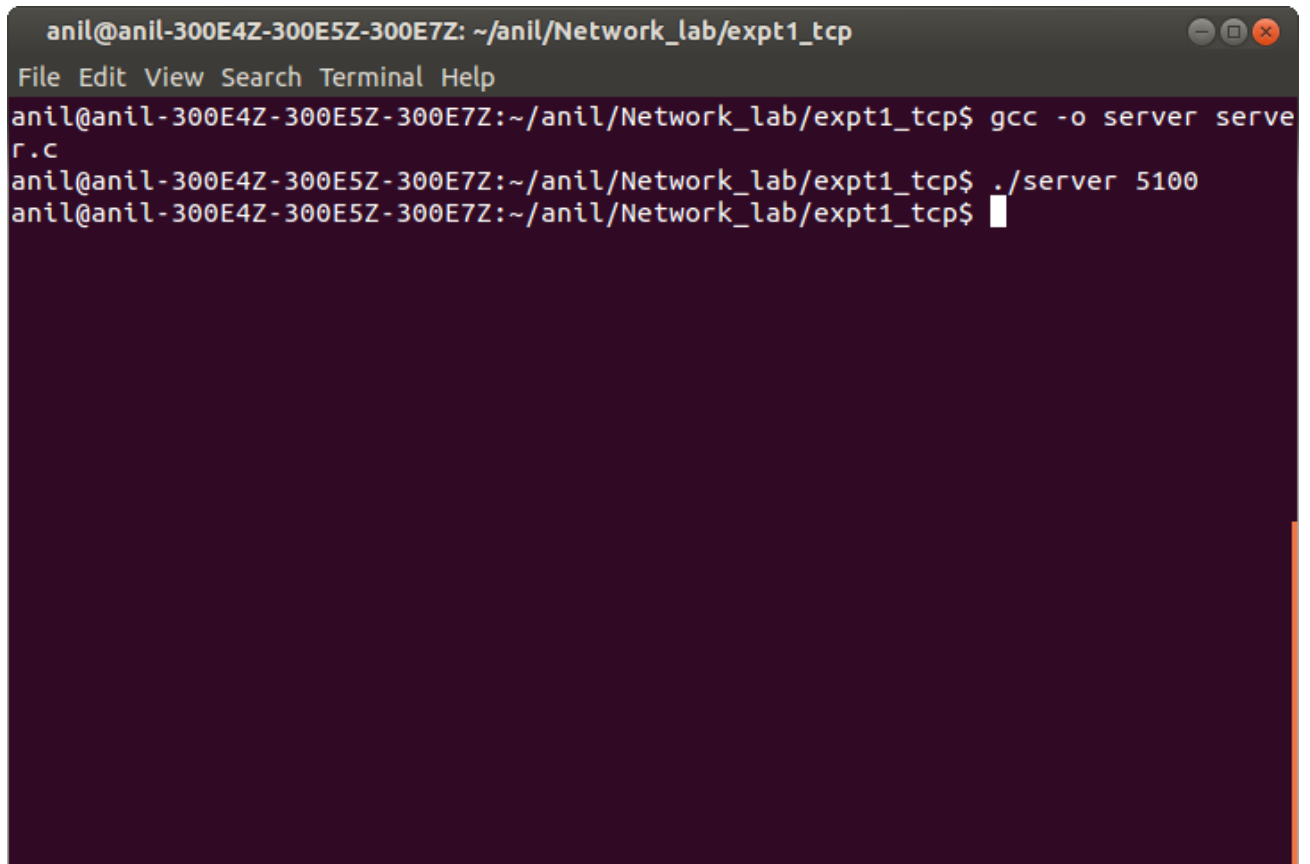
```
close(data);
```

```
close(sd);
```

```
}
```

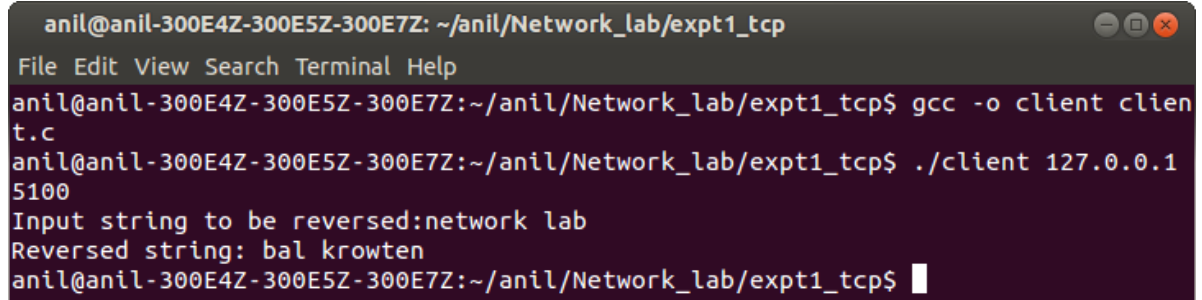
## **Output**

### **Server**

A terminal window with a dark background and light text. The title bar shows the user 'anil' and the directory '~/anil/Network\_lab/expt1\_tcp'. The terminal contains the following commands and output:

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ gcc -o server server.c
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ ./server 5100
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$
```

## Client



A terminal window titled "anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network\_lab/expt1\_tcp" with standard window controls. The terminal shows the following commands and output:

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp$ gcc -o client client.c
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp$ ./client 127.0.0.1 5100
Input string to be reversed:network lab
Reversed string: bal krowten
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp$
```

## 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*

#### **1. 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**.

#### **2. 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.

### **3. 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.

### **4. 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 originating 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.

## 5. 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

1. Create socket
2. bind IP address and port number to the socket
3. Read the matrices socket from the client using socket
4. Find product of matrices
5. Send the product matrix to the client using socket
6. 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++)
```

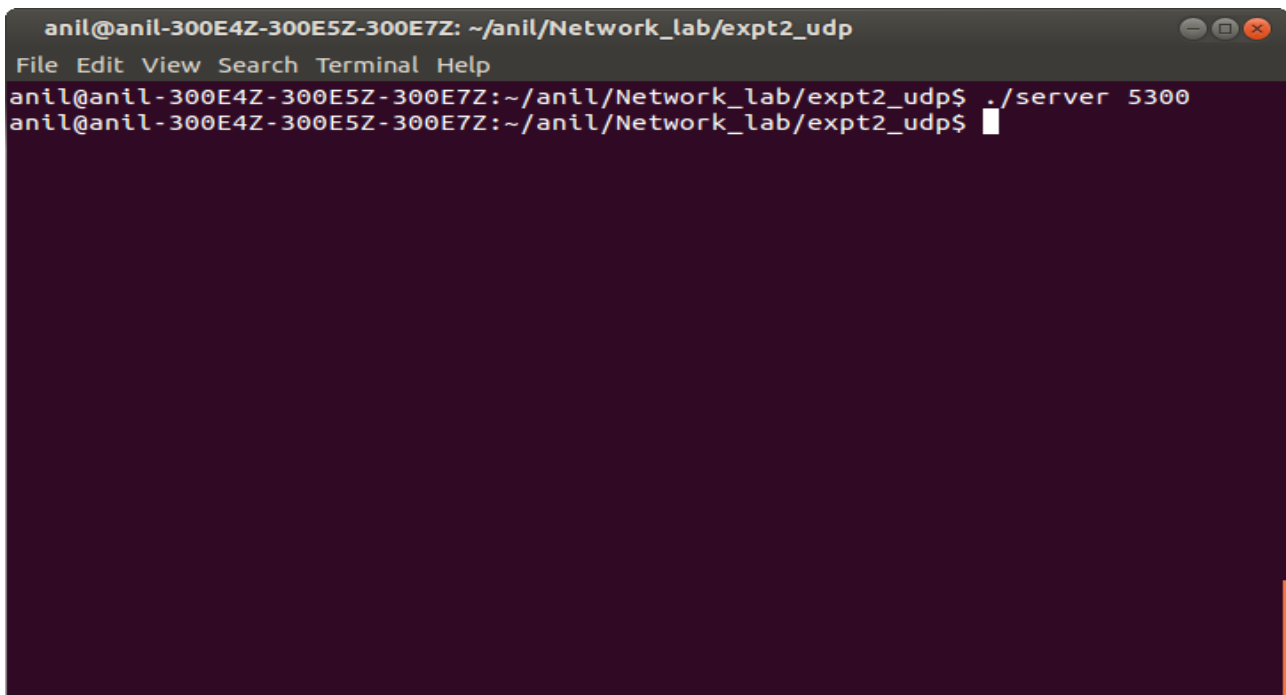
```
{
    matrix_product[i][j] += matrix_1[i][k]*matrix_2[k][j];
}

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);
}
```

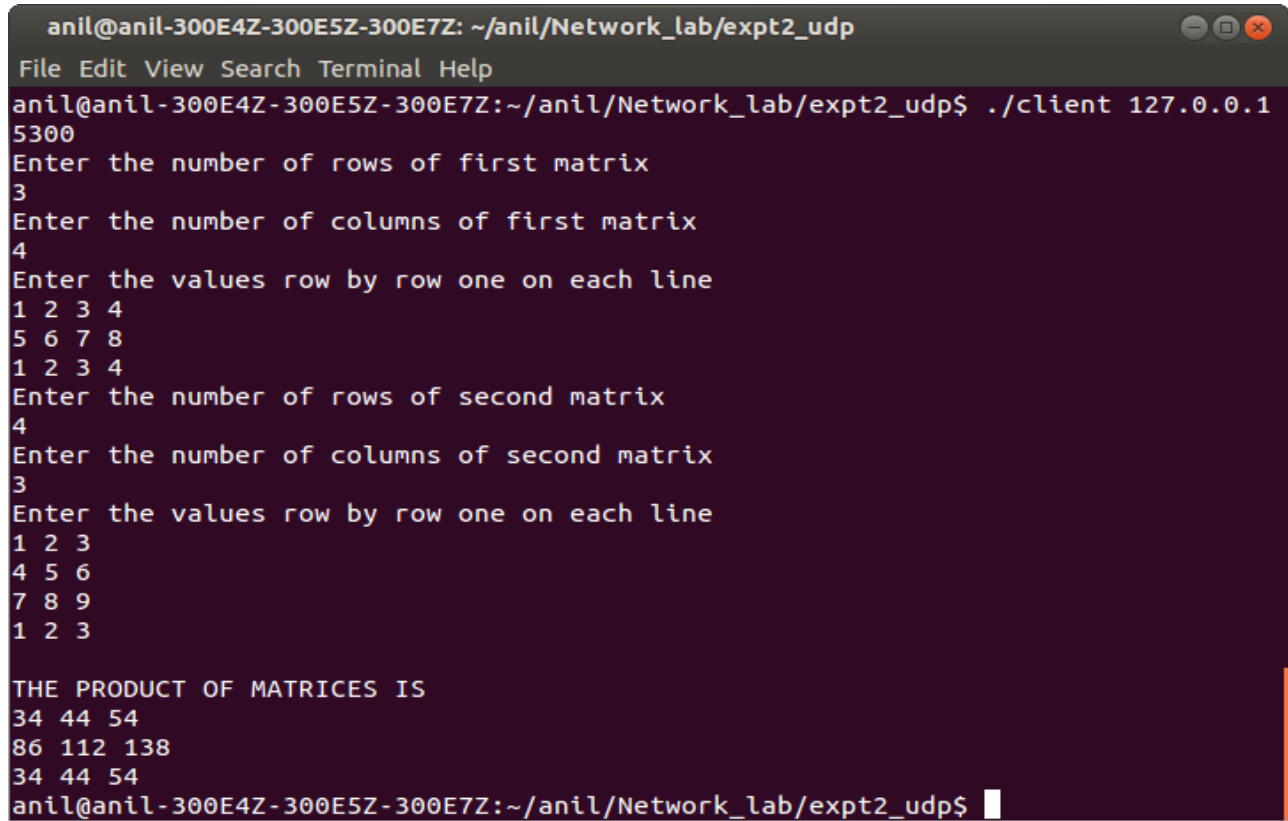
## Output

### Server

A terminal window with a dark purple background and white text. The title bar at the top reads 'anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network\_lab/expt2\_udp'. Below the title bar is a menu bar with 'File Edit View Search Terminal Help'. The terminal shows two lines of command history: 'anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network\_lab/expt2\_udp\$ ./server 5300' and 'anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network\_lab/expt2\_udp\$'. A white cursor is visible at the end of the second line.

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt2_udp
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$ ./server 5300
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$
```

## Client



```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt2_udp
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$ ./client 127.0.0.1
5300
Enter the number of rows of first matrix
3
Enter the number of columns of first matrix
4
Enter the values row by row one on each line
1 2 3 4
5 6 7 8
1 2 3 4
Enter the number of rows of second matrix
4
Enter the number of columns of second matrix
3
Enter the values row by row one on each line
1 2 3
4 5 6
7 8 9
1 2 3

THE PRODUCT OF MATRICES IS
34 44 54
86 112 138
34 44 54
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$
```

## **Experiment 5**

**Simulate sliding window flow control protocols. (Stop and Wait, Go back N, Selective Repeat ARQ protocols)**

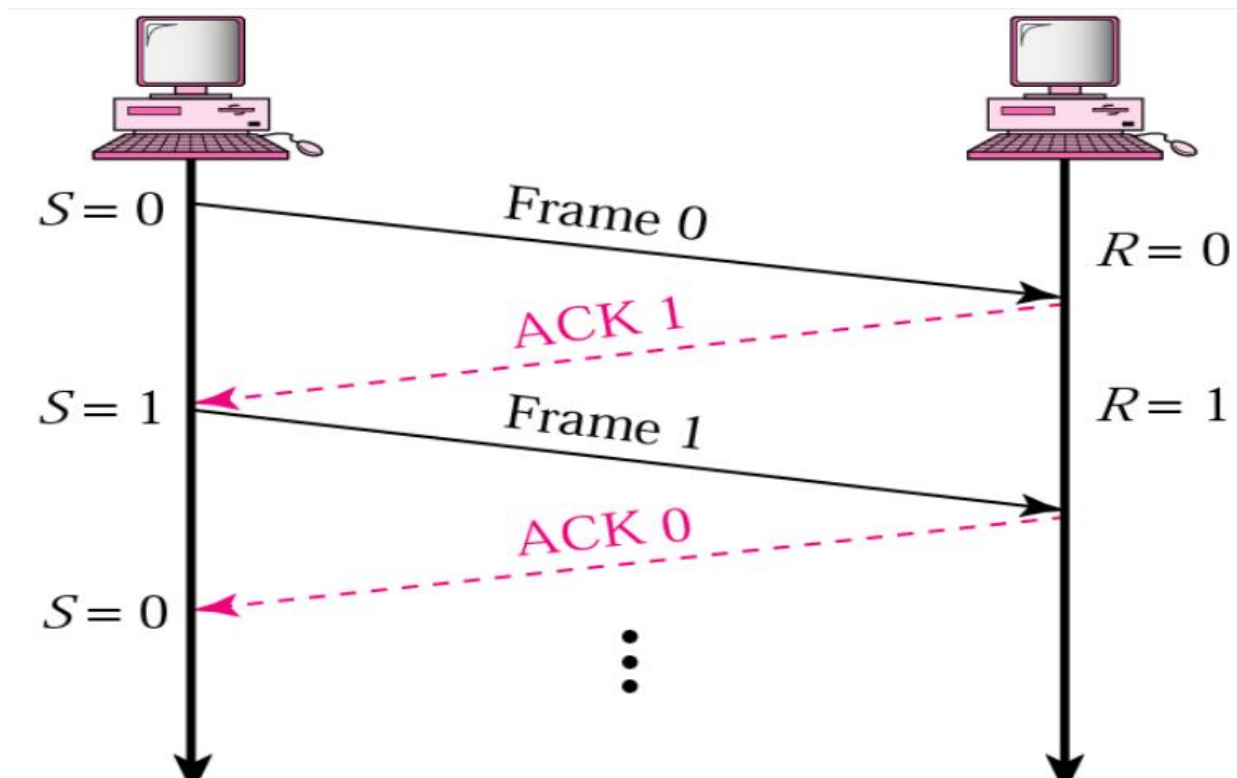
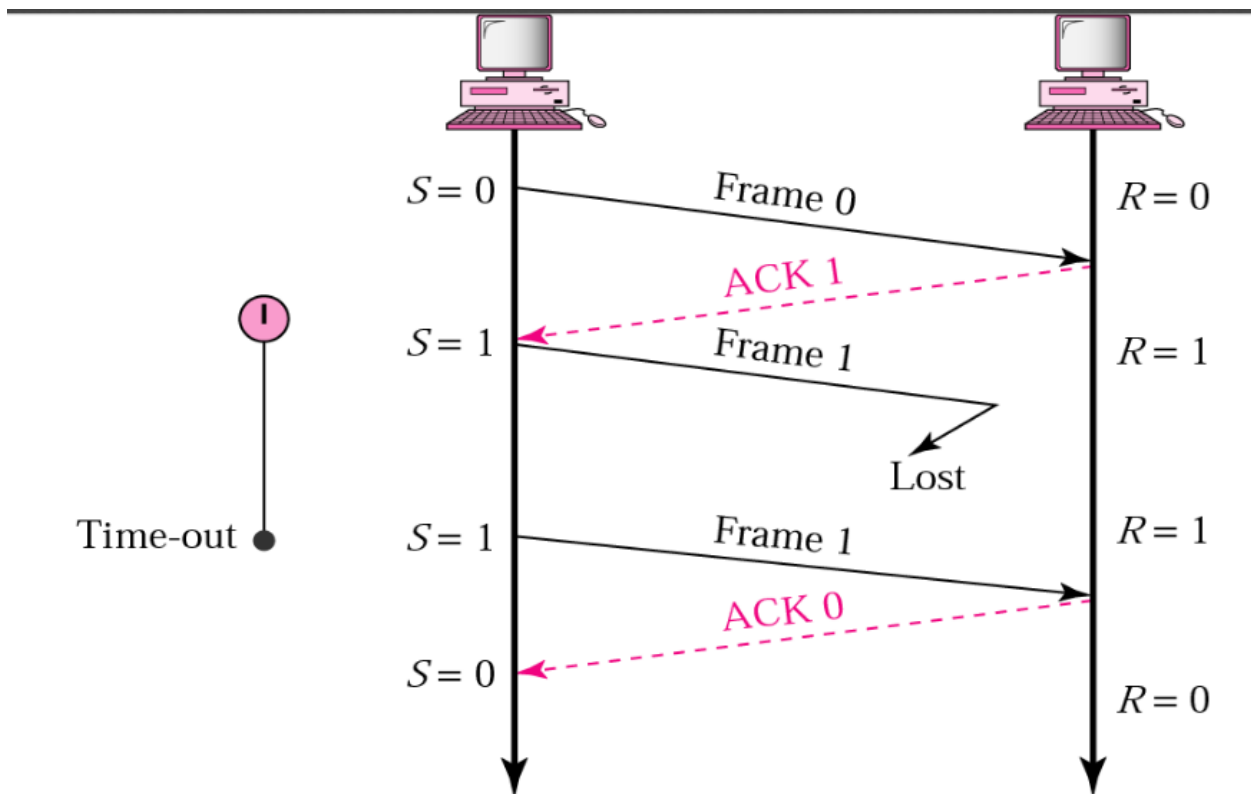
**Aim: To simulate sliding window protocols (Stop and Wait, Go back N, Selective Repeat ARQ protocols)**

### **Description of Stop-and-Wait Protocol**

Stop-and-wait Protocol is a flow control protocol used in the data link layer for transmission of data in noiseless channels. Sender keeps on sending messages to the Receiver. In order to prevent the receiver from overwhelming, there is a need to tell the sender to slow down the transmission of frames. We can make use of feedback from the receiver to the sender. Frames 0 sends to receiver, ACK 1 will be sentback to sender; frame 1 goes to receiver, ACK 0 will be back to sender, and so on

### **Algorithm of Stop-and-Wait Protocol**

1. Start the program
2. Generate a random number that gives the total number of frames to be transmitted.
3. Transmit the first frame
4. Receive the acknowledgement for the first frame
5. Transmit the next frame
6. Find the remaining frames to be sent.
7. If an acknowledgement is not received for a particular frame, retransmit that frame alone again.
8. Repeat the steps 5 to 7 till the number of remaining frames to be sent becomes zero.
9. Stop the program.





