Networks Laboratory

Subject Code: 10CSL77	**I.A. Marks : 25**
Hours/Week : 03	**Exam Hours: 03**
Total Hours : 42	**Exam Marks: 50**

Note: Student is required to solve one problem from PART-A and one problem from PART-B. The questions are allotted based on lots. Both questions carry equal marks.

PART A – Simulation Exercises

The following experiments shall be conducted using either NS228/OPNET or any other suitable simulator.

1. Simulate a three nodes point – to – point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets dropped.
2. Simulate a four node point-to-point network with the links connected as follows:
   n0 – n2, n1 – n2 and n2 – n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP / UDP.
3. Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
4. Simulate an Ethernet LAN using n nodes (6-10), change error rate and data rate and compare throughput.
5. Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
6. Simulate simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.

PART-B

Implement the following in C/C++:

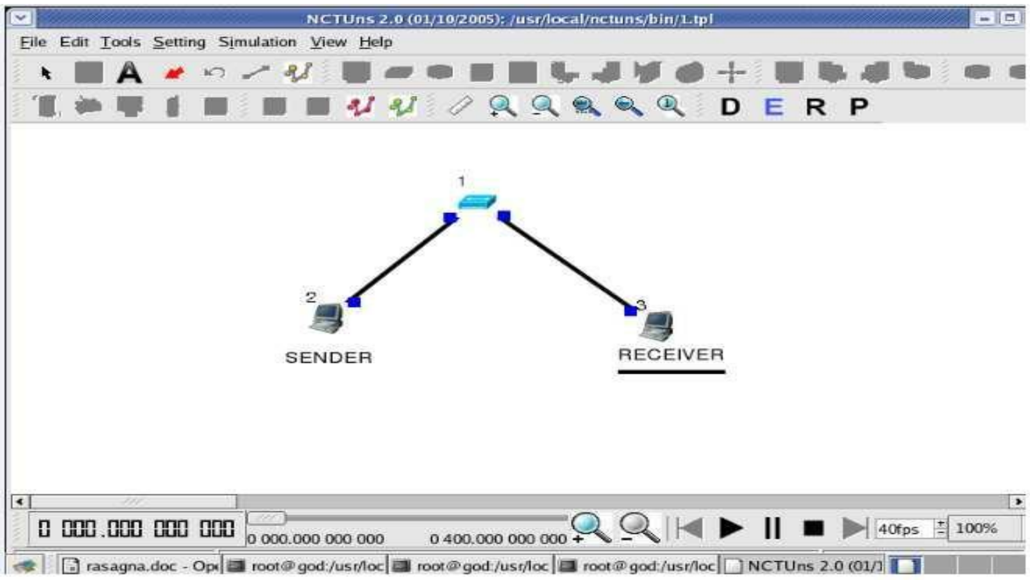
7. Write a program for error detecting code using CRC-CCITT (16- bits).
8. Write a program for distance vector algorithm to find suitable path for transmission.
9. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.
10. Implement the above program using as message queues or FIFOs as IPC channels.
11. Write a program for simple RSA algorithm to encrypt and decrypt the data.
12. Write a program for congestion control using leaky bucket algorithm.

Note:
In the examination, a combination of one problem has to be asked from Part A for a total of 25 marks and one problem from Part B has to be asked for a total of 25 marks. The choice must be based on random selection from the entire lots.

1. Simulate a three nodes point – to – point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

Design Mode:
a) Take a hub and two hosts.
b) Make a Point to Point connection between host1 and hub and also between host2 and hub.
c) Save it.

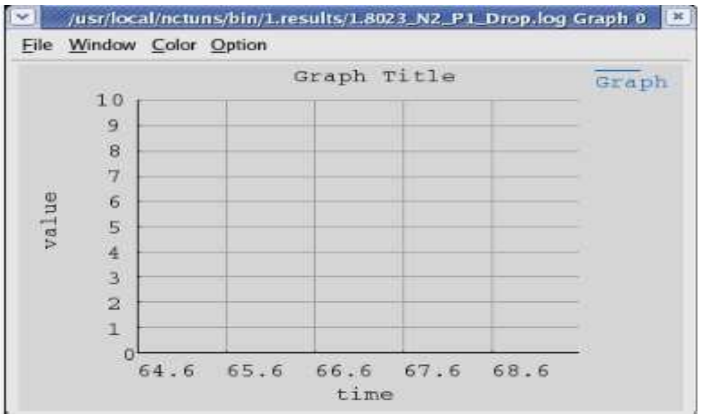
Edit Mode (E):
a) Double click on host 1. A new window will be opened click on add and type the following command:
stg –u 1024 200 (1.0.1.2) Receiver IP address should be given in command.
b) Click on Node editor and select Out-throughput, drop log.
c) Now double click on host 2, add the following command
rtg –u –w log1
d) Click on Node editor and click MAC 8023. Select In-throughput and drop log.



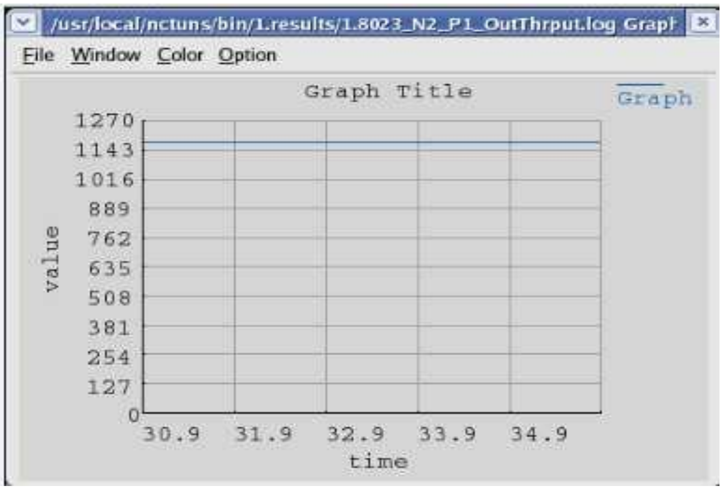
Run Mode (R)
Goto Simulation Tab>run

Play Mode (P):
a) Click on play button in the bar at the bottom.
b) Goto Tools>Plot Graph , and start plotting for graphs for in, out and drop logs. (Note: Logs are present in your filename.results file)

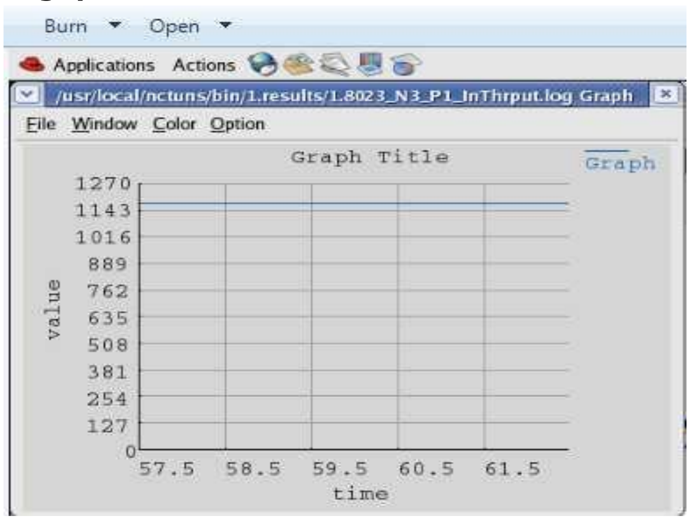
Drop Graph:



Out Throughput of sender:



In Throughput of receiver:



Go Back to edit mode:

a) Double click on Point to point link and change bandwidth to 5(which is by default 10 in previous case)
b) Go to Run Mode.
c) Goto Simulation menu>Run
c) Now play and plot the graph to see the drop, in and out throughput.

INFERENCE:

Bandwidth: 10
Out Throughput: 1174 to 1180 bits/s.
In Throughput: 1174 to 1180 bits/s.
Drop: 0 to 0.

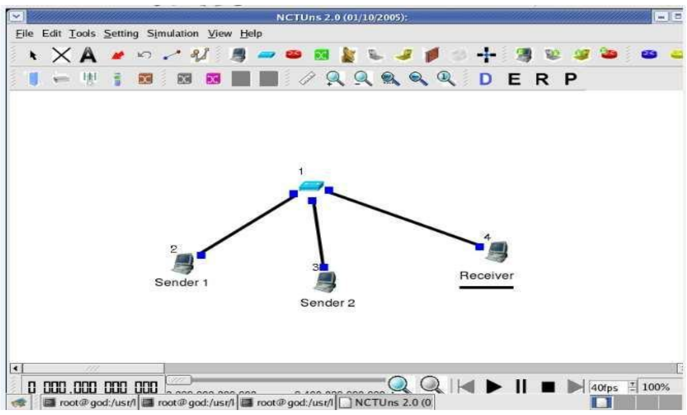
Bandwidth(at sender link): 8
Out Throughput: 942 to 945 bits/s;
In Throughput: 942 to 945 bits/s;
Drop: 0 to 0 bits/s.

