NEW HORIZON
COLLEGE OF ENGINEERING

Computer Network Laboratory Manual

Course Code : 20ISL77A

Semester : VII 2021-22

Department of Information Science and Engineering

Prepared By	Approved By	Authorized By
Mrs. Shoba S	**Dr.Kalaivani D**	**Dr.R J Anandhi**

New Horizon College of Engineering

VISION

To achieve total quality in education and excellent knowledge management through specific, measurable, attainable relevant, time- bound goals and continuous improvement methods.

MISSION

To mould our students into a holistic personality accomplished in emotional, moral, intellectual, social and mental capabilities, besides inculcating a capacity for critical and lateral thinking.

Department of Information Science & Engineering

VISION

To evolve as a centre of academic excellence and advanced research in information science and engineering discipline and to endeavour the computational competence of students for their dream career achievement and enhancing the managerial and technical skills.

MISSION

To inculcate students with profound understanding of fundamentals related to discipline, attitudes, skills and their application in solving real world problems, with an inclination towards societal issues and research.

PROGRAM EDUCATIONAL OBJECTIVES

PEO1: To excel in their professional career with expertise in providing solutions to Information Technology problems.

PEO2: To pursue higher studies with profound knowledge enriched with academia and industrial skill sets.

PEO3: To exhibit adaptive and agile skills in the core area of Information Science & Engineering to meet the technical and managerial challenges.

PEO4: To demonstrate interpersonal skills, professional ethics to work in a team to make a positive impact on society.

COMPUTER NETWORKS LABORATORY

Course Code	**: 20ISL77**	**Credits**	**: 1.5**
L:T:P	**: 0:0:1.5**	**CIE Marks**	**: 25**
Exam Hours	**: 3**	**SEE Marks**	**: 25**

Course Outcomes: At the end of the Course, the Student will be able to:

CO1	Implement different network protocols
CO2	Be familiar with the various routing algorithms
CO3	Learn to communicate between two desktop computers
CO4	Learn and use simulation tools

Mapping of Course Outcomes to Program Outcomes:

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	3		3	2						2
CO2	3	3	3		3	3	3		2			2
CO3	3	3	3	3	3	3	3	3				
CO4	3	3	3	3	3	3	3	3	3		3	3

Experiment No.	Experiment
PART-A	
1	Write a program for error detecting code using CRC-CCITT (16- bits).
2	Write a program for distance vector algorithm to find suitable path for transmission.

3	Implementation of Stop and Wait Protocol.
4	Write a program for congestion control using leaky bucket algorithm.
5	Implement the data link layer framing methods such as character, character stuffing and bit stuffing.
6	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.
PART-B	
7	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.
8	Simulate an Ethernet LAN using n nodes (6-10), change error rate and data rate and compare throughput.
9	Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
10	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.
11.	Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
12	Simulate simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.

For SEE Examination:

- One experiment from part A & One experiment from part B to be given
- Examination will be conducted for 50 marks and scaled down to 25 marks
- Marks Distribution : Procedure write-up – 20%
  Conduction – 60%
  Viva – Voce – 20%
- Change of the experiment is allowed only once and procedure write-up marks will be considered as ‘0’

CIE - Continuous Internal Evaluation (25 Marks)

Bloom's Category	Tests (25 Marks)
Remember	
Understand	
Apply	10
Analyze	
Evaluate	
Create	15

SEE – Semester End Examination (25 Marks)

Bloom's Taxonomy	Marks
Remember	
Understand	
Apply	10
Analyze	
Evaluate	
Create	15

PART A

Program1.Write a program for error detecting code using CRC-CCITT (16- bits).

Generally data gets transmitted from one system to another in a network. While getting transmitted, data may get manipulated, due to noise signals or any other external factor. Hence, there should be certain technique to detect to detect errors at the receiving end. One such technique is called as CYLIC REDUDANCY CHECK(CRC). This method is also known as polynomial code.

CRC is a single bit error detection technique. In other words, during transmission, if a single bit gets manipulated, then the CRC technique can detect the error. A suitable generator polynomial, which is common both to the sender and the receiver, should be selected. The sender uses the generator polynomial (G(p)) , to obtain the checksum bits.It can be obtained by dividing the data by G(p). The division is based upon, polynomial arithmetic which is similar to Exclusive OR(XOR) operation. The generator polynomial G(p) should be selected in such a way that, the data should be longer the G(p). The calculated checksum bits are attached to the end of the original data and are transmitted.

` At the receiver end, the obtained information is again divided by G(p).If the remainder obtained is zero, then the transmission is error free else the receiver should understand that an error is induced during transmission and may request for re-transmission.

The calculation of checksum bits and the error detection procedure is shown in the following example. Lets assume that the data to be transmitted and the generator polynomial to be,

Data input(x)= $x^9+x^8+x^6+x^4+x^3+x+1$ = 1101011011

Generator polynomial - G(p) = x^4+x+1 =10011

At the sender end,Input(x) = 11010110110000 (**Note :** add those many zeros as that of the highest power of G(p), Divide input (x) by G(p).

```
          10011 ) 11010110110000 ( 110000101
                  10011
                  ------------------------
                  010011
                    10011
```

```
                    --------------------
                     0000010110
                           10011
                    -------------------------
                           0010100
                             10011
                           ------------
                                1110
```

So CRC is the remainder i.e CRC = 1110
Now the data transmitted by the sender is the original data + CRC
i.e 1101011011 + 1110 = 11010110111110

At the receiver end
Case 1 : Assume that the data is error free so now data received is 11010110111110

Divide the received data by G(p) and if the remainder is zero it is confirmed that data is error free

```
10011 ) 11010110111110 ( 11000011
          10011
          ---------
          010011
            10011
          ----------
            0000010111
                  10011
                --------------
                 0010011
                    10011
                --------------------
                     000000
```

Remainder is zero. Hence there is no error
Case 2 : If the data is erroneous….
i.e the received data is 11010111111110

Note that error has been inserted at the 8th bit. If error has been induced then after dividing the received data by the generator polynomial, the remainder will not be zero

```
10011 ) 1101011111110 ( 110000111
          10011
          --------
          010011
            10011
```

```
---------------
  0000011111
       10011
-----------------
       010101
        10011
      ---------------
         1100
```

Program-Name: crc.c

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

intcrc(char *input,char *output,char *gp,int mode)
{
intj,k;
strcpy(output,input);
if(mode)
  {
      for(j=1; j<strlen(gp); j++)
      strcat(output,"0");
  }
for(j=0; j<strlen(input); j++)
      if(*(output+j) == '1')
      for(k=0; k<strlen(gp); k++)
            {
```

```
                if (((*(output+j+k) =='0') && (gp[k] == '0')                    ||
                  (*(output+j+k) == '1') && (gp[k] == '1')))
                        *(output+j+k)='0';
                else
                        *(output+j+k)='1';
            }
for(j=0; j<strlen(output); j++)
if(output[j] == '1')
return 1;
return 0;
 }

int main()
{
char input[50],output[50];
charrecv[50], gp[50];
system("clear");
printf("\n Enter the input message in binary\n");
scanf("%s",input);
printf("\n Enter the generator polynomial\n");
scanf("%s",gp);
crc(input,output,gp,1);
printf("\n The transmitted message is %s %s\n",input,
                                                output+strlen (input));
printf("\n\n Enter the received message in binary \n");
scanf("%s",recv);
if(!crc(recv,output,gp,0))
```

```
printf("\n No error in data\n");
else
printf("\n Error in data transmission has occurred\n");
}
```

Compile and run