**Program****CLIENT SIDE**

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8080
void stop_and_wait(int client_socket){
    char frame[100];
    int seqNo = 0;
    char ack[6];
    int i=0;
    int next = 1;
    char exit[4] = {'e', 'x', 'i', 't'};

    while(1){
        int noExit = 0;
        if(next==1){
            bzero(frame,100);
            i=1;
            printf("\nEnter message to server : ");

            while((frame[i++]=getchar())!='\n');
            frame[i-1]='\0';

            frame[0] = seqNo;

        }

        sleep(1.5);
        send(client_socket,frame,sizeof(frame),0); //Send frame

        bzero(ack,6); //Empty ack

        recv(client_socket,ack,sizeof(ack),0); //Receive ack
        if(ack[0]=='A'){

            //If frame acknowledged by server, proceed to next frame by setting next=1

            printf("Acknowledgement received from server for frame %d - [" ,seqNo);
            for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);
            printf("]\n");
            seqNo = seqNo==0?1:0;
        }
    }
}

```

```

        next = 1;
    }
    else{
        //If frame not acknowledged, resend same frame by setting next =0
        next = 0;
        printf("Negative acknowledgement received from server for frame %d - [",seqNo);
        for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);
        printf("\nResending frame %d - [",seqNo);
        for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);
        printf("\n");
    }

    //If frame contains the message "exit", exit the network
    for(int j=1;j<5;j++){
        if(frame[j]!=exit[j-1]) {
            noExit = 1;
            break;
        }
    }
    if(noExit==0){
        printf("Client has successfully exited the network ... \n");
        return;
    }

}

}

```

### Server Side

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8080
#define n 4

//For demonstration of loss of frames, server sends negative ack for every nth frame

void stop_and_wait(int client_socket){
    char frame[100]; //Initializations
    int seqNo = 0;
    char ack[6];
    int i=0;

```

## CSL 332: Networking Lab

```

int count = 1;
char exit[4] = {'e','x','i','t'};
while(1){
    int noExit = 0;
    bzero(frame,100);    //Empty frame

    recv(client_socket,frame,sizeof(frame),0); //Receive frame from client
    bzero(ack,6); //Empty ack

    if(frame[0] == seqNo && count%n!=0){

        //If expected sequence number received, send ack and change seqNo
        ack[0] = 'A';
        ack[1] = 'C';
        ack[2] = 'K';
        ack[3] = seqNo;
        ack[4] = '\0';

        printf("Frame received from client : ");
        for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);
        printf("\n");

        seqNo = seqNo==0?1:0;
    }
    else{

        //If expected sequence number not received, send negative ack
        ack[0] = 'N';
        ack[1] = 'A';
        ack[2] = 'C';
        ack[3] = 'K';
        ack[4] = seqNo;
        ack[5] = '\0';
    }
    sleep(1);
    send(client_socket,ack,sizeof(ack),0);

    //If frame contains the message "exit", exit the network
    for(int j=1;j<5;j++){
        if(frame[j]!=exit[j-1]) {
            noExit = 1;
            break;
        }
    }
    if(noExit==0){
        printf("Server has successfully exited the network ... \n");
        return;
    }
}

```

# CSL 332: Networking Lab

```

        count++;

    }

}

int main(){

    //Initialize socket descriptor
    int server_socket;

    //create a socket
    server_socket = socket(PF_INET,SOCK_STREAM,0);

    if(server_socket<0){
        printf("Error creating socket ...\n");
        exit(1);
    }
    else{
        printf("Socket created successfully ...\n");
    }

    //Binding socket to port
    //Initializing sockaddr_in structure before binding

    struct sockaddr_in sa;
    bzero(&sa,sizeof(sa));
    sa.sin_family = PF_INET; //Refers to anything in the protocol
    sa.sin_port = htons(PORT); //Port number 8080
    sa.sin_addr.s_addr = inet_addr("127.0.0.7");

    if( bind(server_socket,(struct sockaddr*)&sa, sizeof(sa))== 0){
        printf("Socket binded successfully ...\n");
    }
    else{
        printf("Unable to bind server... An error has occurred \n");
        exit(1);
    }

    //Listen

    if (listen(server_socket,10)==0){
        printf("Server listening...\n");
    }
    else{
        printf("Server listen failed\n");
    }

    //Accept connection
    struct sockaddr_in cli;

```

## CSL 332: Networking Lab

```
int len = sizeof(cli);
int client_socket = accept(server_socket, (struct sockaddr*)&cli, &len);

if(client_socket < 0){
    printf("Failed to accept client\n");
    exit(1);
}
else{
    printf("Server accepted client\n");
}
stop_and_wait(client_socket);
close(server_socket);
```

### FIRST EXECUTE SERVER AND THEN ONLY CLIENT

#### Output - Client side

```
net@inlab:~$ gcc client.c -o c
net@inlab:~$ ./c
Socket creation successful
Connected to server
```

```
Enter message to server : hello
Acknowledgement received from server for frame 0 - [hello]
```

```
Enter message to server : welcome to lmcs
Acknowledgement received from server for frame 1 - [welcome to AISAT]
```

```
Enter message to server : God Bless you
Acknowledgement received from server for frame 0 - [God Bless you]
```

```
Enter message to server : Wow!
Negative acknowledgement received from server for frame 1 - [Wow!]
Resending frame 1 - [Wow!]
Acknowledgement received from server for frame 1 - [Wow!]
```

```
Enter message to server : Bye
Acknowledgement received from server for frame 0 - [Bye]
```

```
Enter message to server : exit
Negative acknowledgement received from server for frame 0 - [exit]
Resending frame 0 - [exit]
Client has successfully exited the network ...
```

#### Output - Server side

```
net@inlab:~$ gcc server.c -o s
net@inlab:~$ ./s
Socket created successfully ...
Socket binded successfully ...
Server listening...
```

Server accepted client  
Frame received from client : hello  
Frame received from client : welcome to AISAT  
Frame received from client : God Bless you  
Frame received from client : Wow!  
Frame received from client : Bye  
Frame received from client : EXIT  
Server has successfully exited the network ...  
net@inlab:~\$

### **Description of Go Back N**

**Go-Back-N ARQ** is mainly a specific instance of Automatic Repeat Request (**ARQ**) **protocol** where the sending process continues to send a number of frames as specified by the window size even without receiving an acknowledgement (**ACK**) **packet** from the receiver. The sender keeps a copy of each frame until the acknowledgement arrives.

This protocol is a practical approach to the sliding window.

- In Go-Back-N ARQ, the size of the sender window is N and the size of the receiver window is always 1.
- This protocol makes the use of **cumulative acknowledgements** means here the receiver maintains an acknowledgement timer.
- If the receiver receives a corrupted frame, then it silently discards that corrupted frame and the correct frame is retransmitted by the sender after the timeout timer expires.
- In case if the receiver receives the out of order frame then it simply discards all the frames.
- In case if the sender does not receive any acknowledgement then the frames in the entire window will be retransmitted again.

### **Disadvantages**

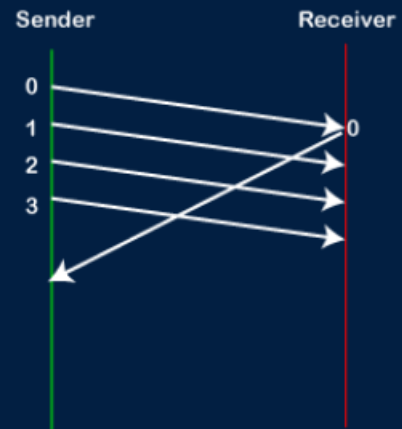
- Timeout timer runs at the receiver side only.
- The transmitter needs to store the last N packets.
- The retransmission of many error-free packets follows an erroneous packet.

## WORKING OF GO-BACK-N ARQ

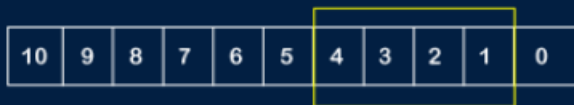


Sliding Window

Window Size: 4

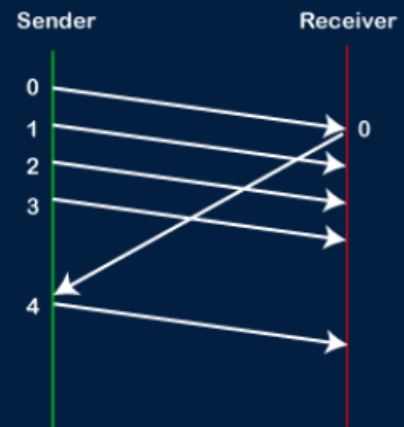


## WORKING OF GO-BACK-N ARQ

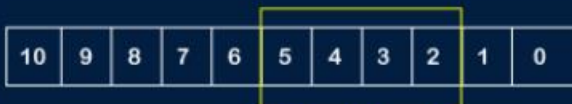


Sliding Window

Window Size: 4

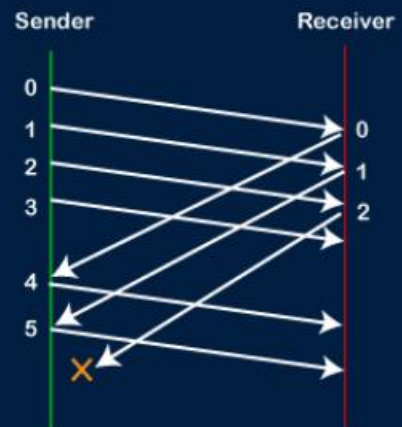


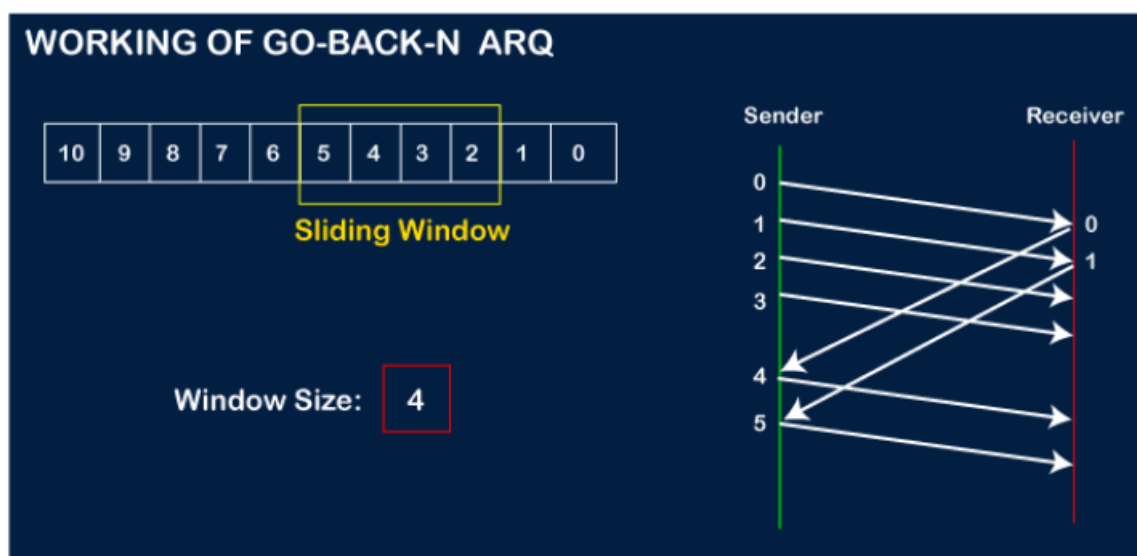
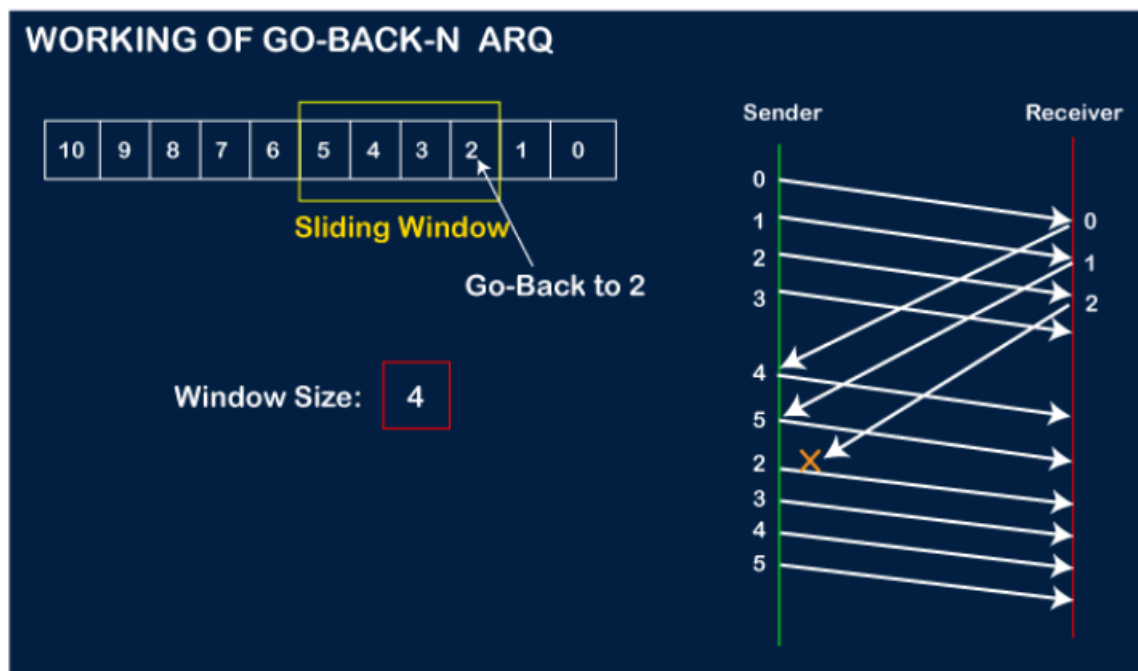
## WORKING OF GO-BACK-N ARQ



