PART A:

1. **Write a program for Distance Vector Algorithm to find suitable path for transmission.**

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms.

Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always upd by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The starting assumption for distance-vector routing is each node knows the cost of the link of each of its directly connected neighbors. Next, every node sends a configured message to its directly connected neighbors containing its own distance table. Now, every node can learn and up its distance table with cost and next hops for all nodes network. Repeat exchanging until no more information between the neighbors.

Consider a node A that is interested in routing to destination H via a directly attached neighbor J. Node A's distance table entry, Dx(Y,Z) is the sum of the cost of the direct-one hop link between A and J, c(A,J), plus neighboring J's currently known minimum-cost path (shortest path) from itself(J) to H. That is

Dx(H,J) = c(A,J) + minw{Dj(H,w)} The minw is taken over all the J's

This equation suggests that the form of neighbor-to-neighbor communication that will take place in the DV algorithm - each node must know the cost of each of its neighbors' minimum-cost path to each destination. Hence, whenever a node computes a new minimum cost to some destination, it must inform its neighbors of this new minimum cost

Implementation Algorithm:

```
1. send my routing table to all my neighbors whenever my link table changes
2. when I get a routing table from a neighbor on port P with link metric M:
   a. add L to each of the neighbor's metrics
   b. for each entry (D, P', M') in the updated neighbor's table:
      i. if I do not have an entry for D, add (D, P, M') to my routing table
      ii. if I have an entry for D with metric M", add (D, P, M') to my routing table if M' < M"
3. if my routing table has changed, send all the new entries to all my neighbors.
```

Source Code:

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

void rout_table();
int d[10][10],via[10][10];
int i,j,k,l,m,n,g[10][10],temp[10][10],ch,cost;
int main()
{
            system("clear");
            printf("enter the value of no. of nodes\n");
            scanf("%d",&n);
            rout_table();
            for(i=0;i<n;i++)
              for(j=0;j<n;j++)
                          temp[i][j]=g[i][j];
            for(i=0;i<n;i++)
              for(j=0;j<n;j++)
                          via[i][j]=i;
            while(1)
            {
               for(i=0;i<n;i++)
                 for(j=0;j<n;j++)
                          if(d[i][j])
               for(k=0;k<n;k++)
                          if(g[i][j]+g[j][k]<g[i][k])
                          {
                                      g[i][k]=g[i][j]+g[j][k];
                                      via[i][k]=j;
                          }
               for(i=0;i<n;i++)
               {
                          printf("table for router %c\n" ,i+97);
                          for(j=0;j<n;j++)
                                      printf("%c:: %d via %c\n" ,j+97,
                                                                  g[i][j],via[i][j]+97);
                          }
                    Break;
                    }

}
void rout_table()
{
        printf("\nEnter the routing table : \n");
               printf("\t|");
        for(i=1;i<=n;i++)
                printf("%c\t",i+96);
        printf("\n");
        for(i=0;i<=n;i++)
                printf("-------");
        printf("\n");
        for(i=0;i<n;i++)
        {
                printf("%c       |",i+97);
                for(j=0;j<n;j++)
                {

                     scanf("%d",&g[i][j]);
                              if(g[i][j]!=999)
                               d[i][j]=1;
                    }
        }
}
```

Output:

```
[root@localhost ]# cc prg1.c
[root@localhost ]# ./a.out

enter the value of no. of nodes
4
Enter the routing table :
        |a      b       c       d
-----------------------------------
a       |0      5       1       4
b       |5      0       6       2
c       |1      6       0       3
d       |4      2       3       0

table for router a
a:: 0 via a
b:: 5 via a
c:: 1 via a
d:: 4 via a

table for router b
a:: 5 via b
b:: 0 via b
c:: 5 via d
d:: 2 via b

table for router c
a:: 1 via c
b:: 5 via d
c:: 0 via c
d:: 3 via c
```

```
table for router d
a:: 4 via d
b:: 2 via d
c:: 3 via d
d:: 0 via d

do you want to change the cost(1/0)
1
enter the vertices which you want to change the cost
1 3
enter the cost
2
table for router a
a:: 0 via a
b:: 5 via a
c:: 2 via a
d:: 4 via a
table for router b
a:: 5 via b
b:: 0 via b
c:: 5 via d
d:: 2 via b
table for router c
a:: 2 via c
b:: 5 via d
c:: 0 via c
d:: 3 via c
table for router d
a:: 4 via d
b:: 2 via b
c:: 3 via d
d:: 0 via d

