EXPERIMENT No. 01:
Construct a HDLC frame using following techniques
i. Bit stuffing
ii. Character stuffing

1.1 Objectives:

Construct a HDLC frame using following techniques

i. Bit stuffing
ii. Character stuffing

1.2 Apparatus Required:

Text Editor, Terminal Window

1.3 Pre-Requisite:

Data link layer, Connection oriented connection.

1.4 Introduction:

Bit stuffing refers to the insertion of one or more bits into a data transmission as a way to provide signalling information to a receiver. The receiver knows how to detect, remove or disregard the stuffed bits. In the data link layer of the Open Systems Interconnection model, a stream of bits is divided into more manageable units, or frames. Each frame contains the sending and receiving information to facilitate transmission. To separate the frames, an 8-bit flag byte is injected at the beginning and end of the sequence. This keeps the receiver from interpreting the flag as part of the transmitted information.

Character stuffing is commonly used in data communication protocols to ensure the receiving system correctly interprets the transmitted data. It helps frame the data so that the receiver can easily identify the start and end of each data frame. Implementing character stuffing involves adding special characters or sequences of characters to the original data before transmission.

1.5 Procedure:

To write C Program:

- Open a terminal and enter the command: **gedit**
- Type the C program and save the program with **filename.c**
- Close the text editor

To Compile the program:

- Go to terminal and type the command: **cc filename.c**

To Check the output:

- By default, output will be stored in "a.out" file.
- In terminal type the command: **./a.out**

1.6 Observations:

i. Bit stuffing

```
#include<stdio.h>
#include<string.h>
int si,di;
char src[30],dst[30],flag[30]="01111110";
int main()
{
  di=strlen(flag);
  printf("Enter message string\n");
  scanf("%s",&src);
  strcpy(dst,flag);
  while(src[si]!='\0')
   if(src[si]=='1' && src[si+1]=='1' && src[si+2]=='1' && src[si+3]=='1' &&src[si+4]=='1')
        {
          dst[di]='1', dst[di+1]='1', dst[di+2]='1', dst[di+3]='1', dst[di+4]='1', dst[di+5]='0';
          di+=6; si+=5;
        }
  else
          dst[di++] =src[si++];
  dst[di++] ='\0';
  printf("string is %s \n stuffed string is %s \n",src,strcat(dst,flag));
}
```

ii. Character stuffing

```
#include<stdio.h>
#include<string.h>
int si,di;
char src[30],dst[30],flag1[30]="DLESTX",flag2[30]="DLEETX";
int main()
{
  di=strlen(flag1);
  printf("Enter message string\n");
  scanf("%s",&src);
  strcpy(dst,flag1);
  while(src[si]!='\0')
        if(src[si]=='D'&&src[si+1]=='L'&&src[si+2]=='E')
        {
              dst[di]='D' , dst[di+1]='L' , dst[di+2]='E' , dst[di+3]='D' ,
              dst[di+4]='L' , dst[di+5]='E';
              di+=6; si+=3;
        }
        else
              dst[di++]=src[si++];
  dst[di++]='\0';
  printf("string is %s \n stuffed string is %s \n",src,strcat(dst,flag2));
}
```

1.7 Results:

i. Enter message string: 001111111100

String is: 001111111100

Stuffed string is: 01111110001111101110001111110

ii. Enter message string: LADLE

String is: LADLE

Stuffed string is: DLESTXLADLEDLEDLEETX

1.8 Discussions:

1.9 Pre – Experimentation Questions:

1. Define DLC.
2. Name the services of DLC layer.
3. What is framing?
4. What is error control?
5. What are the layers of TCP/IP suit?

1.10 Post – Experimentation Questions:

1. What is bit and byte stuffing?
2. Differentiate between bit stuffing & character stuffing?
3. Advantages of bit stuffing?
4. Advantages of character stuffing?
5. How HDLC frame format is?

EXPERIMENT No. 02:
Construct a packet from HDLC frame using following techniques
i. Bit Unstuffing
ii. Character Unstuffing

2.1 Objectives:

Construct a packet from HDLC frame using following techniques

i. Bit Unstuffing

ii. Character Unstuffing

2.2 Apparatus Required:

Text Editor, Terminal Window

2.3 Pre-Requisite:

Data link layer, Connection oriented connection

2.4 Introduction:

HDLC is based on IBM's SDLC protocol, which is the layer 2 protocol for IBM's Systems Network Architecture (SNA). It was extended and standardized by the ITU as LAP (Link Access Procedure), while ANSI named their essentially identical version ADCCP.

The HDLC specification does not specify the full semantics of the frame fields. This allows other fully compliant standards to be derived from it, and derivatives have since appeared in innumerable standards. It was adopted into the X.25 protocol stack as LAPB, into the V.42 protocol as LAPM, into the Frame Relay protocol stack as LAPF and into the ISDN protocol stack as LAPD.

The original ISO standards for HDLC are the following:

ISO 3309-1979 – Frame Structure

ISO 4335-1979 – Elements of Procedure

ISO 6159-1980 – Unbalanced Classes of Procedure

ISO 6256-1981 – Balanced Classes of Procedure

ISO/IEC 13239:2002, the current standard, replaced all of these specifications.

HDLC was the inspiration for the IEEE 802.2 LLC protocol, and it is the basis for the framing mechanism used with the PPP on synchronous lines, as used by many servers to connect to a WAN, most commonly the Internet.

A similar version is used as the control channel for E-carrier (E1) and SONET multichannel telephone lines. Cisco HDLC uses low-level HDLC framing techniques but adds a protocol field to the standard HDLC header.

Character unstuffing means removing special characters or sequences of characters them at the receiving end to extract the original data.

In bit unstuffing the extra stuffed bit is eventually removed from the data by the receiver.

2.5 Procedure:

To write C Program:

- Open a terminal and enter the command: **gedit**
- Type the C program and save the program with **filename.c**
- Close the text editor

To Compile the program:

- Go to terminal and type the command: **cc filename.c**

To Check the output:

- By default, output will be stored in “a.out” file.
- In terminal type the command: **./a.out**

2.6 Observations:

i. Bit unstuffing

```
#include<stdio.h>
int si,di;
char src[50],dst[50];
int main()
{
  printf("enter input frame(0's & 1's only):\n");
        scanf("%s",&src);
  while(src[si]!='\0')
  {
```

```
        if(src[si]=='0' && src[si+1]=='1' && src[si+2]=='1' && src[si+3]=='1' &&
        src[si+4]=='1' && src[si+5]=='1' && src[si+6]=='1' && src[si+7]=='0')
        {
           si+=8;
        }
        else if(src[si]=='1' && src[si+1]=='1' && src[si+2]=='1' && src[si+3]=='1'
        && src[si+4]=='1' && src[si+5]=='0')
        {
          dst[di]='1',dst[di+1]='1',dst[di+2]='1',dst[di+3]='1',dst[di+4]='1';
          si+=6; di+=5;
        }
        else
           dst[di++]=src[si++];
  }
  printf("After unstuffing the frame is %s\n",dst);
  return 0;
}
```

ii. Character unstuffing

```
#include<stdio.h>
char src[50],dst[50];
int si,di;
int main()
{
  printf("enter input frame:\n");
  scanf("%s",&src);
  while(src[si]!='\0')
  {
  if(src[si]=='D' && src[si+1]=='L' && src[si+2]=='E' && src[si+3]=='S' &&
  src[si+4]=='T' && src[si+5]=='X' )
        {
           si=si+6;
        }
  else if(src[si]=='D' && src[si+1]=='L' && src[si+2]=='E' && src[si+3]=='E' &&
  src[si+4]=='T' && src[si+5]=='X' )
```

```
            {
               si=si+6;
            }
    else if(src[si]=='D' && src[si+1]=='L' && src[si+2]=='E' && src[si+3]=='D' &&
    src[si+4]=='L' && src[si+5]=='E' )
            {
               si=si+6;
               dst[di]='D', dst[di+1]='L', dst[di+2]='E';
               di=di+3;
            }
    else
            {
               dst[di++]=src[si++];
            }
    printf("%d\t%d\t%s\n",si,di,dst);
    }
    printf("After unstuffing the frame is %s\n",dst);
    return 0;
}
```

2.7 Results:

i. Enter input frame: 01111110001111101110001111110

After unstuffed the frame is 001111111100

ii. Enter input frame: DLESTXLADLEDLEDLEETX

After unstuffed the frame is LADLE

2.8 Discussions:

2.9 Pre – Experimentation Questions:

1. What is starting and ending flags, with bit stuffing framing method?
2. What is Physical layer coding violations framing method?
3. What is an error control?
4. What is error detection?
5. What is an error correction?

2.10 Post – Experimentation Questions:

1. Difference between byte stuffing and bit stuffing
2. Draw a placement of the data link protocol?
3. How a frame is created from a packet?
4. What is a character count framing method?
5. What are flag bytes with byte stuffing framing method?

EXPERIMENT No. 03:

Implement Ethernet LAN using n (6-10) nodes.

3.1 Objectives:

- Implement Ethernet LAN using n (6-10) nodes.
- Compare the throughput by changing the error rate and data rate.

3.2 Apparatus Required:

NS2

3.3 Pre-Requisite:

Ethernet LAN, error rate, data rate, throughput.

