RajaRajeswari College of Engineering

DEPARTMENT OF MCA

Affiliated to Visvesvaraya Technological University"Jnana Sangama", Belgaum – 560 018

COMPUTER NETWORKS LAB MANUAL

Prof. Shreedhar Kumbhar

"Subject Code: 22MCAL17"

Semester: I

Computer Networks Laboratory			
Course Code	**22MCAL17**	CIE Marks	50
Teaching Hours/Week (L:P: SDA)	0:3:0	SEE Marks	50
Credits	1.5	Exam Hours	03

Course objectives:

- To understand the working principle of various communication protocols.
- To understand the network simulator environment and visualize a network topology and observe its performance.
- To analyze the traffic flow and the contents of protocol frames.

Sl. NO	Experiments
1	Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.
2	Implement the data link layer framing methods such as character, character-stuffing and bit stuffing.
3	Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP
4	Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.
5	Implement Dijsktra's algorithm to compute the shortest path through a network
6	Implement data encryption and data decryption
7	Simulate the network with five nodes n0, n1, n2, n3, n4, forming a star topology. The node n4 is at the centre. Node n0 is a TCP source, which transmits packets to node n3 (a TCP sink) through the node n4. Node n1 is another traffic source, and sends UDP packets to node n2 through n4. The duration of the simulation time is 10 seconds.
8	Simulate to study transmission of packets over Ethernet LAN and determine the number of packets drop destination.
	Demonstration Experiments (For CIE) if any
9	Simulate the different types of internet traffic such as FTP and TELNET over a wired network and analyze the packet drop and packet delivery ratio in the network.

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Implement data link layer farming methods.
- Analyze error detection and error correction codes.
- Implement and analyze routing and congestion issues in network design.
- Implement Encoding and Decoding techniques used in presentation layer.
- To be able to work with different network tools.

Communication and Computer Networks Simulator (NS2)

1.0 INTRODUCTION

NS meaning Network Simulation is an open source network simulation tool or software, which was first, developed Steve McCanne in 1995-1997. NS (Version 2) is an object oriented, discrete event driven simulator written in C++ (core) and Otcl (python). NS was mainly developed in order for researchers to simulate various kinds of wired or wireless LAN and WAN thus implementing network protocols such as TCP and UDP. In addition NS enables researchers to study and understand traffic source behavior of FTP, VBR, CBR, Web and Telnet, not only that but also router queue management mechanism such as DropTail, RED and CBQ, routing algorithms such as Dijkstra, and many more.

NS2 was written in C++ and Otcl in order to differentiate the control and data path implementations. The simulator supports a class hierarchy in C++ (the compiled hierarchy) and a corresponding hierarchy within the Otcl interpreter (interpreted hierarchy). To understand more about the control and data path, take for instance; basically simulation of protocols requires efficient manipulation of bytes and packet headers thus making run-time speed very essential. On the other hand, in network studies where the aim is to vary some parameters and to quickly examine a number of scenarios the time to change the model and run it again is more important.

In NS2 for detailed protocol implementation (new queuing discipline), C++ is the best choice this is because every packet flowing must be processed and C++ is quiet fast. On the other hand Otcl is suitable for configuration and setup. Otcl runs a bit slow compared to C++, but it can be changed very quickly making the construction of simulations easier. Another important aspect is availability of the compiled C++ objects to the Otcl interpreter; this enables the objects to be controlled from the OTcl level.

A network simulation performance for a network topology that consist of 3 sources, a gateway and a receiver is analyzed using FTP and CBR applications connected over TCP and UDP agent

respectively. The benchmark for the simulation is done using the following queuing mechanisms namely; Droptail, Stochastic Fair Queuing (SFQ) and Random Early Detection (RED) in which the performance of this topology is analyzed. The report is organized as follows;

First, an introduction is carried out to give an overview on Network Simulation software (NS2) and the aim of this simulation; this is followed by the methodology which gives an insight on the steps carried out in developing the code script for the simulation, this is then followed by the source code and NAM trace file for TCP Scenario. Afterwards the results of the simulation are analyzed and a conclusion on the overall simulation is performed.

2.0 METHODOLOGY

This section will cover detail description on various sections of the code and their functions. The various code sections are organized as follows;

a. Creation of new simulator object

b. Trace file

c. Source creation

d. Color definition for different traffic flow

e. Link Creation between nodes

f. Queuing size definition

g. Topology layout

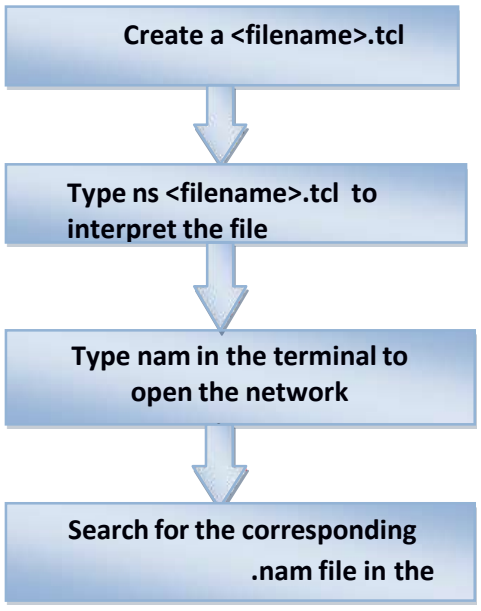
h. Monitoring of queues for the links

i. Creating TCP and UDP agent and attaching to nodes

j. Creating a TCP sink agents and attach them to node

k. Scheduling events for the FTP and CBR agents

NS2 Simulation Flow chart



A simulator object have functions that allow for the creation of nodes, links and connecting agents used in creating a network topology for the simulation in determining network performance . The class Simulator has all the basic functions to perform this task. Functions belonging to the simulator class can be used by appending "$ns" at the beginning of the command since the simulator object are design to be handled by ns. This is show below.

```
#Create a simulator object
set ns [new Simulator]
```

2. Trace file

Trace file is use in collecting data for calculating of simulation results. With the aid of NS2 trace file, drop or arrival of packet that occurs in queue or in a link are recorded. Below is the code that is use in creating a trace file and the nam trace file for the simulation.

```
#Open trace files

set f [open droptail-queue-out.tr w]
$ns trace-all $f

#Open the nam trace file
set nf [open droptail-queue-out.nam w]
$ns namtrace-all $nf
```

3. Source creation

The Simulator class has a member called "node" that is use to create five nodes which are assign to be handle s0, s1, s2, G, r. The handles are used when referring to those nodes. Traffic sources (FTP and CBR) are attached to the traffic agents (TCP and UDP) and these traffic agents are attached to the nodes if a node is not a router but an end system. Below is the code for creating nodes using the Simulator class "node".

```
#s1, s2 and s3 act as sources.
```

```
set s0 [$ns node]
set s1 [$ns node]
set s2 [$ns node]

#G acts as a gateway.
set G [$ns node]

#r acts as a receiver.
set r [$ns node]
```

2.4 Color definition for different traffic flow

In the simulation, there are two types of traffic used. These traffics can be differentiated from one another by assigning color to each. The color assigning is done using the calling the color function from the Simulator class using the commands below;

```
#Define different colors for data flows
$ns color 1 Yallow ;        # the color of packets from s0
$ns color 2 SeaGreen ;    # the color of packets from s1
$ns color 3 Blue ;        # the color of packets from s2
```

It could be seen that the packet sent from node s0 and s1 have the assigned color red because they use the same TCP traffic agent and that of node s2 has green color which uses a UDP traffic agent.