Bandwidth(at receiver link) : 9
Out Throughput: 1177 to 1178 bits/s;
In Throughput: 0 to 0 bits/s;
Drop: 1101 to 1103 bits/s.

(NOTE: To View the Min- Max values of each parameter go down to the results folder i.e go to Computer on desktop, Click on File System, usr>local>nctuns>bin. In Bin folder you can find your simulations result folder. Open the folder to view the parameters which you have set. When you open each parameter file, you can view two columns one is time and other is number of bits. Note the least value and highest value)

2. Simulate a 4 node point-to-point network with the links connected as follows: n0 – n2, n1 – n2 and n2 – n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP / UDP.

DESIGN MODE



EDIT MODE:
Sender:-
stcp –p 3000 –l 1024 1.0.1.3
stg –u 1024 1.0.1.3
Receiver:-
rtcp –p 3000 –l 1024
rtg –u –w log1

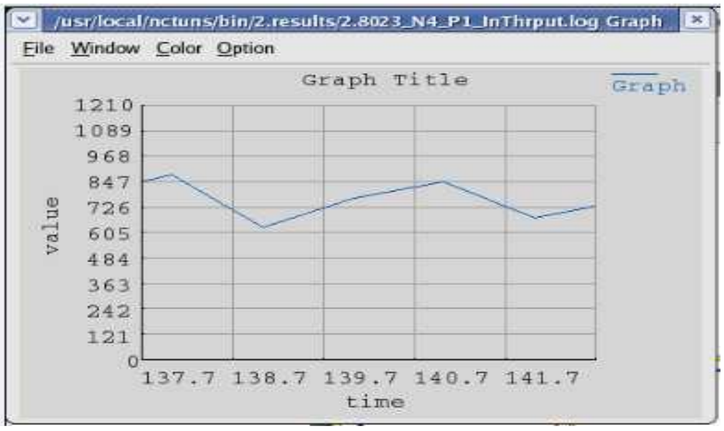
(Note: The receiver IP should be given. To know IP of any host place turtle on blue point on POINT to Point Link near HOST)

Click on Node editor (of respective hosts) click MAC 8023 and set the parameter Out throughputs for Sender 1 and Sender 2. Select In Throughput for Receiver.

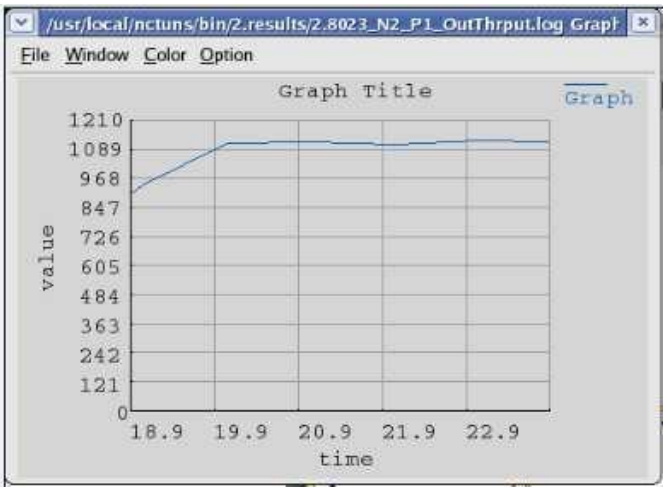
RUN MODE:
Goto Simulation>Run.

PLAY MODE:
Play the model and plot the graphs for IN Throughput of receiver and Out Throughputs of senders.

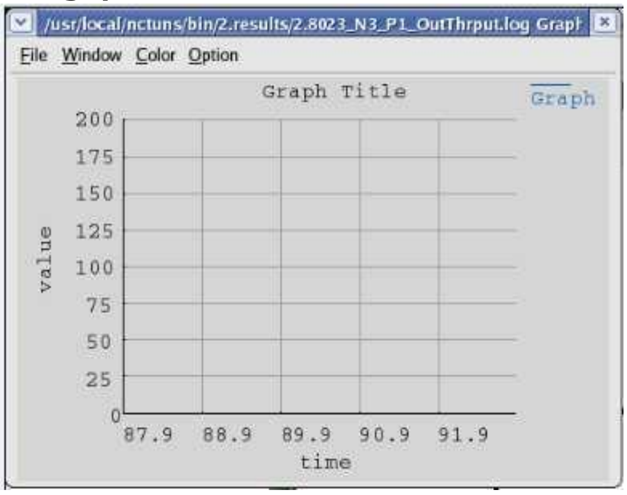
IN Throughput:



OUT Throughput at SENDER 1



OUT Throughput at SENDER 2

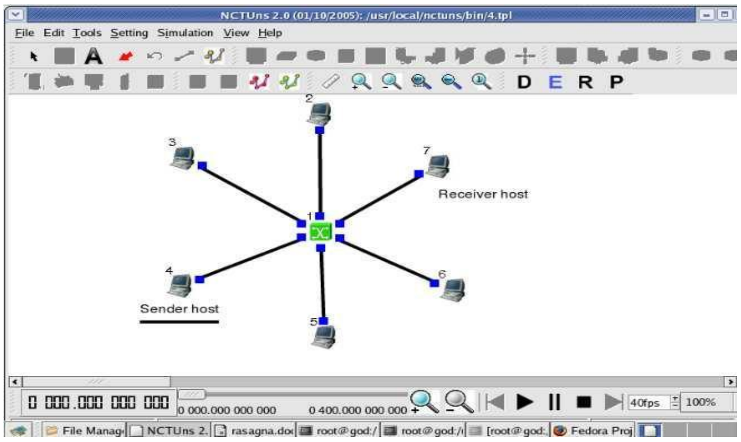


(to know why nothing is seen in graph see inference for sender 2 below)
INFERENCE:
Sender 1: OUT Throughput: 1186 to 1191 bits/s.
Sender 2: OUT Throughput: 0 to 0 bits/s.
Receiver: IN Throughput: 1186 to 1191 bits/s.

3. Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

DESIGN MODE:

SELECT Subnet symbol (plus symbol). Click on window. Enter hosts: 6 and radius as 125.


NCTUns 2.0 (01/10/2005): /usr/local/nctuns/bin/4.tpl
File Edit Tools Setting Simulation View Help
D E R P
2
3
7
Receiver host
1
4
Sender host
6
5
40fps
100%

EDIT MODE

Sender: stcp –p 2000 –l 1024 1.0.1.6

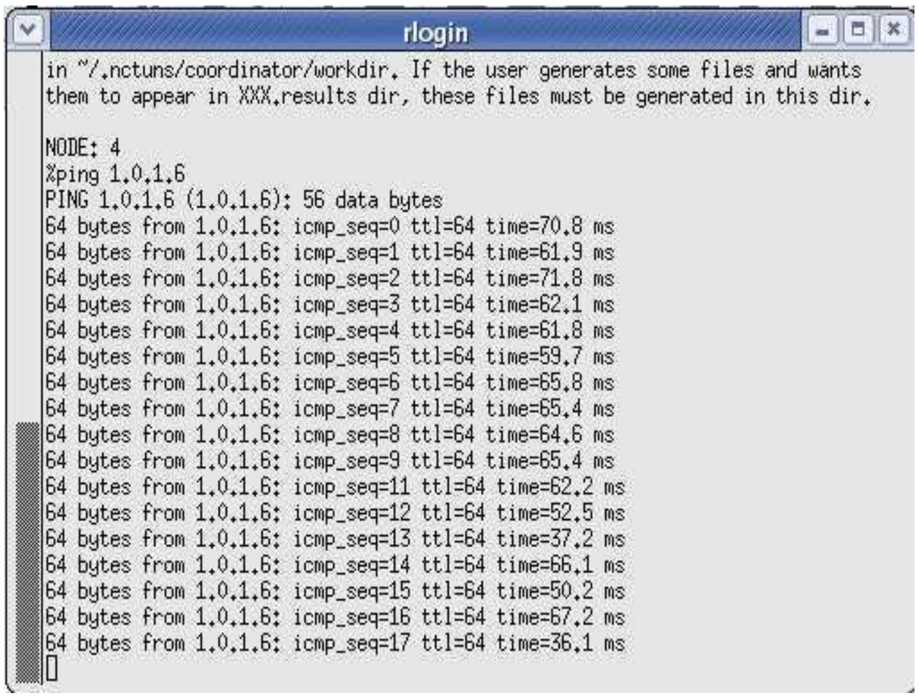
Receiver: rtcp –p 2000 –l 1024

Run MODE

Goto Simulation>Run

When the bar is moving in run mode, click on any host other than receiver and click on COMMAND CONSOLE, when a prompt is seen in window type ping IP of receiver(1.0.1.6).