Sliding Window

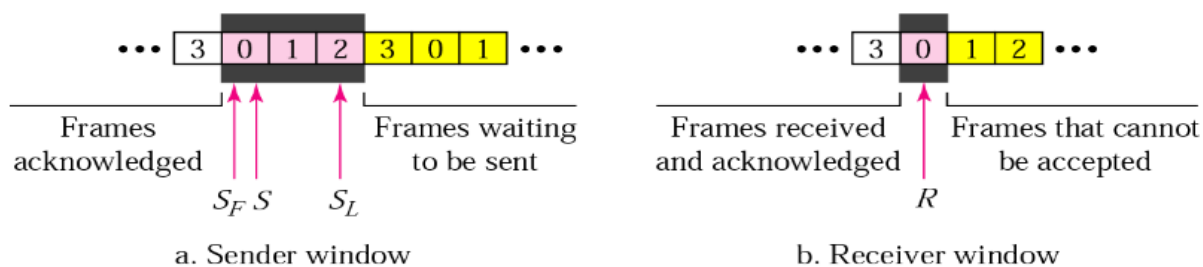
Window Size: 4



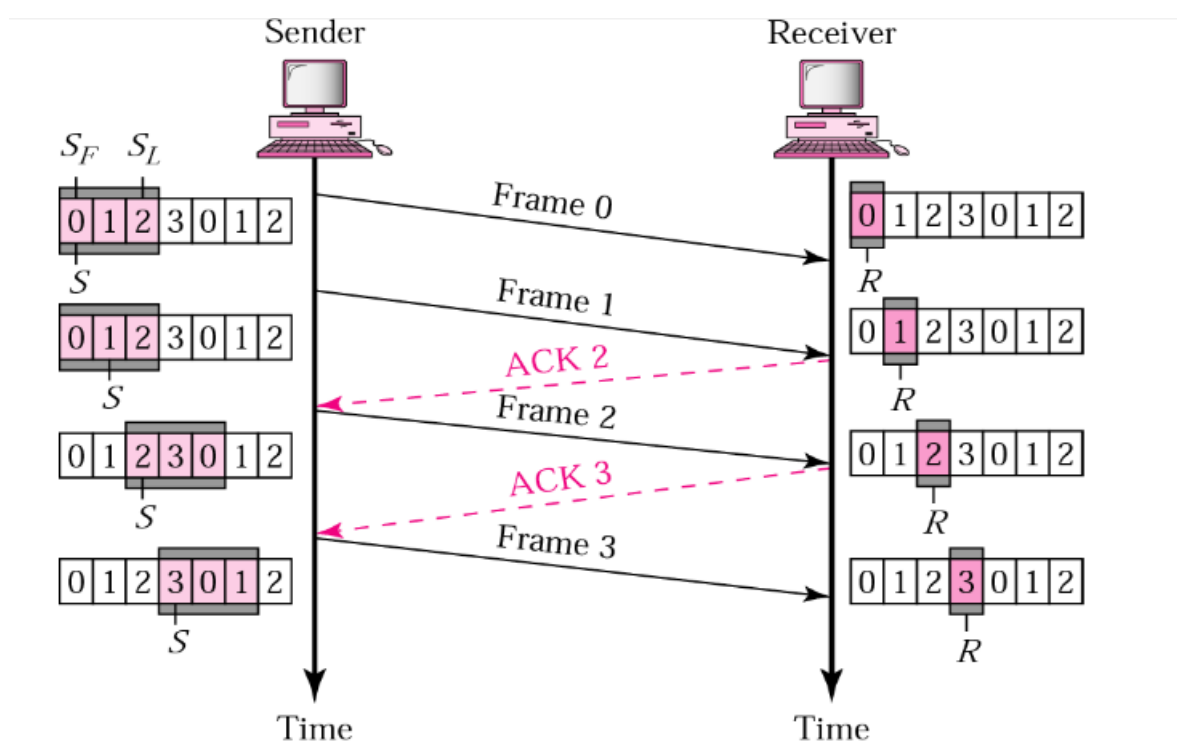


SF is the sequence number of the first frame in the slide window, SL is the sequence number of the last frame in the slide window. R is the sequence number of the excepted frame.  $W = SL - SF + 1 = 2m - 1$ . Only when R and sequence number of received frame are matched, frame accept, otherwise discard it.





## Go-Back-N ARQ control variable [1]



## Go-Back-N ARQ normal operation [1]

Frame 0 & 1 send, ACK 1 & 2 back to sender. Frame 2 send, ACK 3 back to sender.

*Go-Back-N sender algorithm*

```

1   $S_w = 2^n - 1$ ;
2   $S_f = 0$ ;
3   $S_n = 0$ ;
4
5  while (true)                                //Repeat forever
6  {
7      WaitForEvent();
8      if(Event(RequestToSend))                //A packet to send
9      {
10         if( $S_n - S_f \geq S_w$ )                  //If window is full
11             Sleep();
12         GetData();
13         MakeFrame( $S_n$ );
14         StoreFrame( $S_n$ );
15         SendFrame( $S_n$ );
16          $S_n = S_n + 1$ ;
17         if(timer not running)
18             StartTimer();
19     }
20
21     if(Event(ArrivalNotification)) //ACK arrives
22     {
23         Receive(ACK);
24         if(corrupted(ACK))
25             Sleep();
26         if( $\{ackNo > S_f\} \&\& \{ackNo \leq S_n\}$ ) //If a valid ACK
27             While( $S_f \leq ackNo$ )
28             {
29                 PurgeFrame( $S_f$ );
30                  $S_f = S_f + 1$ ;
31             }
32             StopTimer();
33     }
34
35     if(Event(TimeOut))                    //The timer expires
36     {
37         StartTimer();
38         Temp =  $S_f$ ;
39         while(Temp <  $S_n$ );
40         {
41             SendFrame( $S_f$ );
42              $S_f = S_f + 1$ ;
43         }
44     }
45 }

```

*Go-Back-N receiver algorithm*

```

1   $R_n = 0$ ;
2
3  while (true)                                //Repeat forever
4  {
5      WaitForEvent();
6
7      if(Event(ArrivalNotification)) //Data frame arrives
8      {
9          Receive(Frame);
10         if(corrupted(Frame))
11             Sleep();
12         if(seqNo ==  $R_n$ )                //If expected frame
13         {
14             DeliverData();              //Deliver data
15              $R_n = R_n + 1$ ;              //Slide window
16             SendACK( $R_n$ );
17         }
18     }
19 }

```

**Algorithm 8** GoBack-N Protocol - Sender

---

```

1:  $S_w \leftarrow 2^m - 1$ 
2:  $S_f = S_n = 0$ 
3: while True do
4:   WaitForEvent()
5:   if Event(RequestToSend) then
6:     if  $S_n - S_f \geq S_w$  then
7:       Sleep()
8:     end if
9:     GetData()
10:    MakeFrame( $S_n$ )
11:    StoreFrame( $S_n$ )
12:    SendFrame( $S_n$ )
13:     $S_n \leftarrow (S_n + 1) \% S_w$ 
14:    if Timer is not running then
15:      StartTimer()
16:    end if
17:  end if
18:  if Event(ArrivalNotification) then
19:    Receive(ACK)
20:    if Corrupted(ACK) then
21:      Sleep()
22:    end if
23:    if  $ackNo > S_f$  and  $ackNo \leq S_n$  then
24:      while  $S_f \leq ackNo$  do
25:        PurgeFrame( $S_n$ )
26:         $S_f \leftarrow (S_f + 1) \% S_w$ 
27:      end while
28:    end if
29:    StopTimer()
30:  end if
31:  if Event(Timeout) then
32:    StartTimer()
33:     $temp \leftarrow S_f$ 
34:    while  $temp < S_n$  do
35:      SendFrame( $S_n$ )
36:       $S_f \leftarrow (S_f + 1) \% S_w$ 
37:    end while
38:  end if
39: end while

```

---

**Algorithm 9** GoBack-N Receiver

---

```

1:  $R_n \leftarrow 0$ 
2: while True do
3:   WaitForEvent()
4:   if Event(ArrivalNotification) then
5:     Receive(frame)
6:     if Corrupted(frame) then
7:       Sleep()
8:     end if
9:     if  $seqNo == R_n$  then
10:      DeliverData()
11:       $R_n \leftarrow (R_n + 1) \% 2^m$ 
12:    end if
13:    SendACK( $R_n$ )
14:  end if
15: end while

```

---

**GoBackN.c**

```

#include<stdio.h>
#include<time.h>
#include<stdlib.h>

int main()
{
    int nf,N;
    int tr=0;
    srand(time(NULL));
    printf("Enter the number of frames : ");
    scanf("%d",&nf);
    printf("Enter the Window Size : ");
    scanf("%d",&N);
    int i=1;
    while(i<=nf)
    {
        int x=0;
        for(int j=i;j<i+N && j<=nf;j++)
        {
            printf("Sent Frame %d \n", j);
            tr++;
        }
        for(int j=i;j<i+N && j<=nf;j++)
        {
            int flag = rand()%2;
            if(!flag)
            {
                printf("%d : Acknowledged! \n", j);
                x++;
            }
            else
            {
                printf("Frame %d Not Received \n", j);
                printf("Retransmitting Window \n");
                break;
            }
        }
        printf("\n");
        i+=x;
    }
    printf("Total number of transmissions : %d \n", tr);
    return 0;
}

```

**Output**

```

gcc goBackN.c
net@inlab:~$ ./a.out
Enter the number of
frames : 5
Enter the Window Size : 2
Sent Frame 1
Sent Frame 2
1 : Acknowledged!
Frame 2 Not Received
Retransmitting Window

```

## CSL 332: Networking Lab

Sent Frame 2  
 Sent Frame 3  
 2 : Acknowledged!  
 3 : Acknowledged!

Sent Frame 4  
 Sent Frame 5  
 4 : Acknowledged!  
 5 : Acknowledged!

Total number of  
 transmissions : 6

### Go-Back N Client/Server Implementation in C gbnclient.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/time.h>
#include<sys/wait.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
int main() {
    int c_sock;
    c_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in client;
    memset(&client, 0, sizeof(client));
    client.sin_family = AF_INET;
    client.sin_port = htons(9009);
    client.sin_addr.s_addr = inet_addr("127.0.0.1");
    if(connect(c_sock, (struct sockaddr*)&client, sizeof(client)) == -1)
    {
        printf("Connection failed");
        return 0;
    }
    printf("\n\tClient -with individual acknowledgement scheme\n\n");
    char msg1[50]="acknowledgement of :";
    char msg2[50];
    char buff[100];
    int flag=1,flg=1;
```

```

for(int i=0;i<=9;i++) {
    flg=1;
    bzero(buff,sizeof(buff));
    bzero(msg2,sizeof(msg2));
    if(i==8&&flg==1){
        printf("here\n"); //simulating loss
        flg=0;
        read(c_sock,buff,sizeof(buff));
    }
    int n = read(c_sock, buff, sizeof(buff));
    if(buff[strlen(buff)-1]!=i+'0'){ //out of order
        printf("Discarded as out of order \n");
        i--;
    }
    else{
        printf("Message received from server : %s \t %d\n",buff,i);
        printf("Acknowledgement sent for message \n");
        strcpy(msg2,msg1);
        msg2[strlen(msg2)]=i+'0';
        write(c_sock,msg2, sizeof(msg2));
    }
}
close(c_sock);
return 0;
}

```

**gbnserv.c**

```

#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<sys/time.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<fcntl.h>
int main() {
    int s_sock, c_sock;
    s_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server, other;
    memset(&server, 0, sizeof(server));
    memset(&other, 0, sizeof(other));
    server.sin_family = AF_INET;
    server.sin_port = htons(9009);
    server.sin_addr.s_addr = INADDR_ANY;
    socklen_t add;

```

## CSL 332: Networking Lab

```

if(bind(s_sock, (struct sockaddr*)&server, sizeof(server)) == -1) {
printf("Binding failed\n");
return 0;
}
printf("\tServer Up\n Go back n (n=3) used to send 10 messages \n\n");
listen(s_sock, 10);
add = sizeof(other);
c_sock = accept(s_sock, (struct sockaddr*)&other, &add);
time_t t1,t2;
char msg[50]="server message :";
char buff[50];
int flag=0;
fd_set set1,set2,set3;
struct timeval timeout1,timeout2,timeout3;

```

```

int rv1,rv2,rv3;
int i=-1;
qq:
i=i+1;
bzero(buff,sizeof(buff));
char buff2[60];
bzero(buff2,sizeof(buff2));
strcpy(buff2,"server message :");
buff2[strlen(buff2)]=i+'0';
buff2[strlen(buff2)]='\0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
usleep(1000);
i=i+1;
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(msg)]=i+'0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
i=i+1;
usleep(1000);
qqq:
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(msg)]=i+'0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
FD_ZERO(&set1);
FD_SET(c_sock, &set1);
timeout1.tv_sec = 2;
timeout1.tv_usec = 0;
rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);

```

```

if(rv1 == -1)
perror("select error ");
else if(rv1 == 0){
printf("Going back from %d:timeout \n",i);
i=i-3;
goto qq;}
else{
read(c_sock, buff, sizeof(buff));
printf("Message from Client: %s\n", buff);
i++;
if(i<=9)
goto qq;
}
qq2:
FD_ZERO(&set2);
FD_SET(c_sock, &set2);
timeout2.tv_sec = 3;
timeout2.tv_usec = 0;
rv2 = select(c_sock + 1, &set2, NULL, NULL, &timeout2);
if(rv2 == -1)
perror("select error "); // an error accured
else if(rv2 == 0){
printf("Going back from %d:timeout on last 2\n",i-1);
i=i-2;
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(buff2)]=i+'0';
write(c_sock, buff2, sizeof(buff2));
usleep(1000);
bzero(buff2,sizeof(buff2));
i++;
strcpy(buff2,msg);
buff2[strlen(buff2)]=i+'0';
write(c_sock, buff2, sizeof(buff2));
goto qq2;} // a timeout occured
else{
read(c_sock, buff, sizeof(buff));
printf("Message from Client: %s\n", buff);
bzero(buff,sizeof(buff));
read(c_sock, buff, sizeof(buff));
printf("Message from Client: %s\n", buff);
}
close(c_sock);
close(s_sock);
return 0;
}

```



**Output**

```
gcc gbserver.c -o s
```

```
cca@labb30:~$ ./s
```

```
Server Up
```

```
Go back n (n=3) used to send 10 messages
```

```
Message sent to client :server message :0
```

```
Message sent to client :server message :1
```

```
Message sent to client :server message :2
```

```
Message from Client: acknowledgement of0
```

```
Message sent to client :server message :3
```

```
Message from Client: acknowledgement of1
```

```
Message sent to client :server message :4
```

```
Message from Client: acknowledgement of2
```

```
Message sent to client :server message :5
```

```
Message from Client: acknowledgement of3
```

```
Message sent to client :server message :6
```

```
Going back from 6:timeout
```

```
Message sent to client :server message :4
```

```
Message sent to client :server message :5
```

```
Message sent to client :server message :6
```

```
Message from Client: acknowledgement of4
```

```
Message sent to client :server message :7
```

```
Message from Client: acknowledgement of5
```

```
Message sent to client :server message :8
```

```
Message from Client: acknowledgement of6
```

```
Message sent to client :server message :9
```

```
Message from Client: acknowledgement of7
```

```
Going back from 9:timeout on last 2
```

```
Message from Client: acknowledgement of8
```

```
Message from Client: acknowledgement of9
```

```
gcc gbclient.c -o c
```

```
cca@labb30:~$ ./c
```

```
Client -with individual acknowledgement scheme
```

```
Message received from server : server message :0    0
```

```
Acknowledgement sent for message
```

```
Message received from server : server message :1    1
```

```
Acknowledgement sent for message
```

```
Message received from server : server message :2    2
```

```
Acknowledgement sent for message
```

```
Message received from server : server message :3    3
```

```
Acknowledgement sent for message
```

```
Discarded as out of order
```

```
Discarded as out of order
```

```
Discarded as out of order
```

```
Message received from server : server message :4    4
```

```
Acknowledgement sent for message
```

```
Message received from server : server message :5    5
```

```
Acknowledgement sent for message
```

Message received from server : server message :6     6  
 Acknowledgement sent for message  
 Message received from server : server message :7     7  
 Acknowledgement sent for message  
 here  
 Discarded as out of order  
 Message received from server : server message :8     8  
 Acknowledgement sent for message  
 Message received from server : server message :9     9  
 Acknowledgement sent for message

### **Description of Selective Repeat ARQ protocols**

Selective repeat protocol, also called Selective Repeat ARQ (Automatic Repeat reQuest), is a data link layer protocol that uses sliding window method for reliable delivery of data frames. Here, only the erroneous or lost frames are retransmitted, while the good frames are received and buffered.

It uses two windows of equal size: a sending window that stores the frames to be sent and a receiving window that stores the frames received by the receiver. The size is half the maximum sequence number of the frame. For example, if the sequence number is from 0 – 15, the window size will be 8.

### **Working Principle**

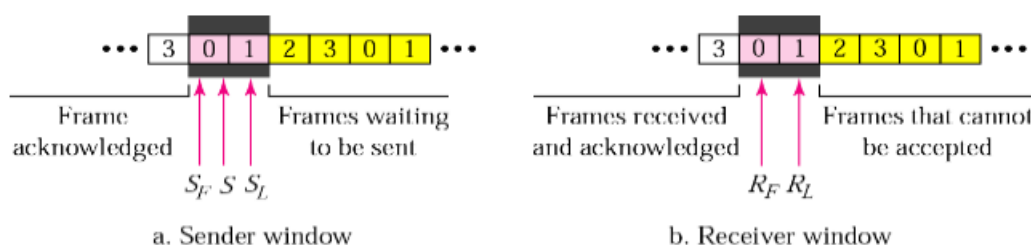
Selective Repeat protocol provides for sending multiple frames depending upon the availability of frames in the sending window, even if it does not receive acknowledgement for any frame in the interim. The maximum number of frames that can be sent depends upon the size of the sending window.

The receiver records the sequence number of the earliest incorrect or un-received frame. It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with every acknowledgement frame.

The sender continues to send frames that are in its sending window. Once, it has sent all the frames in the window, it retransmits the frame whose sequence number is given by the acknowledgements. It then continues sending the other frames.

The control variables in Selective Repeat ARQ are same as in Go-Back-N ARQ:

SF, SL and S. But the sender sliding window size changed into  $2^{m-1}$ . Receiver sliding window has 2 control variables, RF and RL.



Selective Repeat ARQ receiver slide window [1]

**Sender Side Algorithm for Selective Repeat ARQ Protocol**