3.4 Introduction:

Ethernet provides a connectionless service, which means each frame sent is independent of the previous or next frame. Ethernet has no connection establishment or connection termination phases. The sender sends a frame whenever it has it; the receiver may or may not be ready for it. The sender may overwhelm the receiver with frames, which may result in dropping frames. If a frame drops, the sender will not know about it. Since IP, which is using the service of Ethernet, is also connectionless, it will not know about it either. If the transport layer is also a connectionless protocol, such as UDP, the frame is lost and salvation may only come from the application layer. However, if the transport layer is TCP, the sender TCP does not receive acknowledgment for its segment and sends it again. Ethernet is also unreliable like IP and UDP. If a frame is corrupted during transmission and the receiver finds out about the corruption, which has a high level of probability of happening because of the CRC-32, the receiver drops the frame silently. It is the duty of high-level protocols to find out about it. The bit error rate or bit error ratio is the number of bit errors divided by the total number of transferred bits during a studied time interval i.e. BER=Bit errors/Total number of bits.

3.5 Procedure:

1. Open terminal.
2. Check whether the necessary folders are available in the location by typing **ls**
3. If not available change the directory by typing **cd ..**
4. Type **java -jar nsg2.1.jar** in the terminal (A new blue window opens)
5. In new window
   a. Click on Scenario ------> select new wired scenario (for wired connection) or new wireless scenario (for wireless connection)
   b. Click on node —> select single -----> place as many nodes as needed in order
   c. Click on Link —> select duplex link -----> connect the nodes
   d. Click on Agent —> For Connection oriented network attach TCP to source node and TCP Sink to destination node. Similarly, for Connectionless network attach UDP to source node and Null to destination node Click source and destination and do logical connection
   e. Click on Application —> For TCP attach FTP and for UDP attach CBR
   f. Click on Parameter --->Set Simulation time=10.0, Trace and Nam file name should be same ----> click Done
   g. Click on TCL ---> do the necessary changes ---> Save with same filename as before with file type TCL files ---> close all the windows
6. In terminal type **ns filename.tcl**
   a. NAM file opens to show animation of the network. Click on play button, observe packet transfer ---> close all the windows
7. In terminal type **gedit filename.awk**
   a. Type awk program in an editable window, save and close
8. In terminal type **awk -f filename.awk filename.tr** ---> Required output gets displayed on the terminal
9. For alternative cases type **gedit filename.tcl** in terminal ---> Change respective parameter
10. Repeat Step 6
11. Repeat Step 8 ----> Note down the values in a table
12. Trace file can be opened by typing a command **gedit filename.tr** on terminal

3.6 Observations:

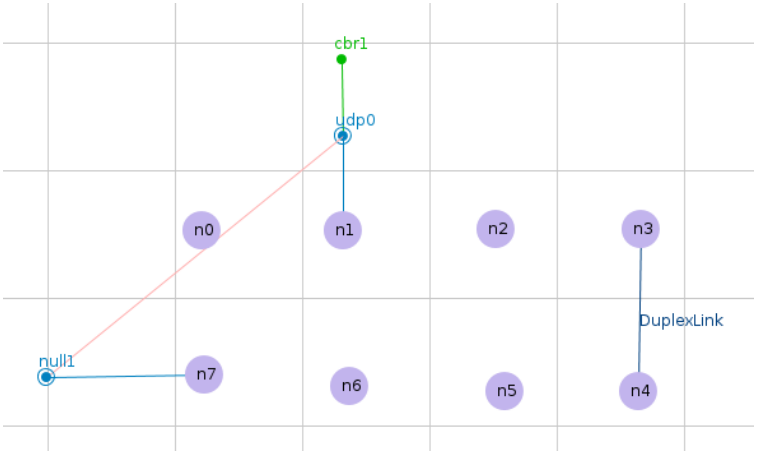


Fig. 3.1 Topology

Program:

```
#=====================================
#     Simulation parameters setup
#=====================================
set val(stop) 99.0                ;# time of simulation end
#=====================================
#        Initialization
#=====================================
#Create a ns simulator
set ns [new Simulator]
#Open the NS trace file
set tracefile [open CN3.tr w]
$ns trace-all $tracefile
#Open the NAM trace file
set namfile [open CN3.nam w]
$ns namtrace-all $namfile
#=====================================
#        Nodes Definition
#=====================================
#Create 8 nodes
set n0 [$ns node]
set n1 [$ns node]
```

```
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
$n1 label "Source/UDP"
$n3 label "Error Node"
$n7 label "Destination"
$ns make-lan "$n0 $n1 $n2 $n3" 100Mb 10ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6 $n7" 100Mb 10ms LL Queue/DropTail Mac/802_3
#===================================
#       Links Definition
#===================================
#Createlinks between nodes
$ns duplex-link $n3 $n4 100.0Mb 10ms DropTail
$ns queue-limit $n3 $n4 50
#Give node position (for NAM)
$ns duplex-link-op $n3 $n4 orient left-down
set err [new ErrorModel]
$ns lossmodel $err $n3 $n4
$err set rate_ 0.3
#===================================
#       Agents Definition
#===================================
#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
set null1 [new Agent/Null]
$ns attach-agent $n7 $null1
$ns connect $udp0 $null1
$udp0 set packetSize_ 1500
#Setup a CBR Application over UDP connection
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
```

```
$cbr0 set packetSize_ 1000
$cbr0 set rate_ 1.0Mb
$cbr0 set random_ null
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr0 stop"
#Define a 'finish' procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam CN3.nam &
    exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\"; $ns halt"
$ns run
```

AWK File

```
BEGIN{
        tcppack=0
        tcppack1=0
        }
        {
        if($1=="r"&&$4=="7"&&$5=="cbr")
        {
        tcppack++;
        }
        }
  END{
        printf("\n total number of  data packets at Node 7: %d\n", tcppack++);

        }
```

3.7 Results:

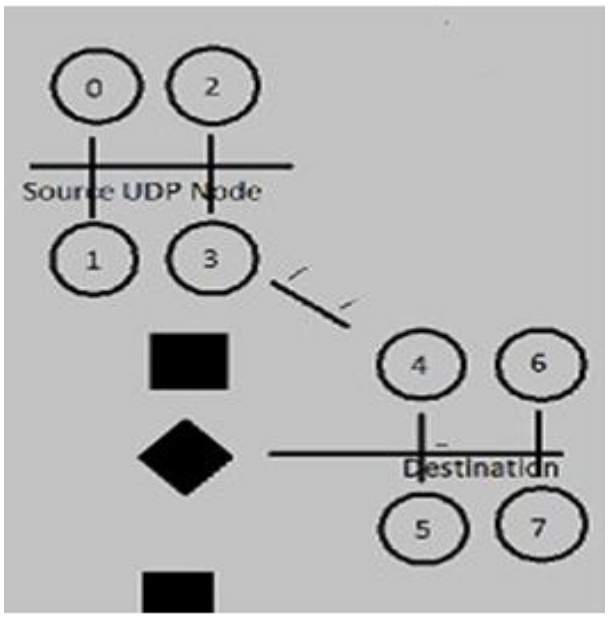


Fig. 3.2 Animation result

```
(base) sahyadri@sahyadri-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~$ awk -f exp33.awk exp33.tr

 total number of  data packets at Node 7: 87
(base) sahyadri@sahyadri-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~$
```

Fig. 3.3 AWK result

Table.3.1 AWK output

Error Rate	Data Rate	Packets Received at n7	Throughput = (PR/EST)*PS
0.1	1MB		
0.3	1.1MB		
0.5	1.2MB		
1.1	1.3MB		

PR = Packet Received at node 7

EST = End Simulation Time

PS = Packet Size

3.8 Discussions:

3.9 Pre – Experimentation Questions:

1. Compare analog and digital signals?
2. Define bandwidth?
3. What are the factors on which data rate depends?
4. Define bit rate and bit interval?
5. What is Nyquist bit rate formula?

3.10 Post – Experimentation Questions:

1. What is a node?
2. What is anonymous FTP?
3. What does 10Base-T mean?
4. Describe Ethernet.
5. What are some drawbacks of implementing a ring topology?

EXPERIMENT No. 04:
Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations

4.1 Objectives:

- Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.

4.2 Apparatus Required:

NS2

4.3 Pre-Requisite:

Media access control, collision detection, Carrier-sense multiple access.

4.4 Introduction:

In Ethernet networks, the CSMA/CD is commonly used to schedule retransmissions after collisions. The retransmission is delayed by an amount of time derived from the slot time and the number of attempts to retransmit. After c collisions, a random number of slot times between 0 and 2^{c-1} is chosen. For the first collision, each sender will wait 0 or 1 slot times. After the second collision, the senders will wait anywhere from 0 to 3 slot times (inclusive). After the third collision, the senders will wait anywhere from 0 to 7 slot times (inclusive), and so forth. As the number of retransmission attempts increases, the number of possibilities for delay increases exponentially. The 'truncated' simply means that after a certain number of increases, the exponentiation stops; i.e. the retransmission timeout reaches a ceiling, and thereafter does not increase any further. For example, if the ceiling is set at i = 10 (as it is in the IEEE 802.3 CSMA/CD standard), then the maximum delay is 1023 slot times. Because these delays cause other stations that are sending to collide as well, there is a possibility that, on a busy network, hundreds of people may be caught in a

single collision set. Because of this possibility, after 16 attempts at transmission, the process is aborted.