2.5 Link Creation between nodes

In other to complete the network topology, links are required to connect all the different nodes together. When creating a link in NS2, the output queue and the queue type of a node are implemented as part of the link. There are two types of links namely: simplex and duplex but in this simulation all links are design to be duplex. The command for creating a duplex or simplex

$ns duplex/simplex-link endpoint1 endpoint2 bandwidth delay queue-type

The code section below shows a duplex link connection between all the sources (S0, S1 and S2) and the gateway and from the gateway to the receiver using Drop tail queuing mechanism.

```
#Create links between the nodes
$ns duplex-link $s0 $G 6Mb 5ms DropTail
$ns duplex-link $s1 $G 6Mb 5ms DropTail
$ns duplex-link $s2 $G 6Mb 5ms DropTail
$ns duplex-link $G $r  3Mb 10ms DropTail
```

From the code section above, it could be seen that the duplex link connection the sources to the gateway have a bandwidth of 6Mb and a delay of 5ms. The duplex link between the gateway and the receiver has a bandwidth of 3Mb and delay of 10ms.

There are several queue management algorithms implemented in ns2, but in this exercise only DropTail, RED and SFQ queuing mechanism will be needed.

2.6 Queuing size definition

The Queuing size between the gateway and the receiver need to be define.

$ns queue-limit node1 node2 number:

The command above sets the queue limit of the two duplex links that connect node1 and node2 to the number specified. In this report, a queuing size of 5 is used between the gateway and the receiver. This is illustrated as shown in the code fragment below.

```
#Define the queue size for the link between node G (gateway) and r (receiver)
$ns queue-limit $G $r 5
```

2.7 Topology layout

For the simulation scenario in ns2 to run successfully, there is need to set up the layout of the network topology. This topology comprises of a collections of links and nodes. The code fragment for creating the topology using 5 nodes is shown below:

```
#Define the layout of the topology
$ns duplex-link-op $s0 $G orient right-down
$ns duplex-link-op $s1 $G orient right
$ns duplex-link-op $s2 $G orient right-up
$ns duplex-link-op $G $r orient right
```

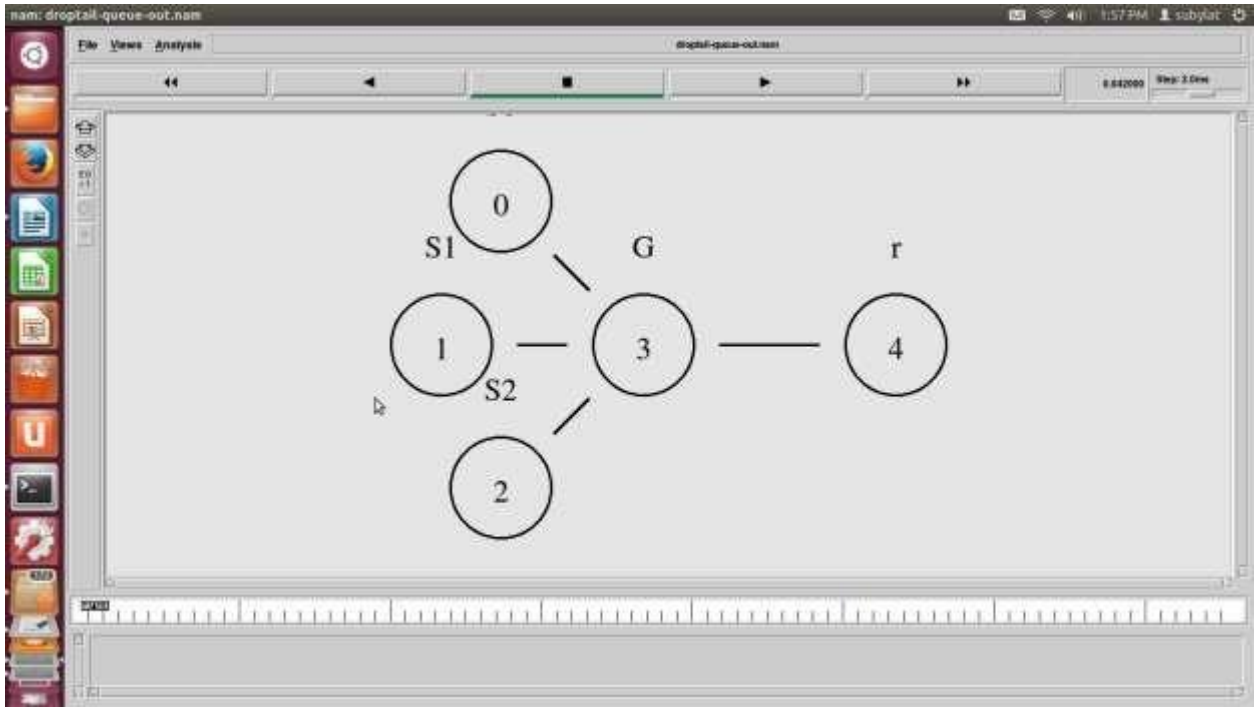
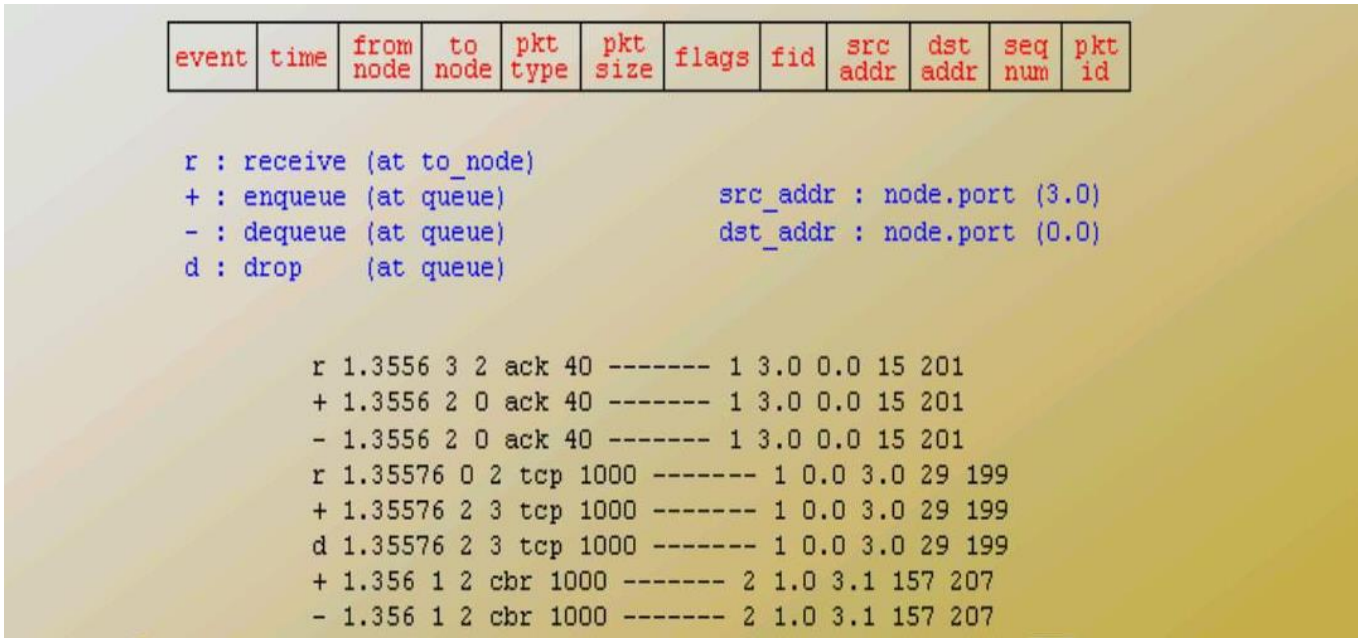


Figure 1: Network topology for simulation

AWK File Reference Table



event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop (at queue)

src_addr : node.port (3.0)
dst_addr : node.port (0.0)