do you want to change the cost(1/0)
0
```

2. **Using TCP/IP sockets, write a client server program to make the client send the file name and to make the server send back the content of the requested file if present.**

Source Code:

Client Side:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>
#define SERV_TCP_PORT 6880
#define SERV_HOST_ADDR "127.0.0.1"

int main()
{       int sockfd;
        struct sockaddr_in serv_addr,cli_addr;
        char filename[100],buf[1000];
        int n;
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=inet_addr(SERV_HOST_ADDR);
        serv_addr.sin_port=htons(SERV_TCP_PORT);
        if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
                printf("Client:cant open stream socket\n");
        else
        printf("Client:stream socket opened successfully\n");
        if(connect(sockfd,(struct sockaddr *)&serv_addr,
        sizeof(serv_addr))<0)
        printf("Client:cant connect to server\n");
        else
        printf("Client:connected to server successfully\n");
        printf("\n Enter the file name to be displayed :");
        scanf("%s",filename);
        write(sockfd,filename,strlen(filename));
        printf("\n filename transfered to server\n");
        n=read(sockfd,buf,1000);
        if(n < 0)
        printf("\n error reading from socket");
        printf("\n Client : Displaying file content of  %s\n",filename);
        fputs(buf,stdout);
        close(sockfd);
        exit(0);
}
```

Output: AT CLIENT SIDE

```
[root@localhost ]# cc tcpc.c
[root@localhost ]# ./a.out
```

Data Sent
File Content....
Sockets are a mechanism for exchanging data between processes. These processes can either be on the same machine, or on different machines connected via a network. Once a socket connection is established, data can be sent in both directions until one of the endpoints closes the connection.
I needed to use sockets for a project I was working on, so I developed and refined a few C++ classes to encapsulate the raw socket API calls. Generally, the application requesting the data is called the client, and the application servicing the request is called the server. I created two primary classes, ClientSocket and ServerSocket, that the client and server could use to exchange data.

Server side:
```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>
#define SERV_TCP_PORT 6880
#define SERV_HOST_ADDR "127.0.0.1"

int main()
{       int sockfd,newsockfd,clilen;
        struct sockaddr_in cli_addr,serv_addr;
        char filename[25],buf[1000];
        int n,m=0;
        int fd;

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

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

        if((bind(sockfd,(struct sockaddr *)
        &serv_addr,sizeof(serv_addr)))<0)
        printf("server:cant bind local address\n");
        else
        printf("server:bound to local address\n");
        listen(sockfd,5);
        printf("\n SERVER : Waiting for client...\n");
        for(;;)
        {
                clilen=sizeof(cli_addr);
                newsockfd=accept(sockfd,(struct sockaddr *) &cli_addr,&clilen);

        if(newsockfd<0)
        printf("server:accept error\n");
        else
        printf("server:accepted\n");
        n=read(newsockfd,filename,25);
        filename[n]='\0';
        printf("\n SERVER : %s is found and ready to transfer
        \n",filename);
        fd=open(filename,O_RDONLY);
        n=read(fd,buf,1000);
        buf[n]='\0';
        write(newsockfd,buf,n);
        printf("\n transfer success\n");
        close(newsockfd);
        exit(0)
}       }
```

Output:

```
[root@localhost ]# cc tcps.c
[root@localhost ]# ./a.out
Received the file name : data.txt
File content sent
```

3. **Write a program for Hamming code generation for error detection and correction.**

Hamming code uses redundant bits (extra bits) which are calculated according to the below formula:-

$2^r \geq m+r+1$

Where **r** is the number of redundant bits required and **m** is the number of data bits.

R is calculated by putting **r = 1, 2, 3 …** until the above equation becomes true.