```
$  cc –o crccrc.c
$ ./crc
$ Enter the input message in binary
  1101011011
$ Enter the generator polynomial
  10011
  The transmitted message is 1101011011 1110

  Enter the received message in binary
  11010110111110
  No error in data

  Enter the received message in binary
  11011110111110
  Error in data transmission has occurred
```

Program 2.Write a program for distance vector algorithm to find suitable path for transmission.

Routing algorithm is a part of network layer, which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision must be made a new for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new established route is being set up. The latter case is

sometimes called session routing, because a rout remains in force for an entire user session (e.g., login session at a terminal or a file).

Routing algorithms can be grouped into two major classes: adaptive and nonadaptive.Nonadaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from X to Y is compute in advance, offline, and downloaded to the routers when the network ids booted. This procedure is sometime called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ΔT sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbors.

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962).It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred out going line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbor.

If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets that the receiver just time stamps and sends back as fast as possible.

Program-Name :dv.c

```
#include<stdio.h>

#include<string.h>

struct node
```

```
{
        intdist[20];
        int from[20];
}rt[10];

int main()
{
        intdmat[20][20];
        int n=i=j=k=0,count=0;
        system("clear");
        printf("Enter The Number 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]);
                        dmat[i][i]=0;
                        rt[i].dist[j]=dmat[i][j];
                        rt[i].from[j]=j;
                }
        do
        {
        count=0;
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        for(k=0;k<n;k++)
```

```
                if(rt[i].dist[j]>                                dmat[i][k]+rt[k].dist[j])
                {
                rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                rt[i].from[j]=k;
                count++;
                }
        }while (count!=0);
        for(i=0;i<n;i++)
        {
                printf("State Value For Router %d Is\n",i);
                for(j=0;j<n;j++)
                {
                printf("\t\tVia %d Distance %d",
rt[i].from[j],rt[i].dist[j]);
                }
        }}
```

Compile and run

```
$  cc –o dv dv.c
$  ./dv

Enter The Number Of Nodes
3
Enter The Cost Matrix
0 3 6
```

7 0 9

5 7 0

State Value For Router 0 Is

Via 0 Distance 0

Via 1 Distance 3

Via 2 Distance 6

State Value For Router 1 Is

Via 0 Distance 7

Via 1 Distance 0

Via 2 Distance 9

State Value For Router 2 Is

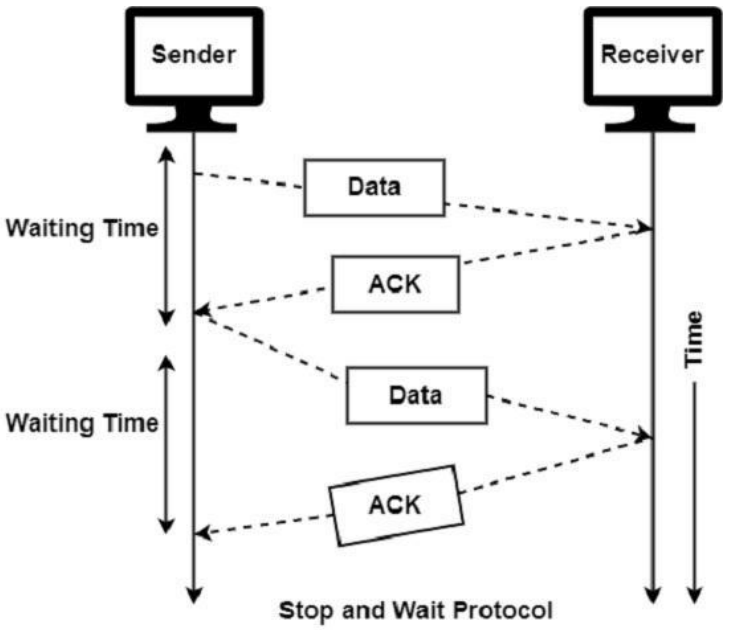
Via 0 Distance 5

Via 1 Distance 7

Via 2 Distance 0

Program 3. Implementation of Stop and Wait Protocol.

It is the simplest flow control method. In this, the sender will transmit one frame at a time to the receiver. The sender will **stop and wait** for the acknowledgement from the receiver.



The main advantage of stop & wait protocols is their accuracy. The next frame is transmitted only when the first frame is acknowledged. So there is no chance of the frame being lost.

The drawback of this approach is that it is inefficient. It makes the transmission process slow. An individual frame travels from source to destination in this method, and a single acknowledgement travels from destination to source. As a result, each frame sent and received uses the entire time needed to traverse the link.

The features of Stop and Wait Protocol are as follows −

- It is used in Connection-oriented communication.
- It offers error and flows control.
- It can be used in data Link and transport Layers.
- Stop and Wait ARQ executes Sliding Window Protocol with Window Size 1.

```
#include <stdio.h>

int main()

{

int framesize,sent=0,ack,i;

printf("enter the number of frames");
```

```
scanf("%d",&framesize);
while(1)
{
   for(i=0;i<framesize;i++)
   {
      printf("frame %d has to be transmitted\n",sent);
      sent++;
      if(sent==framesize)
      break;

   }
printf("\n please enter the last acknowledgement recieved\n");
scanf("%d",&ack);
if(ack>=framesize)
break;
else
sent=ack;
}
return 0;
}
```

Program 4.Write a program for Congestion control using the leaky bucket algorithm

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.

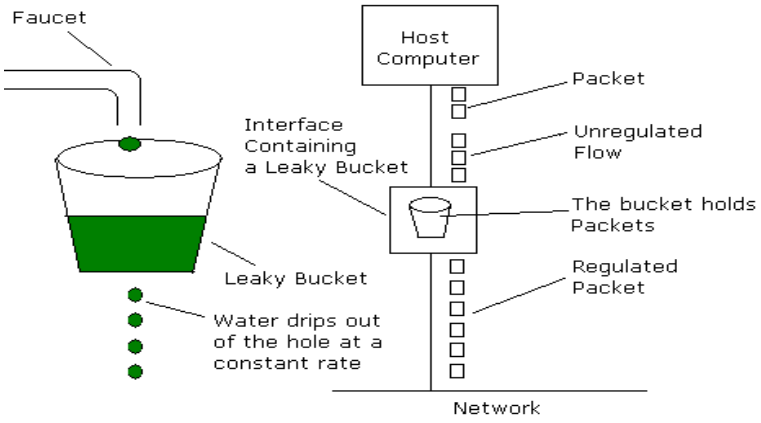
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn 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.



Input for the leaky bucket algorithm :

- Bucket size – the number of packets that the bucket can hold at atime.
- Simulation time – number of clock ticks for which the packets are to be sent to the network.
- Processing rate – number of packets outputted by the bucket per clock tick.

- General steps :
-
- At each tick, a counter is added with the size of the incoming packets.
- Counter holds the total number of packets inside the bucket (i e previously left over packets and the current incoming packets)
- If the counter value is less than the processing rate, then all packets within the bucket will be transmitted out.
- If the processing rate is less than the counter value, then number of packets equal to that of the processing rate will be transmitted out and those many packets will be subtracted from the counter.
- If the sum of the counter and the number of incoming packets is greater than the bucket size, then the dropped packets are calculated as
-
- Drop packets = total packets in counter – bucket size
- Which are allowed to enter the bucket in the next clock tick.
- This process is continued until all the incoming packets are transmitted to the network and the simulation time expires.
- After the simulation time expires, process is continued until all the packets in the bucket are transmitted to the network, here the counter is not incremented,its only decremented until it reaches zero.

Program-Name :Congestion.c