4.5 Procedure:

1. Open terminal.
2. Check whether the necessary folders are available in the location by typing **ls**
3. If not available change the directory by typing **cd ..**
4. Type **java -jar nsg2.1.jar** in the terminal (A new blue window opens)
5. In new window
   h. Click on Scenario ------> select new wired scenario (for wired connection) or new wireless scenario (for wireless connection)
   i. Click on node —> select single -----> place as many nodes as needed in order
   j. Click on Link —> select duplex link -----> connect the nodes
   k. Click on Agent —> For Connection oriented network attach TCP to source node and TCP Sink to destination node. Similarly, for Connectionless network attach UDP to source node and Null to destination node Click source and destination and do logical connection
   l. Click on Application —> For TCP attach FTP and for UDP attach CBR
   m. Click on Parameter --->Set Simulation time=10.0, Trace and Nam file name should be same ----> click Done
   n. Click on TCL ---> do the necessary changes ---> Save with same filename as before with file type TCL files ---> close all the windows
6. In terminal type **ns filename.tcl**
   a. NAM file opens to show animation of the network. Click on play button, observe packet transfer ---> close all the windows
7. In terminal type **gedit filename.awk**
   a. Type awk program in an editable window, save and close
8. In terminal type **awk -f filename.awk filename.tr** ---> Required output gets displayed on the terminal
9. For alternative cases type **gedit filename.tcl** in terminal ---> Change respective parameter
10. Repeat Step 6
11. Repeat Step 8 ----> Note down the values in a table
12. Trace file can be opened by typing a command **gedit filename.tr** on terminal

4.6 Observations:

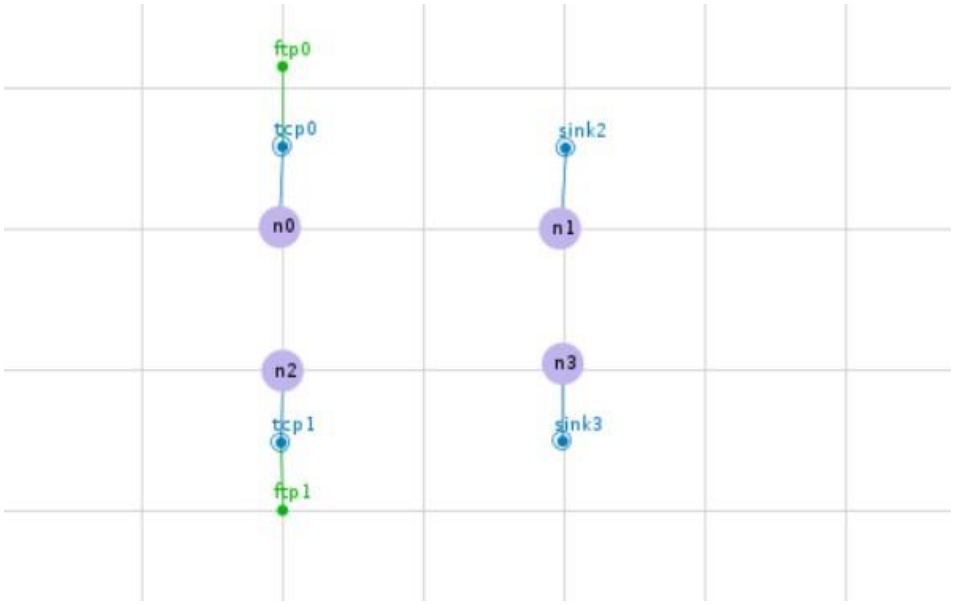


Fig.4.1 Topology

Program:

```
set ns [new Simulator]

set tf [open lab4.tr w]
$ns trace-all $tf
set nf [open lab4.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns make-lan "$n0 $n1 $n2 $n3" 10mb 10ms LL Queue/DropTail
Mac/802_3 set tcp0 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp0 $sink3
```

```
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp2 $sink1
set file1 [open file1.tr w]
$tcp0 attach $file1
$tcp0 trace cwnd_
#$tcp0 set maxcwnd_ 10
set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp2 trace cwnd_
proc finish { } { global nf tf ns
$ns flush-trace
exec nam lab7.nam &
close $nf
close $tf
exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 1.5 "$ftp0 stop"
$ns at 2 "$ftp0 start"
$ns at 3 "$ftp0 stop"
$ns at 0.2 "$ftp2 start"
$ns at 2 "$ftp2 stop"
$ns at 2.5 "$ftp2 start"
$ns at 4 "$ftp2 stop"
$ns at 5.0 "finish"
$ns run
```

4.7 Results:

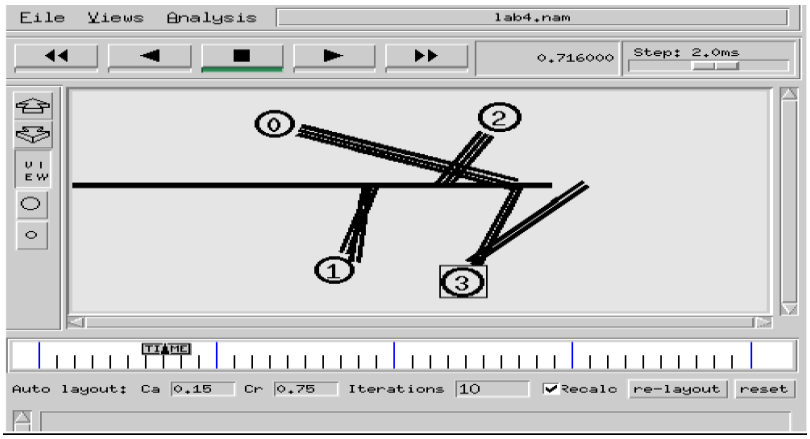


Fig.4.2 Animation result

4.8 Discussions:

4.9 Pre – Experimentation Questions:

1. Define Shannon Capacity?
2. What is sampling?
3. What are the modes of data transmission?
4. What is Asynchronous mode of data transmission?
5. What is Synchronous mode of data transmission?

4.10 Post – Experimentation Questions:

1. What are the different types of multiplexing?
2. What are the different transmission media?
3. What are the different Guided Media?
4. What do you mean by wireless communication?
5. What are the switching methods?

EXPERIMENT No. 05:
Develop Dijkstra's algorithm to compute the shortest routing path.

5.1 Objectives:

- To develop Dijkstra's algorithm to compute the shortest routing path.

5.2 Apparatus Required:

Text Editor, Terminal Window

5.3 Pre-Requisite:

Fundamentals of routing algorithms.

5.4 Introduction:

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. The Dijkstra's algorithm uses labels that are positive integers or real numbers, which are totally ordered. It can be generalized to use any labels that are partially ordered, provided the subsequent labels are monotonically non-decreasing. This generalization is called the generic Dijkstra shortest-path algorithm. To create a least-cost tree for itself, using the shared LSDB, each node needs to run the famous Dijkstra's Algorithm. This iterative algorithm uses the following steps:

1. The node chooses itself as the root of the tree, creating a tree with a single node, and sets the total cost of each node based on the information in the LSDB.
2. The node selects one node, among all nodes not in the tree, which is closest to the root, and adds this to the tree. After this node is added to the tree, the cost of all other nodes not in the tree needs to be updated because the paths may have been changed.
3. The node repeats step 2 until all nodes are added to the tree.

5.5 Procedure:

To write C Program:

- Open a terminal and enter the command: **gedit**
- Type the C program and save the program with **filename.c**
- Close the text editor

To Compile the program:

- Go to terminal and type the command: **cc filename.c**

To Check the output:

- By default, output will be stored in "a.out" file.
- In terminal type the command: **./a.out**

5.6 Observations:

```
#include<stdio.h>

int p[10][10];

int main()

{

int i,j,k,n,t,g;

int m[10][10],w[10][10];

char r;

int path (int a,int b);

printf("enter the number of nodes\n");

scanf("%d",&n);

printf("enter the node connection matrix(to indicate no connection enter 100)\n" );
```

```
for(i=1;i<=n;i++)
        {
        for(j=1;j<=n;j++)
        {
                scanf("%d",&w[i][j]);
                m[i][j]=w[i][j];
                p[i][j]=0;
        }
}
for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        if(m[i][k]+m[k][j]<m[i][j])
                        {
                                m[i][j]=m[i][k]+m[k][j];
                                p[i][j]=k;
                        }
                        for(g=0;g<5;g++)
                        {
                            printf("\n enter the source and destination node\n");
                            scanf("%d%d",&i,&j);
                            printf("\nthe weight is %d",m[i][j]);
                            printf("\n the path is");
                            printf("%d---->",i);
                            path(i,j);
```

```
                    printf("%d",j);
                }
    }
    int path(int i,int j)
    {
    int k;
    k=p[i][j];
    if(k!=0)
    {
        path(i,k);
        printf("%d-->",k);
        path(k,j);
    }
    }
```

5.7 Results:

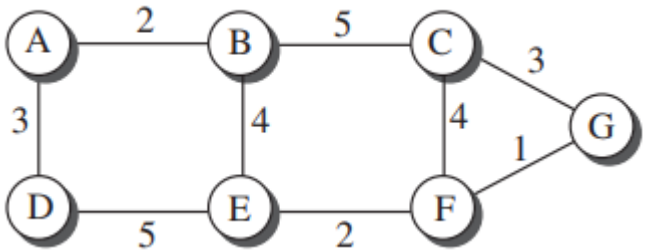


Fig.5.1 Topology

Enter the number of nodes 7

Enter the node connection matrix (to indicate no connection enter 100)

0	2	100	3	100	100	100
2	0	5	100	4	100	100
100	5	0	100	100	4	3
3	100	100	0	5	100	100
100	4	100	5	0	2	100
100	100	4	100	2	0	1
100	100	3	100	100	1	0

Enter the source and destination node

1 5

The weight is 6

The path is **1----->2------->5**

5.8 Discussions:

5.9 Pre – Experimentation Questions:

1. What are drawbacks in Dijkstra's algorithm?
2. Why is Dijkstra's algorithm important?
3. What is the time complexity of Dijkstra's algorithm?
4. What are the design issue of layers?
5. What are the protocols in application layer?