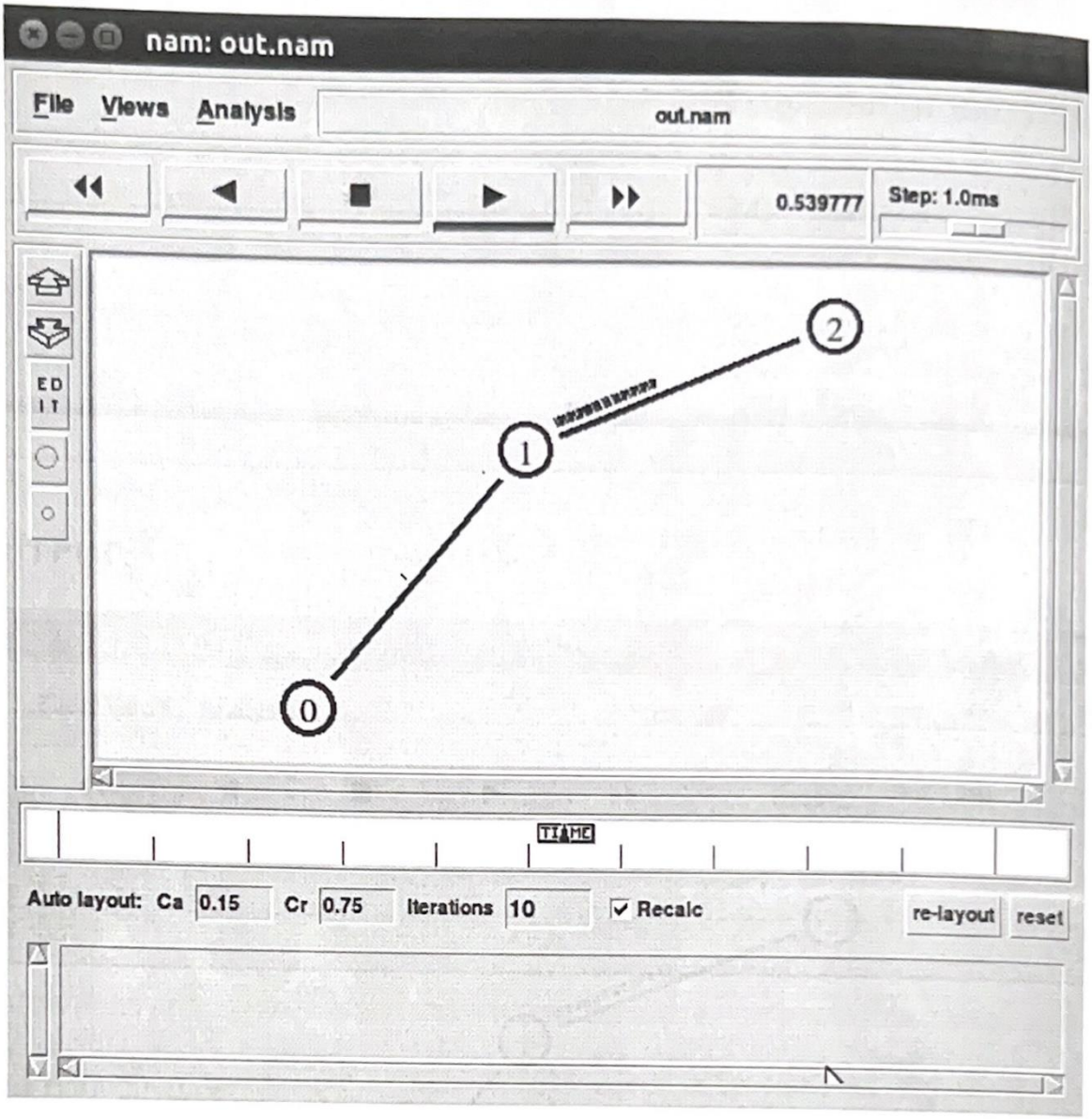
```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

1) Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidthnumber and find the of packets dropped.

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit 0
}
set n0 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 200Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 1000ms DropTail
$ns queue-limit $n0 $n2 10

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

$ns at 0.1 "$cbr0 start"
$ns at 1.0 "finish"
$ns run
```

```
//AWK file

BEGIN
{
    c=0;
   }
{
  if($1=="d")
  {
    c++;
    printf("%s\t%s\n",$5,$11);
  }
}
END{ printf("The number of packets dropped = %d\n",c);
}
```

OUTPUT:




```
saira@saira-Inspiron-3541:~/Desktop/MCA/Sem3/CN_lab$ awk -f pb1.awk out.tr
The number of packets dropped = 0
saira@saira-Inspiron-3541:~/Desktop/MCA/Sem3/CN_lab$
```

2) Implement the data link layer framing methods such as character, character-stuffing and bit stuffing.

Experiment No: 2

BIT STUFFING

AIM: Write a C/C++ program for bit stuffing and de-stuffing in HDLC frame format.

THEORY:

The new technique allows data frames to contain an arbitrary number if bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with special bit pattern, 01111110, called a flag byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

ALGORITHM for BIT STUFFING:

Step1: Input data sequence

Step 2: Add start of frame to output sequence

Step 3: For every bit in input
- a. Append bit to output sequence
- b. Is bit a 1?
  - **Yes:** Increment count,
    If count is 5, append 0 to output sequence and reset count.
  - **No:** Set count to 0

Step 4: Add stop of frame bits to output sequence.

ALGORITHM for BIT DESTUFFING:

Step 1: Input the stuffed sequence.

Step 2: Remove start of frame from sequence.

Step 3: For every bit in input,
- a. Append bit to output sequence.
- b. Is bit a 1?
  - Yes: Increment count. If count is 5, remove next bit (which is 0) & reset count.
  - No: Set count to 0.

Step 4: Remove end of frame from bits from sequence.

C- LANGUAGE PROGRAM CODE

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

void main() { char ch,array[50]={"01111110"},reed_array[50];
int counter=0,i=8,j,k;

// clrscr();
printf("Enter the original data stream for bit stuffing:\n");

while((ch=getchar())!='\n')
{ if(ch=='1')
  ++counter;
```

```
  else
  counter=0; array[i++]=ch;
  if(counter==5)
  {
    array[i++]='0';
          counter=0;
  }
}
 strcat(array,"01111110");
 printf("\n The stuffed data stream is:\n");
for(j=0;j<i+8;++j)
printf("%c",array[j]);
counter=0;
printf("\n The destuffed data stream is:\n");
for(j=8,k=0;j<i+8;++j)
 {
   if(array[j]=='1')
         ++counter;
         else
         counter=0;
         recd_array[k++]=array[j];
         if(counter==6)
         break;
         else if(counter==5 && array[j+1]=='0')
         {
          ++j;
          counter=0;
         }
 }
for(j=0;j<=k-strlen("01111110");++j)
printf("%c",recd_array[j]);
// getch();
}
```

OUTPUT:

```
Enter the original data stream for bit stuffing:
0011011111000011110

The stuffed data stream is:
01111110001101111100000111100111111 0

The destuffed data stream is:
0011011111000011110
```

CHARACTER STUFFING

AIM: Implement the data link layer framing using character-stuffing for a given character data.

THEORY:

The framing method gets around the problem of resynchronization after an error by having each frame start with the ASCII character sequence DLE. If the destination ever losses the track of the frame boundaries all it has to do is look for DLE characters to figure out. The data link layer on the receiving end removes the DLE before the data are given to the network layer. This technique is called character stuffing.

ALGORITHM:

Step 1: Start
Step 2 Read the character string to be transmitted in upper case.
Step 3: For stuffing process, append at begin with _DLE' as starting flag byte and end with _DLE' as ending flag byte of the string.
Step 4: Check the string whether it has _DLE'.
Step 5: If yes then insert the string _ESC' before the character DLE transmit the next character.
Step 6: Continue this process until the completion of string.
Step 7: Stuffed data is obtained.
Step 8: Now destuffing process, remove the appended string at start and end of string.
Step 9: Continue this process until the last character of string.
Step 10: If yes then remove the string _DLE' that is encountered first else transmit the data.
Step 11: Continue this process until the last character of string.
Step 12: Original data has been obtained.
Step 13 Stop.

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
// #include<conio.h>
#include<string.h>
 main()
{
 char a[50],s1[100]={"DLE"},s2[10];
 int i,j,l;
 // clrscr();
 printf("\n character stuffing and unstuffing program\n");
 printf("\n @ SENDER --\n");
 printf("\n enter the message to be sent:\n");
 gets(a);
 l=strlen(a); //string length
 for(i=0;i<l;i++)
```

```
{
 if((a[i]=='D'&&a[i+1]=='L'&&a[i+2]=='E')||
          (a[i]=='E'&&a[i+1]=='S'&&a[i+2]=='C'))
 {
  for(j=l+3;j>=i+3;j--)
   { a[j]=a[j-3];  //shifted data three position to the right
   }
  l=l+3;
  a[i]='E';
  a[i+1]='S';
  a[i+2]='C';
  i=i+5;
 }
}
 printf("\n message after character stuffing:\n");
 printf("%s\n",a);
 strcpy(s2,s1);
 strcat(s1,a);
 strcat(s1,s2);
 printf("\n the transmitted frame:\n");
 printf("%s\n",s1);
printf("\n----------------------------\n");
printf("\n @RECEIVER --\n");
 l=strlen(s1);    //lenght of flag+stuffed_message+flag
 s1[l-3]='\0';    //remove end delimiter(flag)
 l=strlen(s1);
 for(i=0;i<l;i++)
 s1[i]=s1[i+3];    //remove start delimiter(flag)
 l=strlen(s1);
 printf("\nmessage after flag removal at receiver:\n");
 printf("%s\n",s1);
 for(i=0;i<l;i++)
 {
  if(s1[i]=='E'&&s1[i+1]=='S'&&s1[i+2]=='C')
  {
   for(j=i;j<=l;j++)
    { s1[j]=s1[j+3]; //shifted data three position to the left
    }
   l=l-3;
   i=i+2;
  }
 }
 printf("\nmessage after unstuffing:\n");
 printf("%s\n",s1);
```

```
 // getch();
}
   }
```

RESULT: **Character stuffing and Unstuffing program**

@SENDER --
Enter the message to be sent:
ETXWITHSTXCANDLE

Message after Character stuffing:
ETXWITHSTXCANESCDLE
The transmitted frame:
DLE ETXWITHSTXCANESCDLE **DLE**

.....

@RECEIVER –

Message after flag removal at receiver:
ETXWITHSTXCANESCDLE

Message after Unstuffing :

ETXWITHSTXCANDLE

3) Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP

ERROR DETECTING CODE Using CRC-CCITT (16-bit)

AIM: *C Program for ERROR detecting code using CRC-CCITT (16bit).*

THEORY :

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The **CRC** calculation or *cyclic redundancy check* was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation—by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

```
                    1 0 1 = 5
                  ------------
1 0 0 1 1 / 1 1 0 1 1 0 1
            1 0 0 1 1 | |
            --------- | |
              1 0 0 0 0 |
              0 0 0 0 0 |
              --------- |
              1 0 0 0 0 1
                1 0 0 1 1
                ---------
                  1 1 1 0 = 14 = remainder
```

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "*Lammert*"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an *exclusive or* operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

- The message bits are appended with *c* zero bits; this *augmented message* is the dividend
- A predetermined *c+1*-bit binary sequence, called the *generator polynomial*, is the divisor
- The checksum is the *c*-bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

	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

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x)

This is used to generate a CRC = C(x) to be appended to M(x).

Note this G(x) is prime.

Steps: 1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).

2. Divide the result by G(x). The remainder = C(x).

   Special case: This won't work if bitstring =all zeros. We don't allow such an M(x).But M(x) bitstring = 1 will work, for example. Can divide 1101 into 1000.

3. If: x div y gives remainder c

   that means: x = n y + c , Hence (x-c) = n y

   (x-c) div y gives remainder 0

   Here (x-c) = (x+c)

   Hence (x+c) div y gives remainder 0

4. Transmit: T(x) = M(x) + C(x)
5. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

Note if G(x) has order n - highest power is x^n,

then G(x) will cover (n+1) bits

and the *remainder* will cover n bits. i.e. Add n bits (Zeros) to message.

Some CRC polynomials that are actually used

Some CRC polynomials

- CRC-8:
  x^8+x^2+x+1
  - Used in: 802.16 (along with error *correction*).
- CRC-CCITT:
  $x^{16}+x^{12}+x^5+1$
  - Used in: HDLC, SDLC, PPP default
- IBM-CRC-16 (ANSI):
  $x^{16}+x^{15}+x^2+1$
- 802.3:
  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
  - Used in: Ethernet, PPP rootion

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
int a[100],b[100],i,j,len,k,count=0;
//Generator Polynomial:g(x)=x^16+x^12+x^5+1
int gp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,};
int main()
{
        void div();
        system("clear");
        printf("\nEnter the length of Data Frame :");
        scanf("%d",&len);
        printf("\nEnter the Message :");
        for(i=0;i<len;i++)
        scanf("%d",&a[i]);
```

```
        //Append r(16) degree Zeros to Msg bits
        for(i=0;i<16;i++)
                a[len++]=0;
        //Xr.M(x)  (ie. Msg+16 Zeros)
        for(i=0;i<len;i++)
                b[i]=a[i];
        //No of times to be divided ie.Msg Length
        k=len-16;
        div();
        for(i=0;i<len;i++)
                b[i]=b[i]^a[i];  //MOD 2 Substraction
        printf("\nData to be transmitted : ");
        for(i=0;i<len;i++)
                printf("%2d",b[i]);
        printf("\n\nEnter the Reveived Data : ");
        for(i=0;i<len;i++)
                scanf("%d",&a[i]);
        div();
        for(i=0;i<len;i++)
                if(a[i]!=0)
                {
                        printf("\nERROR in Recived Data");
                        return 0;
                }
        printf("\nData Recived is ERROR FREE");
}
void div()
{
        for(i=0;i<k;i++)
        {
                if(a[i]==gp[0])
                {
                        for(j=i;j<17+i;j++)
                        a[j]=a[j]^gp[count++];
                }
                count=0;
        }
}
```

Output:

```
Enter the length of Data Frame :4
Enter the Message :1 0 1 1
Data to be transmitted :  1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Reveived Data : 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1
ERROR in Recived Data
Remender is : 0000000100000000
**********************************************************
```

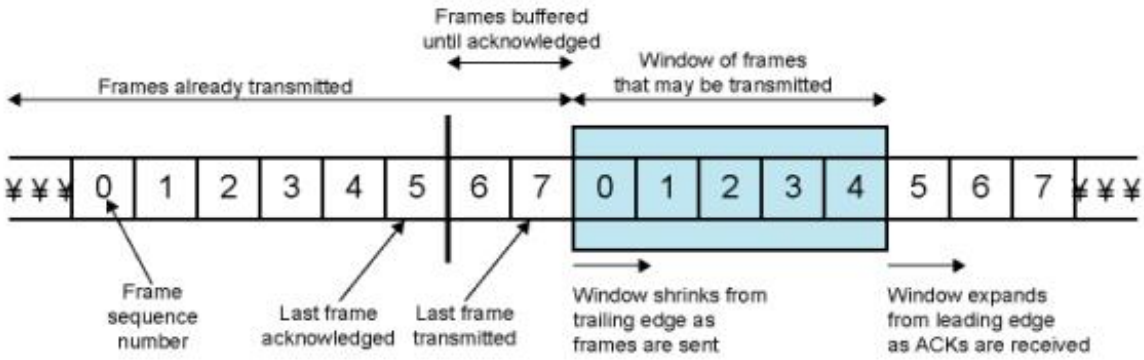
4) Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.

SLIDING WINDOW PROTPCOL

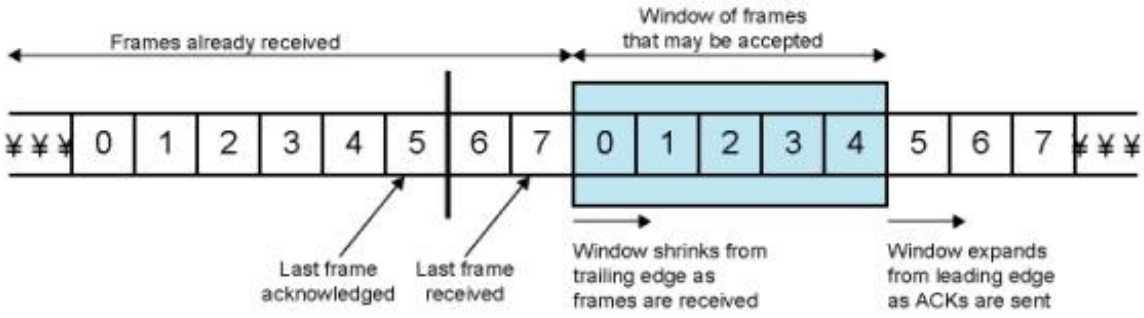
AIM: Implementation of Sliding Window Protocol using C.

THEORY:

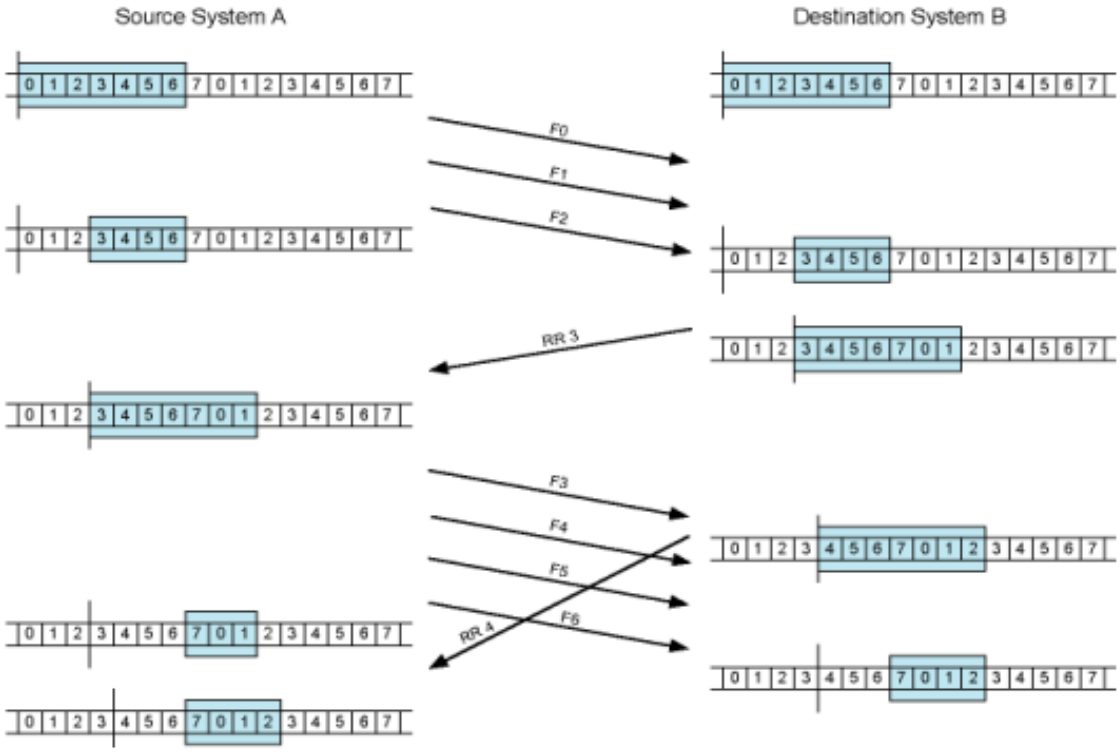
It allows multiple frames to be in transmit as compared to stop and wait protocol. In this the receiver has buffer of length W. Transmitter can send up to W frames without ACK. Each frame is numbered according to modular arithmetic. ACK includes number of next frame expected. Sequence number bounded by size of field (k).



(a) Sender's perspective



(b) Receiver's perspective



C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
#include<conio.h>
void main()
{
char sender[50],receiver[50];
int i,winsize;
// clrscr();
printf("\n ENTER THE WINDOWS SIZE : ");
scanf("%d",&winsize);
 printf("\n SENDER WINDOW IS EXPANDED TO STORE MESSAGE \n");
 printf("\n ENTER THE DATA TO BE SENT: ");
fflush(stdin);
gets(sender);
for(i=0;i<winsize;i++)
receiver[i]=sender[i];
receiver[i]=NULL;
 printf("\n MESSAGE SEND BY THE SENDER:\n");
 puts(sender);
 printf("\n WINDOW SIZE OF RECEIVER IS EXPANDED\n");
printf("\n ACKNOWLEDGEMENT FROM RECEIVER \n");
for(i=0;i<winsize;i++);
printf("\n ACK:%d",i);
printf("\n MESSAGE RECEIVED BY RECEIVER IS : ");
```

```
puts(receiver);
printf("\n WINDOW SIZE OF RECEIVER IS SHRINKED \n");
getch();
}
```

OUTPUT

ENTER THE WINDOW SIZE: 5

SENDER WINDOW IS EXPANDED TO STORE MESSAGE

ENTER THE DATA TO BE SENT: CITGUBBI

MESSAGE SEND BY THE SENDER: CITGUBBI

WINDOW SIZE OF RECEIVER IS EXPANDED

ACKNOWLEDGEMENT FROM RECEIVER

ACK: 5

MESSAGE RECEIVED BY RECEIVER IS: CITGU

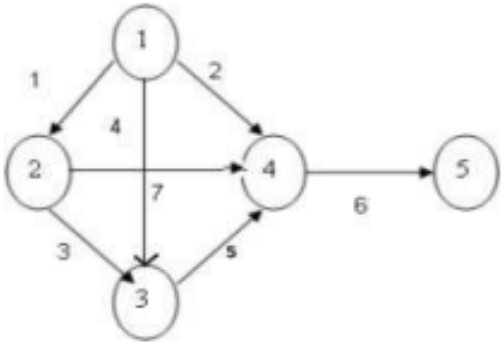
WINDOW SIZE OF RECEIVER IS SHRINKED

5) Implement Dijsktra's algorithm to compute the shortest path through a network.

SHORTEST PATH USING DIJKSTRA ALGORITHM

AIM: Write a C/C++ program to find the shortest path algorithm using Dijkstra's algorithm.

THEORY:



Dijkstra's algorithm progressively identifies the closest no Dijkstra's algorithm progressively identifies the closest nodes from the source node in order of increasing path cost. The algorithm is iterative. The Dijkstra's algorithm calculates the shortest path between two points on a network using a graph made up of nodes and edges The algorithm divides the nodes into two sets : tentative and permanent . It chooses nodes, makes them tentative, examines them and if they pass the criteria makes them permanent.

ALGORITHM:

Step1: Declare array path [5] [5], min, a [5][5], index, t[5];

Step2: Declare and initialize st=1,ed=5

Step 3: Declare variables i, j, stp, p, edp

Step 4: print –enter the cost

—Step 5: i=1

Step 6: Repeat step (7 to 11) until (i<=5)

Step 7: j=1

Step 8: repeat step (9 to 10) until (j<=5)

Step 9: Read a[i] [j]

Step 10: increment j

Step 11: increment i

Step 12: print –Enter the pathl

Step 13: read p

Step 14: print –Enter possible pathsl

Step 15: i=1

Step 16: repeat step(17 to 21) until (i<=p)

Step 17: j=1

Step 18: repeat step(19 to 20) until (i<=5)

Step 19: read path[i][j]

Step 20: increment j

Step 21: increment i

Step 22: j=1

Step 23: repeat step(24 to 34) until(i<=p)

Step 24: t[i]=0

Step 25: stp=st

Step 26: j=1

Step 27: repeat step(26 to 34) until(j<=5)

Step 28: edp=path[i][j+1]

Step 29: t[i]= [ti]+a[stp][edp]

Step 30: if (edp==ed) then

Step 31: break;

Step 32: else

Step 33: stp=edp

Step 34: end if

Step 35: min=t[st]

Step 36: index=st

Step 37: repeat step(38 to 41) until (i<=p)

Step 38: min>t[i]

Step 39: min=t[i]

Step 40: index=i

Step 41: end if

Step 42: printl minimum costl min

Step 43: printl minimum cost pthl

Step 44: repeat step(45 to 48) until (i<=5)