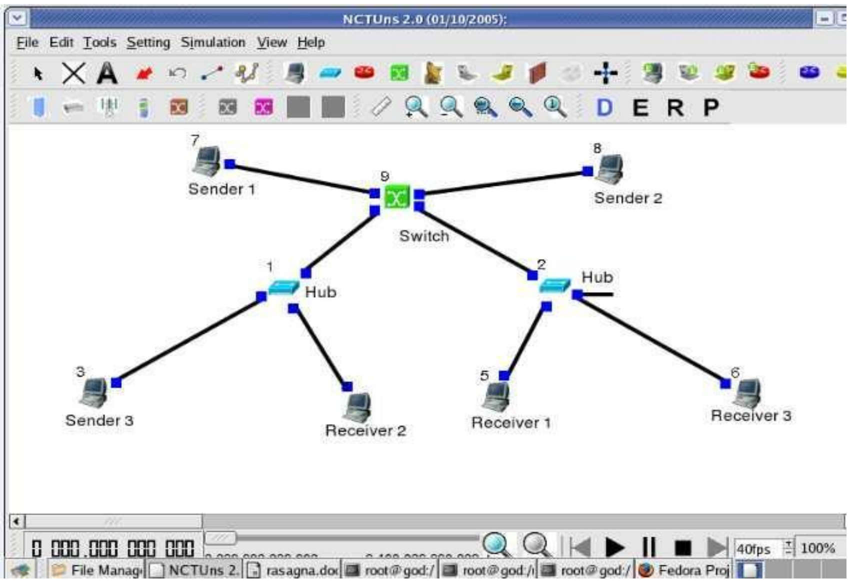
The messages are seen as below.



```
in ~/.nctuns/coordinator/workdir. If the user generates some files and wants
them to appear in XXX.results dir, these files must be generated in this dir.

NODE: 4
%ping 1.0.1.6
PING 1.0.1.6 (1.0.1.6): 56 data bytes
64 bytes from 1.0.1.6: icmp_seq=0 ttl=64 time=70.8 ms
64 bytes from 1.0.1.6: icmp_seq=1 ttl=64 time=61.9 ms
64 bytes from 1.0.1.6: icmp_seq=2 ttl=64 time=71.8 ms
64 bytes from 1.0.1.6: icmp_seq=3 ttl=64 time=62.1 ms
64 bytes from 1.0.1.6: icmp_seq=4 ttl=64 time=61.8 ms
64 bytes from 1.0.1.6: icmp_seq=5 ttl=64 time=59.7 ms
64 bytes from 1.0.1.6: icmp_seq=6 ttl=64 time=65.8 ms
64 bytes from 1.0.1.6: icmp_seq=7 ttl=64 time=65.4 ms
64 bytes from 1.0.1.6: icmp_seq=8 ttl=64 time=64.6 ms
64 bytes from 1.0.1.6: icmp_seq=9 ttl=64 time=65.4 ms
64 bytes from 1.0.1.6: icmp_seq=11 ttl=64 time=62.2 ms
64 bytes from 1.0.1.6: icmp_seq=12 ttl=64 time=52.5 ms
64 bytes from 1.0.1.6: icmp_seq=13 ttl=64 time=37.2 ms
64 bytes from 1.0.1.6: icmp_seq=14 ttl=64 time=66.1 ms
64 bytes from 1.0.1.6: icmp_seq=15 ttl=64 time=50.2 ms
64 bytes from 1.0.1.6: icmp_seq=16 ttl=64 time=67.2 ms
64 bytes from 1.0.1.6: icmp_seq=17 ttl=64 time=36.1 ms
```

4. Simulate an Ethernet LAN using n nodes (6-10), change error rate and data rate and compare throughput.

DESIGN MODE:
Draw the topology as below.



EDIT MODE:
Sender 1: stcp –p 2000 –l 1024 1.0.1.3 ***(Receiver 1 IP)***
Sender 2: stcp –p 2001 –l 1024 1.0.1.2 ***(Receiver 2 IP)***
Sender 3: stcp –p 2002 –l 1024 1.0.1.4 ***(Receiver 3 IP)***

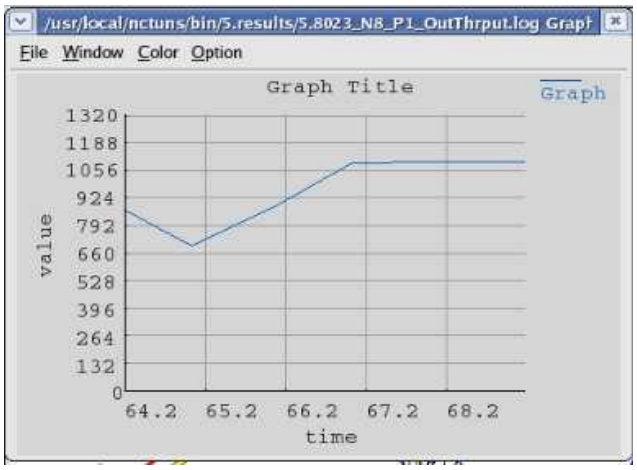
Receiver 1: rtcp –p 2000 –l 1024
Receiver 2: rtcp –p 2001 –l 1024
Receiver 3: rtcp –p 2002 –l 1024

Node Editor:
Set In throughputs in receiver's windows and OUT throughputs in Sender's windows.

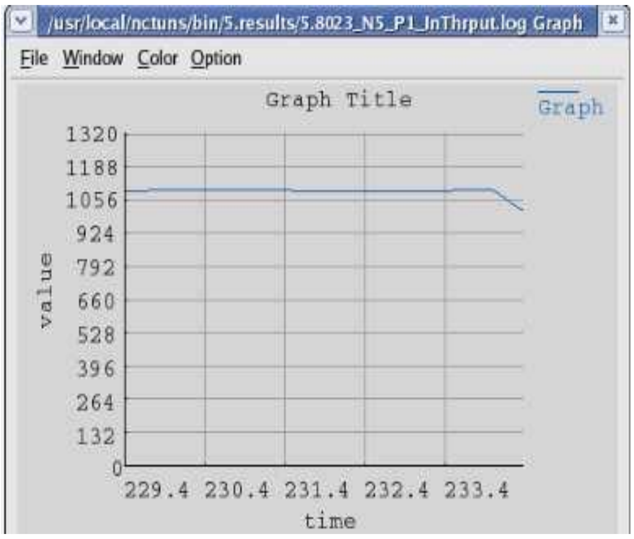
Run MODE:
Run the model

PLAY MODE:
Play the model and plot graphs for above parameters

Sender 1 OUT throughput



Receiver 1 In Throughput



INFERENCE:
Sender 1 out throughput: 1 to 1197 bits/s.
Sender 2 out throughput: 1 to 784 bits/s.
Sender 3 out throughput: 431 to 1103 bits/s.
Receiver 1 in Throughput: 1 to 1197 bits/s.
Receiver 2 in Throughput: 431 to 1100 bits/s.
Receiver 3 in Throughput: 1 to 783 bits/s.

GO BACK TO EDIT MODE *CHANGE <u>BER</u> (Data and Error rate) by double clicking on receivers point to point links to <u>0.000001</u>*

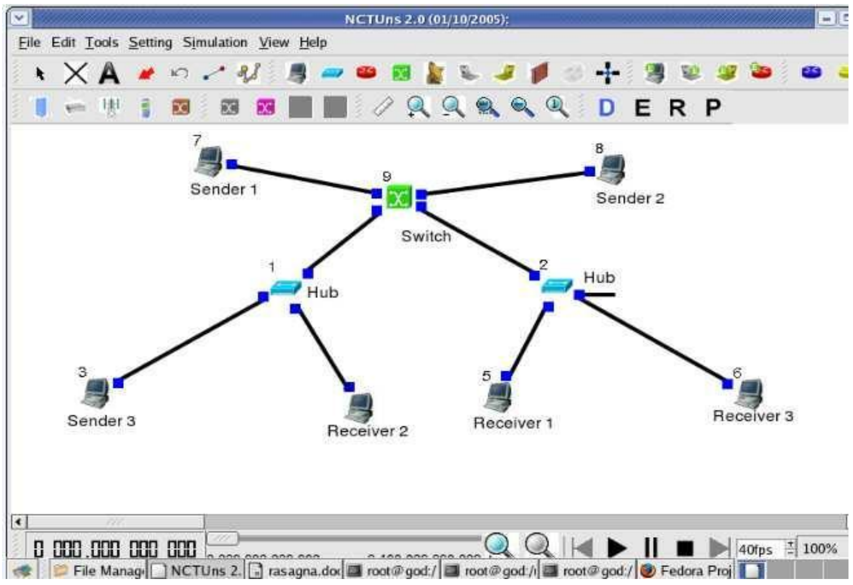
Now save the topology goto run mode and run the topology.