```
#include<stdio.h>

#include<string.h>


int min(intx,int y)

{

        if(x<y)

                return x;
```

```
        else
                return y;
}

int main()
{
        int drop=0,mini,nsec,cap,count=0,i,inp[25],process;
        //syste("clear");
        printf("Enter The Bucket Size\n");
        scanf("%d",&cap);
        printf("Enter The Processing Rate\n");
        scanf("%d",&process);
        printf("Enter The No. Of Seconds You Want To Stimulate\n");
        scanf("%d",&nsec);

        for(i=0;i<nsec;i++)
        {
                printf("Enter The Size Of The Packet Entering At %d
                sec\n",i+1);
                scanf("%d",&inp[i]);
        }

        printf("\nSecond|PacketRecieved|PacketSent|Packet Left|Packet Dropped|\n");
        printf("-------------------------------------------------          ---------\n");

        for(i=0;i<nsec;i++)
        {
```

```
        count+=inp[i];
        if(count>cap)
        {
                drop=count-cap;
                count=cap;
        }
        printf("%d",i+1);
        printf("\t%d",inp[i]);
        mini=min(count,process);
        printf("\t\t%d",mini);
        count=count-mini;
        printf("\t\t%d",count);
        printf("\t\t%d\n",drop);
        drop=0;
}

for(;count!=0;i++)
{
        if(count>cap)
        {
                drop=count-cap;
                count=cap;
        }
        printf("%d",i+1);
        printf("\t0");
        mini=min(count,process);
        printf("\t\t%d",mini);
```

```
                count=count-mini;
                printf("\t\t%d",count);
                printf("\t\t%d\n",drop);

        }
}
```

Compile and run

$ cc –o Congestion Congestion.c

$ ./Congestion

Enter The Bucket Size

5

Enter The Processing Rate

2

Enter The No. Of Seconds You Want To Stimulate

3

Enter The Size Of The Packet Entering At 1 sec

5

Enter The Size Of The Packet Entering At 1 sec

4

Enter The Size Of The Packet Entering At 1 sec

3

Second|PacketRecieved|PacketSent| Packet Left|Packet Dropped|

1	5	2	3	0
2	4	2	3	2
3	3	2	3	1
4	0	2	1	0
5	0	1	0	0

Program 5.Implement the data link layer framing methods such as character, character stuffing and bit stuffing.

Bit stuffing:

- Allows frame to contain arbitrary number of bits and arbitrary character size. The frames are separated by separating flag.
- Each frame begins and ends with a special bit pattern, 01111110 called a flag byte. When five consecutive l's are encountered in the data, it automatically stuffs a '0' bit into outgoing bit stream.
- In this method, frames contain an arbitrary number of bits and allow character codes with an arbitrary number of bits per character. In his case, each frame starts and ends with a special bit pattern, 01111110.
- In the data a 0 bit is automatically stuffed into the outgoing bit stream whenever the sender's data link layer finds five consecutive 1s.
- This bit stuffing is similar to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.
- When the receiver sees five consecutive incoming i bits, followed by a o bit, it automatically destuffs (i.e., deletes) the 0 bit. Bit Stuffing is completely transparent to network layer as byte stuffing. The figure1 below gives an example of bit stuffing.
- This method of framing finds its application in networks in which the change of data into code on the physical medium contains some repeated or duplicate data. For example, some LANs encodes bit of data by using 2 physical bits.

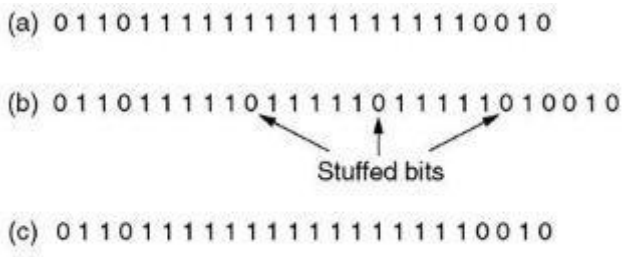


Fig1: Bit stuffing

Byte stuffing:

- In this method, start and end of frame are recognized with the help of flag bytes. Each frames starts with and ends with a flag byte. Two consecutive flag bytes indicate the end of one frame and start of the next one. The flag bytes used in the figure 2 used is named as “ESC” flag byte.
- A frame delimited by flag bytes. This framing method is only applicable in 8-bit character codes which are a major disadvantage of this method as not all character codes use 8-bit characters e.g. Unicode.
- Four example of byte sequences before and after stuffing:

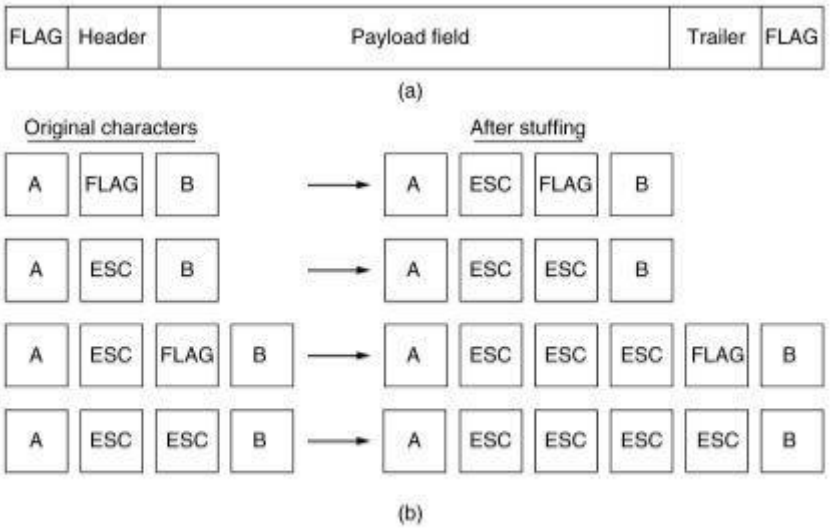


Fig2: Framing with Byte stuffing

It has same idea as bit-stuffing, but operates on bytes instead of bits. Use reserved characters to indicate the start and end of a frame. For instance, use the two-character sequence DLE STX (Data-Link Escape, Start of TeXt) to signal the beginning of a frame, and the sequence DLE ETX (End of TeXt) to flag the frame's end.

Bit Stuffing

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
        int i, j,count=0,nl;
char str[100];
        clrscr();
        printf("enter the bit string:  ");
        gets(str);
        for (i=0;i<strlen(str);i++) {
                count=0;
                //the following code search the six ones in given string
                for (j=i;j<=(i+5);j++) {
                        if(str[j]=='1') {
                                count++;
                        }
                }
                //if there is six ones then folling code execute to bit stuffing after five ones
```

```
                if(count==6) {
                        nl=strlen(str)+2;
                        for (;nl>=(i+5);nl--) {
                                str[nl]=str[nl-1];
                        }
                        str[i+5]='0';
                        i=i+7;
                }
        }
printf("The stuffed bit string:  ");
puts(str);

}
```

Output

enter the bit string: 10101111110

The stuffed bit string: 101011111010

Character Stuffing

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void charc(void);
void main()
{
int choice;
while(1)
{
printf("\n\n\n1.character stuffing");
printf("\n\n2.exit");
printf("\n\n\nenter choice");
scanf("%d",&choice);
printf("%d",choice);
if(choice>2)
printf("\n\n invalid option....please renter");
switch(choice)
{
case 1:
charc();
break;
case 2:
exit(0);
}
}
```