```

begin
  frame s; //s denotes frame to be sent
  frame t; //t is temporary frame
  S_window = power(2,m-1); //Assign maximum window size
  SeqFirst = 0; // Sequence number of first frame in window
  SeqN = 0; // Sequence number of Nth frame window
  while (true) //check repeatedly
  do
    Wait_For_Event(); //wait for availability of packet
    if ( Event(Request_For_Transfer)) then
      //check if window is full
      if (SeqN-SeqFirst >= S_window) then
        doNothing();
      end if;
      Get_Data_From_Network_Layer();
      s = Make_Frame();
      s.seq = SeqN;
      Store_Copy_Frame(s);
      Send_Frame(s);
      Start_Timer(s);
      SeqN = SeqN + 1;
    end if;
    if ( Event(Frame_Arrival) then
      r = Receive_Acknowledgement();
      //Resend frame whose sequence number is with ACK
      if ( r.type = NAK) then
        if ( NAK_No > SeqFirst && NAK_No < SeqN ) then
          Retransmit( s.seq(NAK_No));
          Start_Timer(s);
        end if
      //Remove frames from sending window with positive ACK
      else if ( r.type = ACK ) then
        Remove_Frame(s.seq(SeqFirst));
        Stop_Timer(s);
        SeqFirst = SeqFirst + 1;
      end if
    end if
  end while
end begin

```

```

    end if
    end if
    // Resend frame if acknowledgement haven't been received
    if ( Event(Time_Out)) then
        Start_Timer(s);
        Retransmit_Frame(s);
    end if
end

```

### Receiver Side Algorithm for Selective Repeat ARQ Protocol

Begin

```

frame f;
RSeqNo = 0; // Initialise sequence number of expected frame
NAKsent = false;
ACK = false;
For each slot in receive_window
Mark(slot)=false;
while (true) //check repeatedly
do
    Wait_For_Event(); //wait for arrival of frame
    if ( Event(Frame_Arrival) then
        Receive_Frame_From_Physical_Layer();
        if ( Corrupted ( f.SeqNo ) AND NAKsent = false) then
            SendNAK(f.SeqNo);
            NAKsent = true;
        end if
        if ( f.SeqNo != RSeqNo AND NAKsent = false ) then
            SendNAK(f.SeqNo);
            NAKsent = true;
        end if
        if ( f.SeqNo is in receive_window ) then
            if ( Mark(RSeqNo) = false ) then
                Store_frame(f.SeqNo);
                Mark(RSeqNo) = true;
            end if
        end if
    else
        while ( Mark(RSeqNo))
            Extract_Data(RSeqNo);

```

```

        Deliver_Data_To_Network_Layer();
        RSeqNo = RSeqNo + 1;
        Send_ACK(RSeqNo);
    end while
end if
end if
end while
end

```

## **Program**

### **Selective-Repeat Client/Server Implementation in C**

#### **srclient.c**

```

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

int isfaulty(){ //simulating corruption of message

int d=rand()%4;
return (d>2);

}

int main() {
    srand(time(0));
    int c_sock;
    c_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in client;
    memset(&client, 0, sizeof(client));
    client.sin_family = AF_INET;
    client.sin_port = htons(9009);
    client.sin_addr.s_addr = inet_addr("127.0.0.1");

    if(connect(c_sock, (struct sockaddr*)&client, sizeof(client)) == -1) {
        printf("Connection failed");
        return 0;
    }
}

```

```

}
printf("\n\tClient -with individual acknowledgement scheme\n\n");
char msg1[50]="acknowledgement of";
char msg3[50]="negative ack ";
char msg2[50];
char buff[100];
int count=-1,flag=1;
while(count<8){
bzero(buff,sizeof(buff));
bzero(msg2,sizeof(msg2));
if(count==7&&flag==1){
printf("here\n"); //simulate loss
flag=0;
read(c_sock,buff,sizeof(buff));
continue;
}
int n = read(c_sock, buff, sizeof(buff));
char i=buff[strlen(buff)-1];
printf("Message received from server : %s \n",buff);
int isfault=isfaulty();
printf("corruption status : %d \n",isfault);
printf("Response/acknowledgement sent for message \n");
if(isfault)
strcpy(msg2,msg3);
else{
strcpy(msg2,msg1);
count++;}
msg2[strlen(msg2)]=i;
write(c_sock,msg2, sizeof(msg2));
}

```

**srsrserver.c**

```

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

```

```

void rsendd(int ch,int c_sock){
char buff2[60];
bzero(buff2,sizeof(buff2));
strcpy(buff2,"reserver message :");
buff2[strlen(buff2)]=(ch)+'0';
buff2[strlen(buff2)]='\0';
printf("Resending Message to client :%s \n",buff2);

```

## CSL 332: Networking Lab

```

write(c_sock, buff2, sizeof(buff2));
usleep(1000);
}

nt main() {
int s_sock, c_sock;
s_sock = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in server, other;
memset(&server, 0, sizeof(server));
memset(&other, 0, sizeof(other));
server.sin_family = AF_INET;
server.sin_port = htons(9009);
server.sin_addr.s_addr = INADDR_ANY;
socklen_t add;

if(bind(s_sock, (struct sockaddr*)&server, sizeof(server)) == -1) {
printf("Binding failed\n");
return 0;
}
printf("\tServer Up\n Selective repeat scheme\n\n");
listen(s_sock, 10);
add = sizeof(other);
c_sock = accept(s_sock, (struct sockaddr*)&other, &add);
time_t t1,t2;
char msg[50]="server message :";
char buff[50];
int flag=0;

fd_set set1,set2,set3;
struct timeval timeout1,timeout2,timeout3;
int rv1,rv2,rv3;

int tot=0;
int ok[20];
memset(ok,0,sizeof(ok));

while(tot<9){
int toti=tot;
for(int j=(0+toti);j<(3+toti);j++){
bzero(buff,sizeof(buff));
char buff2[60];
bzero(buff2,sizeof(buff2));
strcpy(buff2,"server message :");
buff2[strlen(buff2)]=(j)+'0';
buff2[strlen(buff2)]='\0';
printf("Message sent to client :%s \t%d\t%d\n",buff2,tot,j);
write(c_sock, buff2, sizeof(buff2));
usleep(1000);
}
}

```

```

for(int k=0+toti;k<(toti+3);k++){
    qq:
    FD_ZERO(&set1);
    FD_SET(c_sock, &set1);
    timeout1.tv_sec = 2;
    timeout1.tv_usec = 0;
    rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);
    if(rv1 == -1)
        perror("select error ");
    else if(rv1 == 0){
        printf("Timeout for message :%d \n",k);
        rsendd(k,c_sock);
        goto qq;} // a timeout occurred
    else{
        read(c_sock, buff, sizeof(buff));
        printf("Message from Client: %s\n", buff);
        if(buff[0]=='n'){
            printf(" corrupt message acknowledgement (msg %d) \n",buff[strlen(buff)-1]-'0');
            rsendd((buff[strlen(buff)-1]-'0'),c_sock);
            goto qq;}
        else
            tot++;
    }
}

close(c_sock);
close(s_sock);
return 0;
}

```

### **Output**

```

gcc srserver.c -o s
cca@lab30:~$ ./s
Server Up
Selective repeat scheme

```

```

Message sent to client :server message :0   0   0
Message sent to client :server message :1   0   1
Message sent to client :server message :2   0   2
Message from Client: acknowledgement of0
Message from Client: acknowledgement of1
Message from Client: acknowledgement of2
Message sent to client :server message :3   3   3
Message sent to client :server message :4   3   4
Message sent to client :server message :5   3   5
Message from Client: negative ack 3

```



```

corrupt message awk (msg 3)
Resending Message to client :reserver message :3
Message from Client: acknowledgement of4
Message from Client: negative ack 5
corrupt message awk (msg 5)
Resending Message to client :reserver message :5
Message from Client: acknowledgement of3
Message from Client: acknowledgement of5
Message sent to client :server message :6  6  6
Message sent to client :server message :7  6  7
Message sent to client :server message :8  6  8
Message from Client: negative ack 6
corrupt message awk (msg 6)
Resending Message to client :reserver message :6
Message from Client: acknowledgement of7
Message from Client: acknowledgement of8
Timeout for message :8
Resending Message to client :reserver message :8
Message from Client: acknowledgement of8

```

```
cca@labb30:~$ gcc srclient.c -o c
```

```
cca@labb30:~$ ./c
```

Client -with individual acknowledgement scheme

```

Message received from server : server message :0
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :1
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :2
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :3
corruption status : 1
Response/acknowledgement sent for message
Message received from server : server message :4
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :5
corruption status : 1
Response/acknowledgement sent for message
Message received from server : reserver message :3
corruption status : 0
Response/acknowledgement sent for message
Message received from server : reserver message :5
corruption status : 0

```

Response/acknowledgement sent for message

*CSL 332: Networking Lab*

Message received from server : server message :6  
corruption status : 1  
Response/acknowledgement sent for message  
Message received from server : server message :7  
corruption status : 0  
Response/acknowledgement sent for message  
Message received from server : server message :8  
corruption status : 0  
Response/acknowledgement sent for message  
here  
Message received from server : reserver message :8  
corruption status : 0  
Response/acknowledgement sent for message

## **Experiment 6**

### **Implementation 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 from its directly attached neighbours, performs some calculations and returns 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 algorithm is applied.

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

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

where  $v$  is a neighbour of node  $x$ .  $d_v(y)$  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  $c(x,v) + d_v(y)$  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  $N$ :

$$D_x(y) = c(x,y) \text{ /* if } y \text{ is not a neighbour of } x, \text{ then } c(x,y) = \infty \text{ */}$$

for each neighbour  $w$ ,

send distance vector  $D_x = \{ D_x(y): y \text{ in } N \}$  to  $w$

loop:

for each  $y$  in  $N$ :

$$D_x(y) = \min \{ c(x,v) + D_v(y) \}$$

If  $D_x(y)$  changed for any destination  $y$  send distance vector  $D_x = \{D_x(y) : y \text{ in } N\}$  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 distance 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)
        msg.RT[i][j][k] = cost[j][k];
    else
        msg.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]] +

```



```

    msg.RT[node_sel][N[to_node_it][h]][i]
        temp1 = cost[to_node_it][N[to_node_it][h]] +
msg.RT[node_sel][N[to_node_it][h]][i];
    }
    }
    h++;
}

msg.RT[to_node_it][to_node_it][i] = temp1;
h=0;

for(i=0; i<num_nodes ; i++)
{
    if( prev[i] != msg.RT[to_node_it][to_node_it][i])
    {
        p=0;
        while(N[to_node_it][p] !=-1 )
        {
            if(p != to_node_it)
            {
                msg.to_node = N[to_node_it][p];
                msg.from_node = to_node_it;
                write(fd[j], &msg, sizeof(msg));
            }
            p++;
        }
        break;
    }

    }
    temp1 =999;
}

if(--nready == 0)
break;
}
}

for(i = 0; i < num_nodes; i++)
{
    for(j = 0; j < num_nodes; j++)
    {
        printf("%d ", msg.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**

#### **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 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 from the 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

Initialization :

$N' = \{u\}$

for all nodes  $v$

if  $v$  is a neighbour of  $u$

then  $D(v) = c(u,v)$

else

$D(v) = \infty$

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);
    }
}

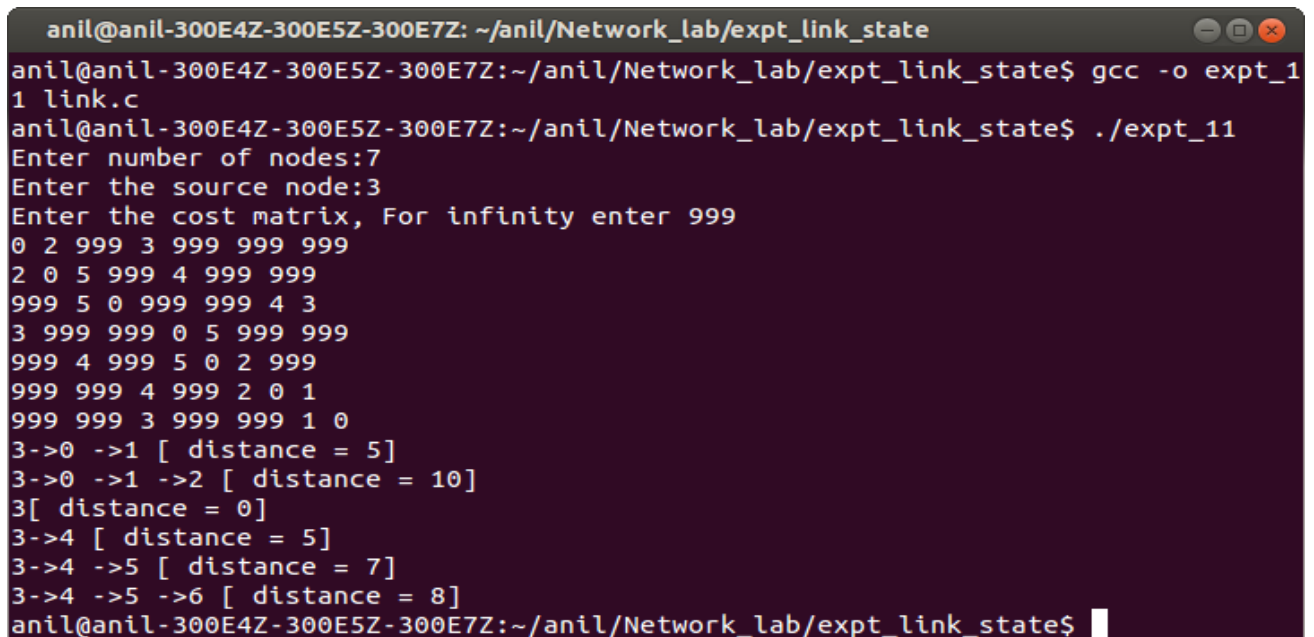
int search(_i)
{
    int i = 0;
```

```
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



```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt_link_state
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_link_state$ gcc -o expt_1
1 link.c
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_link_state$ ./expt_11
Enter number of nodes:7
Enter the source node:3
Enter the cost matrix, For infinity enter 999
0 2 999 3 999 999 999
2 0 5 999 4 999 999
999 5 0 999 999 4 3
3 999 999 0 5 999 999
999 4 999 5 0 2 999
999 999 4 999 2 0 1
999 999 3 999 999 1 0
3->0 ->1 [ distance = 5]
3->0 ->1 ->2 [ distance = 10]
3[ distance = 0]
3->4 [ distance = 5]
3->4 ->5 [ distance = 7]
3->4 ->5 ->6 [ distance = 8]
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_link_state$
```

## **Experiment 8**

### **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 establishment, 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”*
4. Read the first message from the client and print it.
5. Send the second command to the client.  
*250 Hello “client name”*
6. Read the second message from client and print it.
7. Send the third command to the client.  
*250 “client email address “..... Sender ok*
8. Read the third message from client and print it
9. Send the fourth command to the client  
*250 “server email address”..... Receptient ok*
10. Read the fourth message from client and print it
11. Send the fifth command to the client  
*354 Enter mail, end with “.” on a line by itself*
12. Read the email text from the client till a “.” is reached
13. Send the sixth command to the client  
*250 Message accepted for delivery*

14. Read the fifth message from the client and print it.
15. 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. .... Receipient 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;

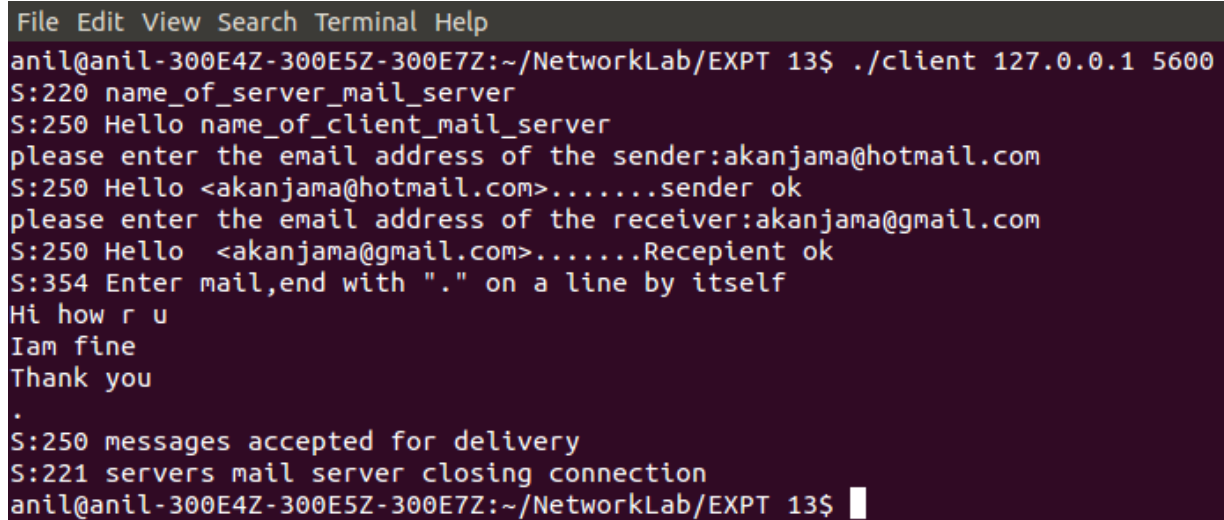
    sprintf(mesg, “250 Message accepted for delivery\n”);
    sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

    n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len);
    mesg[n] = '\0';
    printf(“C:%s\n”, mesg);
```

```
printf(msg, "221 Server's mail server closing connection\n");  
sendto(sock_fd, msg, MAXLINE, 0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));  
  
}
```

### **Output**

### **Client**



```
File Edit View Search Terminal Help  
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$ ./client 127.0.0.1 5600  
S:220 name_of_server_mail_server  
S:250 Hello name_of_client_mail_server  
please enter the email address of the sender:akanjama@hotmail.com  
S:250 Hello <akanjama@hotmail.com>.....sender ok  
please enter the email address of the receiver:akanjama@gmail.com  
S:250 Hello <akanjama@gmail.com>.....Receipient ok  
S:354 Enter mail,end with "." on a line by itself  
Hi how r u  
Iam fine  
Thank you  
.  
S:250 messages accepted for delivery  
S:221 servers mail server closing connection  
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$
```

**Server**

```
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$ ./server 5600
mesg:SMTP REQUEST FROM CLIENT

C:HELLO name_of_client_mail_server

C:MAIL FROM :<akanjama@hotmail.com>
C:RCPT TO : <akanjama@gmail.com>
C:DATA

C:Hi how r u

C:Iam fine

C:Thank you

C:.

C:QUIT

^C
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$
```

## **Experiment 9**

### **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)`

{

2.1 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.