Step 45: print path[index][i]

Step 46: if(path[idex][i]==ed) then

Step 47: break

Step 48: end if

End

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
// #include<conio.h>
void main()
{
        int path[5][5], i, j, min, a[5][5], p, st=1,ed=5,stp,edp,t[5],index;
        // clrscr();
        printf("Enter the cost matrix\n");
        for(i=1;i<=5;i++)
        for(j=1;j<=5;j++)
        scanf("%d",&a[i][j]);
        printf("Enter the paths\n");
        scanf("%d",&p);
        printf("Enter possible paths\n");
        for(i=1;i<=p;i++)
        for(j=1;j<=5;j++)
        scanf("%d",&path[i][j]);
        for(i=1;i<=p;i++)
        {
                t[i]=0;
                stp=st;
                for(j=1;j<=5;j++)
                {
                        edp=path[i][j+1];
                        t[i]=t[i]+a[stp][edp];
                        if(edp==ed)
                        break;
                        else
                        stp=edp;
                }
        }
        min=t[st];index=st;
        for(i=1;i<=p;i++)
        {
                if(min>t[i])
                {
                        min=t[i];
                        index=i;
                }
        }
        printf("Minimum cost %d",min);
        printf("\n Minimum cost path ");
        for(i=1;i<=5;i++)
        {
```

```
            printf("--> %d",path[index][i]);
            if(path[index][i]==ed)
            break;
        }
        getch();
}
```

OUTPUT:

Enter the cost matrix

0 1 4 2 0
1 0 3 7 0
4 3 0 5 0
2 7 5 0 6
0 0 0 6 0

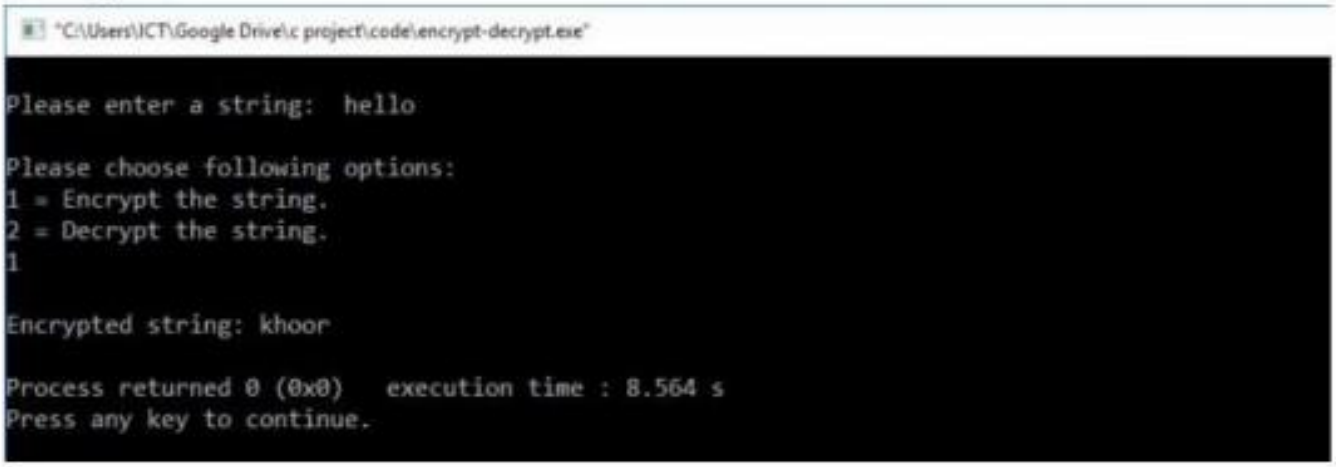
Enter the paths
4
Enter possible paths
1 2 3 4 5
1 2 4 5 0
1 3 4 5 0
1 4 5 0 0
Minimum cost 8
Minimum cost path → 1→ 4 →5

6) Implement data encryption and data decryption.

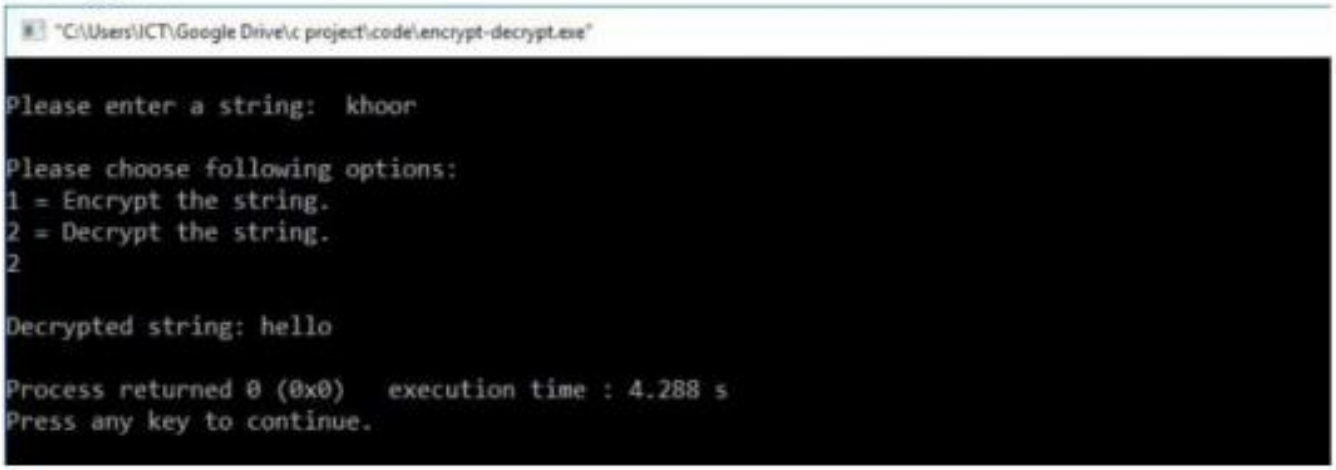
```
#include <stdio.h>
int main()
{
  int i, x;
  char str[100];
  printf("\nPlease enter a string:\t");
  gets(str);

  printf("\nPlease choose following options:\n");
  printf("1 = Encrypt the string.\n");
  printf("2 = Decrypt the string.\n");
  scanf("%d", &x);

  //using switch case statements
  switch(x)
  {
  case 1:
    for(i = 0; (i < 100 && str[i] != '\0'); i++)
      str[i] = str[i] + 3; //the key for encryption is 3 that is added to ASCII value

    printf("\nEncrypted string: %s\n", str);
    break;

  case 2:
    for(i = 0; (i < 100 && str[i] != '\0'); i++)
      str[i] = str[i] - 3; //the key for encryption is 3 that is subtracted to ASCII value

    printf("\nDecrypted string: %s\n", str);
    break;