```
}
void charc(void)
{
char c[50],d[50],t[50];
int i,m,j;
clrscr();
printf("enter the number of characters\n");
scanf("%d",&m);
printf("\n enter the characters\n");
for(i=0;i<m+1;i++)
 {
scanf("%c",&c[i]);
}
printf("\n original data\n");
for(i=0;i<m+1;i++)
 printf("%c",c[i]);
d[0]='d';
d[1]='l';
d[2]='e';
d[3]='s';
d[4]='t';
d[5]='x';
for(i=0,j=6;i<m+1;i++,j++)
 {
if((c[i]=='d'&&c[i+1]=='l'&& c[i+2]=='e'))
{
d[j]='d';
j++;
d[j]='l';
j++;
d[j]='e';
j++;
m=m+3;
}
d[j]=c[i];
}
m=m+6;
m++;
d[m]='d';
m++;
d[m]='l';
m++;
d[m]='e';
m++;
```

```
d[m]='e';
m++;
d[m]='t';
m++;
d[m]='x';
m++;
printf("\n\n transmitted data: \n");
for(i=0;i<m;i++)
 {
printf("%c",d[i]);
}
for(i=6,j=0;i<m-6;i++,j++)
 {
if(d[i]=='d'&&d[i+1]=='l'&&d[i+2]=='e'&&d[i+3]=='d'&&d[i+4]=='l'&&d[i+5]=='e')
i=i+3;
t[j]=d[i];
}
printf("\n\nreceived data:");
for(i=0;i<j;i++)
 {printf("%c",t[i]);
}
}
```

Output:

1.character stuffing

2.exit

enter choice1

enter the number of characters 9

enter the characters dleabcdef

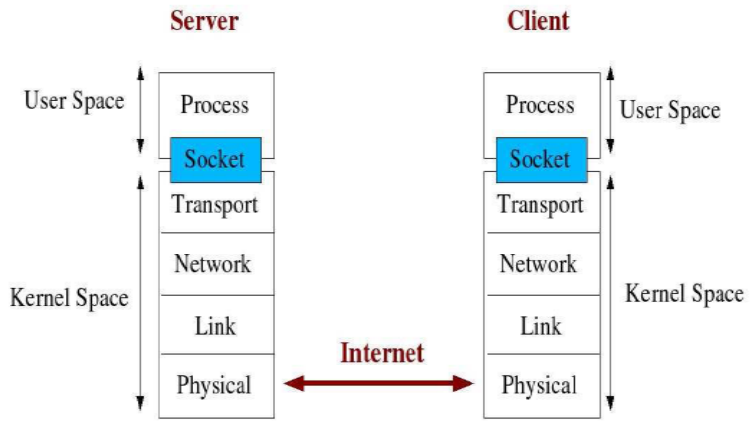
original data dleabcdef

transmitted data: dlestxdledleabcdefdleetx

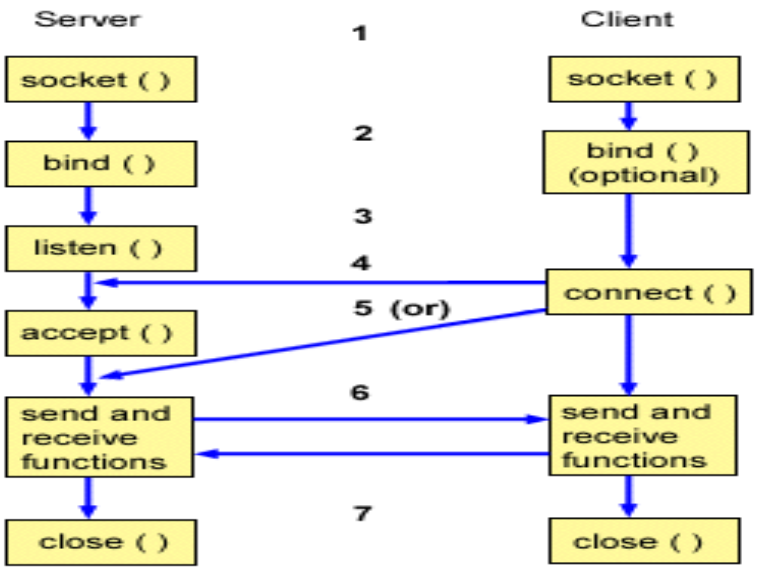
received data: dleabcdef

Program 6.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.

Sockets provide point-to-point, two-way communication between two processes. Sockets are very versatile and are a basic component of inter processes and intersystem communication. A socket is an endpoint of communication to which a name can be bound. It has a type and one or more associated processes. Socket may also defined as an interface between an application process and transport layer



The application process can send/receive messages to/from another application process (local or remote)via a socket.The diagram below shows the architecture of a connection oriented application.



- **Socket** primitive creates a new end point and allocates tavle space for it within the transport entity.

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

- **Bind** primitive is used by the server to establish a local address. Used to associate a socket with a port on the local machine._ The port number is used by the kernel to match an incoming packet to a process

int bind(intsockfd, structsockaddr *my_addr, intaddrlen)

- sockfd is the socket descriptor returned by socket()

- my_addr is pointer to structsockaddr that contains information

about your IP address and port

- addrlen is set to sizeof(structsockaddr)

returns -1 on error

my_addr.sin_port = 5000; //choose an unused port at random

my_addr.sin_addr.s_addr = INADDR_ANY; //use my IP addr

structsockaddr: Holds socket address information formany types of sockets

structsockaddr_in: A parallel structure that makes iteasy to reference elements of the socket address

structsockaddr

{

unsigned short sa_family; //address family AF_xxx

unsigned short sa_data[14]; //14 bytes of protocol addr

}

structsockaddr_in

{

shortintsin_family; // set to AF_INET

unsigned short intsin_port; // Port number

structin_addrsin_addr; // Internet address

unsigned char sin_zero[8]; //set to all zeros

}

- **Connect** prmitive is used by an application program to establish a connection before transfer of data takes place through a reliable stream socket or it connects to a remote host.

int connect(intsockfd, structsockaddr *serv_addr, intaddrlen)

- sockfd is the socket descriptor returned by socket()

- serv_addr is pointer to structsockaddr that contains information on destination IP address and port

- addrlen is set to sizeof(structsockaddr)

returns -1 on error

- **Listen** primitive allows the server to prepare a socket for incoming connections.

int listen(intsockfd, int backlog)

- sockfd is the socket file descriptor returned by socket()

- backlog is the number of connections allowed on the incoming queue

returns -1 on error

- **Accept** primitive is used basically by the server to wait for a connection request to arrive on the socket created or accept gets the pending connection on the port you are listening on.

int accept(intsockfd, void *addr, int *addrlen)

- sockfd is the listening socket descriptorinformation about incoming connection is stored in.

- addr which is a pointer to a local structsockaddr_in

- addrlen is set to sizeof(structsockaddr_in)

accept returns a new socket file descriptor to use for this accepted connection and -1 on error

- **Close** Closes connection corresponding to the socket descriptor and frees the socket descriptor.Will prevent any more sends and recvs

int close(intsockfd)

Header Filename: inet.h

```
#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>

#define SERV_TCP_PORT 6880

#define SERV_HOST_ADDR "127.0.0.1"
```

Program-Name: Client.c

```
#include"inet.h"

int main()
{
intsockfd;
structsockaddr_inserv_addr,cli_addr;
char filename[100],buf[1000];
int n;

serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=inet_addr(SERV_HOST_ADDR);
```