Play and check for throughputs as above.

INFERENCE:
Sender 1 out throughput: 3 to 443 bits/s.
Sender 2 out throughput: 22 to 447 bits/s.
Sender 3 out throughput: 1 to 481 bits/s.
Receiver 1 in Throughput: 7 to 766 bits/s.
Receiver 2 in Throughput: 22 to 447 bits/s.
Receiver 3 in Throughput: 1 to 428 bits/s.

5. Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

DESIGN MODE:
Draw the topology as below.



EDIT MODE:
Sender 1: stcp –p 2000 –l 1024 1.0.1.3 ***(Receiver 1 IP)***
Sender 2: stcp –p 2001 –l 1024 1.0.1.2 ***(Receiver 2 IP)***
Sender 3: stcp –p 2002 –l 1024 1.0.1.4 ***(Receiver 3 IP)***

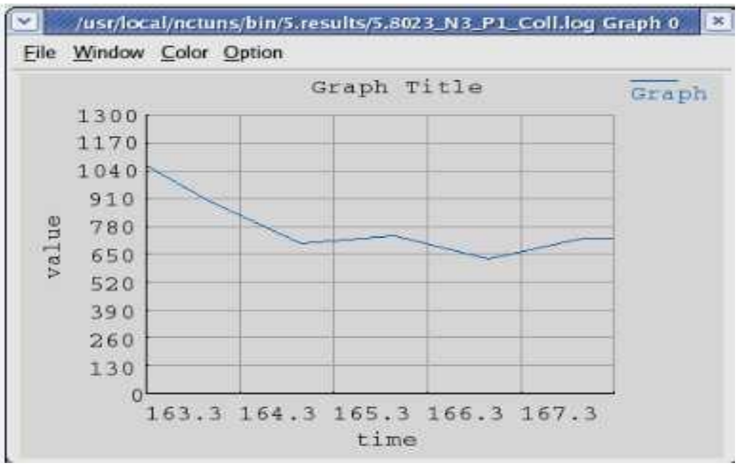
Receiver 1: rtcp –p 2000 –l 1024
Receiver 2: rtcp –p 2001 –l 1024
Receiver 3: rtcp –p 2002 –l 1024

Node Editor:
Set Collision and drop parameters for different nodes and check for the values.

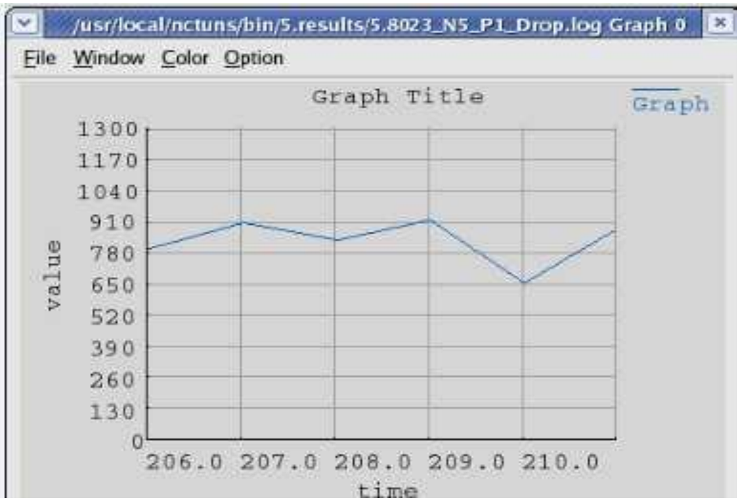
Run MODE:
Run the model

PLAY MODE:
Play the model and plot graphs for above parameters.

Collisions:



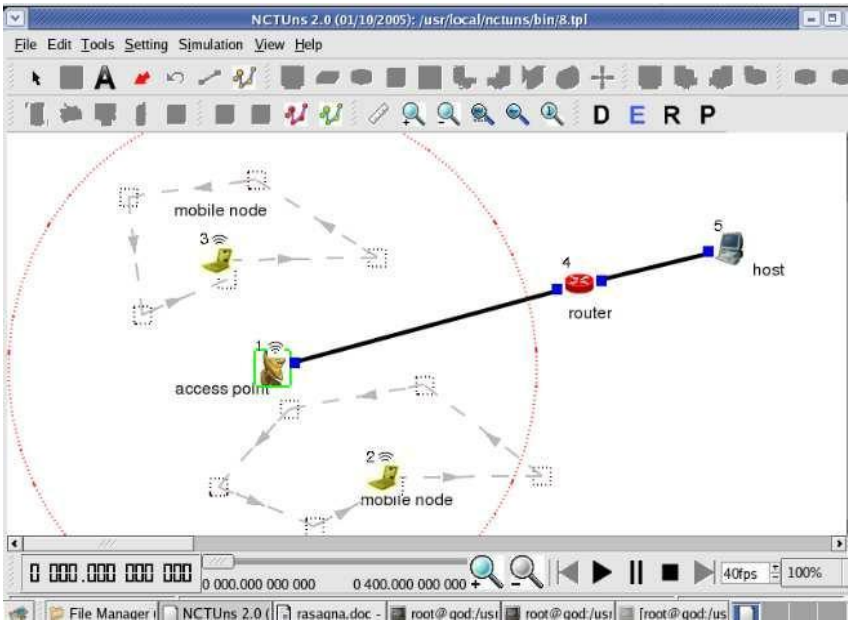
Drop:



INFERENCE:
Collision: 606 to 1190 bits/s.
Drop: 606 to 1190 bits/s.

6. Simulate simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.

DESIGN MODE:
a) Drag and drop Access Point, two--Mobile Node in Infrastructure Mode(yellow mobile symbol), router and one host.
b) Make a Point to Point Connection between Host and Router, And also Router to Access Point.



EDIT MODE:
a) Double click on access point, go to Wireless Interface tab.
b) At bottom select **show transmission range.**
c) Go to Node editor MAC 8023 and select Out Throughput parameter.
d) Now double click on MOBILE NODE 1 and go to Application tab. Click on ADD and add the following command:

ttcp –t –u –s –p 2000 1.0.1.1 (HOST IP)

Click OK.

e) Now double click on MOBILE NODE 2 and go to Application tab. Click on ADD and add the following command:

ttcp –t –u –s –p 2001 1.0.1.1 (HOST IP)

Click OK.

f) Double Click on router and select Node editor, Click on MAC 8023 and select IN and OUT throughputs. Click on other MAC 8023 and do the same. (Since there are two MAC 8023 select parameters for both).

e) Now double click on host and add command as follows:

ttcp –r –u –s –p 2000

click ok

f) Once again click on ADD and add the command

ttcp –r –u –s –p 2001

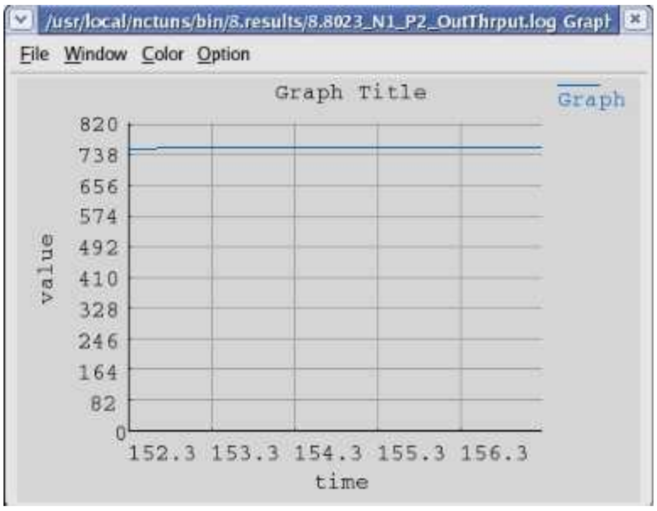
Click ok.

g) Goto Node Editor and click MAC 8023, select IN throughput.

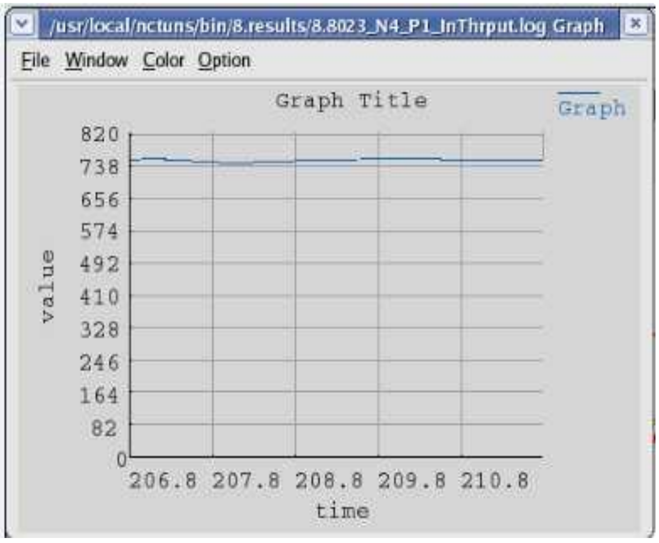
RUN MODE:
Simulation>Run the Model.