  default:
    printf("\nError\n");
  }
  return 0;
}
```

Output

#Encryption



#Decryption



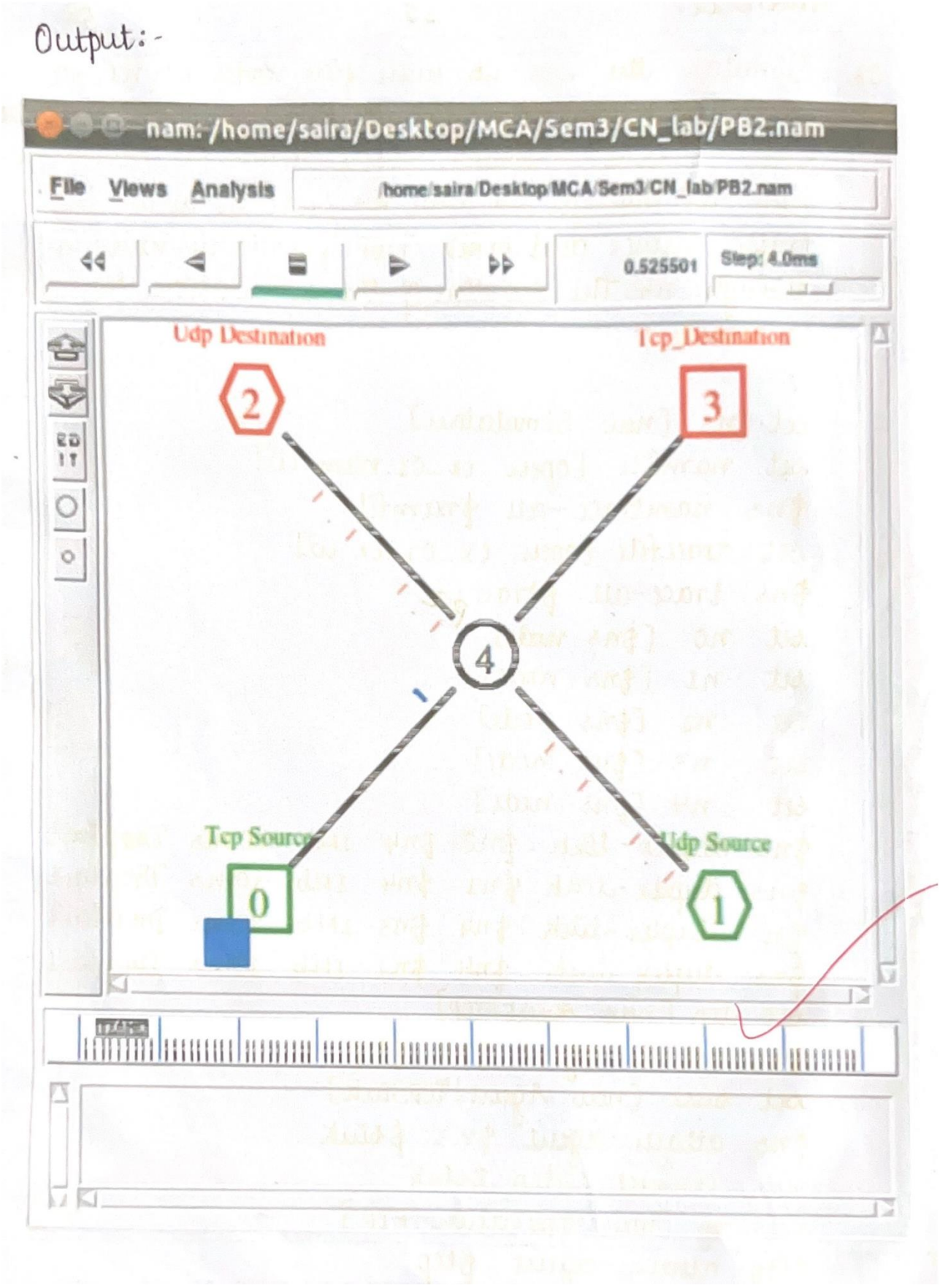
7) Simulate the network with five nodes n0,n1,n2,n3,n4, forming a star topology. The node n4 is at the center. Node n0 is a TCP source, which transmits packets to node n3 (a TCP sink) through the node n4. Node n1 is another traffic source and sends UDP packets to node n2 through n4. The duration of the simulation time is 10 seconds.

Simulate the network with five nodes n0, n1, n2, n3, n4 forming a star topology. The node n4 is at the center. Node n0 is a TCP source, which transmits packets to node n3 through the node n4. Node n1 is another traffic source and sends UDP packets to node n2 through n4. The duration of the simulation time is 10 seconds :-