```
serv_addr.sin_port=htons(SERV_TCP_PORT);


if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
        {
printf("Client:cant open stream socket\n");
         exit(0);
        }
else
printf("Client:stream socket opened successfully\n");


if(connect(sockfd,(structsockaddr *)&serv_addr,
                        sizeof(serv_addr))<0)
       {
printf("Client:cant connect to server\n");
exit(0);
       }
else
printf("Client:connected to server successfully\n");


printf("\n Enter the file name to be displayed :");
scanf("%s",filename);
write(sockfd,filename,strlen(filename));
printf("\n filename transferred to server\n");
      n=read(sockfd,buf,1000);
buf[n]='\0';
printf("\n Client : Displaying file content of %s\n",filename);
fputs(buf,stdout);
close(sockfd);
```

```
exit(0);
}
```

Program-Name: Server.c

```
#include"inet.h"
int main()
{
intsockfd,newsockfd,clilen;
structsockaddr_incli_addr,serv_addr;
char filename[25],buf[1000];
intn,m,fd,size;

serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
serv_addr.sin_port=htons(SERV_TCP_PORT);


if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
        {
printf("Server:cant open stream socket\n");
        exit(0);
        }
else
printf("Server:stream socket opened successfully\n");


if((bind(sockfd,(structsockaddr *)                    &serv_addr,sizeof(serv_addr)))<0)
```

```
        {
printf("Server:cant bind local address\n");
exit(0);
        }
else
printf("Server:bind to local address\n");

listen(sockfd,5);
printf("\n SERVER : Waiting for client...\n");
clilen=sizeof(cli_addr);
newsockfd=accept(sockfd,(structsockaddr *)&cli_addr,&clilen);
if(newsockfd<0)
       {
printf("server:accept error\n");
exit(0);
       }
else
printf("Server: accepted\n");
            n=read(newsockfd,filename,25);
filename[n]='\0';
    printf("\n SERVER : %s is found and ready to transfer \n",filename);
fd=open(filename,O_RDONLY);
if(fd==-1)
       {
         write(newsockfd,"File doesn't exists",25);
         exit(0);
       }
```

```
size=lseek(fd,0,2);
lseek(fd,0,0);
      n=read(fd,buf,size);
buf[n]='\0';
write(newsockfd,buf,n);
printf("\n transfer success\n");
close(newsockfd);
exit(0);

}
```

Compile and run

- **open first terminal, compile and run server there**
- **open second terminal, compile and run client there**

```
$ cc –o serServer.c
$ ./ser
Server:stream socket opened successfully
Server:bind to local address
SERVER : Waiting for client...
Server: accepted
SERVER :vit is found and ready to transfer
transfer success