5.10 Post – Experimentation Questions:

1. Explain Dijkstra's algorithm?
2. Differentiate Dijkstra's algorithm and distance vector algorithm?
3. How routers update distances to each of its neighbour?
4. How do you overcome count to infinity problem?
5. What is Dynamic routing?

EXPERIMENT No. 06:
Interpret routing protocols of network layer through C programming

6.1 Objectives:

- Interpret routing protocols of network layer through C programming

6.2 Apparatus Required:

Text Editor, Terminal Window

6.3 Pre-Requisite:

Data network, routers, routing tables of routers

6.4 Introduction:

Distance Vector Routing is one of the routing algorithms used in a Wide Area Network for computing shortest path between source and destination. Each router initially has information about its all neighbours (i.e., it is directly connected). After a period of time, each router exchanges its routing table among its neighbours. After certain number of exchanges, all routers will have the full routing information about the area of the network. After each table exchange, router re-computes the shortest path between the routers. The algorithm used for this routing is called Distance Vector Routing.

6.5 Procedure:

To write C Program:

- Open a terminal and enter the command: **gedit**
- Type the C program and save the program with **filename.c**
- Close the text editor

To Compile the program:

- Go to terminal and type the command: **cc filename.c**

To Check the output:

- By default, output will be stored in “a.out” file.
- In terminal type the command: **./a.out**

6.6 Observations:

```
#include<stdio.h>
int dist[10][10],nextnode[10][10],cost[10][10],n;
void distvector()
{
  int i,j,k;
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    {
      dist[i][j]=cost[i][j];
      nextnode[i][j]=j;
    }
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
        for(k=0;k<n;k++)
          if(dist[i][j]>cost[i][k]+dist[k][j])
          {
            dist[i][j]=dist[i][k]+dist[k][j];
            nextnode[i][j]=k;
          }
}
int main()
{
  int i,j;
  printf("\nenter the no of vertices:\t");
  scanf("%d",&n);
  printf("\nenter the cost matrix\n");
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      scanf("%d",&cost[i][j]);
  distvector();
```

```
for(i=0;i<n;i++)
{
  printf("\nstate value for router %c \n",i+65);
  printf("\ndestnode\tnextnode\tdistance\n");
  for(j=0;j<n;j++)
  {
    if(dist[i][j]==99)
      printf("%c\t\t-\t\tinfinite\n",j+65);
    else
      printf("%c\t\t%c\t\t%d\n",j+65,nextnode[i][j]+65,dist[i][j]);
  } }
}
```

6.7 Results:

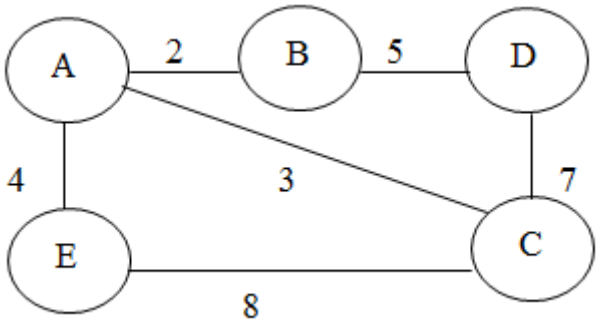


Fig.6.1 Topology

Enter no of vertices 5

Enter the cost matrix

99	2	3	99	4
2	99	99	5	99
3	99	99	7	8
99	5	7	99	99
4	99	8	99	99

State value for router A

destnode	nextnode	distance
A	B	4
B	B	2
C	C	3
D	B	7
E	E	4

State value for router B

destnode	nextnode	distance
A	A	2
B	A	4
C	A	5
D	D	5
E	A	6

State value for router C

destnode	nextnode	distance
A	A	3
B	A	5
C	A	6
D	D	7
E	A	7

State value for router D

destnode	nextnode	distance
A	B	7
B	B	5
C	C	7
D	B	10
E	B	11

State value for router E

destnode	nextnode	distance
A	A	4
B	A	6
C	A	7
D	A	11
E	A	8

6.8 Discussions:

6.9 Pre – Experimentation Questions:

1. What are Routing algorithms?
2. How do you classify routing algorithms? Give examples for each.
3. What are drawbacks in distance vector algorithm?
4. How routers update distances to each of its neighbour?
5. How do you overcome count to infinity problem?

6.10 Post – Experimentation Questions:

1. What are the design issue of layers?
2. What are the protocols in application layer?
3. What is domain name system (DNS)?
4. Differentiate between Connectionless and connection oriented connection.
5. What are protocols running in different layers?

EXPERIMENT No. 07:
Implement Stop and Wait Protocol and Sliding Window Protocol.

7.1 Objectives:

- Implementation of Stop and Wait Protocol and Sliding Window Protocol.

7.2 Apparatus Required:

Text Editor, Terminal Window

7.3 Pre-Requisite:

Connection Oriented (Point to Point) Transmission, Data Link and Transport Layers

7.4 Introduction:

Stop and Wait Protocol:

Stop and Wait is a reliable transmission flow control protocol. This protocol works only in Connection Oriented (Point to Point) Transmission. The Source node has window size of ONE. After transmission of a frame the transmitting (Source) node waits for an Acknowledgement from the destination node. If the transmitted frame reaches the destination without error, the destination transmits a positive acknowledgement. If the transmitted frame reaches the Destination with error, the receiver destination does not transmit an acknowledgement. If the transmitter receives a positive acknowledgement it transmits the next frame if any. Else if its acknowledgement receive timer expires, it retransmits the same frame.

1. Start with the window size of 1 from the transmitting (Source) node
2. After transmission of a frame the transmitting (Source) node waits for a reply (Acknowledgement) from the receiving (Destination) node.
3. If the transmitted frame reaches the receiver (Destination) without error, the receiver (Destination) transmits a Positive Acknowledgement.
4. If the transmitted frame reaches the receiver (Destination) with error, the receiver (Destination) do not transmit acknowledgement.

5. If the transmitter receives a positive acknowledgement it transmits the next frame if any. Else if the transmission timer expires, it retransmits the same frame again.

6. If the transmitted acknowledgment reaches the Transmitter (Destination) without error, the Transmitter (Destination) transmits the next frame if any.

7. If the transmitted frame reaches the Transmitter (Destination) with error, the Transmitter (Destination) transmits the same frame.

8. This concept of the Transmitting (Source) node waiting after transmission for a reply from the receiver is known as STOP and WAIT.

Sliding Window Protocol:

A sliding window protocol is a feature of packet-based data transmission protocols. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the Data Link Layer (OSI model) as well as in the Transmission Control Protocol (TCP). Conceptually, each portion of the transmission (packets in most data link layers, but bytes in TCP) is assigned a unique consecutive sequence number, and the receiver uses the numbers to place received packets in the correct order, discarding duplicate packets and identifying missing ones. The problem with this is that there is no limit on the size of the sequence number that can be required. By placing limits on the number of packets that can be transmitted or received at any given time, a sliding window protocol allows an unlimited number of packets to be communicated using fixed-size sequence numbers. The term "window" on the transmitter side represents the logical boundary of the total number of packets yet to be acknowledged by the receiver. The receiver informs the transmitter in each acknowledgment packet the current maximum receiver buffer size (window boundary). The TCP header uses a 16-bit field to report the receive window size to the sender. Therefore, the largest window that can be used is 216 = 64 kilobytes. In slow-start mode, the transmitter starts with low packet count and increases the number of packets in each transmission after receiving acknowledgment packets from receiver. For every ack packet received, the window slides by one packet (logically) to transmit one new packet. When the window threshold is reached, the transmitter sends one packet for one ack packet received. If the window limit is 10 packets then in slow start mode the transmitter may start transmitting one packet followed by two packets (before transmitting two packets, one packet ack has to be received), followed by three packets and so on until 10 packets. But after reaching 10 packets, further transmissions are restricted to one packet transmitted for one ack packet received. In a simulation this appears as if the window is moving by one packet distance for every ack packet received. On the receiver side also the window moves one packet for every packet received. The sliding window method ensures that

traffic congestion on the network is avoided. The application layer will still be offering data for transmission to TCP without worrying about the network traffic congestion issues as the TCP on sender and receiver side implement sliding windows of packet buffer. The window size may vary dynamically depending on network traffic. For the highest possible throughput, it is important that the transmitter is not forced to stop sending by the sliding window protocol earlier than one round-trip delay time (RTT). The limit on the amount of data that it can send before stopping to wait for an acknowledgment should be larger than the bandwidth-delay product of the communications link. If it is not, the protocol will limit the effective bandwidth of the link.