PLAY MODE:
Play the model and plot graphs for OUT throughput of Access Point and IN throughputs for both Router and Host.

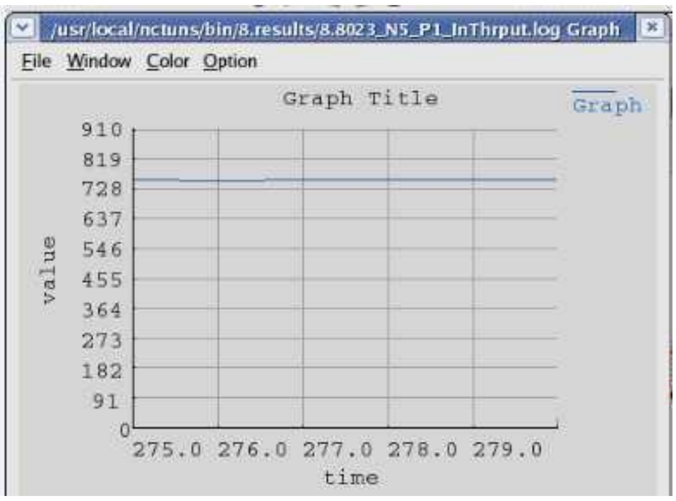
Access Point OUT Throughput:



Router IN Throughput:



Host IN Throughput:



INFERENCE:
Access Point: OUT Throughput: 255 to 760 bits/s.
Router: IN Throughput: 255 to 760 bits/s.
Host: IN Throughput: 255 to 760 bits/s.

7. Write a program for error detecting code using CRC-CCITT (16-bits).

Error detection is the detection of errors caused by noise or other impairments during transmission from the transmitter to receiver.
Error correction is the detection of errors and reconstruction of the original error free data.

Error detection schemes:-

Repetition codes.
Parity bit.
Checksum.
Cyclic Redundancy Check (CRC).

Error correcting codes:-

An error correcting code can be used for error detection and for error correction.

Error correction schemes:-

Automatic Repeat Request.
Forward Error Correction.

CRC-CCITT(16-bits):-

CCITT stands for "***International Telegraph and Telephonique Consultative Committee".***
A Cyclic Redundancy Check(CRC) is a single burst-error-detecting code. CRC's are particularly easy to implement in hardware and are commonly used in digital networks and storage devices like hard disk drives.

It is characterized by specification of a so-called generator polynomial, which is used as the divisor in a polynomial long division, taking the input data as the dividend and where the remainder becomes the result.

	CRC-CCITT	**CRC-16**	**CRC-32**
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	100000100110000010001110110110111

Table 1. International Standard CRC Polynomials

Example:
Data : 110101
Polynomial: 101

```
1 0 1)1 1 0 1 0 1 0 0( 1 1 1 0 1
      1 0 1
      0 1 1 1
        1 0 1
        0 1 0 0
          1 0 1
          0 0 1 1 0
              1 0 1
              0 1 1 0
                1 0 1
                  1 1  =  checksum.
```

Final codeword is 1 1 0 1 0 1 1 1

```
1 0 1)  1 1 0 1 0 1 1 1(1 1 1 0 1
        1 0 1
        0 1 1 1
          1 0 1
          0 1 0 0
            1 0 1
            0 0 1 1 1
                1 0 1
                0 1 0 1
                  1 0 1
                  0 0 0  =  no error in transmitted data.
```

/*program for error detection using CRC-CCITT*/

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

char data[50], crc[50],gen[20];
//char gen[] = "10001000000100001";
//G(x): x^16+x^12+x^5+1 for CRC-CCITT 16bit
int len, i, j;

void calc_crc()
{
        for (i = 0; i < strlen(gen); i++)
                crc[i] = data[i];
        do
        {
                if (crc[0] == '1')
                {
                        for(j = 1; j < strlen(gen); j++)
                                crc[j] = ((crc[j] == gen[j]) ? '0' : '1');
                }
                for (j = 0; j < strlen(gen) - 1; j++)
                        crc[j] = crc[j+1];

                crc[j] = data[i++];
        } while (i <= len + strlen(gen) - 1);
}

int main()
{
        printf("Enter Bit string\t: ");
        scanf("%s", data);
        len = strlen(data);
        printf(”Enter generating polynomial\n”);
        scanf(“%s”,gen);

        printf("Generating Polynomial\t: %s\n", gen);
        for (i = len; i < len + strlen(gen) - 1; i++)
             data[i] = '0';

        printf("Modified Data is\t: %s\n", data);
        calc_crc();

        printf("Checksum is\t\t: %s\n", crc);
        for (i = len;i < len + strlen(gen)-1; i++)
                data[i] = crc[i - len];
        printf("Final Codeword is\t: %s\n", data);

        printf("Test Error detection\n1(Yes) / 0(No)? : ");
        scanf("%d", &i);
        if (i == 1)
        {
                printf("Enter position to insert an error : ");
                scanf("%d",&i);
                data[i] = (data[i] == '0') ? '1' : '0';
                printf("Erroneous data\t\t: %s\n", data);
        }
        calc_crc();

        for (i=0;(i < strlen(gen)-1)&&(crc[i]!='1');i++);
        if (i < strlen(gen) - 1)
                printf("Error detected.\n");
        else
                printf("No Error Detected.\n");
        return 0;
}
```

8. Write a program for distance vector algorithm to find suitable path for transmission.

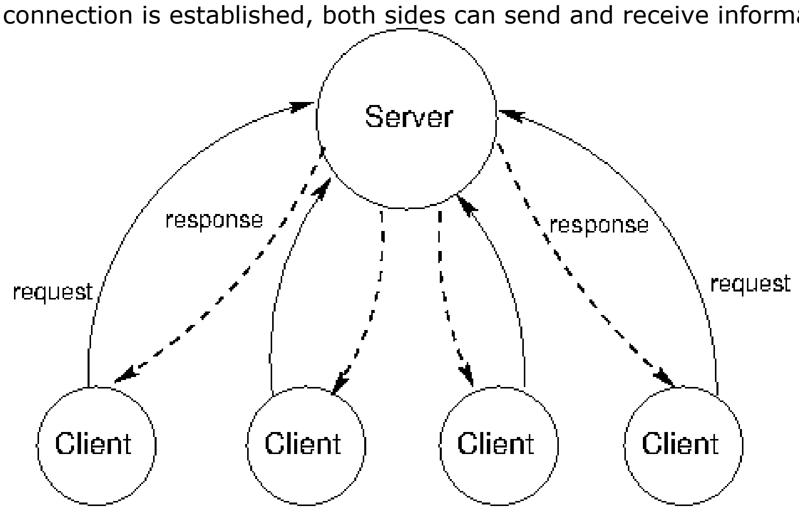
```
#include<stdio.h>
#include<conio.h>
struct node
{
        unsigned dist[20];
        unsigned from[20];
}rt[10];

void main()
{
        int n,i,j,k;
        int dmat[20][20];
        clrscr();
        printf("enter the no of nodes:\n");
        scanf("%d",&n);
        printf("enter the cost matrix:\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                {
                        scanf("%d",&dmat[i][j]);
                        rt[i].dist[j]=dmat[i][j];
                        rt[i].from[j]=j;
                }


                for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                                for(k=0;k<n;k++)
                                if(rt[i].dist[j]> rt[i].dist[k]+rt[k].dist[j])
                                {
                                        rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                                        rt[i].from[j]=k;
                                }