$ cc –o cli Client.c
$ ./cli
Client:stream socket opened successfully
Client:connected to server successfully
```

Enter the file name to be displayed :vit

filename transferred to server

Client : Displaying file content of vit

Welcome to vitcomputr science department.

NOTE : If the program has to be executed on the network say on two different machines then

- Change the ipddress in the header file(inet.h) from 127.0.0.1 to the ipaddress of the machine(first machine), where the server program has to be typed , compiled and executed.

Type the client program on the other machine(second machine) , compile it and then execute

PART B

SIMULATION

Setting up the environment

A user using the NCTUns in single machine mode, needs to do the following steps before he/she starts the GUI program:

Set up environment variables:

Before the user can run up the dispatcher, coordinator, or NCTUns GUI

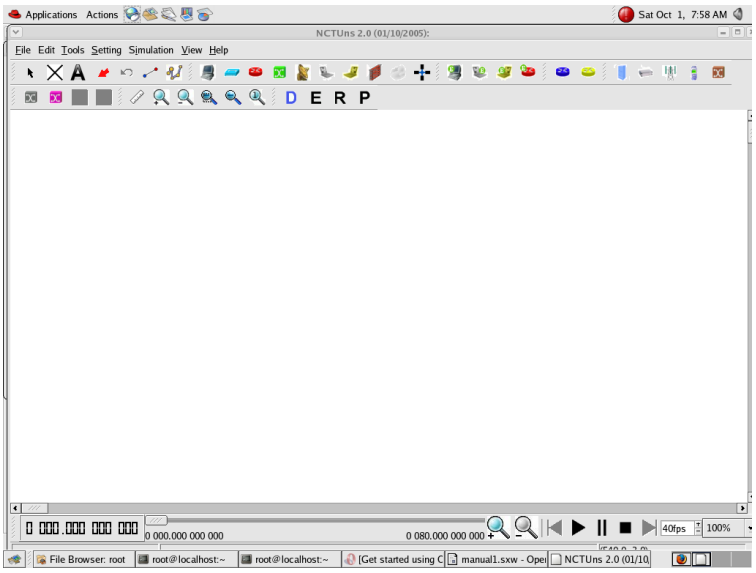
program he/she must set up the NCTUNSHOME environment variable.

Start up the dispatcher on terminal 1.

Start up the coordinator on terminal 2.

Start up the nctunsclient on terminal 3.

After the above steps are followed, the starting screen of NCTUns disappears and the user is presented with the working window as shown below:



Drawing A Network Topology

To draw a new network topology, a user can perform the following steps:

Choose Menu->File->Operating Mode-> and make sure that the “Draw Topology” mode is checked. This is the default mode of NCTUns when it is launched. It is only in this mode that a user can draw a new network topology or change an existing simulation topology. When a user switches the mode to the next mode “ Edit Property”, the simulation network topology can no longer be changed.

Move the cursor to the toolbar.

Left-Click the router icon on the toolbar.

Left-Click anywhere in the blank working area to add a router to the current network topology. In the same way we can add switch, hub,WLAN access point,WLAN mobile node , wall (wireless signal obstacle) etc.

Left-Click the host icon on the toolbar. Like in step 4, add the required number of hosts to the current topology.

To add links between the hosts and the router, left-click the link icon on the toolbar to select it.

Left-Click a host and hold the mouse button. Drag this link to the router and then release the mouse left button on top of the router. Now a link between the selected host and the router has been created.

Add the other, required number of links in the same way. This completes the creation of a simple network topology.

Save this network topology by choosing Menu->File->Save. It is saved with a

.tpl extension.

Take the snapshot of the above topology.

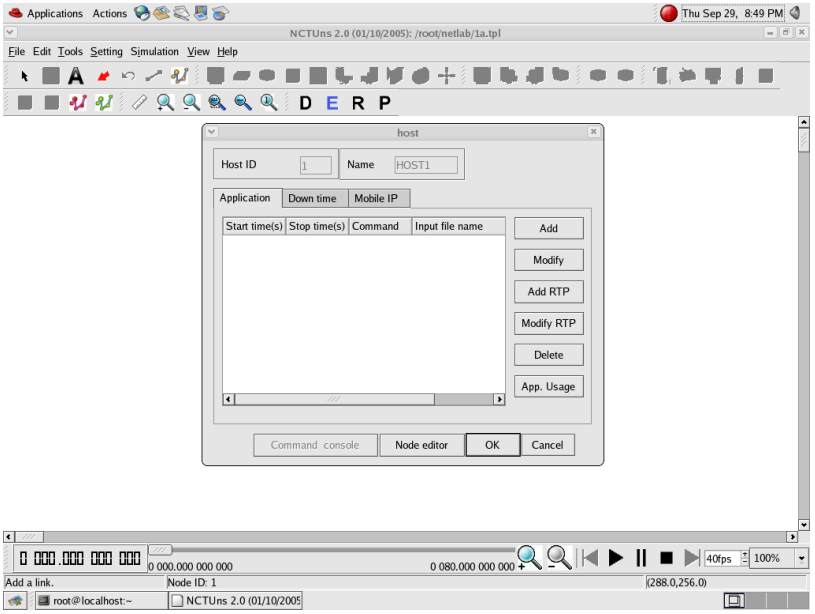
Editing Node's Properties

A network node (device) may have many parameters to set. For example, we may have to set the maximum bandwidth, maximum queue size etc to be used in a network interface. For another example, we may want to specify that some application programs (traffic generators) should be run on some hosts or routers to generate network traffic.

Before a user can start editing the properties of a node, he/she should switch the mode from the “Draw Topology” to “Edit Property” mode. In this mode, topology changes can no longer be made. That is, a user cannot add or delete nodes or links at this time.

The GUI automatically finds subnets in a network and generates and assigns IP and MAC addresses to layer 3 network interfaces.

A user should be aware that if he/she switches the mode back to the “Draw Topology” mode when he/she again switches the mode back to the “Edit Topology” mode, node's IP and MAC addresses will be regenerated and assigned to layer 3 interfaces. Therefore the application programs now may use wrong IP addresses to communicate with their partners.



Running the Simulation

1. When a user finishes editing the properties of network nodes and specifying application programs to be executed during a simulation, he/she can start running the simulation.

In order to do so, the user must switch mode explicitly from “Edit Property” to “Run Simulation”. Entering this mode indicates that no more changes can (should) be made to the

simulation case, which is reasonable. This simulation is about to be started at this moment; of course, any of its settings should be fixed.

Whenever the mode is switched to the " Run Simulation" mode, the many simulation files that collectively describe the simulation case will be exported. These simulation files will be transferred to the (either remote or local) simulation server for it to execute the simulation. These files are stored in the " main File Name.sim" directory, where main Filename is the name of the simulation case chosen in the "Draw Topology" mode.

Playing Back the Packet Animation Trace

After the simulation is finished, the simulation server will send back the simulation result files to the GUI program after receiving these files, the GUI program will store these files in the "results directory" .It will then automatically switch to "play back mode".

These files include a packet animation trace file and all performance log files that the user specifies to generate. Outputting these performance log files can be specified by checking some output options in some protocol modules in the node editor. In addition to this, application programs can generate their own data files.