7.5 Procedure:

To write C Program:

- Open a terminal and enter the command: **gedit**
- Type the C program and save the program with **filename.c**
- Close the text editor

To Compile the program:

- Go to terminal and type the command: **cc filename.c**

To Check the output:

- By default, output will be stored in "a.out" file.
- In terminal type the command: **./a.out**

7.6 Observations:

Stop and Wait Protocol

```
#include <stdio.h>
#include <stdlib.h>
#define RTT 4
#define TIMEOUT 4
#define TOT_FRAMES 7
enum {NO,YES} ACK;
int main()
{
  int wait_time,i=1;
  ACK=YES;
  for(;i<=TOT_FRAMES;)
```

```
  {
   if (ACK==YES && i!=1)
    printf("\nSENDER: ACK for Frame %d Received.\n",i-1);
   printf("\nSENDER: Frame %d sent, Waiting for ACK...\n",i);
   ACK=NO;
   wait_time= rand() % 4+1;
   if (wait_time==TIMEOUT)
    printf("SENDER: ACK not received for Frame %d=>TIMEOUT Resending Frame...",i);
   else
   {
    sleep(RTT);
    printf("\nRECEIVER: Frame %d received, ACK sent\n\n",i);
    ACK=YES;
    i++;
   }
  }
  return 0;
}
```

Sliding Window Protocol

```
#include <stdio.h>
#include <stdlib.h>
#define RTT 5
int main()
{
  int window_size,i,j,frames[50];
  printf("Enter window size: ");
  scanf("%d",&window_size);
  printf("\nEnter number of frames to transmit: ");
  scanf("%d",&j);
  printf("\nEnter %d frames: ",j);
  for(i=1;i<=j;i++)
   scanf("%d",&frames[i]);
  printf("\nAfter sending %d frames at each stage sender waits for ACK",window_size);
  printf("\nSending frames in the following manner....\n\n");
```

```
    for(i=1;i<=j;i++)
    {
     if(i%window_size!=0)
       printf(" %d",frames[i]);
     else
     {
       printf(" %d\n",frames[i]);
       printf("SENDER: waiting for ACK...\n\n");
       sleep(RTT/2);
       printf("RECEIVER: Frames Received, ACK Sent\n\n");
       sleep(RTT/2);
       printf("SENDER:ACK received\n");
     }
    }
    if(j%window_size!=0)
    {
     printf("\nSENDER: waiting for ACK...\n");
     sleep(RTT/2);
     printf("\nRECEIVER:Frames Received, ACK Sent\n\n");
     sleep(RTT/2);
     printf("SENDER:ACK received.");
    }
    return 0;
  }
```

7.7 Results:

Stop and Wait Protocol:

SENDER: Frame1 sent, waiting for ACK….

SENDER: ACK not received for Frame1=>TIMEOUT Resending Frame…

SENDER: Frame1 sent, waiting for ACK…

RECEIVER: Frame1 received, ACK sent

SENDER: ACK for Frame1 received ….

SENDER: Frame2 sent, waiting for ACK……

RECEIVER: Frame2 received, ACK sent

SENDER: ACK for Frame2 received.

SENDER: Frame3 sent, waiting for ACK…

SENDER: ACK not received for Frame3=>TIMEOUT Resending Frame…

SENDER: Frame3 sent, waiting for ACK…

RECEIVER: Frame3 received, ACK sent

SENDER: ACK for Frame3 received

SENDER: Frame4 sent, waiting for ACK…

SENDER: ACK not received for Frame4=>TIMEOUT Resending Frame…

SENDER: Frame4 sent, waiting for ACK…

RECEIVER: Frame4 received, ACK sent

SENDER: ACK for Frame4 received

SENDER: Frame5sent, waiting for ACK…

RECEIVER: Frame5 received, ACK sent

SENDER: ACK for Frame5 received

SENDER: Frame6 sent, waiting for ACK…

RECEIVER: Frame6 received, ACK sent

SENDER: ACK for Frame6 received

SENDER: Frame7 sent, waiting for ACK…

RECEIVER: Frame7 received, ACK sent

Sliding Window Protocol:

Enter Window Size : 3
Enter number of frames to transmit : 8
Enter 8 frames : 0
1
2
3
4
5
6
7
After sending 3 frames at each stage sender waits for ACK
Sending frames in the following manner…
0 1 2
SENDER: Waiting for ACK…
RECEIVER: Frames Received, ACK sent
SENDER: ACK received
3 4 5
SENDER: Waiting for ACK…

RECEIVER: Frames Received, ACK sent
SENDER: ACK received
6 7
SENDER: Waiting for ACK..
RECEIVER: Frames Received, ACK sent

7.8 Discussions:

7.9 Pre – Experimentation Questions:

1. What is ARQ?
2. What is stop and wait protocol?
3. What is stop and wait ARQ?
4. What is usage of sequence number in reliable transmission?
5. What is sliding window?

7.10 Post – Experimentation Questions:

1. Define Go-Back-N ARQ?
2. Define Selective Repeat ARQ?
3. What do you mean by pipelining, is there any pipelining in error control?
4. What do you mean by line control protocol?
5. What do you mean by Authentication protocol?

EXPERIMENT No. 08:
Build a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth

8.1 Objectives:

- Build a point to point network with four nodes and duplex links between them.
- Analyze the network performance by setting the queue size and varying the bandwidth

8.2 Apparatus Required:

NS2

8.3 Pre-Requisite:

Computer networks, network topology.

8.4 Introduction:

Point-to-point networks are used to connect one location to another location. Point to Point topology is the simplest topology that connects two nodes directly together with a common link. The entire bandwidth of the common link is reserved for transmission between those two nodes. The point-to-point connections use an actual length of wire or cable to connect the two ends, but other options, such as satellite links, or microwaves are also possible. The typical applications include Internet, Intranet or Extranet configurations, ISP access networks, LAN to LAN applications.

The transfer of data in a point-to-point topology can be in multiple ways across the network: in a simplex, in full duplex, or half duplex. In Simplex mode of communication, signal flows in ONE direction and only one node transmit and the other receives. In Half duplex mode of communication, each node can transmit and receive but NOT at the same time. In Full-duplex mode of communication, both stations transmit and receive simultaneously.

8.5 Procedure:

1. Open terminal.
2. Check whether the necessary folders are available in the location by typing **ls**
3. If not available change the directory by typing **cd ..**
4. Type **java -jar nsg2.1.jar** in the terminal (A new blue window opens)
5. In new window
   a. Click on Scenario ------> select new wired scenario (for wired connection) or new wireless scenario (for wireless connection)
   b. Click on node —> select single ----> place as many nodes as needed in order
   c. Click on Link —> select duplex link -----> connect the nodes
   d. Click on Agent —> For Connection oriented network attach TCP to source node and TCP Sink to destination node. Similarly, for Connectionless network attach UDP to source node and Null to destination node Click source and destination and do logical connection
   e. Click on Application —> For TCP attach FTP and for UDP attach CBR
   f. Click on Parameter --->Set Simulation time=10.0, Trace and Nam file name should be same ----> click Done
   g. Click on TCL ---> do the necessary changes ---> Save with same filename as before with file type TCL files ---> close all the windows
6. In terminal type **ns filename.tcl**
   a. NAM file opens to show animation of the network. Click on play button, observe packet transfer ---> close all the windows
7. In terminal type **gedit filename.awk**
   a. Type awk program in an editable window, save and close
8. In terminal type **awk -f filename.awk filename.tr** ---> Required output gets displayed on the terminal
9. For alternative cases type **gedit filename.tcl** in terminal ---> Change respective parameter
10. Repeat Step 6
11. Repeat Step 8 ----> Note down the values in a table
12. Trace file can be opened by typing a command **gedit filename.tr** on terminal

8.6 Observations:

Case 1:

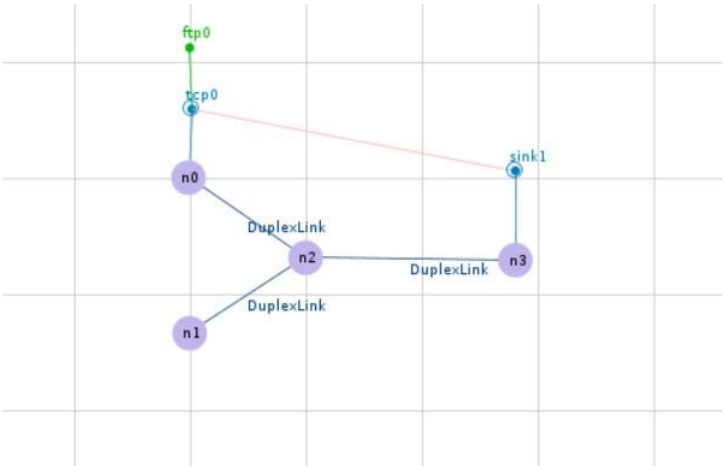


Fig. 8.1 Point to point network with TCP

Program:

```
# This script is created by NSG2 beta1
# <http://wushoupong.googlepages.com/nsg>
#===================================
#     Simulation parameters setup
#===================================
set val(stop)  10.0                ;# time of simulation end
#===================================
#        Initialization
#===================================
#Create a ns simulator
set ns [new Simulator]
#Open the NS trace file
set tracefile [open cn1.tr w]
$ns trace-all $tracefile
#Open the NAM trace file
set namfile [open cn1.nam w]
$ns namtrace-all $namfile
#===================================
#        Nodes Definition
```

```
#=================================
#Create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#=================================
#       Links Definition
#=================================
#Createlinks between nodes
$ns duplex-link $n0 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 50
$ns duplex-link $n1 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 50
$ns duplex-link $n3 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n3 $n2 50
#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n3 $n2 orient left
#=================================
#       Agents Definition
#=================================
#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp0 $sink1
$tcp0 set packetSize_ 1500
#=================================
#       Applications Definition
#=================================
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
```

```
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
$ns at 2.0 "$ftp0 stop"
#=================================
#       Termination
#=================================
#Define a 'finish' procedure
proc finish {} {
   global ns tracefile namfile
   $ns flush-trace
   close $tracefile
   close $namfile
   exec nam cn1.nam &
   exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

AWK File:

```
BEGIN{
        tcppack=0
        tcppack1=0
        }
        {
        if($1=="r"&&$3=="2"&&$4=="3"&&$5=="tcp")
        {
        tcppack++;
        }
        if($1=="d"&&$3=="2"&&$4=="3"&&$5=="tcp")
        {
        tcppack1++;
        }
        }
```

```
END{
        printf("\n total number of data packets received at Node 3: %d\n", tcppack++);
        printf("\n total number of packets dropped at Node 2: %d\n", tcppack1++);
        }
```

Case 2:

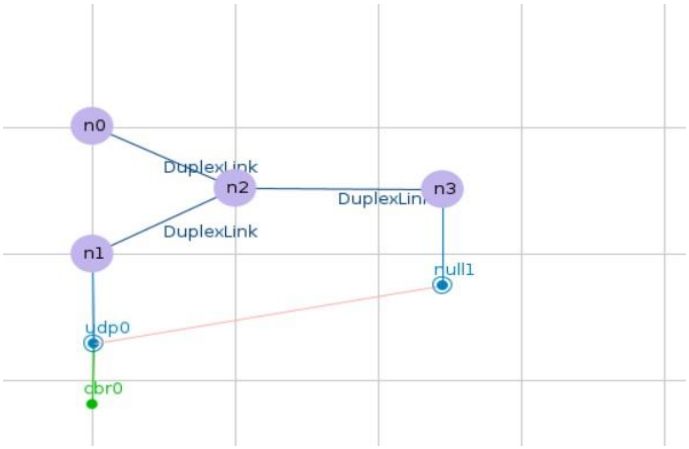


Fig.8.2 Point to point network with UDP

AWK File:

```
BEGIN{
        tcppack=0
        tcppack1=0
        }
        {
        if($1=="r"&&$3=="2"&&$4=="3"&&$5=="cbr")
        {
        tcppack++;
        }
        if($1=="d"&&$3=="2"&&$4=="3"&&$5=="cbr")
        {
        tcppack1++;
        }
        }
 END{
```

```
printf("\n total number of data packets received at Node 3: %d\n", tcppack++);
printf("\n total number of packets dropped at Node 2: %d\n", tcppack1++);
}
```

8.7 Results:

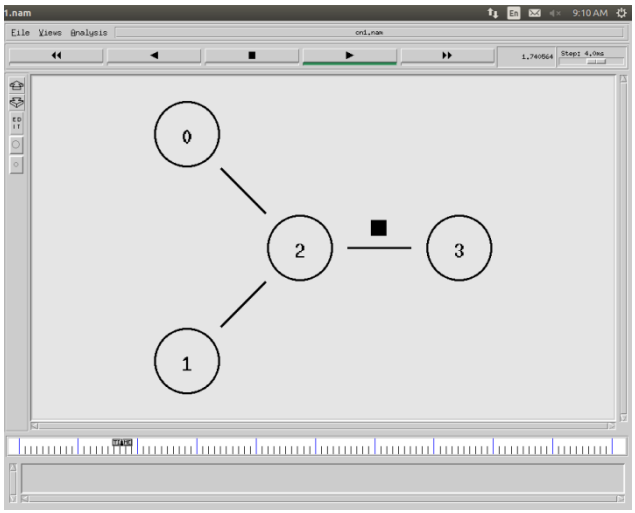


Fig.8.2 NAM result:

```
sahyadri@sahyadri-IP4BL-ME:~$ gedit cn1.awk
sahyadri@sahyadri-IP4BL-ME:~$ awk -f cn1.awk cn1.tr

 total number of data packets received at Node 3: 430

 total number of packets dropped at Node 2: 0
sahyadri@sahyadri-IP4BL-ME:~$
```

Fig.8.3 Performance of network for various values of Bandwidth & Queue sizes

Table 8.1.

Bandwidth (MB) N0-N2, N1-N2, N2-N3	**Queue Size Q1, Q2, Q3**	**Packets Received at Node 3**	**Packets Dropped at Node 2**
100, 100, 100	20, 20, 20		
100, 100, 1	20, 20, 2		
300, 300, 10	50, 50, 3		

8.8 Discussions:

8.9 Pre – Experimentation Questions:

1. What do you mean by data communication?
2. What is simplex?
3. What is half-duplex?
4. What is full duplex?
5. What is a network?

8.10 Post – Experimentation Questions:

1. What is distributed processing?
2. What is point to point connection?
3. What is multipoint connection?
4. What is a topology?
5. Define LAN

EXPERIMENT No. 09:
SIMULATE TO FIND THE NUMBER OF PACKETS DROPPED BY TCP/UDP

9.1 Objectives:

- Implement a four node point to point network with links 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

9.2 Apparatus Required:

NS2.

9.3 Pre-Requisite:

TCP/IP, Transport Layer

9.4 Introduction:

TCP (Transmission Control Protocol):

TCP recovers data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver side sequence number is used to eliminate the duplicates as well as to order the segments in correct order since there is a chance of "out of order" reception. Therefore, in TCP no transmission errors will affect the correct delivery of data.

UDP (User Datagram Protocol):

UDP uses a simple transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network

protocol to the user's program. As this is normally IP over unreliable media, there is no guarantee of delivery, ordering or duplicate protection.

9.5 Procedure:

1. Open terminal.
2. Check whether the necessary folders are available in the location by typing **ls**
3. If not available change the directory by typing **cd ..**
4. Type **java -jar nsg2.1.jar** in the terminal (A new blue window opens)
5. In new window
   a. Click on Scenario ------> select new wired scenario (for wired connection) or new wireless scenario (for wireless connection)
   b. Click on node —> select single ----> place as many nodes as needed in order
   c. Click on Link —> select duplex link -----> connect the nodes
   d. Click on Agent —> For Connection oriented network attach TCP to source node and TCP Sink to destination node. Similarly, for Connectionless network attach UDP to source node and Null to destination node Click source and destination and do logical connection
   e. Click on Application —> For TCP attach FTP and for UDP attach CBR
   f. Click on Parameter --->Set Simulation time=10.0, Trace and Nam file name should be same ----> click Done
   g. Click on TCL ---> do the necessary changes ---> Save with same filename as before with file type TCL files ---> close all the windows
6. In terminal type **ns filename.tcl**
   a. NAM file opens to show animation of the network. Click on play button, observe packet transfer ---> close all the windows
7. In terminal type **gedit filename.awk**
   a. Type awk program in an editable window, save and close
8. In terminal type **awk -f filename.awk filename.tr** ---> Required output gets displayed on the terminal
9. For alternative cases type **gedit filename.tcl** in terminal ---> Change respective parameter
10. Repeat Step 6
11. Repeat Step 8 ----> Note down the values in a table
12. Trace file can be opened by typing a command **gedit filename.tr** on terminal

9.6 Observations:

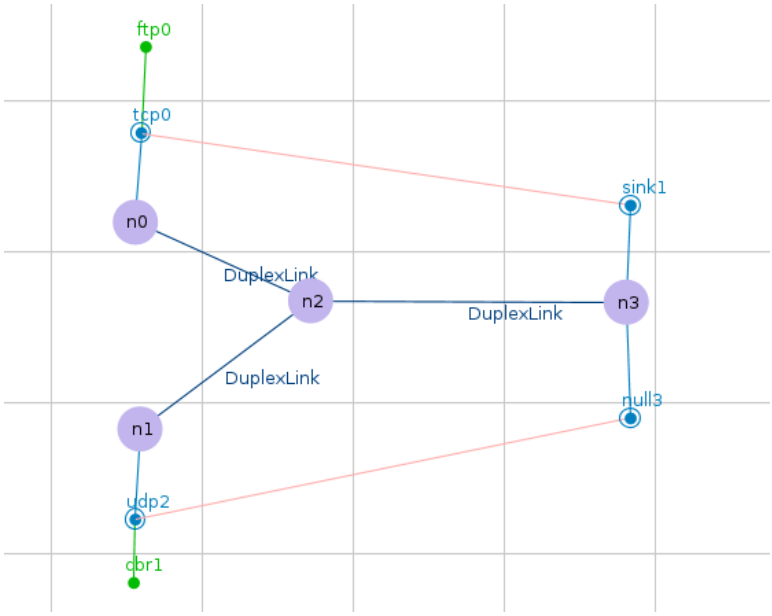


Fig.9.1 Point to point network with UDP & TCP agent

Program:

```
#=====================================
#     Simulation parameters setup
#=====================================
set val(stop)   10.0                ;# time of simulation end
#=====================================
#        Initialization
#=====================================
#Create a ns simulator
set ns [new Simulator]
#Open the NS trace file
set tracefile [open 2.tr w]
$ns trace-all $tracefile
#Open the NAM trace file
set namfile [open 2.nam w]
$ns namtrace-all $namfile
#=====================================
```