R1 bit is appended at position **2^0**

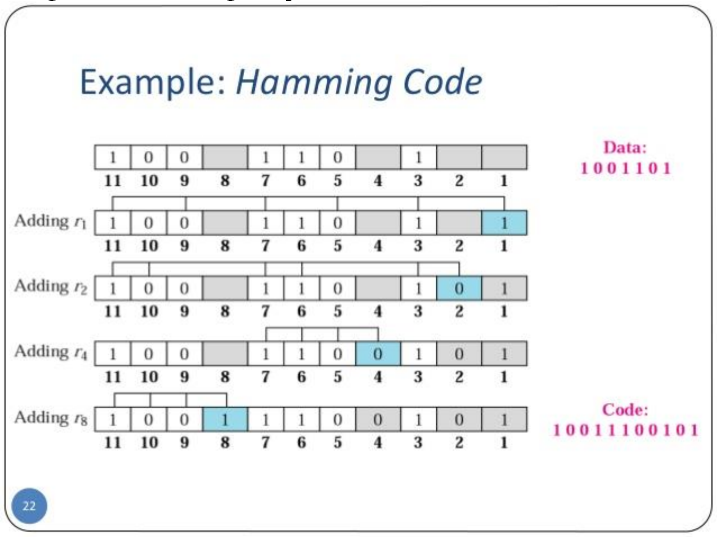
R2 bit is appended at position **2^1**

R3 bit is appended at position **2^2** and so on.

These redundant bits are then added to the original data for the calculation of error at receiver’s end.

At receiver’s end with the help of even parity (generally) the erroneous bit position is identified and since data is in binary we take complement of the erroneous bit position to correct received data.

Respective index parity is calculated for **r1, r2, r3, r4** and so on.



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

char data[5];
int encoded[8],edata[7],syndrome[3];
int hmatrix[3][7] = {
                    1,0,0,0,1,1,1,
                    0,1,0,1,0,1,1,
                    0,0,1,1,1,0,1
                    };
char gmatrix[4][8]={"0111000","1010100","1100010","1110001"};

int main(){
    int i,j;
    system("clear");
    printf("\nHamming code----- Encoding\n");
    printf("Enter 4 bit data : ");
    scanf("%s",data);
    printf("\nGenerator matrix\n");
    for(i=0;i<4;i++)
```

```
        printf("%s\n",gmatrix[i]);
    printf("\nEncoded data ");
    for(i=0;i<7;i++)
    {
        for(j=0;j<4;j++)
            encoded[i]+=((data[j]-'0')*(gmatrix[j][i]-'0'));
        encoded[i]=encoded[i]%2;
        printf("%d ",encoded[i]);
    }
    printf("\nHamming code----- Decoding\n");
    printf("Enter encoded bits as recieved : ");
    for(i=0;i<7;i++)
        scanf("%d",&edata[i]);
    for(i=0;i<3;i++)
    {
        for(j=0;j<7;j++)
        syndrome[i]+=(edata[j]*hmatrix[i][j]);
        syndrome[i]=syndrome[i]%2;
    }
    for(j=0;j<7;j++)
    if((syndrome[0]==hmatrix[0][j]) && (syndrome[1]==hmatrix[1][j])&& (syndrome[2]==hmatrix[2][j]))
    break;
    if(j==7)
    printf("\nError free\n");
    else
    {
        printf("\nError recieved at bit number %d of data\n",j+1);
        edata[j]=!edata[j];
        printf("\nCorrect data should be : ");
        for(i=0;i<7;i++)
            printf("%d",edata[i]);
    }
    return 0;
}
```

OUTPUT:

```
[root@localhost ]# cc prg4.c
[root@localhost ]#./a.out

 Enter 4 bit data : 1011
 Generator matrix
 0111000
 1010100
 1100010
 1110001

 Encoded data 0 1 0 1 0 1 1
 Hamming code----- Decoding
 Enter encoded bits as received : 0 1 0 1 1 1 1
 Error received at bit number 5 of data
 Correct data should be : 0101011
```

4. **Write a program for congestion control using leaky bucket algorithm.**

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero.

From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).

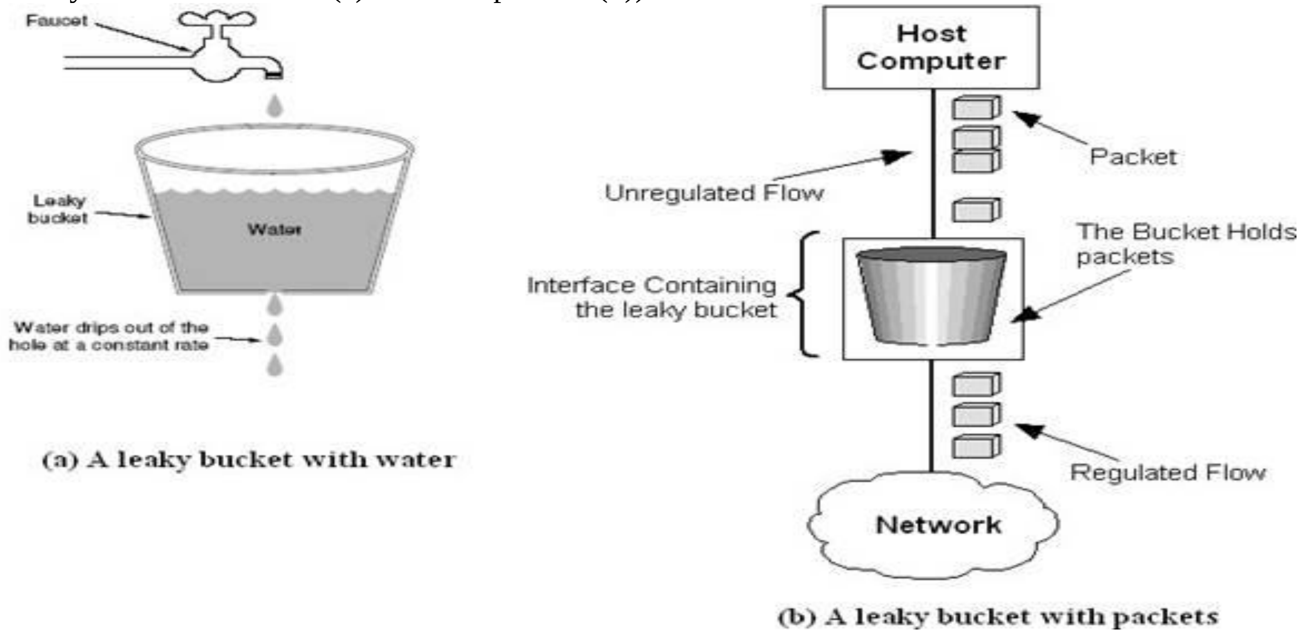

Faucet
Leaky bucket
Water
Water drips out of the hole at a constant rate
(a) A leaky bucket with water
Host Computer
Packet
Unregulated Flow
The Bucket Holds packets
Interface Containing the leaky bucket
Regulated Flow
Network
(b) A leaky bucket with packets

Figure: The leaky bucket traffic shaping algorithm

While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn't take into account the idle process of the sender which means that if the host doesn't transmit data for some time the bucket becomes empty without permitting the transmission of any packet

Implementation Algorithm:

```
Steps:
1. Read The Data For Packets
2. Read The Queue Size
3. Divide the Data into Packets
4. Assign the random Propagation delays for each packets to input into the bucket (input_packet).
5. wlile((Clock++<5*total_packets)and
         (out_packets< total_paclets))
   a. if (clock == input_packet)
         i. insert into Queue
   b. if (clock % 5 == 0 )
         i. Remove paclet from Queue
6. End
```

Source Code:

```
#include<stdio.h>
#include<strings.h>
#include<stdio.h>
int min(int x,int y)
{
if(x<y)
return x;
else
return y;
}
int main()
{
int drop=0,mini,nsec,cap,count=0,i,inp[25],process;
syste("clear");
printf("Enter The Bucket Size\n");
scanf("%d",&cap);
printf("Enter The Operation Rate\n");
scanf("%d",&process);
printf("Enter The No. Of Seconds You Want To Stimulate\n");
scanf("%d",&nsec);
for(i=0;i<nsec;i++)
{
printf("Enter The Size Of The Packet Entering At %dsec\n",i+1);
scanf("%d",&inp[i]);
}
printf("\nSecond|Packet Recieved|Packet Sent|PacketLeft|Packet Dropped|\n");
printf("--------------------------------------------------------------\n");
for(i=0;i<nsec;i++)
{
count+=inp[i];
if(count>cap)
{
drop=count-cap;
count=cap;
}
printf("%d",i+1);
printf("\t%d",inp[i]);
mini=min(count,process);
printf("\t\t%d",mini);
count=count-mini;
printf("\t\t%d",count);
printf("\t\t%d\n",drop);
drop=0;
}
for(;count!=0;i++)
{
if(count>cap)
{
drop=count-cap;
count=cap;
}
printf("%d",i+1);
printf("\t0");
mini=min(count,process);
printf("\t\t%d",mini);
count=count-mini;
printf("\t\t%d",count);
printf("\t\t%d\n",drop);
}
}
```

<u>Output:</u>

Compile and run

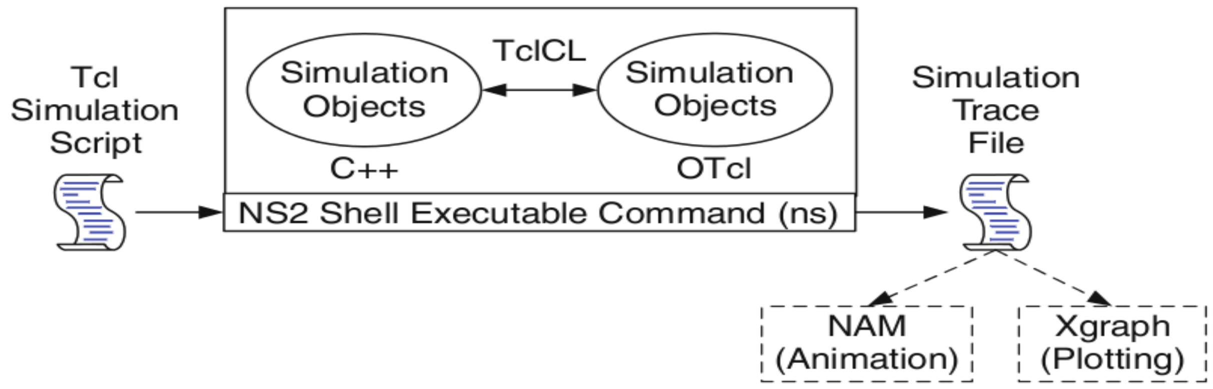
```
$ cc –o Congestion Congestion.c
$ ./Congestion
Enter The Bucket Size
5
Enter The Operation Rate
2
Enter The No. Of Seconds You Want To Stimulate
3
Enter The Size Of The Packet Entering At 1 sec
5
Enter The Size Of The Packet Entering At 1 sec
4