2.2 Read the command from the terminal given by the user

2.3 Send the command to the server using control socket *sock\_ctrl*.

2.4 If command == "close"

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

break

2.5 Enter the filename using the keyboard

2.6 Write the filename using control socket *sock\_ctrl* to the server.

2.7 Use *while(connect(...) < 0)* to wait for the data connection.

2.8 read data using *sock\_data* (file contents) from the server and write to the file

2.9 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*.

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

4. while (1)

{

4.1 accepts the client control connection and returns the connect socket descriptor

*sock\_ctrl*

4.2 fork a child process

4.2.1 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 = getpid 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 = getpid 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)
```

```
}
```

5. close(*listen\_control*);

6. 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**

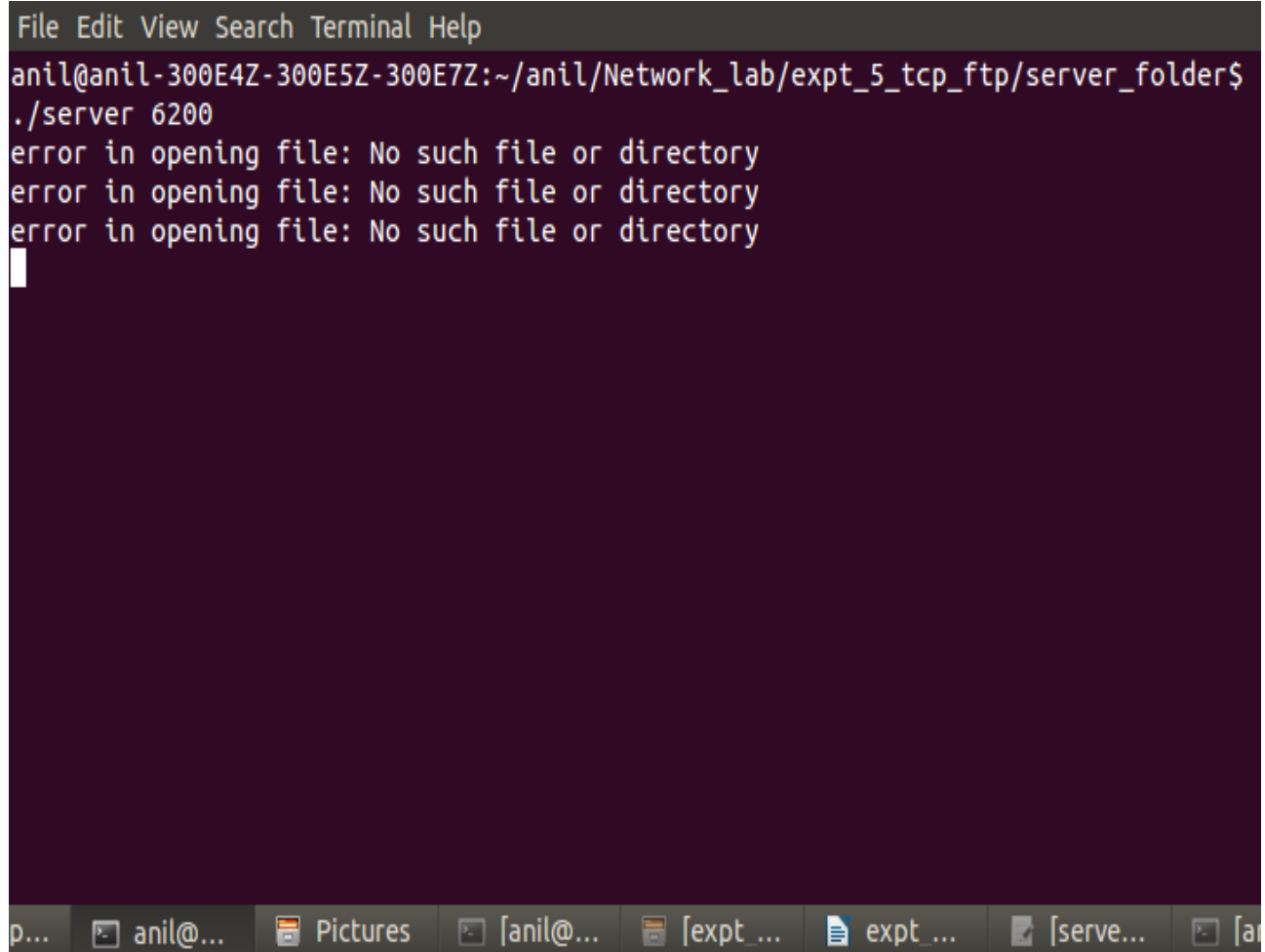
The files stored in the server are *a1*, *b1*, and *c1*. Client 1 connects to the server and gets files *a1* and *b1*. Client 2 connects to server and receives *c1*. The server process id is stored in the files received.

## Client 1

```
File Edit View Search Terminal Help
$ ./client 127.0.0.1 6200
  Enter "get" for receiving file from server
  Enter "close" for closing connection
Enter command:get
Enter filename:a1
name of file = a1
Enter command:get
Enter filename:b1
name of file = b1
Enter command:get
Enter filename:e1
name of file = e1
@FILE NOT FOUND PROCESS ID = 10902
Enter command:close
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/client_1_folder
$ cat a1
Iam file a1
Iam in server
FILE a1 RECEIVED FROM SERVER WITH PROCESS ID = 10902anil@anil-300E4Z-300E5Z-300E
$ cat b1
Iam file b2
Iam in server
FILE b1 RECEIVED FROM SERVER WITH PROCESS ID = 10902anil@anil-300E4Z-300E5Z-300E
$ !~/anil/Network_lab/expt_5_tcp_ftp/client_1_folder$
```

## Client 2

```
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/client_2_folder
$ ./client 127.0.0.1 6200
  Enter "get" for receiving file from server
  Enter "close" for closing connection
Enter command:get
Enter filename:c1
name of file = c1
Enter command:get
Enter filename:d1
name of file = d1
@FILE NOT FOUND PROCESS ID = 10899
Enter command:close
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/client_2_folder
$ cat c1
Iam file b3
Iam in server
FILE c1 RECEIVED FROM SERVER WITH PROCESS ID = 10899anil@anil-300E4Z-300E5Z-300E
$ !~/anil/Network_lab/expt_5_tcp_ftp/client_2_folder$
```

**Server**

The screenshot shows a terminal window with a menu bar at the top containing 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal text shows a user 'anil' at a machine with IP 'anil-300E4Z-300E5Z-300E7Z' in the directory '~/anil/Network\_lab/expt\_5\_tcp\_ftp/server\_folder'. The user has entered the command './server 6200'. The terminal output shows three consecutive error messages: 'error in opening file: No such file or directory'. The terminal window has a dark purple background and a white cursor. At the bottom, there is a taskbar with several open windows: 'p...', 'anil@...', 'Pictures', '[anil@...', '[expt ...', 'expt ...', '[serve...', and '[a...'. The taskbar is light gray with icons for each window.

```
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/server_folder$
./server 6200
error in opening file: No such file or directory
error in opening file: No such file or directory
error in opening file: No such file or directory

```

## **Experiment 10**

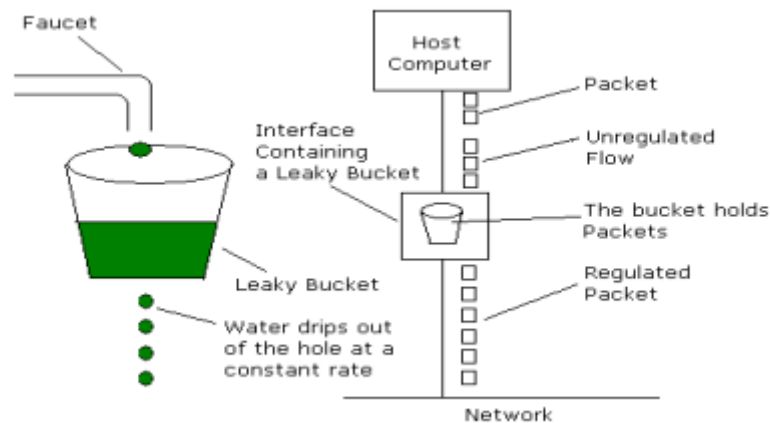
### **Implement Congestion control using a leaky bucket algorithm**

**Aim:** 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 simulated 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
- 7.

**Program**

```
#include<stdio.h>
int main(){
    int incoming, outgoing, buck_size, n, store = 0;
    printf("Enter bucket size, outgoing rate and no of inputs: ");
    scanf("%d %d %d", &buck_size, &outgoing, &n);

    while (n != 0) {
        printf("Enter the incoming packet size : ");
        scanf("%d", &incoming);
        printf("Incoming packet size %d\n", incoming);
        if (incoming <= (buck_size - store)){
            store += incoming;
            printf("Bucket buffer size %d out of %d\n", store, buck_size);
        } else {
            printf("Dropped %d no of packets\n", incoming - (buck_size - store));
            printf("Bucket buffer size %d out of %d\n", store, buck_size);
            store = buck_size;
        }
        store = store - outgoing;
        printf("After outgoing %d packets left out of %d in buffer\n", store, buck_size);
        n--;
    }
}
```

**Output**

Enter bucket size, outgoing rate and no of inputs: 50 100 3

Enter the incoming packet size : 50

Incoming packet size 50

Bucket buffer size 50 out of 50

After outgoing -50 packets left out of 50 in buffer

Enter the incoming packet size : 100

Incoming packet size 100

Bucket buffer size 50 out of 50

After outgoing -50 packets left out of 50 in buffer

Enter the incoming packet size : 20

Incoming packet size 20

Bucket buffer size -30 out of 50

After outgoing -130 packets left out of 50 in buffer

## **Experiment 11**

### **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.

2. 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.

3. Packet-header details window

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

4. 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.

## Output

The image shows a Wireshark network traffic capture. The top pane displays a list of packets. Packet 47 is highlighted, showing it is a UDP datagram from 192.168.43.15 to 52.114.14.0. The bottom pane shows the details of frame 65 (which is packet 47), including Ethernet II, Internet Protocol Version 4, and User Datagram Protocol (UDP) fields.

No.	Time	Source	Destination	Protocol	Length	Info
32	3.438689	2405:204:d30d:dbf6::...	2405:200:800::1	DNS	96	Standard query 0x00e0 A api.cc.skype.com
33	3.446726	192.168.43.15	13.107.17.20	STUN	175	Allocate Request user: 7 bandwidth: 12000 with nonce
35	3.447321	192.168.43.15	13.107.17.20	STUN	175	Allocate Request user: 7 bandwidth: 12000 with nonce
38	3.476777	2405:200:800::1	2405:204:d30d:dbf6::...	DNS	199	Standard query response 0x00e0 A api.cc.skype.com CNAME api-cc-skype.trafficmanager.net CNAME b-cc-asse-01-...
39	3.477210	2405:204:d30d:dbf6::...	2405:200:800::1	DNS	96	Standard query 0xf8ae AAAA api.cc.skype.com
40	3.497062	13.107.17.20	192.168.43.15	STUN	229	Allocate Error Response error-code: 401 (Unauthorized) The request did not contain a Message-Integrity attr...
41	3.498046	13.107.17.20	192.168.43.15	STUN	229	Allocate Error Response error-code: 401 (Unauthorized) The request did not contain a Message-Integrity attr...
46	3.516615	2405:200:800::1	2405:204:d30d:dbf6::...	DNS	260	Standard query response 0xf8ae AAAA api.cc.skype.com CNAME api-cc-skype.trafficmanager.net CNAME b-cc-asse-...
47	3.517132	192.168.43.15	52.114.14.0	UDP	747	64163 → 3478 Len=705
50	3.552597	192.168.43.15	104.44.201.26	STUN	236	Allocate Request user: 7 bandwidth: 12000 realm: 0'0001 0000B00W00200 with nonce
52	3.553158	192.168.43.15	104.44.201.25	STUN	236	Allocate Request user: 7 bandwidth: 12000 realm: 01200108w003100Ha000k with nonce

Frame 65: 299 bytes on wire (2392 bits), 299 bytes captured (2392 bits) on interface 0  
 Ethernet II, Src: XiaomiCo\_fa:19:07 (04:b1:67:fa:19:07), Dst: IntelCor\_ad:6b:23 (b8:03:05:ad:6b:23)  
 Internet Protocol Version 4, Src: 52.114.14.0, Dst: 192.168.43.15  
 User Datagram Protocol, Src Port: 3478, Dst Port: 64163  
 Source Port: 3478  
 Destination Port: 64163  
 Length: 265  
 Checksum: 0x371e [unverified]  
 [Checksum Status: Unverified]

## **Experiment 12**

### **3 way handshake during TCP connection establishment using Wireshark**

**Aim:** To observe 3 way handshake during TCP connection establishment using Wireshark

**Description:**

In this experiment, we upload a file to a server.

The steps are

1. Create a file named input.txt
2. Start Wireshark and begin packet capturing
3. Upload input.txt to a server with IP address 128.119.245.12 using a web page.
4. Stop packet capturing when the file is completely uploaded.

**Output:**

IP address of the client computer is 192.168.43.15 and the source port is 53001. The destination has IP address 128.119.245.12 with port number 80.

#### *Connection Establishment*

The TCP SYN segment is used to initiate the TCP connection between the client computer and server. Frame 24 in the trace shows the transfer of SYN segment from the source to destination computer. It has sequence number 0. The SYN flag is set to 1 and it indicates that this segment is a SYN segment.

The sequence number of the SYN/ACK segment from server to the client computer in reply to the SYN has the value of 0 in the trace and is given by frame 27. The value of the acknowledgement field in the SYN/ACK segment is 1. The value of the acknowledgement field in the SYN/ACK segment is determined by the server by adding 1 to the initial sequence number of SYN segment

from the client computer. The SYN flag and Acknowledgement flag in the segment are set to 1 and they indicate that this segment is a SYNACK segment. Finally an acknowledgement is sent by the client computer to the server with ack number 1 and sequence number 1 as shown in frame 28. This completes TCP connection establishment.

## Data Transfer

Let us look at packet numbers 42 and 43. They are TCP segments carrying data from the source to destination. The sequence number of former segment = 7510, while that of the latter = 8880. The difference between the two sequence numbers =  $8880 - 7510 = 1370$  bytes is the length of the TCP segment.

## Connection Termination

Packet 204 is the TCP segment with FIN and ACK fields set from the destination to source. The source then sends an ACK given by packet 2015

## Output

tcp\_trace\_feb\_2019.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
22	9.762431	2405:204:d208:92bd::...	2001:1af8:4100:b100::...	TCP	74	52998 → 443 [FIN, ACK] Seq=1 Ack=1 Win=4110 Len=0
23	9.763068	192.168.43.15	128.119.245.12	TCP	66	53000 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
24	9.763493	192.168.43.15	128.119.245.12	TCP	66	53001 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
25	10.014400	192.168.43.15	128.119.245.12	TCP	66	53002 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
26	10.061033	2001:1af8:4100:b100::...	2405:204:d208:92bd::...	TCP	74	443 → 52998 [RST] Seq=1 Win=0 Len=0
27	10.141249	128.119.245.12	192.168.43.15	TCP	66	80 → 53001 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1370 SACK_PERM=1 WS=128
28	10.141436	192.168.43.15	128.119.245.12	TCP	54	53001 → 80 [ACK] Seq=1 Ack=1 Win=16440 Len=0
29	10.141701	128.119.245.12	192.168.43.15	TCP	66	80 → 53000 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1370 SACK_PERM=1 WS=128
30	10.141793	192.168.43.15	128.119.245.12	TCP	54	53000 → 80 [ACK] Seq=1 Ack=1 Win=16440 Len=0
31	10.144161	192.168.43.15	128.119.245.12	TCP	713	53001 → 80 [PSH, ACK] Seq=1 Ack=1 Win=16440 Len=659 [TCP segment of a reassembled PDU]
32	10.144718	192.168.43.15	128.119.245.12	TCP	1424	53001 → 80 [ACK] Seq=660 Ack=1 Win=16440 Len=1370 [TCP segment of a reassembled PDU]
33	10.397573	128.119.245.12	192.168.43.15	TCP	66	80 → 53002 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1370 SACK_PERM=1 WS=128
34	10.397750	192.168.43.15	128.119.245.12	TCP	54	53002 → 80 [ACK] Seq=1 Ack=1 Win=16440 Len=0

> Frame 1: 714 bytes on wire (5712 bits), 714 bytes captured (5712 bits) on interface 0

> Ethernet II, Src: IntelCor\_ad:6b:23 (b8:03:05:ad:6b:23), Dst: XiaomiCo\_fa:19:07 (04:b1:67:fa:19:07)

> Internet Protocol Version 6, Src: 2405:204:d208:92bd:5da8:c5c1:16a8:6678, Dst: 2001:1af8:4100:b100::102

> Transmission Control Protocol, Src Port: 52997, Dst Port: 443, Seq: 1, Ack: 1, Len: 640

> Secure Sockets Layer

tcp\_trace\_feb\_2019.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help



Apply a display filter ... &lt;Ctrl-/&gt;

No.	Time	Source	Destination	Protocol	Length	Info
198	13.712773	128.119.245.12	192.168.43.15	TCP	54	80 → 53001 [ACK] Seq=1 Ack=148172 Win=274304 Len=0
199	13.712907	128.119.245.12	192.168.43.15	TCP	54	80 → 53001 [ACK] Seq=1 Ack=149542 Win=277248 Len=0
200	13.717823	128.119.245.12	192.168.43.15	TCP	54	80 → 53001 [ACK] Seq=1 Ack=150912 Win=280064 Len=0
201	13.718117	128.119.245.12	192.168.43.15	TCP	54	80 → 53001 [ACK] Seq=1 Ack=152976 Win=284288 Len=0
202	13.718172	128.119.245.12	192.168.43.15	HTTP	831	HTTP/1.1 200 OK (text/html)
203	13.918034	192.168.43.15	128.119.245.12	TCP	54	53001 → 80 [ACK] Seq=152976 Ack=778 Win=15660 Len=0
204	19.025755	128.119.245.12	192.168.43.15	TCP	54	80 → 53001 [FIN, ACK] Seq=778 Ack=152976 Win=284288 Len=0
205	19.025904	192.168.43.15	128.119.245.12	TCP	54	53001 → 80 [ACK] Seq=152976 Ack=779 Win=15660 Len=0
206	20.002288	2405:204:d208:92bd::...	2001:1af8:4100:b100::...	TLSv1.2	714	Application Data
207	20.167977	XiaomiCo_fa:19:07	IntelCor_ad:6b:23	ARP	42	Who has 192.168.43.15? Tell 192.168.43.1
208	20.168021	IntelCor_ad:6b:23	XiaomiCo_fa:19:07	ARP	42	192.168.43.15 is at b8:03:05:ad:6b:23
209	20.306533	2001:1af8:4100:b100::...	2405:204:d208:92bd::...	TLSv1.2	332	Application Data
210	20.506108	2405:204:d208:92bd::...	2001:1af8:4100:b100::...	TCP	74	52997 → 443 [ACK] Seq=2561 Ack=1033 Win=4045 Len=0

- > Frame 1: 714 bytes on wire (5712 bits), 714 bytes captured (5712 bits) on interface 0
- > Ethernet II, Src: IntelCor\_ad:6b:23 (b8:03:05:ad:6b:23), Dst: XiaomiCo\_fa:19:07 (04:b1:67:fa:19:07)
- > Internet Protocol Version 6, Src: 2405:204:d208:92bd:5da8:c5c1:16a8:6678, Dst: 2001:1af8:4100:b100::102
- > Transmission Control Protocol, Src Port: 52997, Dst Port: 443, Seq: 1, Ack: 1, Len: 640
- > Secure Sockets Layer

## **Experiment 13**

### **Packet capturing and filtering application using raw sockets.**

**Aim:** To develop a packet capturing and filtering application using raw sockets

**Description:**

Normally, an application accesses the network layer using the transport layer. However, if a raw socket is used an application can bypass the transport layer and access the network layer directly.

The raw socket is created using the socket function