```
#       Nodes Definition
#=====================================
#Create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#=====================================
#       Links Definition
#=====================================
#Createlinks between nodes
$ns duplex-link $n0 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 50
$ns duplex-link $n1 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 50
$ns duplex-link $n2 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 50
#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
#=====================================
#       Agents Definition
#=====================================
#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp0 $sink1
$tcp0 set packetSize_ 1500
#Setup a UDP connection
set udp2 [new Agent/UDP]
$ns attach-agent $n1 $udp2
set null3 [new Agent/Null]
```

```
$ns attach-agent $n3 $null3
$ns connect $udp2 $null3
$udp2 set packetSize_ 1500
#=================================
#        Applications Definition
#=================================
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
$ns at 2.0 "$ftp0 stop"
#Setup a CBR Application over UDP connection
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp2
$cbr1 set packetSize_ 1000
$cbr1 set rate_ 1.0Mb
$cbr1 set random_ null
$ns at 1.0 "$cbr1 start"
$ns at 2.0 "$cbr1 stop"
#=================================
#        Termination
#=================================
#Define a 'finish' procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam 2.nam &
    exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

AWK File:

```
BEGIN{
        tcppack=0
        tcppack1=0
        }
        {
        if($1=="r"&&$4=="2"&&$5=="tcp")
        {
        tcppack++;
        }
        if($1=="r"&&$4=="2"&&$5=="cbr")
        {
        tcppack1++;
        }
        }
  END{
        printf("\n total number of TCP data packets sent between Node 0 and Node 2: %d\n", tcppack++);
        printf("\n total number of UDP data packets sent between Node 1 and Node 2: %d\n", tcppack1++);
        }
```

9.7 Results:

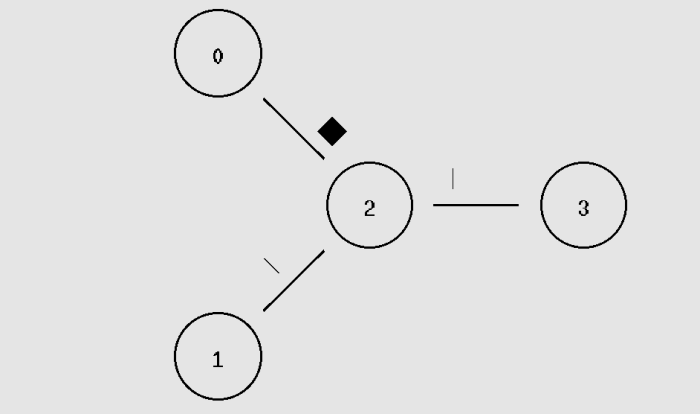


Fig. 9.2 Animation Result

```
(base) sahyadri@sahyadri-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~$ awk -f 2.awk 2.tr

 total number of TCP data packets sent between Node 0 and Node 2: 431

 total number of UDP data packets sent between Node 1 and Node 2: 125
(base) sahyadri@sahyadri-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~$
```

Fig. 9.3 AWK Results

Table. 9.1 Packet reception at node 2

Packet Size (Byte)	Sent at Node 2
TCP: 1500 UDP: 1500	
TCP: 1000 UDP: 1500	
TCP: 1300 UDP: 1500	

9.8 Discussions:

9.9 Pre – Experimentation Questions:

1. Define MAN
2. Define WAN
3. Define internet?
4. What is a protocol?
5. What is TCP/IP protocol model?

9.10 Post – Experimentation Questions:

1. Describe the functions of five layers in TCP/IP protocol?
2. What is ISO-OSI model?
3. What is multiplexing?
4. What is switching?
5. How data is transmitted over a medium?

EXPERIMENT No. 10:
Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters

10.1 Objectives:

- Implement ESS with transmission nodes in Wireless LAN
- Obtain the performance parameters.

10.2 Apparatus Required:

NS2

10.3 Pre-Requisite:

Wireless LAN, transmission nodes, ESS

10.4 Introduction:

Wireless LAN is basically a LAN that transmits data over air, without any physical connection between devices. The transmission medium is a form of electromagnetic radiation. Wireless LAN is ratified by IEEE in the IEEE 802.11 standard. In most of the WLAN products on the market based on the IEEE 802.11b technology the transmitter is designed as a Direct Sequence Spread Spectrum Phase Shift Keying (DSSS PSK) modulator, which is capable of handling data rates of up to 11 Mbps. The underlying algorithm used in Wireless LAN is known as the CSMA/CA – Carrier Sense Multiple Access/Collision Avoidance algorithm.

10.5 Procedure:

1. Open terminal.
2. Check whether the necessary folders are available in the location by typing **ls**
3. If not available change the directory by typing **cd ..**
4. Type **java -jar nsg2.1.jar** in the terminal (A new blue window opens)
5. In new window
   a. Click on Scenario ------> select new wired scenario (for wired connection) or new wireless scenario (for wireless connection)
   b. Click on node —> select single ----> place as many nodes as needed in order
   c. Click on Link —> select duplex link -----> connect the nodes
   d. Click on Agent —> For Connection oriented network attach TCP to source node and TCP Sink to destination node. Similarly, for Connectionless network attach UDP to source node and Null to destination node Click source and destination and do logical connection
   e. Click on Application —> For TCP attach FTP and for UDP attach CBR
   f. Click on Parameter --->Set Simulation time=10.0, Trace and Nam file name should be same ----> click Done
   g. Click on TCL ---> do the necessary changes ---> Save with same filename as before with file type TCL files ---> close all the windows
6. In terminal type **ns filename.tcl**
   a. NAM file opens to show animation of the network. Click on play button, observe packet transfer ---> close all the windows
7. In terminal type **gedit filename.awk**
   a. Type awk program in an editable window, save and close
8. In terminal type **awk -f filename.awk filename.tr** ---> Required output gets displayed on the terminal
9. For alternative cases type **gedit filename.tcl** in terminal ---> Change respective parameter
10. Repeat Step 6
11. Repeat Step 8 ----> Note down the values in a table
12. Trace file can be opened by typing a command **gedit filename.tr** on terminal

10.6 Observations:

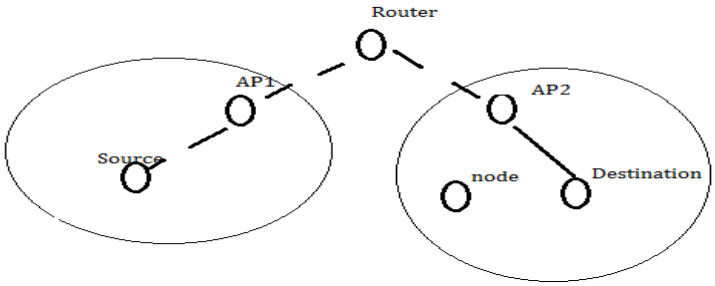


Fig. 10.1 Wireless LAN

Program:

```
set ns [new Simulator]
set tf [open expt55.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1000 1000
set nf [open expt55.nam w]
$ns namtrace-all-wireless $nf 2000 2000
set chan [new Channel/WirelessChannel];#Create wireless channel
$ns node-config -adhocRouting AODV \
                -llType LL \
                -macType Mac/802_11 \
                -ifqType Queue/DropTail \
                -ifqLen 50 \
                -phyType Phy/WirelessPhy \
                -channel $chan \
-propType Propagation/TwoRayGround \
                -antType Antenna/OmniAntenna \
                -topoInstance $topo \
                -agentTrace ON\
                -routerTrace ON \
                -macTrace    ON
create-god 6
set n0 [$ns node]
```

```
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
#set n5 [$ns node]
set n6 [$ns node]
#set n7 [$ns node]
$n0 label "tcp-Source"
$n1 label "Access Point1"
$n2 label "Router"
$n3 label "Access Point2"
$n4 label "Destination"
#$n5 label "node1"
$n6 label "node2"
#$n7 label "gateway"
$n0 set X_ 10
$n0 set Y_ 50
$n0 set Z_ 0
$ns initial_node_pos $n0 20
$n1 set X_ 120
$n1 set Y_ 130
$n1 set Z_ 0
$ns initial_node_pos $n1 20
$n2 set X_ 200
$n2 set Y_ 230
$n2 set Z_ 0
$ns initial_node_pos $n2 20
$n3 set X_ 300
$n3 set Y_ 130
$n3 set Z_ 0
$ns initial_node_pos $n3 20
$n4 set X_ 350
$n4 set Y_ 20
$n4 set Z_ 0
$ns initial_node_pos $n4 20
```

```
$n6 set X_ 600
$n6 set Y_ 20
$n6 set Z_ 0
$ns initial_node_pos $n6 20
$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n4 setdest 900 50 20"
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4
$ns connect $tcp0 $sink4
$ns at 5 "$ftp0 start"
$ns at 50 "$ftp0 stop"
#=================================
#       Termination
#=================================
proc finish { } {
        global ns nf tf
        $ns flush-trace
        exec nam expt55.nam  &
        close $tf
        exit 0
}
$ns at 80 "finish"
$ns run
```