```
set ns [new Simulator]
set namfile [open ex_01.nam w]
$ns namtrace-all $namfile
set tracefile [open ex_01.tr w]
$ns trace-all $tracefile
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n1 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n3 1Mb 10ms DropTail
$ns duplex-link $n4 $n2 1Mb 10ms DropTail
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n2 $null
$ns connect $udp $null
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 500
$cbr set interval_ 0.005
$cbr attach-agent $udp
$ns at 0.0 "$cbr start"
$ns at 0.0 "$ftp start"
$ns at 9.0 "$cbr stop"
$ns at 9.0 "$ftp stop"
proc finish {} {
global ns namfile tracefile
$ns flush-trace
close $namfile
close $tracefile
exec nam ex_01.nam &
exit 0
}
$ns at 10.0 "finish"
$ns run
```

Output:-


nam: /home/saira/Desktop/MCA/Sem3/CN_lab/PB2.nam
File
Views
Analysis
/home/saira/Desktop/MCA/Sem3/CN_lab/PB2.nam
0.525501
Step: 4.0ms
Udp Destination
Tcp_Destination
2
3
4
Tcp Source
Udp Source
0
1

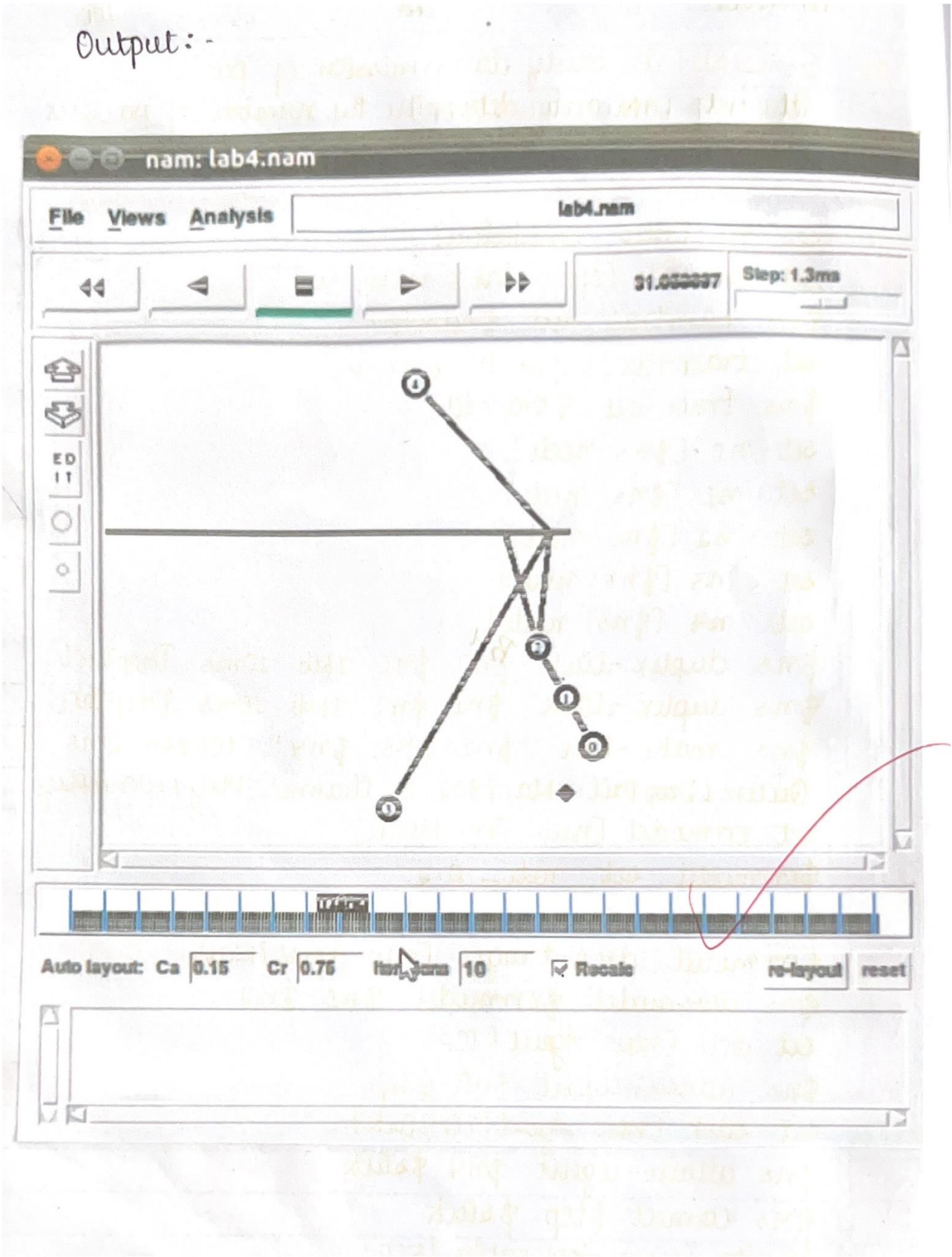
8) Simulate to study transmission of packets over Ethernet LAN and determine the number of packets drop destination.

Simulate to study transmission of packets over ethernet LAN and determine the number of packets drop destination:-

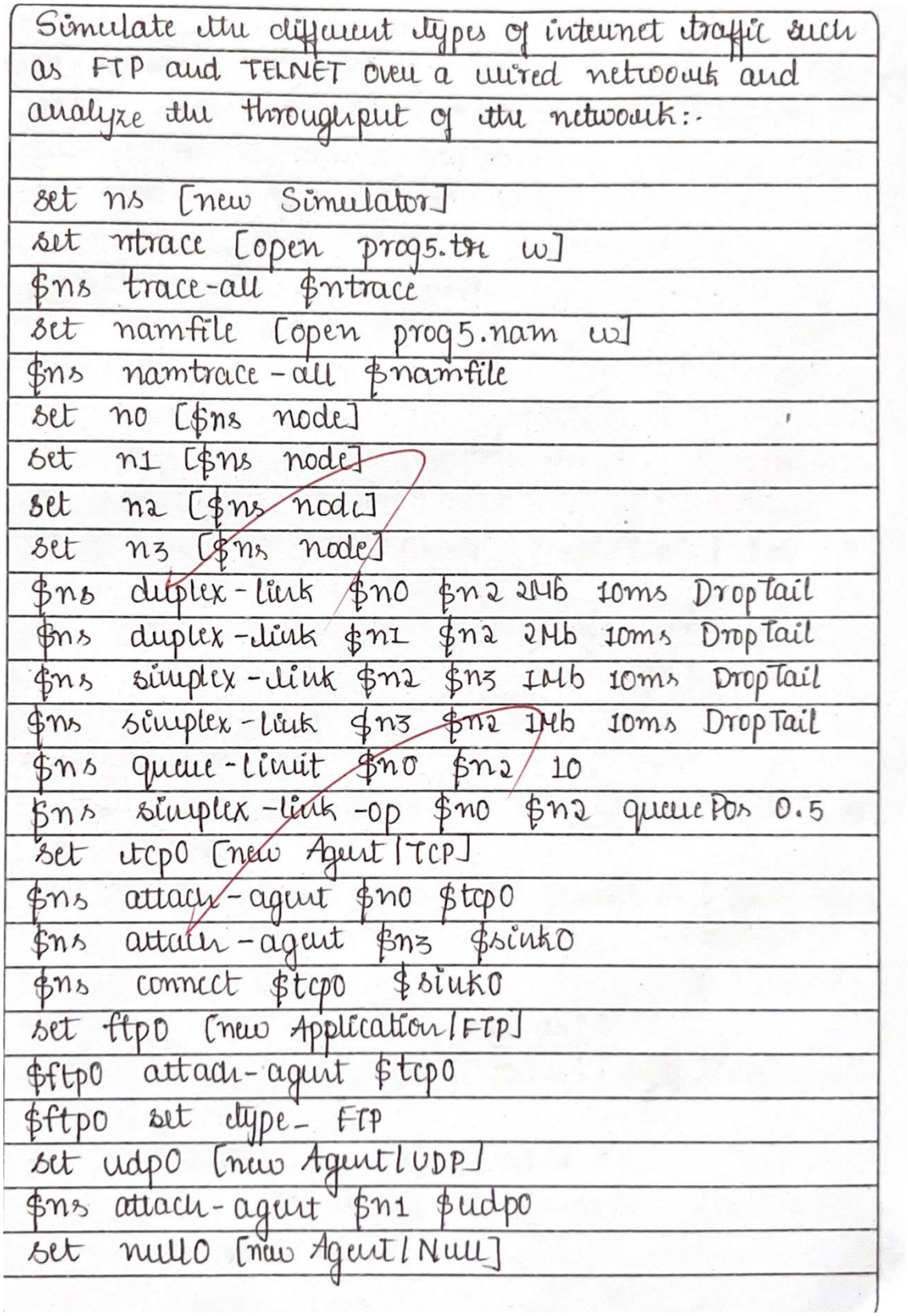
```
set ns [new Simulator]
set namfile [open lab3.nam w]
$ns namtrace-all $namfile
set tracefile [open lab3.tr w]
$ns trace-all $tracefile
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns make-lan "$n2 $n3 $n4" 100Mb 1ms LL
Queue/DropTail/Mac/802_3 Channel Phy/WiredPhy
set errmodel [new ErrorModel]
$errmodel set rate_ 0.2
$errmodel ranvar [new RandomVariable/Uniform]
$errmodel drop-target [new Agent/Null]
$ns lossmodel $errmodel $n1 $n2
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
set filesize [expr 4*1024*1024]
$ns at 0.0 "$ftp send $filesize"
proc finish { } {
global ns namfile tracefile
$ns flush-trace
close $namfile
close $tracefile
set awkCode {
BEGIN { }
{
if ($1 == "d" && $5 == "tcp" && $6 >1460) {
count_packets++;
puts $2, count_packets >> "lab3.data"
}
}
END { }
}
exec awk $awkCode lab3.tr
exec nam lab3.nam &
exec xgraph -bb -tk -x Time -y Dropped_Packets
 lab3.data -bg white &
exit 0
}
$ns at 100.0 "finish"
$ns run
```

Output:-


nam: lab4.nam
File
Views
Analysis
lab4.nam
Step: 1.3ms
ED
IT
Auto layout: Ca 0.15 Cr 0.75
Recalc
re-layout
reset

9) Simulate the different types of internet traffic such as FTP and TELNET over a wired network and analyze the packet drop and packet delivery ratio in the network.



```
Simulate the different types of internet traffic such
as FTP and TELNET over a wired network and
analyze the throughput of the network:-

set ns [new Simulator]
set ntrace [open prog5.tr w]
$ns trace-all $ntrace
set namfile [open prog5.nam w]
$ns namtrace-all $namfile
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 1Mb 10ms DropTail
$ns simplex-link $n3 $n2 1Mb 10ms DropTail
$ns queue-limit $n0 $n2 10
$ns simplex-link -op $n0 $n2 queuePos 0.5
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type- FTP
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
set null0 [new Agent/Null]
```

```
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0
set telnet [new Application/Telnet]
$telnet attach-agent $udp0
$telnet set type_ Telnet
proc Finish {} {
global ns ntrace namfile
$ns flush-trace
close $ntrace
close $namfile
exec nam prog5.nam &
set numTcp
[exec grep "^r" prog5.tr | grep "tcp" | tail -n
    1 | cut -d " " -f 6]
set TcpSize [exec grep "^r" prog5.tr | grep -c
                                    "tcp"]
set tcpTime 24.0
set numUdp
[exec grep "^r" prog5.tr | grep "udp" | tail -n
    1 | cut -d " " -f 6]
set UdpSize [exec grep "^r" prog5.tr | grep -c
                                    "udp" ]
set udpTime 23.9
puts "the throughput of FTP is"
puts "[expr ($numTcp * $TcpSize) / $tcpTime] bytes
    per second"
puts "the throughput of telnet is"
puts "[expr ($numUdp * $UdpSize) /$udpTime]
    bytes per second"
```

```
exit 0
}
$ns at 0.1 "$telnet start"
$ns at 0.5 "$ftp0 start"
$ns at 23.9 "$telnet stop"
$ns at 24.0 "$ftp0 stop"
$ns at 25.0 "Finish"
$ns run
```

Output:-

```
saira@saira-Inspiron-3541:~/Desktop/MCA/Sem3/CN_lab$ ns prog6.tcl
The throughput of FTP is
243706.66666666666 bytes per second
The throughput of Telnet is
1422.5941422594144 bytes per second
```