```
int sockfd;
```

```
sockfd = socket(AF_INET, SOCK_RAW, protocol);
```

where protocol is one of the constants IPPROTO\_XXX defined in <netinet/in.h> header. The socket created above is an IPv4 raw socket. Only the superuser can create a raw socket. There is no concept of a port number in a raw socket. It is possible to read and write any type of datalink frame also, if, in the above socket creation function, the first parameter is AF\_PACKET and the third parameter has value htons (ETH\_P\_ALL).

The program described here receives frames from an application that sends UDP datagrams. Using the raw socket, we read the frame using the raw socket and filter out the UDP datagrams based on the protocol field in the IP header. The data link frames are read in by the raw socket using *recvfrom*.

**Program**

The program for filtering the UDP packets is given in filter\_new.c

*filter\_new.c*

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

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

```
#include<stdlib.h>
```

```
#include<sys/socket.h>
```

```
#include<sys/types.h>
```

```
#include<linux/if_packet.h>
```

```
#include<netinet/in.h>
```

```
#include<netinet/if_ether.h>
```

```
#include<netinet/ip.h>
```

```
#include<netinet/udp.h>
```

```
#include<netinet/tcp.h>
```

```
#include<arpa/inet.h>
```

```
#include<arpa/inet.h>
```

```
void print_headers(unsigned char * , int );
void udp_header(unsigned char*,int );

int main()
{

    int sock_r_fd, len, n;

    struct sockaddr_in servaddr;

    len = sizeof(servaddr);

    unsigned char * buffer;
    buffer = (unsigned char *)malloc(65536);
    memset(buffer,0, 65536);

    sock_r_fd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));

    if(sock_r_fd < 0)
    {
        perror("Error in socket creation");
        exit(1);
    }

    while(1)
    {

        n = recvfrom(sock_r_fd, buffer, 65536, 0, (struct sockaddr*)&servaddr, &len);

        print_headers(buffer, n);

    }
}

void print_headers(unsigned char * _buffer, int _n)
{
    static int tcp =0;
    static int udp =0;
    static int other_headers =0;
```

```
struct iphdr * ip = (struct iphdr *) (_buffer + sizeof(struct ethhdr));

if(ip->protocol == 6)
{
    ++tcp;
}
else if(ip->protocol == 17)
{
    ++udp;
    udp_header(_buffer, _n);
}
else
{
    other_headers++;
}

printf("\nTCP = %d, UDP = %d, others = %d\n", tcp, udp, other_headers);
}

void udp_header(unsigned char* buffer,int buflen)
{
    int i=0;
    int remaining_data, iphdrlen;
    struct sockaddr_in src, dest;
    unsigned char * data;
    struct iphdr *ip;
    struct udphdr *udp;

    struct ethhdr *eth = (struct ethhdr *)(buffer);
    printf("Ethernet Header\n");

    printf("Source address: %.2x.%.2x.%.2x.%.2x.%.2x.%.2x\n", eth->h_source[0], eth->h_source[1],
eth->h_source[2], eth->h_source[3], eth->h_source[4], eth->h_source[5]);

    printf("Dest address: %.2x.%.2x.%.2x.%.2x.%.2x.%.2x\n", eth->h_source[0], eth->h_source[1], eth-
>h_source[2], eth->h_source[3], eth->h_source[4], eth->h_source[5]);

    printf("Protocol: = %d\n", eth->h_proto);
```

```
ip = (struct iphdr*)(buffer + sizeof(struct ethhdr));
iphdrln = ip->ihl*4;
memset(&src, 0, sizeof(src));
src.sin_addr.s_addr = ip->saddr;
memset(&dest, 0, sizeof(dest));
dest.sin_addr.s_addr = ip->daddr;

printf(" Source IP = %s\n", inet_ntoa(src.sin_addr));

udp = (struct udphdr*)(buffer + iphdrln + sizeof(struct ethhdr));

printf("Source Port: %d\n", ntohs(udp->source));
printf("Destination Port: %d\n", ntohs(udp->dest));
printf("UDP Length: %d\n", ntohs(udp->len));
printf("UDP Checksum: %d\n", ntohs(udp->check));

}
```

### Output

```
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 17$ sudo ./filter
```

akanjama

Ethernet Header

Source address: 00.00.00.00. 0. 0

Dest address: 00.00.00.00. 0. 0

Protocol: = 8

Source IP = 127.0.0.1

Source Port: 53562

Destination Port: 5016

UDP Length: 24

UDP Checksum: 65067

TCP = 0, UDP = 1, others = 0

Ethernet Header

Source address: 00.00.00.00. 0. 0

Dest address: 00.00.00.00. 0. 0

Protocol: = 8

Source IP = 127.0.0.1

Source Port: 53562

Destination Port: 5016

UDP Length: 24

UDP Checksum: 65067

TCP = 0, UDP = 2, others = 0



TCP = 0, UDP = 2, others = 1

TCP = 0, UDP = 2, others = 2

Ethernet Header

Source address: 00.00.00.00. 0. 0

Dest address: 00.00.00.00. 0. 0

Protocol: = 8

Source IP = 127.0.0.1

Source Port: 53562

Destination Port: 5016

UDP Length: 408

UDP Checksum: 65451

TCP = 0, UDP = 3, others = 2

Ethernet Header

Source address: 00.00.00.00. 0. 0

Dest address: 00.00.00.00. 0. 0

Protocol: = 8

Source IP = 127.0.0.1

Source Port: 53562

Destination Port: 5016

UDP Length: 408

UDP Checksum: 65451

TCP = 0, UDP = 4, others = 2

TCP = 0, UDP = 4, others = 3

TCP = 0, UDP = 4, others = 4

Ethernet Header

Source address: 00.00.00.00. 0. 0

Dest address: 00.00.00.00. 0. 0

Protocol: = 8

Source IP = 127.0.0.1

Source Port: 53562

Destination Port: 5016

UDP Length: 408

UDP Checksum: 65451

TCP = 0, UDP = 5, others = 4

Ethernet Header

Source address: 00.00.00.00. 0. 0

Dest address: 00.00.00.00. 0. 0

Protocol: = 8

Source IP = 127.0.0.1

Source Port: 53562

Destination Port: 5016

UDP Length: 408

UDP Checksum: 65451

TCP = 0, UDP = 6, others = 4

TCP = 0, UDP = 6, others = 5

TCP = 0, UDP = 6, others = 6

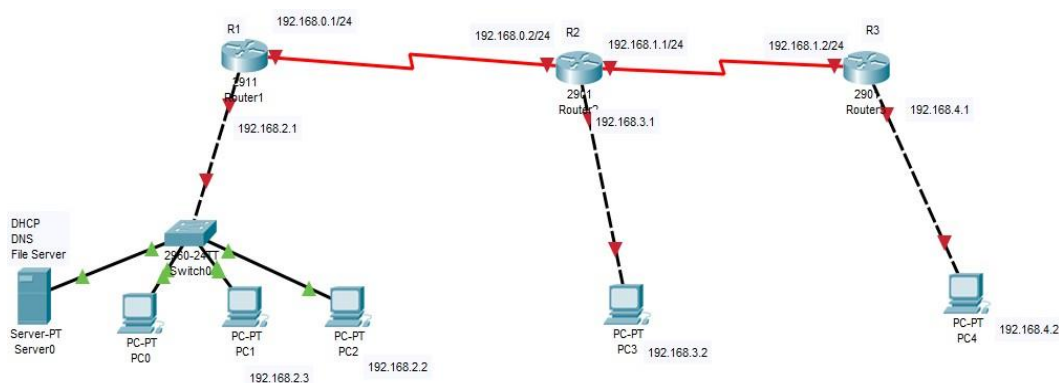
^Canil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 17\$

## Experiment 14

### 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 15**

### **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>

2) Extract the downloaded zip file 'ns-allinone-2.35.tar.gz file' to desktop.

3) 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"
```

4) 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.....**

5) After compleating the installation type the following command in the command window

```
gedit ~/.bashrc
```

6) 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/octl-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
```

7) Save and close the text editor and then type the following command on the terminal

```
source ~/.bashrc
```

8) 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 executing the following command ( it takes 30 to 45 minutes)

```
cd ns-2.35
./validate
```

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

```
ns
```

10) Now type

```
exit
```

## ***Ns2 Commands***

### *Creating wired node*

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

```
[simulator-instance] node
```

### *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:

```
[node-instance] shape <circle|hexagon|box>
```

---

### *Reset all agents of Node*

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

Syntax:

```
[node-instance] reset
```

---

### *Fetch id of Node*

Node id can be fetched in ns2.

Syntax:

```
[node-instance] id
```

---

### *Attach agent to node on a specific port*

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

Syntax:

```
[node-instance] attach-agent [agent-instance] optional:<Port_no
```

---

### *Attach label to node*

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

Syntax:

```
[node-instance] label [label]
```

---

### *Fetch next available port number*

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

Syntax:

```
[node-instance] alloc_port null_agent
```



### *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
```