AWK File

```
BEGIN{
        cbrpack=0
        cbrpack1=0
        }
        {
                if($1=="r"&&$4=="AGT")
        {
        cbrpack++;
        }
                if($1=="s"&&$4=="AGT")
        {
        cbrpack1++;
        }
        }

END{
                printf("\n total number of packets sent: %d\n", cbrpack1++);
                printf("\n total number of packets received: %d\n", cbrpack++);
        }
```

10.7 Results:

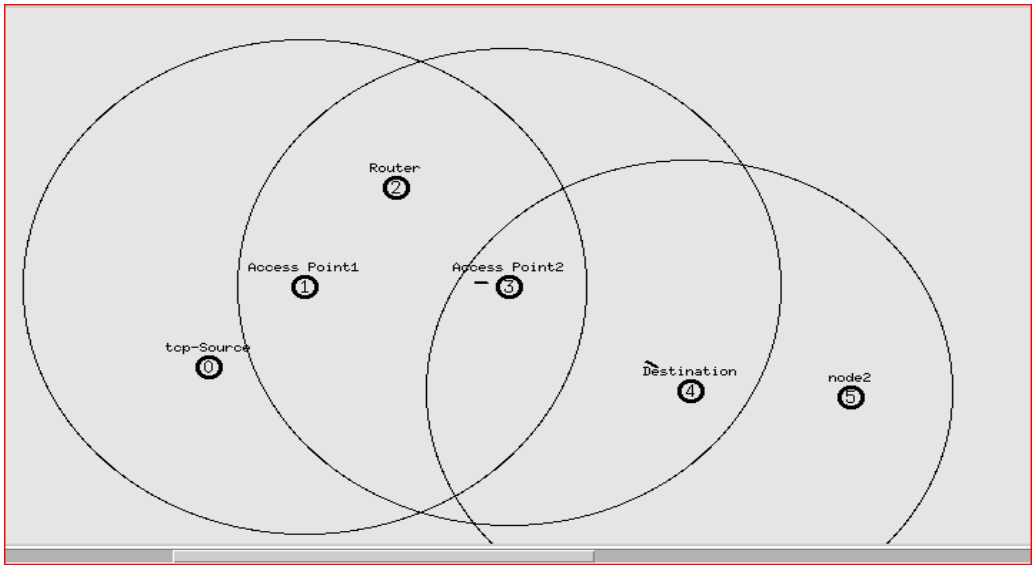


Fig. 10.2 Animation result

10.8 Discussions:

10.9 Pre – Experimentation Questions:

1. What are the duties of data link layer?
2. What are the types of errors?
3. What do you mean by redundancy?
4. Define parity check
5. What do you mean by flow control?

10.10 Post – Experimentation Questions:

1. What are functions of different layers?
2. Differentiate between TCP/IP Layers and OSI Layers
3. What is the use of adding header and trailer to frames?
4. What is encapsulation?
5. Why fragmentation requires?

SIMULATION-INTRODUCTION

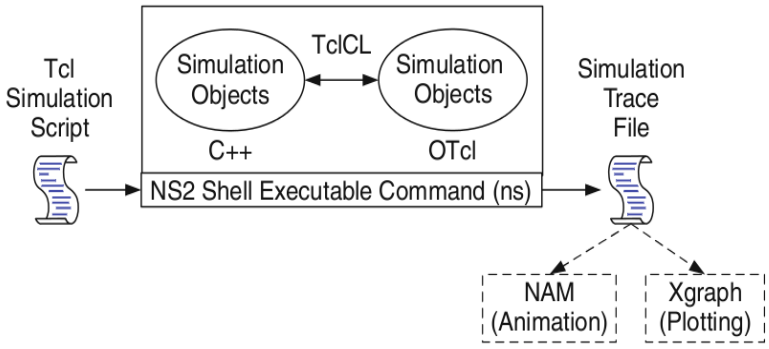
Network simulation is an important tool in developing, testing and evaluating network protocols. Simulation can be used without the target physical hardware, making it economical and practical for almost any scale of network topology and setup. It is possible to simulate a link of any bandwidth and delay, even if such a link is currently impossible in the real world. With simulation, it is possible to set each simulated node to use any desired software. This means that meaning deploying software is not an issue. Results are also easier to obtain and analyze, because extracting information from important points in the simulated network is as done by simply parsing the generated trace files.

Simulation is only of use if the results are accurate, an inaccurate simulator is not useful at all. Most network simulators use abstractions of network protocols, rather than the real thing, making their results less convincing. S.Y. Wang reports that the simulator OPNET uses a simplified finite state machine to model complex TCP protocol processing. NS-2 uses a model based on BSD TCP, it is implemented as a set of classes using inheritance. Neither uses protocol code that is used in real world networking.

INTRODUCTION TO NS-2

- Widely known as NS2, is simply an event driven simulation tool.
- Useful in studying the dynamic nature of communication networks.
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviours.

BASIC ARCHITECTURE OF NS2



TCL SCRIPTING

- Tcl is a general purpose scripting language. [Interpreter]
- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

NS SIMULATOR PRELIMINARIES

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

INITIALIZATION AND TERMINATION OF TCL SCRIPT IN NS-2

An ns simulation starts with the command

set ns [new Simulator]

This line declares a new variable as using the set command, you can call this variable as you wish, in general people declares it as ns because it is an instance of the Simulator class, so an object the code [new Simulator] is indeed the installation of the class Simulator using the reserved word new.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using "open" command:

#Open the Trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

```
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

The above creates a dta trace file called "out.tr" and a nam visualization trace file called "out.nam". Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called "tracefile1" and "namfile" respectively.

Remark that they begin with a # symbol. The second line open the file "out.tr" to be used for writing, declared with the letter "w". The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. In our case, this will be the file pointed at by the pointer "$namfile", i.e the file "out.tr".

The termination of the program is done using a "finish" procedure.

#Define a 'finish' procedure

```
proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
close $tracefile1
close $namfile
exec nam out.nam &
exit 0
}
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method "**flush-trace"** will dump the traces on the respective files. The tcl command "**close"** closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that is an exit because something fails.

At the end of ns program, we should call the procedure "finish" and specify at what time the termination should occur. For example,

```
$ns at 125.0 "finish"
```

will be used to call "**finish**" at time 125sec. Indeed, the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command

```
$ns run
```

DEFINITION OF A NETWORK OF LINKS AND NODES

The way to define a node is

```
set n0 [$ns node]
```

The node is created which is printed by the variable n0. When we shall refer to that node in the script we shall thus write $n0.

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

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

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a directional link instead of a bi-directional one, we should replace “duplex-link” by “simplex-link”.

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

AGENTS AND APPLICATIONS

We need to define routing (sources, destinations) the agents (protocols) the application that use them.

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line

```
set tcp [new Agent/TCP]
```

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.

The command

```
set sink [new Agent /TCPSink]
```

defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#Setup a UDP connection

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2
```

#setup a CBR over UDP connection

The below shows the definition of a CBR application using a UDP agent.

The command **$ns attach-agent $n4 $sink** defines the destination node.

The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false
```

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP.

Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

```
$cbr set packetSize_ <packet size>
```

SCHEDULING EVENTS

NS is a discrete event based simulation. The tcp script defines when event should occur. The initializing command set ns [new Simulator] creates an event scheduler, and events are then scheduled using the format.

$ns at <time><event>

The scheduler is started when running ns that is through the command $ns run. The beginning and end of the FTP and CBR application can be done through the following command.

```
$ns at 0.1 "$cbr start"
$ns at 1.0 " $ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"
```

STRUCTURE OF TRACE FILES

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, the meaning of the fields is:

Event	Time	From Node	To Node	PKT Type	PKT Size	Flags	Fid	Src Addr	Dest Addr	Seq Num	Pkt id

1. **Event or Type Identifier**
   - +: a packet enque event
   - -: a packet deque event
   - r: a packet reception event
   - d: a packet drop (e.g., sent to dropHead_) event
   - c: a packet collision at the MAC level
2. **Time**: at which the packet tracing string is created.

3-4. Source and Destination Node: source and destination ID's of tracing objects.

5. **Packet Name**: Name of the packet type.
6. **Packet Size**: Size of packet in bytes.
7. **Flags**: 7-digit flag string.
   - "-": disable

1st = “E”: ECN (Explicit Congestion Notification) echo is enabled.

2nd = “P”: the priority in the IP header is enabled.

3rd : Not in use

4th = “A”: Congestion action

5th = “E”: Congestion has occurred.

6th = “F”: The TCP fast start is used.

7th = “N”: Explicit Congestion Notification (ECN) is on.

8. Flow ID

9-10. Source and Destination Address:

The format of these two fields is “a.b”, where “a" is the address and "b" is the port.

11. Sequence Number

12. Packet Unique ID

Each trace line starts with an event (+, -, *d*, *r*) descriptor followed by the simulation time (*in seconds*) of that event, and from and to node, which identify the link on which the event occurred. Look at Figure 4 in the *Network Components* to see where in a link each type of event is traced. The next information in the line before flags (appeared as *"------"* since no flag is set) is packet type and size (in *Bytes*). Currently, NS implements only the Explicit Congestion Notification (ECN) bit, and the remaining bits are not used. The next field is flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script. Even though fid field may not use in a simulation, users can use this field for analysis purposes. The fid field is also used when specifying stream color for the NAM display. The next two fields are source and destination address in forms of "node.port". The next field shows the network layer protocol's packet sequence number. Note that even though UDP implementations do not use sequence number, NS keeps track of UDP packet sequence number for analysis purposes. The last field shows the unique id of the packet.