             for(i=0;i<n;i++)
            {
              printf("state values of route %d is\n",i+1);
              for(j=0;j<n;j++)
              {
                 printf("\n node %d to %d via %d distance %d\n",i+1,j+1,rt[i].from[j]+1,rt[i].dist[j]);
              }
            }
           printf("\n\n");
           getch();
}
```

9. Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

Client-server model
Most of the Net Applications use the Client Server architecture. These terms refer to the two processes or two applications which will be communicating with each other to exchange some information. One of the two processes acts as a client process and another process acts as a server.

Client Process:
This is the process which typically makes a request for information. After getting the response this process may terminate or may do some other processing.
For example: Internet Browser works as a client application which sends a request to Web Server to get one HTML web page.

Server Process:
This is the process which takes a request from the clients. After getting a request from the client, this process will do required processing and will gather requested information and will send it to the requestor client. Once done, it becomes ready to serve another client. Server process are always alert and ready to serve incoming requests.
For example: Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.
Notice that the client needs to know of the existence and the address of the server, but the server does not need to know the address or even the existence of the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

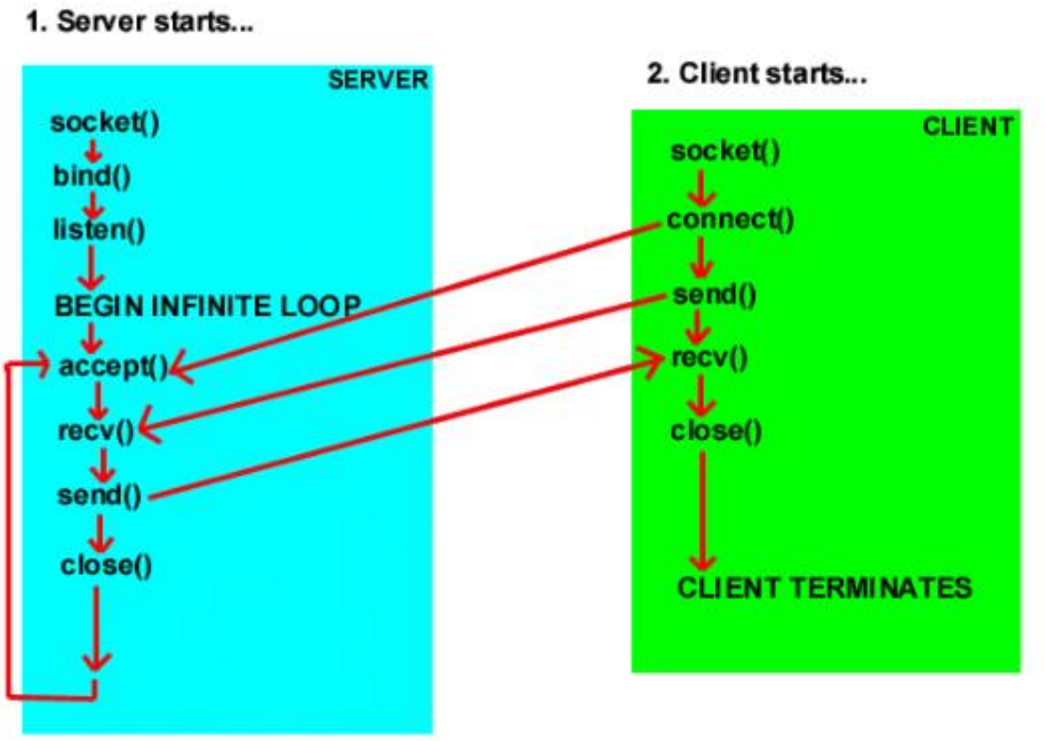


Socket APIs
- **socket:**- creates a socket of a given domain, type, protocol (buy a phone)
- **bind:**- assigns a name to the socket (get a telephone number)
- **listen:-** specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)
- **accept:**- server accepts a connection request from a client (answer phone)
- **connect:**- client requests a connection request to a server (call)
- **send:-** write to connection (speak)
- **recv:**- read from connection (listen)
- **close**:- end the call

Connection-based communication
Server performs the following actions
- **socket**
- **bind**
- **listen**
- **accept**
- **send, recv**
- **close**

Client performs the following actions
- **socket**
- **connect**
- **send, recv**
- **close**

TCP-based sockets



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

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

- domain should be set to AF_INET
- type can be SOCK_STREAM or SOCK_DGRAM
- set protocol to 0 to have socket choose the correct protocol based on type
- socket() returns a socket descriptor for use in later system calls or -1 on error

int bind(int sockfd, struct sockaddr *my_addr, int addrlen)

- sockfd is the socket descriptor returned by socket()
- my_addr is pointer to struct sockaddr that contains information about your IP address and port
- addrlen is set to sizeof(struct sockaddr)
- returns -1 on error

sockaddr

The sockaddr structure and sockaddr_in structures below are used with IPv4.

```
struct sockaddr
{
      ushort  sa_family;
      char    sa_data[14];
};

struct sockaddr_in
{
          short int sin_family;              // set to AF_INET
          unsigned short int sin_port;       // Port number
          struct in_addr sin_addr;            // Internet address
};

struct in_addr
{
   unsigned long s_addr;  // load with IP address
};
```

int listen(int sockfd, int backlog);
- sockfd is the socket file descriptor returned by socket()
- backlog is the number of connections allowed on the incoming queue
- listen() returns -1 on error
- Need to call bind() before you can listen()

int accept(int sockfd, void *addr, int *addrlen);
- sockfd is the listening socket descriptor
- information about incoming connection is stored in addr which is a pointer to a local struct sockaddr_in
- addrlen is set to sizeof(struct sockaddr_in)
- accept returns a new socket file descriptor to use for this accepted connection and -1 on error

int send(int sockfd, const void *msg, int len, int flags);
- sockfd is the socket descriptor you want to send data to (returned by socket() or got with accept())
- msg is a pointer to the data you want to send
- len is the length of that data in bytes
- set flags to 0 for now
- sent() returns the number of bytes actually sent (may be less than the number you told it to send) or -1 on error

int recv(int sockfd, void *buf, int len, int flags);
- sockfd is the socket descriptor to read from
- buf is the buffer to read the information into
- len is the maximum length of the buffer
- set flags to 0 for now
- recv() returns the number of bytes actually read into the buffer or -1 on error
- If recv() returns 0, the remote side has closed connection on you

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)
- sockfd is the socket descriptor returned by socket()
- serv_addr is pointer to struct sockaddr that contains information on destination IP address and port
- addrlen is set to sizeof(struct sockaddr)
- returns -1 on error

int close(int sockfd);
- Closes connection corresponding to the socket descriptor and frees the socket descriptor
- Will prevent any more sends and recvs

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int oflag);
- The open() function establishes the connection between a file and a file descriptor.
- It creates an open file description that refers to a file and a file descriptor that refers to that open file description.
- The file descriptor is used by other I/O functions to refer to that file.
- The path argument points to a pathname naming the file.
- values of oflag can be:
  O_RDONLY: Open for reading only. O_WRONLY: Open for writing only.
  O_RDWR: Open for reading and writing.

#include <unistd.h>
ssize_t read(int fd, void *buf, size_t nbyte);
- The read() function attempts to read nbyte bytes from the file associated with the open file descriptor, fd, into the buffer pointed to by buf.
- If nbyte is 0, read() will return 0 and have no other results.

#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t nbyte);
- The write() function attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with the open file descriptor, fd.
- If nbyte is 0, write() will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.

#include <netinet/in.h>

int inet_pton(int af, const char *src, void *dst);

- This function converts the character string src into a network address structure in the af address family, then copies the network address structure to dst. The af argument must be either AF_INET or AF_INET6.

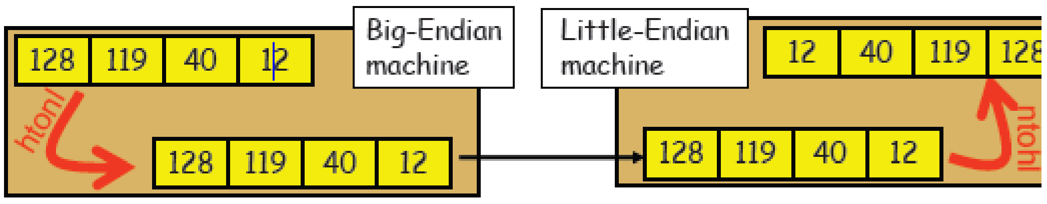
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr *in*);

- This converts a network address in a struct in_addr to a dots-and-numbers format string.
- The "n" in "ntoa" stands for network, and the "a" stands for ASCII

Two types of Byte ordering

- Host Byte-Ordering: the byte ordering used by a host (big or little)
- Network Byte-Ordering: the byte ordering used by the network –always big-endian
- Conversions:
  - htons() - Host to Network Short
  - htonl() - Host to Network Long
  - ntohs() - Network to Host Short
  - ntohl() - Network to Host Long
- On big-endian machines, these routines do nothing.
- On little-endian machines, they reverse the byte order



Algorithm (Server Side)

1. Start.
2. Create a socket using socket().
3. Bind the socket to a server-address using bind().
4. Listen to the connection using listen().
5. accept connection using accept()
6. Receive filename from client using recv()
7. Open file, then read & finally transfer{using send()} its content to client.
8. Stop.

/*Server side program*/