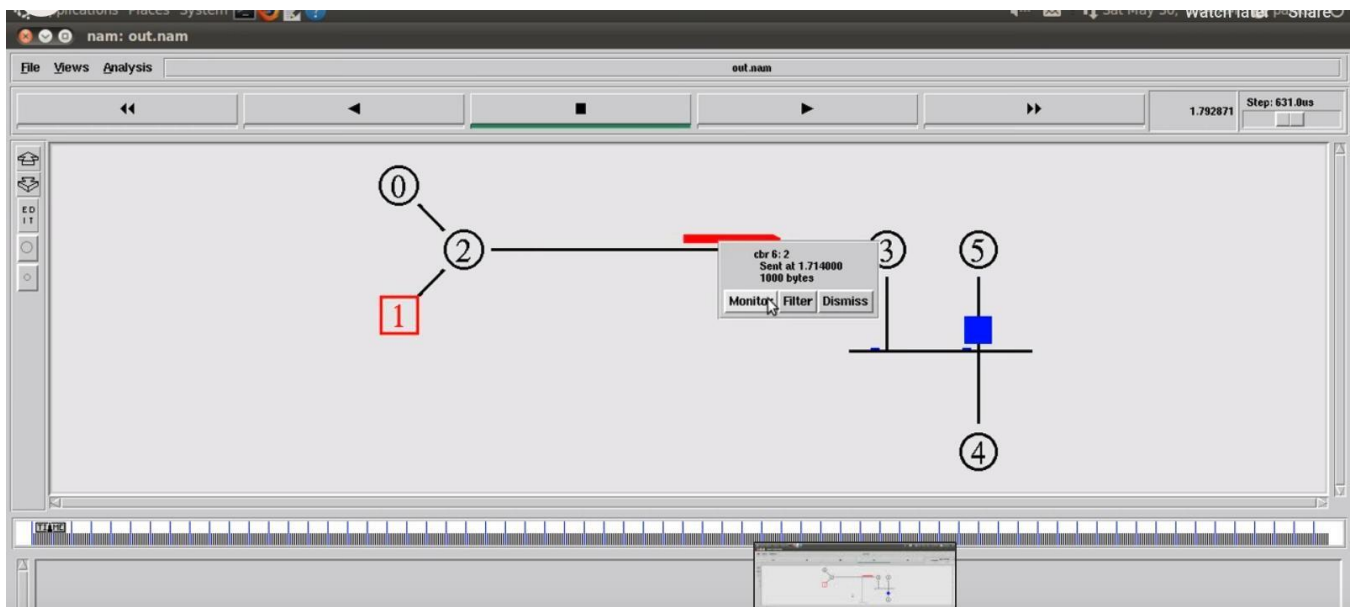
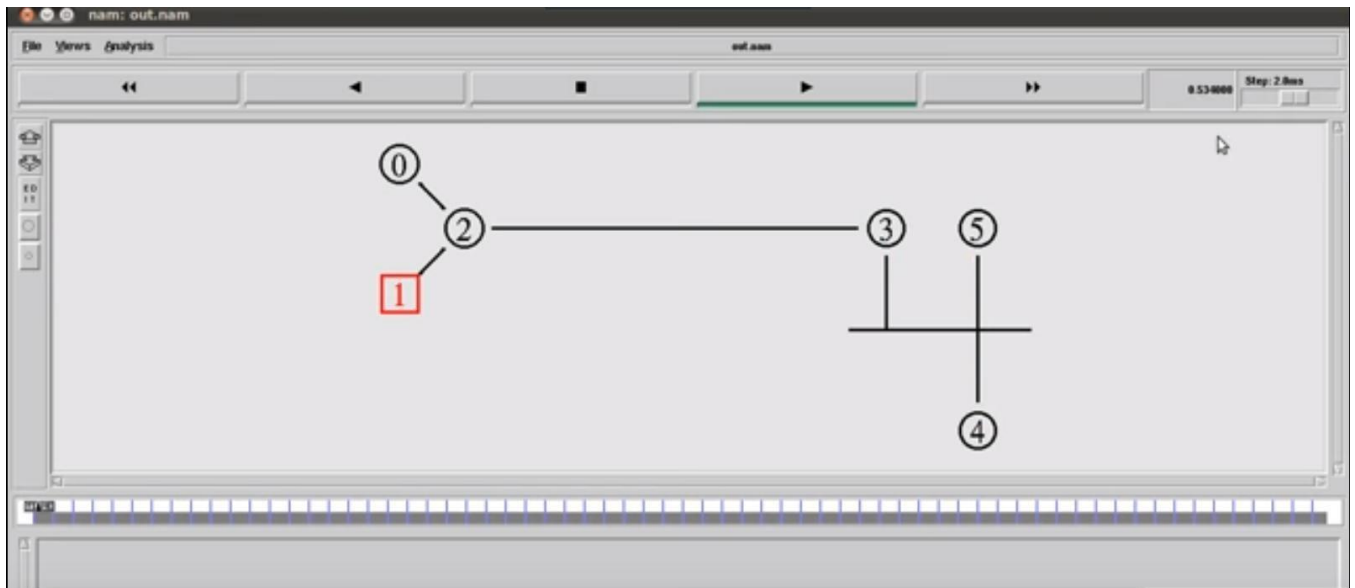
```
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set packet_size_ 552

#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
}

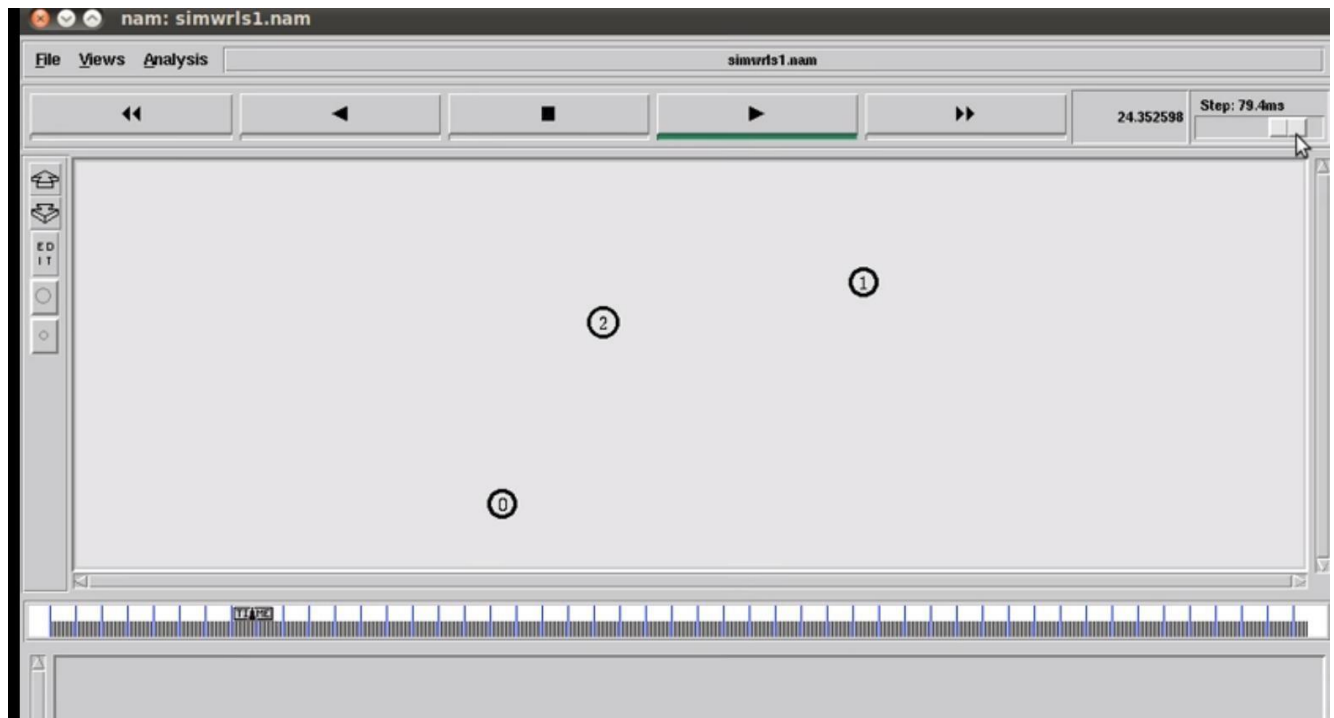
# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# 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



## Viva Questions

### Basics of network configuration files and networking commands in Linux

1. Explain about the network configuration file `/etc/hosts`?

Ans: The main purpose of this file is to resolve hostnames that cannot be resolved in any other way. This file should contain a line specifying the IP address of the loopback device (127.0.0.1) as `localhost.localdomain`.

2. Explain about the network configuration file `/etc/resolv.conf`?

Ans: This file specifies the IP addresses of the DNS servers.

3. Explain about the network configuration file `/etc/sysconfig/network`?

Ans: Specifies routing and host information for all network interfaces.

4. Explain `ifconfig` Command?

Ans: `ifconfig` is a command-line interface tool for network interface configuration and is also used to initialize interfaces at system boot time. Once a server is up and running, it can be used to assign an IP Address to an interface and enable or disable the interface on demand.

5. Explain Ping Command?

Ans: `ping` (Packet INternet Groper) is a utility normally used for testing connectivity between two systems on a network (Local Area Network (LAN) or Wide Area Network (WAN)). It uses ICMP (Internet Control Message Protocol) to communicate to nodes on a network.

6. Explain Netstat Command in Linux?

Ans: Netstat is a command-line tool used by system administrators to evaluate network configuration and activity. The term Netstat is results from network and statistics. It shows open ports on the host device and their corresponding addresses, the routing table, and masquerade connections.

### System calls used for network programming in Linux

1. Explain `fork()` system call in Linux?

Ans: A new process is created by the `fork()` system call.

A new process may be created with `fork()` without a new program being run-the new sub process simply continues to execute exactly the same program that the first (parent) process was running. It is one of the most widely used system calls under process management.

2. Explain `exec()` system call in Linux?

Ans: A new program will start executing after a call to `exec()`

Running a new program does not require that a new process be created first: any process may call `exec()` at any time. The currently running program is immediately terminated, and the new program starts executing in the context of the existing process.

3. Explain `exit()` system call in Linux?

Ans: The `exit()` system call is used by a program to terminate its execution.

The operating system reclaims resources that were used by the process after the `exit()` system call.

4. Explain open() system call in Linux?

Ans: It is the system call to open a file.

This system call just opens the file, to perform operations such as read and write, we need to execute different system call to perform the operations.

5. Explain the use of read() system call?

Ans: This system call opens the file in reading mode

We can not edit the files with this system call.

Multiple processes can execute the read() system call on the same file simultaneously.

6. Explain write() system call?

Ans: This system call opens the file in writing mode

We can edit the files with this system call.

Multiple processes can not execute the write() system call on the same file simultaneously.

7. Use of close() function?

Ans: This system call closes the opened file.

8. Illustrate socket() function with syntax?

Ans:

#### SYNOPSIS

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

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

#### DESCRIPTION

socket() creates an endpoint for communication and returns a 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>.

### **Implement client-server communication using socket programming and TCP as transport layer protocol**

1. What Is The Difference Between Tcp And Udp?

Ans: TCP and UDP are both transport-level protocols. TCP is designed to provide reliable communication across a variety of reliable and unreliable networks and internets.

UDP provides a connectionless service for application-level procedures. Thus, UDP is basically an unreliable service; delivery and duplicate protection are not guaranteed.

2. What Is Socket?

Ans: A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

3. Explain the functions of TCP server for socket communication?

Ans: using create(), Create TCP socket.

using bind(), Bind the socket to server address.

using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection

using accept(), At this point, connection is established between client and server, and they are ready to transfer data.

4. Explain the functions of TCP client for socket communication?

Ans: Create TCP socket.

connect newly created client socket to server

5. Describe bind() function?

Ans: Bind socket to IP address/port

int bind(int socket, struct sockaddr \*addr, int addr\_len)

6. Describe listen() function?

Ans: Mark the socket as accepting connections

int listen(int socket, int backlog)

7. Describe accept() function?

Ans: nt accept(int socket, struct sockaddr \*addr, int addr\_len) (returns a new socket to talk to the client)

8. Describe the syntax of connect() function?

Ans: nt connect(int socket, struct sockaddr \*addr, int addr\_len)

9. Explain Transmission Control Protocol, TCP.

Ans: TCP is a connection-oriented protocol. It simply means when data is transferring from source to destination, the protocol takes care of data integrity by sending the data packet again if it is lost during transmission. TCP ensures reliability and an error-free data stream. TCP packets contain fields such as Sequence Number, AcK number, Data offset, Reserved, Control bit, Window, Urgent Pointer, Options, Padding, checksum, Source Port, and Destination port.

9. What is the TCP packet format?

Ans: The TCP packet format consists of these fields:

Source Port and Destination Port fields (16 bits each); Sequence Number field (32 bits); Acknowledgement Number field (32 bits); Data Offset (a.k.a. Header Length) field (variable length); Reserved field (6 bits); Flags field (6 bits) contains the various flags: URG, ACK, PSH, RST, SYN, FIN; Window field (16 bits); Checksum field (16 bits) ; Urgent pointer field (16 bits) ; Options field (variable length) & Data field (variable length).

Source Port		Destination Port	
Sequence Number		Acknowledgment Number	
Data Offset	Reserved	Code	Window
Checksum		Urgent Pointer	
Options			Padding
Data			

## Implement client-server communication using socket programming and TCP as transport layer protocol

1. Explain the functions of UDP Server?

Ans: Create a UDP socket.

Bind the socket to the server address.

Wait until the datagram packet arrives from the client.

Process the datagram packet and send a reply to the client.

2. Explain the functions of UDP Client?

Ans: Create a UDP socket.

Send a message to the server.

Wait until response from the server is received.

Process reply and go back to step 2, if necessary.

Close socket descriptor and exit.

2. Explain the syntax of sendto function?

Ans: Send a message on the socket

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)
```

Arguments :

sockfd – File descriptor of the socket

buf – Application buffer containing the data to be sent

len – Size of buf application buffer

flags – Bitwise OR of flags to modify socket behavior

dest\_addr – Structure containing the address of the destination

addrlen – Size of dest\_addr structure

3. Explain the syntax of recvfrom function?

Ans: Receive a message from the socket.