Enter The Size Of The Packet Entering At 1 sec
3
Second|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|
--------------------------------------------------------------------------------
    1      5      2               3           0
    2      4      2               3           2
    3      3      2               3           1
    4      0      2               1           0
    5      0      1               0           0
```

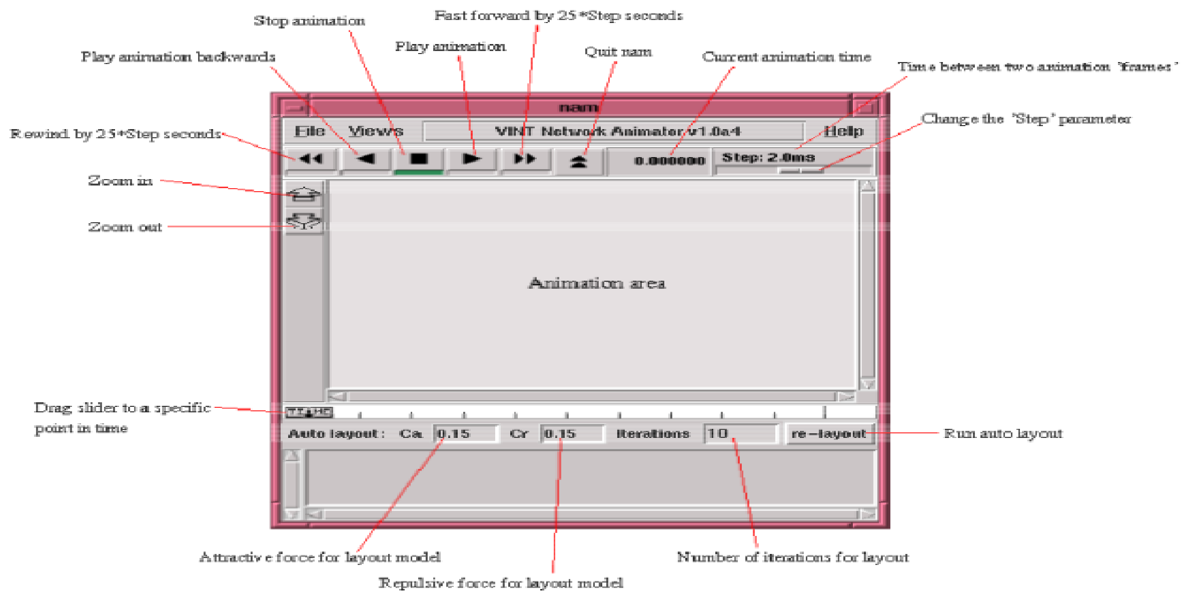
PART B:

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 behaviors.

Basic Architecture of NS2




Stop animation
Fast forward by 25*Step seconds
Play animation backwards
Play animation
Quit nam
Current animation time
Time between two animation 'frames'
Rewind by 25*Step seconds
Change the 'Step' parameter
Zoom in
Zoom out
nam
File
Views
VINT Network Animator v1.0a4
Help
0.000000
Step: 2.0ms
Animation area
Drag slider to a specific point in time
Auto layout:
Ca
0.15
Cr
0.15
Iterations
10
re-layout
Run auto layout
Attractive force for layout model
Repulsive force for layout model
Number of iterations for layout

Wired TCL Script Components

Create the event scheduler
Open new files & turn on the tracing
Create the nodes
Setup the links
Configure the traffic type (e.g., TCP, UDP, etc)
Set the time of traffic generation (e.g., CBR, FTP)
Terminate the simulation

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]
```