3. The packet animation trace file can be replayed later by the packet animation player. The performance curve of these log files can be plotted by the performance monitor.

Post Analysis

1. When the user wants to review the simulation results of a simulation case that has been finished before, he /she can run up the GUI program again and then open the case's topology file

2. The user can switch the mode directly to the "Play Back" mode. The GUI program will then automatically reload the results (including the packet animation trace file and performance log file.

After the loading process is finished, the user can use the control buttons located at the bottom of the screen to view the animation.

Simulation Commands

The following explains the meaning of each job control command:

Run: Start to run the simulation.

Pause: Pause the currently -running simulation.

Continue: Continue the simulation that was just paused.

Stop: Stop the currently -running simulation

Abort: Abort the currently running simulation. The difference between “stop” and “abort” is that a stopped simulation job's partial results will be transferred back to GUI files.

Reconnect: The Reconnect command can be executed to reconnect to a simulation job that was previously disconnected. All disconnected jobs that have not finished their simulations or have finished their simulations but the results have not been retrieved back to be a GUI program by the user will appear in a session table next to the “Reconnect” command. When executing the reconnect command, a user can choose a disconnected job to reconnect from this session table.

Disconnect: Disconnect the GUI from the currently running simulation job. The GUI now can be used to service another simulation job. A disconnected simulation will be given a session name and stored in a session table.

EXPERIMENT 1

Simulate a three-node point-to-point network with a duplex link between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

STEPS:

Step1: Select the hub icon on the toolbar and drag it onto the working window.

Step2: Select the host icon on the toolbar and drag it onto the working window. Repeat this for another host icon.

Step 3: Select the link icon on the toolbar and drag it on the screen from host (node 1) to the hub and again from host(node 2) to the hub. Here the hub acts as node 3 in the point-to-point network. This leads to the creation of the 3-node point-to-point network topology. Save this topology as a .tpl file.

Step 4:Double-click on host(node 1), a host dialog box will open up. Click on Node editor and you can see the different layers- interface, ARP, FIFO, MAC, TCPDUMP, Physical layers. Select MAC and then select full- duplex for switches and routers and half duplex for hubs, and in log Statistics, select Number of Drop Packets, Number of Collisions, Throughput of incoming packets and Throughput of outgoing packets. Select FIFO and set the queue size to 50 and press OK. Then click on Add. Another dialog box pops up. Click on the Command box and type the Command according to the following syntax:

stg [-t duration(sec)] [-p port number]HostIPaddr

and click OK.

Step 5: Double-click on host (node 2), and follow the same step as above with only change in command according to the following syntax:

rtg [-t] [-w log] [-p port number]

and click OK.

Step 6: Double click on the link between node 1 and the hub to set the bandwidth to some initial value say, 10 Mbps. Repeat the same for the other node.

Step 7: Click on the E button (Edit Property) present on the toolbar in order to save the changes made to the topology. Now click on the R button (Run Simulation).

By doing so a user canrun/pause/continue/stop/abort/disconnect/reconnect/submit a simulation. No simulation settings can be changed in this mode.

Step 8: Now go to Menu->Simulation->Run. Executing this command will submit the current simulation job to one available simulation server managed by the dispatcher. When the simulation server is executing, the user will see the time knot at the bottom of the screen move. The time knot reflects the current virtual time (progress) of the simulation case.

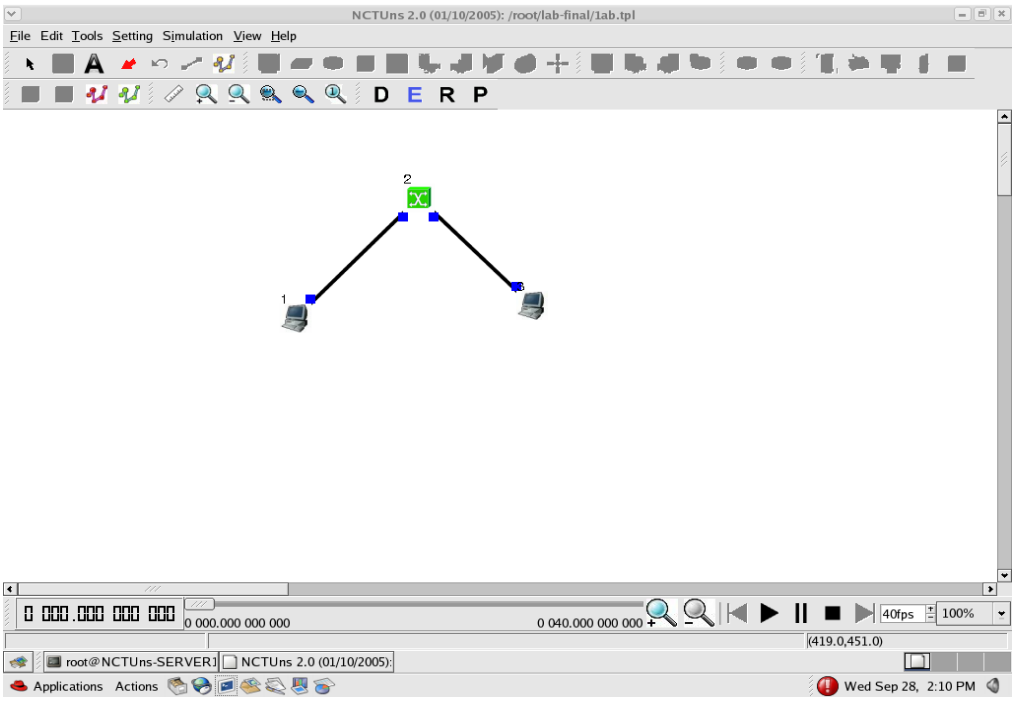
Step 9:To start the playback, the user can left-click the start icon(|>) of the time bar located at the bottom. The animation player will then start playing the recorded packet animation.

Step 10: Change the bandwidth say, 9 Mbps, and run the simulation and compare the two results.

Step 12: To view the results, go to the filename. results folder.

Note: To get the syntax of any command, double click on the host icon. Host dialog boxes appear and then choose App. Usage.

The screenshot below explain the topology.



EXAMPLE AND RESULTS OF EXPT 1

By using HUB:

Commands used: stg -u 1024 100 1.0.1.2

rtg -u -w log1

By setting the bandwidth as 10 Mbps on both the links and queue size as 50 weobtain the following results:

Output throughput n1-p1= 1177

Input throughput n3-p1 = 1177

Collision and drop = 0

By changing bandwidth to 9Mbps in the destination link, we obtain the following results:

Output throughput n1-p1 =1177

Input throughput n3-p1 = ~ 0

Collision and drop = 1100

EXPERIMENT 2

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

STEPS:

Step 1: Connect one set of hosts with a hub and another set of hosts also through a hub and connect these two hubs through a switch. This forms an Ethernet LAN.