```
#include<stdlib.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<unistd.h>
int main()
{
        int sockfd, newsockfd, fd;
        int count, cli_addrlen, b;
        int bufsize = 1024;
        char *buffer = malloc(bufsize);
        char fname[256];
        struct sockaddr_in serv_addr, cli_addr;

        sockfd = socket(AF_INET,SOCK_STREAM,0);
        if (sockfd > 0)
                printf("The socket is created\n");

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(15000);
        serv_addr.sin_addr.s_addr = INADDR_ANY;
        b=bind(sockfd,(struct sockaddr *)&serv_addr, sizeof(serv_addr));
        if(b!=0)
                printf(" error in Binding Socket\n");

        listen(sockfd,3);

        cli_addrlen = sizeof(struct sockaddr_in);
        newsockfd =accept(sockfd,(struct sockaddr *)&cli_addr,&cli_addrlen);
        if (newsockfd > 0)
                printf("The Client %s is Connected...\n", inet_ntoa(cli_addr.sin_addr));

        recv(newsockfd, fname, 255,0);
        printf("A request for filename %s Received..\n", fname);

        fd=open(fname, O_RDONLY);
        if(fd<0)
        {
                perror("File Open Failed");
                exit(0);
        }

        while((count =read(fd, buffer, bufsize))>0)
                send(newsockfd,buffer,count,0);

        printf("Request Completed\n");
        close(newsockfd);

        return close(sockfd);
}
```

Algorithm (Client Side)

1. Start.
2. Create a socket using socket().
3. Connect the socket to the address of the server using connect().
4. Send the filename of required file to server using send().
5. Read the contents of the file sent by server using recv().
6. Stop.

/*Client side program*/

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
int main(int argc,char *argv[])
{
        int sockfd, count, c;
        int bufsize = 1024;
        char *buffer = malloc(bufsize);
        char fname[256];
        struct sockaddr_in serv_addr;

        sockfd = socket(AF_INET,SOCK_STREAM,0);
        if (sockfd > 0)
                printf("The socket is created\n");

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(15000);
        inet_pton(AF_INET, argv[1],&serv_addr.sin_addr);

        c=connect(sockfd,(struct sockaddr*)&serv_addr, sizeof(address));
        if(c== 0)
                printf("The connection was accepted with the server %s...\n", argv[1]);

        printf("Enter The Filename to Request : ");
        scanf("%s",fname);

        send(sockfd, fname, sizeof(fname), 0);

        printf("Request Accepted... Receiving File...\n\n");

        printf("The contents of file are...\n\n");

        while((count=recv(sockfd, buffer, bufsize, 0))>0)
                write(1, buffer, count);

        printf("\nEOF\n");

        return close(sockfd);
}
```

10. Implement the client-server program using as message queues or FIFOs as IPC channels.

A named pipe is really just a special kind of file (a FIFO file) on the local hard drive. Unlike a regular file, a FIFO file does not contain any user information. Instead, it allows two or more processes to communicate with each other by reading/writing to/from this file.

Creating a FIFO File
int mkfifo(const char *filename, mode_t mode)

The function call creates a new named pipe or returns an error if the named pipe already exists. The normal, successful return value from mkfifo is **0**. In the case of an error, **-1** is returned. A default mode allows read, write, and search permission for the owner, group, and others (0777)

Using a FIFO File
Since this named pipe looks like a file, you can use all the system calls associated with files to interact with it. In particular, you can use the open, read, write, and close system calls. The following are prototypes for each of these calls as you will need to use them.

```
int open(const char *pathname, int flags);
int read(int fd, void *buf, size_t count);
int write(int fd, const void *buf, size_t count);
```

Algorithm (Client Side)

1. Start.
2. Open well known server FIFO in write mode.
3. Write the pathname of the file in this FIFO and send the request.
4. Open the client specified FIFO in read mode and wait for reply.
5. When the contents of the file are available in FIFO, display it on the terminal
6. Stop.

/*Client side program*/

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
char fname[256];
int main()
{
        size_t count;
        char buff[512];
        int readfd,writefd;
        printf("Trying to Connect to Server..\n");
        writefd = open("FIFO1", O_WRONLY, 0);
        readfd  = open("FIFO2", O_RDONLY, 0);
        printf("Connected..\n");
        printf("Enter the filename to request from server: ");
        scanf("%s",fname);
        write(writefd, fname, strlen(fname));
        printf("Waiting for Server to reply..\n");
        while((count =read(readfd,buff,512))>0)
                write(1,buff, count);
        close(readfd);
        close(writefd);
        return 0;
}
```

Algorithm (Server Side)

1. Start.
2. Create a well known FIFO using mkfifo()
3. Open FIFO in read only mode to accept request from clients.
4. When client opens the other end of FIFO in write only mode, then read the contents and store it in buffer.
5. Create another FIFO in write mode to send replies.
6. Open the requested file by the client and write the contents into the client specified FIFO and terminate the connection.
7. Stop.

/*Server side program*/

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>
#include<string.h>
char fname[256];

int main()
{
        int readfd, writefd, fd;
        size_t count;
        char buff[512];
        if (mkfifo(“FIFO1”, 0666)<0)
                {
                        printf("Cant Create FIFO Files\n");
                        exit(0);
                }
        if (mkfifo(“FIFO2”, 0666)<0)
                {
                        printf("Cant Create FIFO Files\n");
                        exit(0);
                }
        printf("Waiting for connection Request..\n");
        readfd =open(“FIFO1”, O_RDONLY, 0);
        writefd=open(“FIFO2”, O_WRONLY, 0);
        printf("Connection Established..\n");
        read(readfd, fname, 255);
        printf("Client has requested file %s\n", fname);
        fd=open(fname,O_RDWR);
        if ((fd<0)
        {
                strcpy(buff,"File does not exist..\n");
                write(writefd, buff, strlen(buff));
        }
        else
        {
                while((count=read(fd, buff,512))>0)
                write(writefd, buff, count);
         }
        close(readfd);
        //unlink(FIFO1);
        close(writefd);
        //unlink(FIFO2);
}
```

11. Write a program for simple RSA algorithm to encrypt and decrypt the data.

The RSA public-key Encryption Algorithm:-

It is one of the first public key scheme was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adlemen hence the name RSA. It is first published in 1978.

RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and N-1 for some N.

Plaintext block M:

Ciphertext block C:

$C=M^e \bmod N$

$M=C^d \bmod N = (M^e)^d \bmod N= M^{ed} \bmod N$

Both sender and receiver must know the values of N and e and only receiver knows the value of d.

This is a public-key algorithm with a public key of KU={e,N} and a private key of KR={d,n}.

For this algorithm to be satisfactory for public key encryption, the following requirements must be met.

- It is possible to find values of e, d, N such that $M^{ed} \bmod N = M$ for all M<N.
- It is relatively easy to calculate M^e and C^d for all values of M<N.
- It is infeasible to determine d given e and N.

The first two requirements are easily met. The third requirement can be met for large values of e and N. We need the quantity Q(n) called euler totient of N, which is the number of positive integers less than N and relatively prime to N.

Then select an integer e that is relatively prime to Q(N)[i.e. the greatest common divisor of e and Q(N) is 1]. Finally calculate d as multiplicative inverse of e, modulo Q(N).

Suppose that user A has published its public key and that user B wishes to send the message M to A.

Then B calculates $C=M^e \bmod N$ and transmits C.

On receipt of this cipher text, user A decrypts by calculating

$M=C^d \bmod N$.

e.g. the keys are generated as follows:-

1. select two prime numbers, p=17 and q=11
2. Calculate N=p*q=17*11=187.
3. Calculate Q(N)=(p-1)(q-1)=(17-1)*(11-1)=16*10=160.
4. Select e such that e is relatively prime to Q(N)=160 and less than Q(N);we choose e=7.
5. Determine d such that (de mod 160)=1 and d<160. The correct value is d=23, because 23*7=161=(1*160)+1 .
   Public key KU= {7,187}
   Private key KR= {23,187}

M=88 for encryption, we need to calculate

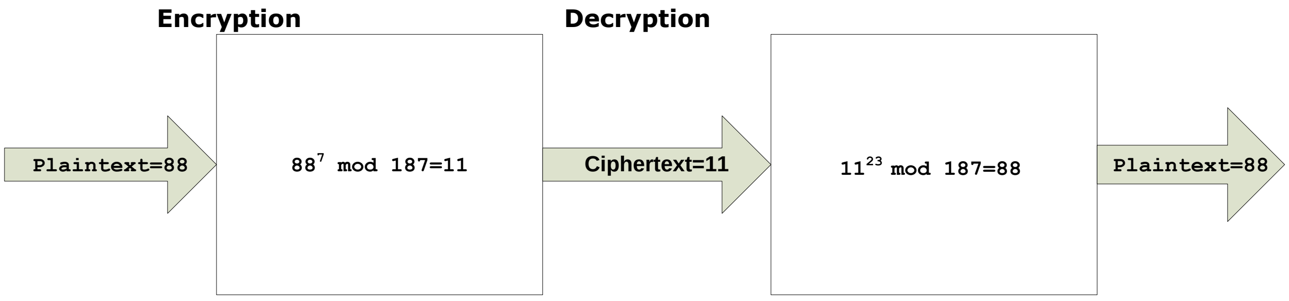
$C=88^7 \bmod 187$

C=11

For decryption $M= c^d \bmod n$

$M= 11^{23} \bmod 187$

M=88



<u>Key generation</u>

Select p,q	p and q both prime, p!=q
Calculate N=p*q	
Calculate Q(N)=(p-1)*(q-1).	
Select integer e	gcd(Q(n),e)=1;1<e<Q(N)
Calculate d	de mod Q(N)=1
Public key	KU={e,N}
Private key	KR={d,N}

e.g.