Which is thus the first line in the tcl script? 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 begins 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 a 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

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 are:

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

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of "node.port".
10. This is the destination address, given in the same form.
11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes
12. The last field shows the Unique id of the packet.

XGRAPH

The xgraph program draws a graph on an x-display given data read from either data file or from standard input if no files are specified. It can display upto 64 independent data sets using different colors and line styles for each set. It annotates the graph with a title, axis labels, grid lines or tick marks, grid labels and a legend.

Syntax:
Xgraph [options] file-name

Options are listed here

/-bd <color> (Border)
This specifies the border color of the xgraph window.
/-bg <color> (Background)
This specifies the background color of the xgraph window.
/-fg<color> (Foreground)
This specifies the foreground color of the xgraph window.
/-lf <fontname> (LabelFont)
All axis labels and grid labels are drawn using this font.
/-t<string> (Title Text)
This string is centered at the top of the graph.
/-x <unit name> (XunitText)
This is the unit name for the x-axis. Its default is "X".
/-y <unit name> (YunitText)
This is the unit name for the y-axis. Its default is "Y".

Awk- An Advanced

awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers.

awk is not just a command, but a programming language too. In other words, awk utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

Syntax:
awk option 'selection_criteria {action}' file(s)

Here, selection_criteria filters input and select lines for the action component to act upon. The selection_criteria is enclosed within single quotes and the action within the curly braces. Both the selection_criteria and action forms an awk program.

Example: $ awk '/manager/ {print}' emp.lst

Variables

Awk allows the user to use variables of there choice. You can now print a serial number, using the variable kount, and apply it those directors drawing a salary exceeding 6700:

```
$ awk –F"|" '$3 == "director" && $6 > 6700 {
kount =kount+1
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }' empn.lst
```

THE –f OPTION: STORING awk PROGRAMS IN A FILE

You should holds large awk programs in separate file and provide them with the awk extension for easier identification. Let's first store the previous program in the file empawk.awk:

$ cat empawk.awk

Observe that this time we haven't used quotes to enclose the awk program. You can now use awk with the –f *filename* option to obtain the same output:

Awk –F"|" –f empawk.awk empn.lst

The Begin And End Sections

Awk statements are usually applied to all lines selected by the address, and if there are no addresses, then they are applied to every line of input. But, if you have to print something before processing the first line, for example, a heading, then the BEGIN section can be used gainfully. Similarly, the end section useful in printing some totals after processing is over.

The BEGIN and END sections are optional and take the form

BEGIN {action}

END {action}

These two sections, when present, are delimited by the body of the awk program. You can use them to print a suitable heading at the beginning and the average salary at the end.

BUILT-IN VARIABLES

Awk has several built-in variables. They are all assigned automatically, though it is also possible for a user to reassign some of them. You have already used NR, which signifies the record number of the current line. We'll now have a brief look at some of the other variable.

The FS Variable: as stated elsewhere, awk uses a contiguous string of spaces as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

BEGIN {FS="|"}

This is an alternative to the –F option which does the same thing.

The OFS Variable: when you used the print statement with comma-separated arguments, each argument was separated from the other by a space. This is awk's default output field separator, and can reassigned using the variable OFS in the BEGIN section:

BEGIN { OFS="~" }

When you reassign this variable with a ~ (tilde), awk will use this character for delimiting the print arguments. This is a useful variable for creating lines with delimited fields.