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen)
```

**Arguments :**

sockfd – File descriptor of the socket

buf – Application buffer in which to receive data

len – Size of buf application buffer

flags – Bitwise OR of flags to modify socket behavior

src\_addr – Structure containing source address is returned

addrlen – Variable in which size of src\_addr structure is returned

**4. Explain User Datagram Protocol, UDP.**

Ans: UDP is a connection-less protocol. In simple terms, if one data packet is lost during transmission, it will not send that packet again.

This protocol is suitable where minor data loss is not a major issue.

**5. Is UDP better than TCP?**

Ans: Both protocols are used for different purposes. If the user wants error-free and guarantees to deliver data, TCP is the choice. If the user wants fast transmission of data and little loss of data is not a problem, UDP is the choice.

**6. What is the UDP packet format?**

Ans: The UDP packet format contains four fields:

Source Port and Destination Port fields (16 bits each): Endpoints of the connection.

Length field (16 bits): Length of the header and data.

Checksum field (16 bits): It allows packet integrity checking (optional).

<b>Source port</b>	<b>Destination Port</b>
<b>Length</b>	<b>CheckSum</b>

**Simulate sliding window flow control protocols****1. What is Stop-and-Wait Protocol?**

Ans: In Stop and wait protocol, sender sends one frame, waits until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.

**2. What is Stop-and-Wait Automatic Repeat Request?**

Ans: Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.

**3. What is usage of Sequence Number in Reliable Transmission?**

Ans: The protocol specifies that frames need to be numbered. This is done by using sequence numbers. A field is added to the data frame to hold the sequence number of that frame. Since we want to minimize the frame size, the smallest range that provides unambiguous communication. The sequence numbers can wrap around.

#### 4. What is Sliding Window?

Ans: The sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers.

#### 5. Difference Between Go-Back-N and Selective Repeat Protocol

S.NO	Go-Back-N Protocol	Selective Repeat Protocol
1.	In Go-Back-N Protocol, if the sent frame are found suspected then all the frames are re-transmitted from the lost packet to the last packet transmitted.	In selective Repeat protocol, only those frames are re-transmitted which are found suspected.
2.	Sender window size of Go-Back-N Protocol is N.	Sender window size of selective Repeat protocol is also N.
3.	Receiver window size of Go-Back-N Protocol is 1.	Receiver window size of selective Repeat protocol is N.
4.	Go-Back-N Protocol is less complex.	Selective Repeat protocol is more complex.
5.	In Go-Back-N Protocol, neither sender nor at receiver need sorting.	In selective Repeat protocol, receiver side needs sorting to sort the frames.
6.	In Go-Back-N Protocol, type of Acknowledgement is cumulative.	In selective Repeat protocol, type of Acknowledgement is individual.
7.	In Go-Back-N Protocol, Out-of-Order packets are NOT Accepted (discarded) and the entire window is re-transmitted.	In selective Repeat protocol, Out-of-Order packets are Accepted.
8.	In Go-Back-N Protocol, if Receiver receives a corrupt packet, then also, the entire window is re-transmitted.	In selective Repeat protocol, if Receiver receives a corrupt packet, it immediately sends a negative acknowledgement and hence only the selective packet is retransmitted.
9.	Efficiency of Go-Back-N Protocol is $N / (1 + 2 * a)$	Efficiency of selective Repeat protocol is also $N / (1 + 2 * a)$

Ans:

#### 6. Explain the functions of server in Selective Repeat?

Ans:

- Establish connection (recommended UDP)
- Accept the window size from the client (should be  $\leq 40$ )
- Accept the packets from the network layer.
- Calculate the total frames/windows required.
- Send the details to the client (total packets, total frames.)
- Initialise the transmit buffer.
- Build the frame/window depending on the window size.
- Transmit the frame.
- Wait for the acknowledgement frame.
- Check for the acknowledgement of each packet and repeat the process for the packet for which the negative acknowledgement is received. Else continue as usual.
- Increment the frame count and repeat steps until all packets are transmitted.
- Close the connection

7. Explain the functions of client in Selective Repeat?

Ans:

- Establish a connection.(recommended UDP)
- Send the window size on server request.
- Accept the details from the server(totalpackets,totalframes).
- Initialise the receive buffer with the expected packets.
- Accept the frame/window from the server.
- Check for validity of the packets and construct the acknowledgement frame depending on the validity.(Here the acknowledgement is accepted from the users)
- Depending on the acknowledgement frame readjust the process.
- Increment the frame count and repeat steps until all packets are received.
- Close the connection.

8. Explain the functions of server in Go Back N?

Ans:

- Establish connection (recommended UDP)
- Accept the window size from the client(should be  $\leq 40$ )
- Accept the packets from the network layer.
- Calculate the total frames/windows required.
- Send the details to the client(totalpackets,totalframes.)
- Initialise the transmit buffer.
- Build the frame/window depending on the window size.
- Transmit the frame.
- Wait for the acknowledgement frame.
- Check for the acknowledgement of each packet and repeat the process from the packet for which the first negative acknowledgement is received.Else continue as usual.
- Increment the frame count and repeat steps until all packets are transmitted.
- Close the connection.

9. Explain the functions of client in Go Back N?

Ans:

- Establish a connection.(recommended UDP)
- Send the window size on server request.
- Accept the details from the server(totalpackets,totalframes).
- Initialise the receive buffer with the expected packets.
- Accept the frame/window from the server.
- Check for validity of the packets and construct the acknowledgement frame depending on the validity.(Here the acknowledgement is accepted from the users)
- Depending on the acknowledgement frame readjust the process.
- Increment the frame count and repeat steps 5-9 until all packets are received.
- Close the connection.

**Implement and simulate algorithm for Distance Vector Routing protocol or Link State Routing protocol**

1. What Is Routing?

Ans: The ROUTING TABLE is a table maintained in the kernel that determines how packets are routed to other systems. A number of programs may add or delete routes from the routing tables, including route, ifconfig, in.routed and in.rdisc.



The routing table consists of three types of routes: HOST ROUTES are checked first and define a route to just one host. NETWORK ROUTES are checked second and define a route to all the hosts on one network. DEFAULT ROUTES are used as a catch-all, when no host or network routes are found to a destination. They usually send to a more knowledgeable routing machine, which has a better chance of being able to find a host or network route to the destination.

## 2. Difference between Distance vector routing and Link State routing?

Ans:

Distance Vector Routing –

It is a dynamic routing algorithm in which each router computes a distance between itself and each possible destination i.e. its immediate neighbors.

The router shares its knowledge about the whole network to its neighbors and accordingly updates the table based on its neighbors.

The sharing of information with the neighbors takes place at regular intervals.

It makes use of Bellman-Ford Algorithm for making routing tables.

Problems – Count to infinity problem which can be solved by splitting horizon.

– Good news spread fast and bad news spread slowly.

– Persistent looping problem i.e. loop will be there forever

Link State Routing –

It is a dynamic routing algorithm in which each router shares knowledge of its neighbors with every other router in the network.

A router sends its information about its neighbors only to all the routers through flooding.

Information sharing takes place only whenever there is a change.

It makes use of Dijkstra's Algorithm for making routing tables.

Problems – Heavy traffic due to flooding of packets.

– Flooding can result in infinite looping which can be solved by using the Time to live (TTL) field

## 3. Explain the routing information in node x by using the Distance Vector Routing algorithm?

Ans:

Let  $dx(y)$  be the cost of the least-cost path from node x to node y. The least costs are related by Bellman-Ford equation:

$$dx(y) = \min_v \{c(x,v) + dv(y)\}$$

Where the  $\min_v$  is the equation taken for all x neighbors. After traveling from x to v, if we consider the least-cost path from v to y, the path cost will be  $c(x,v)+dv(y)$ . The least cost from x to y is the minimum of  $c(x,v)+dv(y)$  taken over all neighbors.

With the Distance Vector Routing algorithm, the node x contains the following routing information:

For each neighbor v, the cost  $c(x,v)$  is the path cost from x to directly attached neighbor, v.

The distance vector x, i.e.,  $Dx = [ Dx(y) : y \text{ in } N ]$ , containing its cost to all destinations, y, in N.

The distance vector of each of its neighbors, i.e.,  $Dv = [ Dv(y) : y \text{ in } N ]$  for each neighbor v of x.

Distance vector routing is an asynchronous algorithm in which node x sends the copy of its distance vector to all its neighbors. When node x receives the new distance vector from one of its neighboring vector, v, it saves the distance vector of v and uses the Bellman-Ford equation to update its own distance vector.

The equation is given below:

$$dx(y) = \min_v \{ c(x,v) + dv(y) \} \quad \text{for each node } y \text{ in } N$$

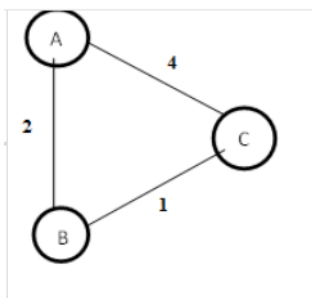
The node x has updated its own distance vector table by using the above equation and sends its updated table to all its neighbors so that they can update their own distance vectors.

4. Explain Distance vector routing algorithm with suitable example?

Ans:

In the network shown below, there are three routers, A, B, and C, with the following weights – AB =2, BC =3 and CA =5.

**Step 1** – In this DVR network, each router shares its routing table with every neighbor. For example, A will share its routing table with neighbors B and C and neighbors B and C will share their routing table with A.



Form A	A	B	C
A	0	2	3
B			
C			

Form B	A	B	C
A			
B	2	0	1
C			

Form C	A	B	C
A			
B			
C	3	1	0

**Step 2** – If the path via a neighbor has a lower cost, then the router updates its local table to forward packets to the neighbor. In this table, the router updates the lower cost for A and C by updating the new weight from 4 to 3 in router A and from 4 to 3 in router C.

Form A	A	B	C
A	0	2	3
B			
C			

Form B	A	B	C
A			
B	2	0	1
C			

Form C	A	B	C
A			
B			
C	3	1	0

**Step 3** – The final updated routing table with lower cost distance vector routing protocol for all routers A, B, and C is given below

#### Router A

Form A	A	B	C
A	0	2	3
B	2	0	1
C	3	1	0

#### Router B

Form B	A	B	C
A	0	2	3
B	2	0	1
C	3	1	0

#### Router C

Form C	A	B	C
A	0	2	3
B	2	0	1
C	3	1	0

## **Implement Simple Mail Transfer Protocol**

### **1. What Is Smtplib?**

Ans: Logging into your email account to retrieve email seems fairly simple. However, there is technology working behind the scenes, such as simple mail transfer protocol (SMTP) and internet networks to ensure you can send and receive email. Think about SMTP as a mail carrier, sorting and delivering messages to mail servers. These messages are then delivered directly to your email inbox.

### **2. Explain How Does Smtplib Work?**

Ans: SMTP is responsible for transmitting email across Internet networks (IPs). This technology is used specifically for sending outgoing email. Clients typically use applications such as Internet Message Access Protocol (IMAP) or Post Office Protocol (POP) to access to their email box. For example, if you send an email it goes to a mail server using SMTP. The mail client will then deliver it to the user's mailbox.

### **3. Explain The Smtplib Basic Functions?**

Ans:SMTP is responsible for a little more than just deliver messages to servers. It performs several functions that streamlines the delivery process.

It evaluates the configuration and grants permission to the computer who is trying to send a message. It can also track if the message was sent successfully. If it isn't, an error message is sent to the sender. Extended SMTP is a little more advanced than older versions. It helps cut back on email spam.

### **4. Explain Resolved Limiting Factors Of Smtplib?**

Ans: Previous versions of this technology had limiting figures, such as the location of the client within the network. SMTP allows clients to submit emails quickly regardless of the recipients location. That's because current SMTP technology uses a client's authentication (which are their accreditations) instead of the client's IP address to send email.

### **5.What Do You Know About Other Protocols?**

Ans : In order for computers to communicate with one another, standard methods of information transfer and processing have been devised. These are referred to as "protocols" and some of the more common ones such as TCP, IP, UDP, POP, SMTP, HTTP, and FTP

## **Implement File Transfer Protocol**

### **1.What Is Ftp?**

Answer : FTP stands for File Transfer Protocol. An FTP server allows clients to connect to it either anonymously or with a username and password combination. After successful authentication, files can be transferred back and forth between the server and client. The files are neither encrypted nor compressed.

### **2. How To Deny Specific Users Access To The Ftp Server ?**

Answer : To deny specific users access to the FTP server, add their usernames to the /etc/vsftpd/ ftpusers file. By default, system users such as root and nobody are included in this list.

### 3. Can We Create Logs For Ftp Authenticated Sessions?

Answer :Yes, If the xferlog\_enable directive in vsftpd.conf is set to YES, file transfers using the FTP protocol are logged to /var/log/xferlog. Information such as a time stamp, IP address of the client, the file being transferred, and the username of the person who authenticated the connection is included in the log entry.

### 4. What Is Meaning Of Max\_clients Parameter?

Answer : Maximum number of clients that can connect at one time. If set to 0, the number of clients is unlimited.

### 5. Explain Ftp Spoofing Attack?

Answer :In the context of network security, a spoofing attack is a situation in which one person or program successfully masquerades as another by falsifying data and thereby gaining an illegitimate advantage.

### 6. Explain Ftp Bounce Attack?

Answer : FTP bounce attack is an exploit of the FTP protocol whereby an attacker is able to use the PORT command to request access to ports indirectly through the use of the victim machine as a middle man for the request. This technique can be used to port scan hosts discreetly, and to access specific ports that the attacker cannot access through a direct connection. nmap is a port scanner that can utilize an FTP bounce attack to scan other servers.

### 7. FTP works on which layer of the OSI model?

Answer: FTP is an application layer protocol.

### 8. What are the components of the FTP Client?

Answer:Control process, Data transfer process, User Interface

## **Implement congestion control using a leaky bucket algorithm**

### 1. Two broad categories of congestion control are

Ans: Open loop congestion control techniques are used to prevent congestion before it even happens by enforcing certain policies. Closed loop congestion control techniques are used to treat congestion after it has happened.

### 2. In open-loop control, policies are applied to \_\_\_\_\_

Ans: Open loop congestion control techniques are used to prevent congestion before it even happens by enforcing certain policies. Retransmission policy, window policy and acknowledgement policy are some policies that might be enforced.

### 3. Retransmission of packets must not be done when \_\_\_\_\_

Ans: Retransmission refers to the sender having to resend the packet to the receiver. It needs to be done only when some anomaly occurs with the packet like when the packet is lost or corrupted.

### 4. Discarding policy is mainly done by \_\_\_\_\_

Ans: The discarding policy adopted by the routers mainly states that the routers discard sensitive or corrupted packets that it receives, thus controlling the integrity of the packet flow. The discarding policy is adopted as an open loop congestion control technique.

5. Closed-Loop control mechanisms try to \_\_\_\_\_

Ans: In closed loop congestion control, methods are implemented to remove congestion after it occurs. Some of the methods used are backpressure and choke packet.

6. In the congestion avoidance algorithm, the size of the congestion window increases \_\_\_\_\_ until congestion is detected.

Ans: In the congestion avoidance algorithm, the size of the congestion window increases additively until congestion is detected. Once congestion is detected, the size of congestion window is decreased once and then the packets are transmitted to achieve congestion avoidance.

7.Explain congestion?

Ans: A state occurring in the network layer when the message traffic is so heavy that it slows down network response time is known as congestion.

8.What is retransmission?

Ans: The TCP retransmission means resending the packets over the network that have been either lost or damaged. Here, retransmission is a mechanism used by protocols such as TCP to provide reliable communication. Here, reliable communication means that the protocol guarantees the packet's delivery even if the data packet has been lost or damaged. The networks are unreliable and do not guarantee the delay or the retransmission of the lost or damaged packets. The network, which uses a combination of acknowledgment and retransmission of damaged or lost packets, offers reliability

9. Difference between Leaky bucket and Token bucket algorithm?

Leaky Bucket	Token Bucket
When the host has to send a packet, the packet is thrown in a bucket.	This leaky bucket holds tokens generated at regular intervals of time.
Bucket leaks at a constant rate	The bucket has a maximum capacity.
Bursty traffic is converted into uniform traffic by leaky buckets.	Bucket there is a ready packet, a token is removed from the bucket and the packet is sent.
In practice, a bucket is a finite queue output at a finite rate.	If there is no token in the bucket, the packet can not be sent.

Ans:

10..Write advantages of the token bucket over the Leaky bucket algorithm?

Ans: If a bucket is full in tokens, tokens are discarded, not packets. While in a leaky bucket, packets are discarded.

Token Bucket can send large bursts at a faster rate, while buckets always send packets at a constant rate.

10. What is the difference between connection-oriented service and connection-less?

Ans:

S.NO	Connection-oriented Service	Connection-less Service
1.	Connection-oriented service is related to the telephone system.	Connection-less service is related to the postal system.
2.	Connection-oriented service is preferred by long and steady communication.	Connection-less Service is preferred by bursty communication.
3.	Connection-oriented Service is necessary.	Connection-less Service is not compulsory.
4.	Connection-oriented Service is feasible.	Connection-less Service is not feasible.
5.	In connection-oriented Service, Congestion is not possible.	In connection-less Service, Congestion is possible.
6.	Connection-oriented Service gives the guarantee of reliability.	Connection-less Service does not give the guarantee of reliability.
7.	In connection-oriented Service, Packets follow the same route.	In connection-less Service, Packets do not follow the same route.
8.	Connection-oriented services require a bandwidth of a high range.	Connection-less Service requires a bandwidth of low range

11. What are open loop congestion control and closed-loop congestion control techniques?

Ans: Open-loop congestion control policies are applied to prevent congestion before it happens. The congestion control is handled either by the source or the destination. Closed-loop congestion control technique is used to treat or alleviate congestion after it happens

## Understanding the Wireshark tool

1. What is Wireshark?

Ans: Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

2. Give some examples people uses Wireshark for?

Ans: □

- network administrators use it to troubleshoot network problems
- network security engineers use it to examine security problems
- developers use it to debug protocol implementations
- people use it to learn network protocol internals

3. Explain some of the features Wireshark provides?

Ans:

- □ Available for UNIX and Windows.
- □ Capture live packet data from a network interface.
- □ Display packets with very detailed protocol information.
- □ Open and Save packet data captured.
- □ Import and Export packet data from and to a lot of other capture programs.
- □ Filter packets on many criteria.
- □ Search for packets on many criteria.
- □ Colorize packet display based on filters.
- □ Create various statistics.

4. Explain Endpoints in Wireshark?

Ans:

A network endpoint is the logical endpoint of separate protocol traffic of a specific protocol layer. The endpoint statistics of Wireshark will take the following endpoints into account:

- Ethernet: an Ethernet endpoint is identical to the Ethernet's MAC address.
- Fibre Channel: XXX - insert info here.
- FDDI: a FDDI endpoint is identical to the FDDI MAC address.

- IPv4: an IP endpoint is identical to its IP address.
- IPX: an IPX endpoint is concatenation of a 32 bit network number and 48 bit node address, be default the Ethernets' MAC address.
- JXTA: a JXTA endpoint is a 160 bit SHA-1 URN.
- NCP: XXX - insert info here.
- RSVP: XXX - insert info here.
- SCTP: a SCTP endpoint is a combination of the host IP addresses (plural) and the SCTP port used. So different SCTP ports on the same IP address are different SCTP endpoints, but the same SCTP port on different IP addresses of the same host are still the same endpoint.
- TCP: a TCP endpoint is a combination of the IP address and the TCP port used, so different TCP ports on the same IP address are different TCP endpoints.
- Token Ring: a Token Ring endpoint is identical to the Token Ring MAC address.
- UDP: a UDP endpoint is a combination of the IP address and the UDP port used, so different UDP ports on the same IP address are different UDP endpoints.
- USB: XXX - insert info here.
- WLAN: XXX - insert info here.

5. A user is unable to ping a system on the network. How can wireshark be used to solve the problem.

Ans: Ping uses ICMP. Wireshark can be used to check if ICMP packets are being sent out from the system. If it is sent out, it can also be checked if the packets are being received.

6. How do I modify Wireshark packets on the fly?

Ans: Wireshark is a packet capture & inspection program, not a packet injection program. What you're asking about appears to be a man-in-the-middle attack so I'm not going to assist you in that.

### **Design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure commonly used services in the network**

1.What is a network?

Ans. A network consists of two or more separate devices linked together such that they can communicate. Networks can be classified according to different criteria such as scope, type of connection, functional relationship, topology, or function, among others.

2.What are the different types of networks?

Ans. Considering the size or span of a network, we can classify them as follows:

PAN (Personal Area Network) – PAN is made up of devices used by a single person. It has a range of a few meters.

WPAN (Wireless Personal Area Network) – It is a PAN network that uses wireless technologies as a medium.

LAN (Local Area Network) – LAN is a network whose range is limited to a relatively small area, such as a room, a building, an aeroplane, etc.

WLAN (Wireless Local Area Network) – WLAN is a LAN network that uses wireless means of communication. It is a widely used configuration due to its scalability and because it does not require the installation of cables.

CAN (Campus Area Network) – A network of high-speed devices that connects LANs in a limited geographical area, such as a university campus, a military base, etc.



MAN (Metropolitan Area Network) or metropolitan area network – It is a high-speed (broadband) network providing coverage in a larger geographic area than a campus, but still limited.

WAN (Wide Area Network) – WAN extends over a large geographical area using unusual means of communication, such as satellites, interoceanic cables, fibre optics, etc. Use public media.

VLAN – It is a type of logical or virtual LAN, mounted on a physical network, in order to increase security and performance. In special cases, thanks to the 802.11Q protocol (also called QinQ), it is possible to mount virtual networks on WAN networks. It is important not to confuse this implementation with VPN technology.

### 3.What is a ‘subnet’?

Ans. A ‘subnet’ is a generic term for a section of an extensive network, usually separated by a bridge or a router. It also works for the network’s broadcast domains, manages traffic flow, and helps increasing network performance. Uses of the subnet in networking include:

Relieving network congestion

Reallocating IP addresses

Improving network security

### 4.What is DNS?

Ans. The Domain Name System (DNS) is a central part of the internet, providing a way to match names (a website you’re seeking) to numbers (the address for the website). Anything connected to the internet – laptops, tablets, mobile phones, and websites – has an Internet Protocol (IP) address made up of numbers.

### 5.What is Network Topology?

Ans. This is among the important networking interview questions. Network topology is the physical or logical arrangement in which the devices or nodes of a network (e.g. computers, printers, servers, hubs, switches, routers, etc.) are interconnected with each other over a communication medium. It consists of two parts – the physical topology, which is the actual arrangement of the cables (the media), and the logical topology, which defines how the hosts access the media.

Types of network topologies –

Bus – In the bus network topology, each workstation is connected to a main cable called a bus. Therefore, in effect, each workstation is directly connected to every other workstation on the network.

Star – In the star network topology, there is a central computer or server to which all workstations are directly connected. Each workstation is indirectly connected to the other through the central computer.

Ring – In the ring network topology, the workstations are connected in a closed-loop configuration. Adjacent workstation pairs are directly connected. Other pairs of workstations are indirectly connected, passing data through one or more intermediate nodes.

Mesh – Mesh network topology has two forms – full and partial mesh. In the full mesh topology, each workstation is directly connected. In the partial mesh topology, some workstations are connected to all the others, and some are connected only to the other nodes with which they exchange more data.

Tree – The tree network topology uses two or more star networks connected to each other. The central computers in star networks are connected to the main bus. Thus, a tree network is a bus network of star networks.

6. What is an IP address?

Ans. An Internet Protocol address (IP address) is a numerical unique address of a device in a network. IP is a datagram-oriented connectionless protocol, therefore each packet must contain a header with the source IP address, the destination IP address, and other data in order to be delivered successfully.

There are two types of IPs –

**Private IP Address** – A private IP address is a set of numbers that are assigned to each computer or system, connected to a private network. An example of a private IP address is your mobile phone or your home router which have a default local address.

**Public IP Address** – Public IP addresses are global addresses, visible to anyone browsing the Internet. A user just needs an internet connection to connect to such devices.

7. How to find the IP address of a website?

Ans. Finding the IP address of a website or a domain is not a tricky task and involves the below steps –

Press the “Start” button on your computer

Type in the program and file browser “cmd”

Hit “Enter”

The MS-DOS console will open, where you must type “nslookup google.com”. Instead of “google.com”, you must write the domain name of the page you want to consult

Next, you will be able to see the IP address

8. What are routers?

Ans. Routers connect two or more network segments. These intelligent network devices store information in their routing tables such as paths, hops, and bottlenecks. They determine the most accurate data transfer paths and operate in Open Systems Interconnection (OSI) Network Layer.

The roles of a router include:

Inter-network communication

Best path selection

Packet forwarding

Packet filtering

9. What are Nodes and Links?

Ans. Nodes – Devices or data points on a more extensive network are known as nodes. They are individual parts of a larger data structure and contain data. They also link other nodes.

**Links**- A link is the physical and logical network component for interconnecting hosts or nodes in a network. It is a physical communication medium such as a coaxial cable or optical fibre.

## Study of NS2 simulator

### 1. What is NS2?

Ans: NS2 stands for Network Simulator version 2. It is a discrete event simulator for networking research that works at packet level. It provides substantial support to simulate a bunch of protocols like TCP, UDP, FTP, HTTP and DSR. NS2 simulates wired and wireless networks. It is primarily Unix based and uses TCL as its scripting language. ns2 is a standard experiment environment in the research community. Network simulator 2 (ns2) is a popular open source discrete event simulator for computer networks. It is a package of tools that simulates the behavior of networks.

- ☐ Create Network Topologies
- ☐ Log events that happen under any load
- ☐ Analyze events to understand the network behavior

### 2. What are the components of NS2?

Ans: Four major classes of ns2 components are:

- 1) Network objects responsible for sending, creating, receiving and destroying packet-related objects.
- 2) Packet-related objects are those which pass around a network.
- 3) Simulation-related objects control simulation timing and supervise the entire simulation. Simulation-related objects are events, handlers, the Scheduler and the Simulator.
- 4) Helper objects do not explicitly participate in packet forwarding. They implicitly help to complete the simulation. For example, a routing module calculates the routes from a source to destination, while network address identifies each of the network objects.

### 3. What Languages are used with NS-2?

Ans: ☐ Split-Language programming is used

1. Scripting Language (Tcl - Tool Command Language and pronounced 'tickle')
  2. System Programming Language (C/C++)
- ☐ ns is a Tcl interpreter to run Tcl Scripts
  - ☐ By using C++/OTcl, the network simulator is completely Object-oriented.

### 4. What is Tcl?

Ans: TCL is the language used to provide a linkage between C++ and OTcl. Toolkit Command Language (Tcl/OTcl) scripts are written to set up/configure network topologies. Tcl provides linkage for class hierarchy, object instantiation, variable binding and command dispatching. OTcl is used for periodic or triggered events. The following is written and compiled with C++.

- ☐ Event Scheduler
- ☐ Basic network component objects

### 5. Explain event scheduling in NS2?

Ans: Event scheduling

**Create scheduler** -> set ns [new Simulator]

**Schedule Event** -> \$ns at <time> <event>

**Start scheduler** -> \$ns run

6. Explain node creation in NS2?

Ans: A unidirectional link between the two nodes is created as follows

A Simplex link (one way) -> \$ns simplex-link \$n0 \$n1 <bandwidth>  
<delay> <queue\_type>

A bi-directional link between the two nodes is created as follows

A duplex link (both ways) -> \$ns duplex-link \$n0 \$n1  
<bandwidth> <delay> <queue\_type>

7. What is trace file in NS2?

Ans: A Trace file contains all information needed for animation purposes- both on a static network layout on dynamic events such as packet arrivals, departures, drops and link failures.