p=5, q=11, e=3 and M=65

N=p*q=5*11=55

Q(N)=(P-1)*(q-1)=(5-1)*(11-1)=4*10=40

e*d=1+k*Q(N)

3*27=1+(2*40)

81=81

Therefore d=27.
Ciphertext c= M^e mod N
C=65^3 mod 55
k = (k * x) % N;[repeat this for three times because e=3]
initially k=1,
k=(1*65)mod55
k=10
k=(10*65) mod 55
k=45
k=(45*65)mod 55
k=10 is the ciphertext
To get plaintext we need to calculate M=c^d mod N=10^{27}mod 55.
For calculating this use formula k=(k*x)mod N initially K=1,repeat this for 27 times, finally we get K=M=65 that is the original text(plaintext).

Difference between encoding and encryption

Encoding transforms data into another format using a scheme that is publicly available so that it can be easily reversed, no keys required.
Encryption transforms data into another format in such a way that only specific individual(s) can reverse the transformation. Encryption is for maintaining data confidentiality.

/* program for RSA algorithm */

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

long int e,d,n;
long int val[100];

int encode(char ch)
{
        int i; long int temp=ch;
        for(i=1;i<e;i++)
                temp=temp*ch%n;
        return temp;
}

char decode(long int ch)
{
        int i; long int temp=ch;
        for(i=1;i<d;i++)
                ch=temp*ch%n;
        return ch;
  }

int gcd(long a,long b)
{
        if(b==0)
                return a;
        else
                return (gcd(b,a%b));
}

int prime(int a)
{
        int i;
        for(i=2;i<(a/2);i++)
        {
                if((a%i)==0)
                        return 0;
        }
        return 1;

}
```

```
void main()
{
        int i,flag;
        long int p,q,z;
        char pt[100],ct[100];

        clrscr();
        printf("\nenter PLAIN TEXT to be encoded.....\n\t");
        gets(pt);

        printf("\nENTER FIRST PRIME NUMBER\n");
        scanf("%ld",&p);
        flag=prime(p);
        if(flag==0)
        {
                printf("\nWRONG INPUT\n");
                getch();
                exit(1);
        }
        printf("\nENTER ANOTHER PRIME NUMBER\n");
        scanf("%ld",&q);
        flag=prime(q);
        if(flag==0||p==q)
        {
                printf("\nWRONG INPUT\n");
                getch();
                exit(1);
        }

        n=p*q;
        z=(p-1)*(q-1);

        do
        {
                e=rand()%z;
        }while(gcd(e,z)!=1);

        do
        {
                d=rand()%z;
        }while((d*e%z)!=1);

        printf("\n\nThe values computed are :\n");
        printf("p=%ld   q=%ld  n=%ld  z= %ld  e=%ld  d=%ld",p,q,n,z,e,d);
        printf("\n The PUBLIC Key computed is { %ld , %ld }",e,n);
        printf("\n The PRIVATE Key computed is { %ld , %ld }\n",d,n);

        for(i=0;pt[i]!='\0';i++)
                val[i]=encode(pt[i]);
        val[i]=-999;

        printf("  Encoded message is\n\t");
        for(i=0;val[i]!=-999;i++)
                printf("%ld\t",val[i]);


        for(i=0;val[i]!=-999;i++)
                ct[i]=decode(val[i]);

        ct[i]='\0';
        printf("\n  Decoded message is\n\t%s\n",ct);

        getch();
}
```

Output:-

```
nenter PLAIN TEXT to be encoded.....
a
ENTER FIRST PRIME NUMBER
11
ENTER ANOTHER PRIME NUMBER
13
The values computed are
p=11  q=13  n=143  z=120  e=37  d=13
The PUBLIC Key computed is (37,143)
The PRIVATE Key computed is {13,143}
Encoded message is
136
Decoded message is:
a
```

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

Theory:-

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.
The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.
Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called **traffic shaping**.
The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.
The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

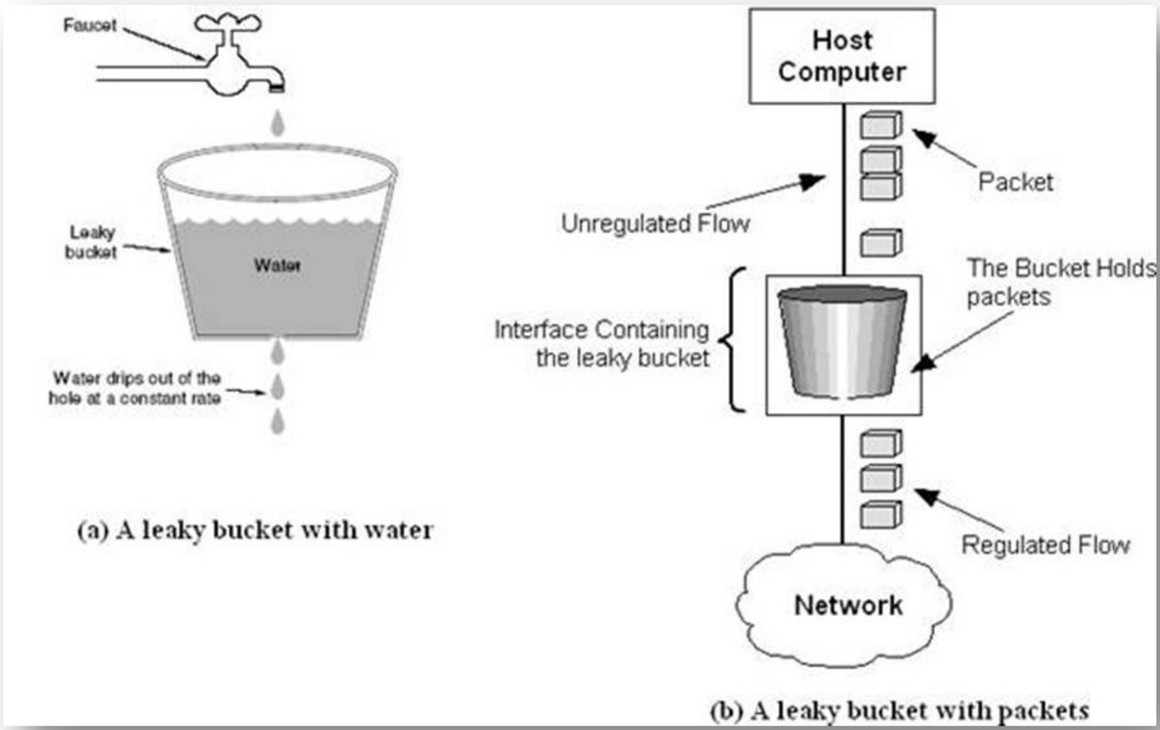

Faucet
Leaky bucket
Water
Water drips out of the hole at a constant rate
(a) A leaky bucket with water
Host Computer
Packet
Unregulated Flow
The Bucket Holds packets
Interface Containing the leaky bucket
Regulated Flow
Network
(b) A leaky bucket with packets

Figure:-The leaky bucket traffic shaping algorithm

/*program for congestion control using leaky bucket algorithm*/

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<stdlib.h>
#define bucketSize 512
void bktInput(int a,int b)
{
	if(a>bucketSize)
		printf("\n\t\tBucket overflow");
	else
	{
		delay(500);
		while(a>b)
		{
			printf("\n %d bytes transmitted.",b);
			a=a-b;
			delay(500);
		}
		if (a>0)
			printf("\n%d Last  bytes transmitted \t",a);
		printf("\n Bucket output successful");
	}
}

void main()
{
		int op, pktSize;
		clrscr();
		randomize();
		printf("Enter output rate : ");
		scanf("%d",&op);
		for(int i=1;i<=5;i++)
		{
			delay(random(1000));
			pktSize=random(1000);
			printf("\n Packet size = %d",pktSize);
			bktInput(pktSize,op);
		}
		getch();
}
```