Step 2: Setup a TCP connection between a host on one hub and host on another hub using the following command:

stcp [-p port] [-l writesize] hostIPaddr

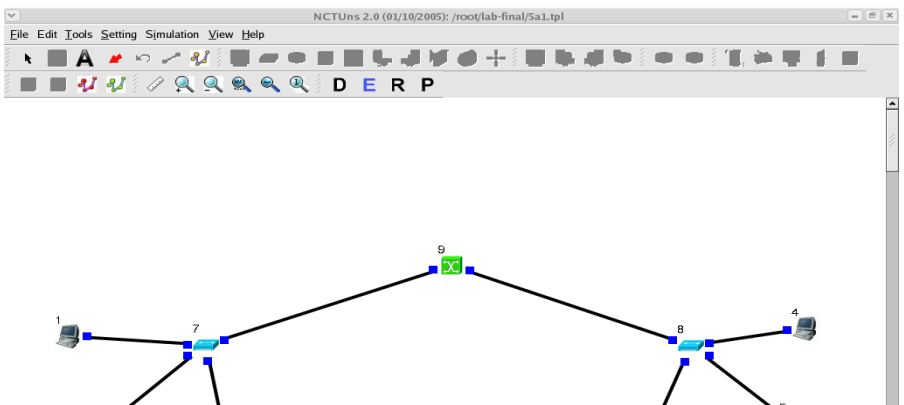
rtcp [-p port] [-l readsize]

Step 3: Setup the error rate, data rate in the physical layer, input and output throughput in the mac layer as described above.

Step 4:Change error rate and data rate and compare the throughputs.

Step 5: View the results in the filename.results.

The screenshot of the topology is shown below:



EXAMPLE AND RESULTS OF EXPT 2

Results:

Commands: stcp -p 7000 -l 1024 1.0.1.6

rtcp -p 7000 -l 1024

For first 6 nodes:

Initial error rate: 0.0

Initial data rate: 10 Mbps

Output Throughput: 654-1091

Input Throughput: 654-1091

Changed error rate: 1.0

Changed data rate: 10 Mbps

Output Throughput: 654-1091

Input Throughput: 654-1091

Error rate: 1.0

Data rate: 100 Mbps

Output Throughput: 1547-9765

Input Throughput: 1547-9765

For 6-10 nodes:

Initial error rate: 0

Initial data rate: 10 Mbps

Output Throughput: 654-1091

Input Throughput: 654-1091

Changed error rate: 1.0

Data rate: 10 Mbps

Output Throughput: 654-1091

Input Throughput: 654-1091

EXPERIMENT 3

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

STEPS:

Step 1: Connect one set of hosts with a hub and another set of hosts also through a hub and connect these two hubs through a switch. This forms an Ethernet LAN.

Step 2: Setup multiple traffic connections between the hosts on one hub and hosts on another hub using the following command:

stcp [-p port] [-l writesize] hostIPaddr

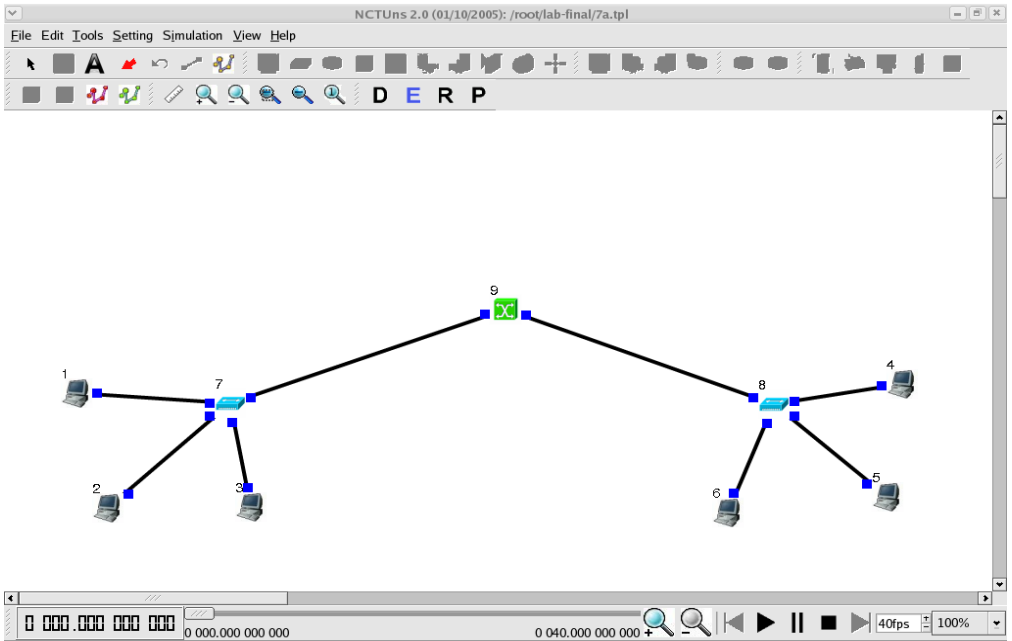
rtcp [-p port] [-l readsize]

Step 3: Setup the collision log at the destination hosts in the MAC layer as described in the earlier experiments.

Step 4: To plot the congestion window go to Menu->Tools->Plot Graph->File->open->filename.results->filename.coll.log

Step 5: View the results in the filename.results.

The screenshot of the topology is shown below:



EXAMPLE AND RESULTS OF EXPT 3

Results:

Commands: stcp -p 7000 -l 1024 1.0.1.6

rtcp -p 7000 -l 1024

Drops at node 5 : 324-750

Drops at node 4 : 274-930

EXPERIMENT 4

Simulate a four-node point-to-point network and connect the link as follows: Apply a TCP agent between n0 to n3 and apply a UDP agent between n1 and n3. Apply relevant applications over TCP and UDP agents changing the parameters and determine the number of packets sent by two agents.

STEPS:

Step 1: Create the topology as specified in the question, in the draw mode of the simulator.

Step 2: Go to edit mode and save the topology.

Step 3: Setup a TCP connection between node 1 and node 3 using the following commands:

stcp [-p port] [-l writesize] hostIPaddr

rtcp [-p port] [-l readsize]

Step 4: Setup a UDP connection between node 2 and node 3 using the following commands:

stg [-u payload size duration] [Hostaddress]

rtg [-u] [-w log]

Step 5:Set the output throughput log to determine the number of packets sent by TCP/UDP as described in experiment 1.

Step 6:To view the results, go to the filename.results folder.

The screenshot of the topology is shown below:

EXAMPLE AND RESULTS OF EXPT 4

Commands used: stg -u 1024 100 1.0.1.3

rtg -u -w log1

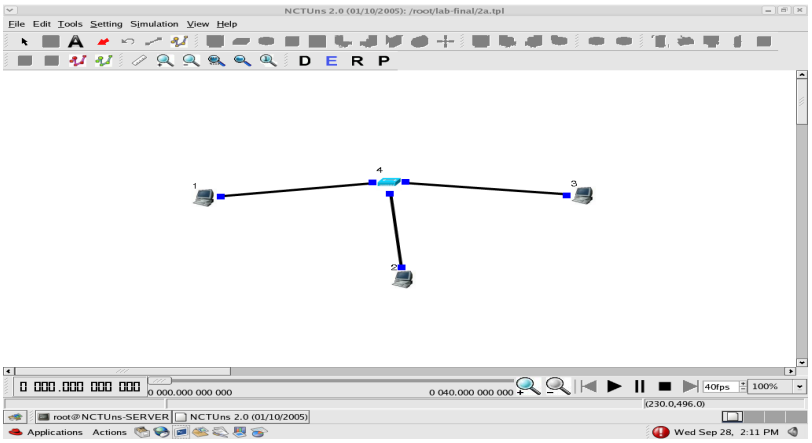
stcp -p 7000 -l 1024 1.0.1.3

rtcp -p 7000 -l 1024

Results: By setting the bandwidth as 100 Mbps on the TCP link and queue size as 50 we obtain the following results:

Average no: of TCP packets sent = varying (348 to 1100)

Average no: of UDP packets sent = 1180



EXPERIMENT 5

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

STEPS:

Step 1: Click on the subnet icon on the toolbar and then click on the screen of the working window.

Step 2: Select the required number of hosts and a suitable radius between the host and the switch.

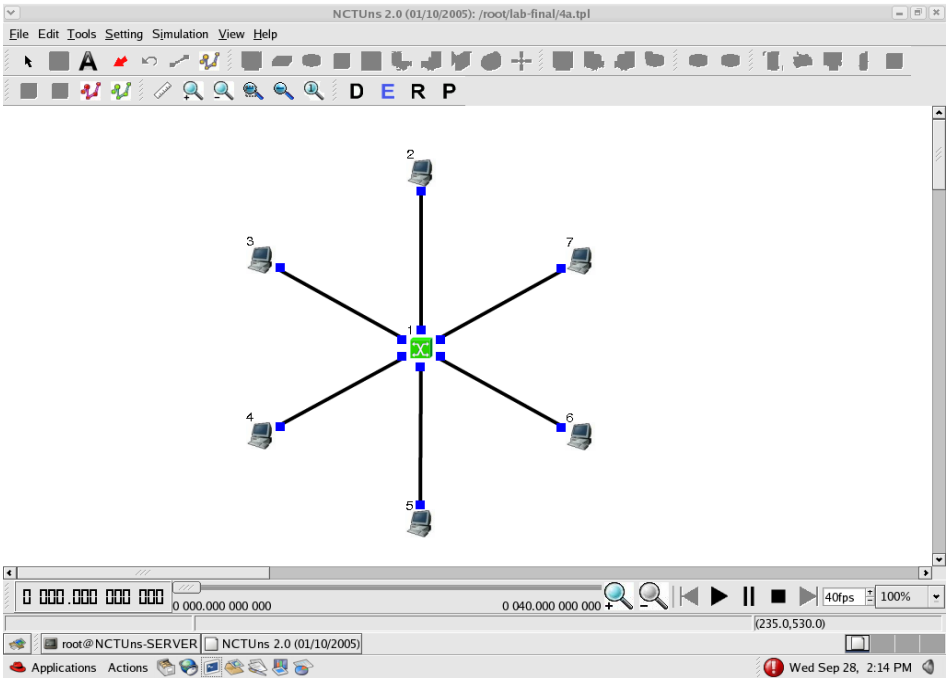
Step 3: In the edit mode, get the IP address of one of the hosts say, host 1 and then for the other host say, host2 set the drop packet and no: of collisions statistics as described in the earlier experiments.

Step 4: Now run the simulation.

Step 5: Now click on any one of the hosts and click on command console and ping the destination node.

Note: The no: of drop packets are obtained only when the traffic is more in the network.

The screenshot of the topology is shown below:



EXAMPLE AND RESULTS OF EXPT 5

Results:

No need to setup any commands on the node editor.

During Run mode open command console and ping 1.0.1.6.

EXPERIMENT 6

Simulate simple BSS and with transmitting nodes in wireless LAN by simulation and determine the performance with respect to transmission of packets.

STEPS:

Step 1: Connect a host and two WLAN access points to a router.

Step 2: Setup multiple mobile nodes around the two WLAN access points and set the path for each mobile node.

Step 3: Setup a ttcp connection between the mobile nodes and host using the following command:

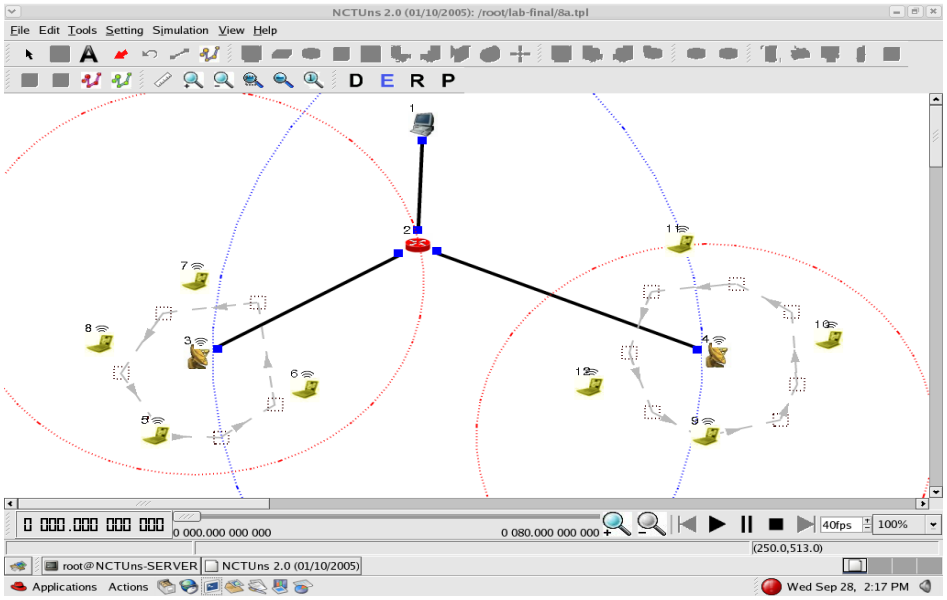
ttcp -r -u -s [-p port number]

Step 4: Setup the input throughput log at the destination host.

Step 5: To set the transmission range go to Menu->Settings->WLAN mobile node->Show transmission range.

Step 5:View the results in the filename. results.

The screenshot of the topology is shown below:



EXAMPLE AND RESULTS OF EXPT 6

Results:

Command: ttcp -r -u -s -p 7000

Output Throughput : 1190