The NF variable: NF comes in quite handy for cleaning up a database of lines that don't contain the right number of fields. By using it on a file, say emp.lst, you can locate those lines not having 6 fields, and which have crept in due to faulty data entry:

$awk 'BEGIN {FS = "|"}
NF! =6 {Print "Record No ", NR, "has", "fields"}' empx.lst

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

Program:

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

$ns color 2 red


#Tell the simulator to use static routing
$ns rtproto Static
#set up trace files
set traceFile [open 1.tr w]
$ns trace-all $traceFile
set namFile [open out1.nam w]
$ns namtrace-all $namFile
proc finish {} {
     global ns namFile traceFile
    $ns flush-trace
     #Close the trace files
   close $traceFile
   close $namFile
     #exec awk -f stats.awk 1.tr &
     exec nam out1.nam &
   exit 0
}

#Set up 3 nodes
set n(1) [$ns node]
set n(2) [$ns node]
set n(3) [$ns node]
#set up duplex links
$ns duplex-link $n(1) $n(2) 0.5Mb 20ms DropTail
$ns duplex-link $n(2) $n(3) 0.5Mb 20ms DropTail
$ns queue-limit $n(1) $n(2) 10
$ns queue-limit $n(2) $n(3) 10
#aesthetics
#source (udp)
$n(1) shape hexagon
$n(1) color red
#destination (udp)
$n(3) shape square
$n(3) color blue
#Create a UDP agent and attach it to node n(1)
set udp0 [new Agent/UDP]
```

```
$ns attach-agent $n(1) $udp0
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 512
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n(3)
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0
#Connect the traffic source with the traffic sink and assign flow id color
$ns connect $udp0 $null0
$udp0 set fid_ 2
#sim events
$ns at 0.5 "$cbr0 start"
$ns at 2.0 "$cbr0 stop"
$ns at 2.0 "finish"
```

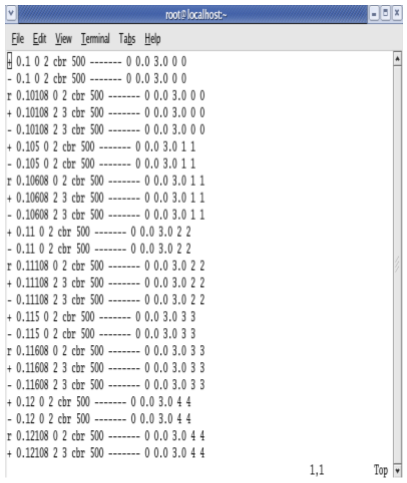
AWK Script:

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

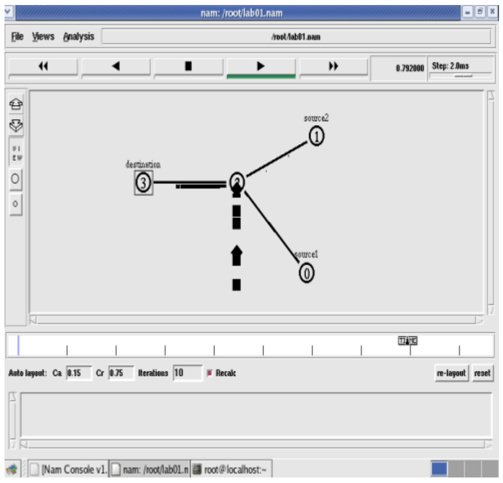
Steps for execution:

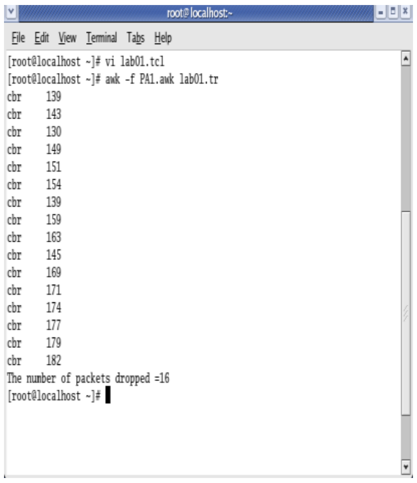
- *Open vi editor and type program. Program name should have the extension “ **.tcl** ”*
  [root@localhost ~]# vi lab1.tcl
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension “**.awk** ”*
  [root@localhost ~]# vi lab1.awk
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Run the simulation program*
  [root@localhost~]# ns lab1.tcl
- *Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.*
- *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output ,*
  [root@localhost~]# awk –f lab1.awk lab1.tr
- *To see the trace file contents open the file as ,*
  [root@localhost~]# vi lab1.tr

Output:



```
+ 0.1 0 2 cbr 500 ------- 0 0.0 3.0 0 0
- 0.1 0 2 cbr 500 ------- 0 0.0 3.0 0 0
r 0.10108 0 2 cbr 500 ------- 0 0.0 3.0 0 0
+ 0.10108 2 3 cbr 500 ------- 0 0.0 3.0 0 0
- 0.10108 2 3 cbr 500 ------- 0 0.0 3.0 0 0
+ 0.105 0 2 cbr 500 ------- 0 0.0 3.0 1 1
- 0.105 0 2 cbr 500 ------- 0 0.0 3.0 1 1
r 0.10608 0 2 cbr 500 ------- 0 0.0 3.0 1 1
+ 0.10608 2 3 cbr 500 ------- 0 0.0 3.0 1 1
- 0.10608 2 3 cbr 500 ------- 0 0.0 3.0 1 1
+ 0.11 0 2 cbr 500 ------- 0 0.0 3.0 2 2
- 0.11 0 2 cbr 500 ------- 0 0.0 3.0 2 2
r 0.11108 0 2 cbr 500 ------- 0 0.0 3.0 2 2
+ 0.11108 2 3 cbr 500 ------- 0 0.0 3.0 2 2
- 0.11108 2 3 cbr 500 ------- 0 0.0 3.0 2 2
+ 0.115 0 2 cbr 500 ------- 0 0.0 3.0 3 3
- 0.115 0 2 cbr 500 ------- 0 0.0 3.0 3 3
r 0.11608 0 2 cbr 500 ------- 0 0.0 3.0 3 3
+ 0.11608 2 3 cbr 500 ------- 0 0.0 3.0 3 3
- 0.11608 2 3 cbr 500 ------- 0 0.0 3.0 3 3
+ 0.12 0 2 cbr 500 ------- 0 0.0 3.0 4 4
- 0.12 0 2 cbr 500 ------- 0 0.0 3.0 4 4
r 0.12108 0 2 cbr 500 ------- 0 0.0 3.0 4 4
+ 0.12108 2 3 cbr 500 ------- 0 0.0 3.0 4 4
                                              1,1          Top
```





```
[root@localhost ~]# vi lab01.tcl
[root@localhost ~]# awk -f PA1.awk lab01.tr
cbr     139
cbr     143
cbr     130
cbr     149
cbr     151
cbr     154
cbr     139
cbr     159
cbr     163
cbr     145
cbr     169
cbr     171
cbr     174
cbr     177
cbr     179
cbr     182
The number of packets dropped =16
[root@localhost ~]#
```

2. **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.**

Program:

```
set ns [new Simulator]
set nf [open lab2.nam w]
$ns namtrace-all $nf
set tf [open lab2.tr w]
$ns trace-all $tf
proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab2.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 10Mb 1ms DropTail
$ns duplex-link $n1 $n2 10Mb 1ms DropTail
$ns duplex-link $n2 $n3 10Mb 1ms DropTail
set tcp0 [new Agent/TCP]                    # letters A,T,C,P are capital
$ns attach-agent $n0 $tcp0
set udp1 [new Agent/UDP]                    # letters A,U,D,P are capital
$ns attach-agent $n1 $udp1
set null0 [new Agent/Null]                  # letters A and N are capital
$ns attach-agent $n3 $null0
set sink0 [new Agent/TCPSink]           # letters A,T,C,P,S are capital
$ns attach-agent $n3 $sink0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$ns connect $tcp0 $sink0
$ns connect $udp1 $null0
$ns at 0.1 "$cbr1 start"
$ns at 0.2 "$ftp0 start"
$ns at 0.5 "finish"
$ns run
```

AWK Script:

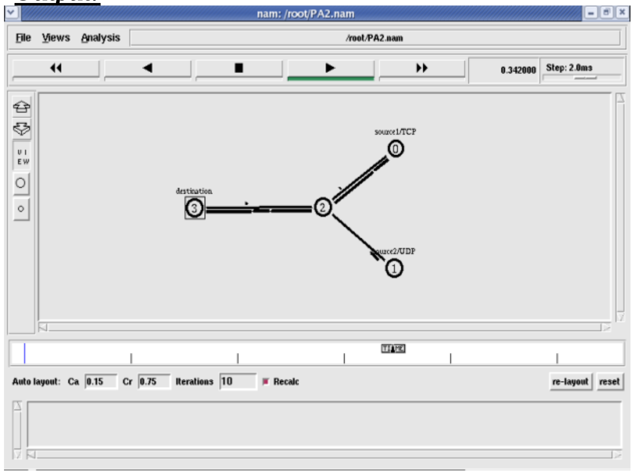
```
BEGIN{
```

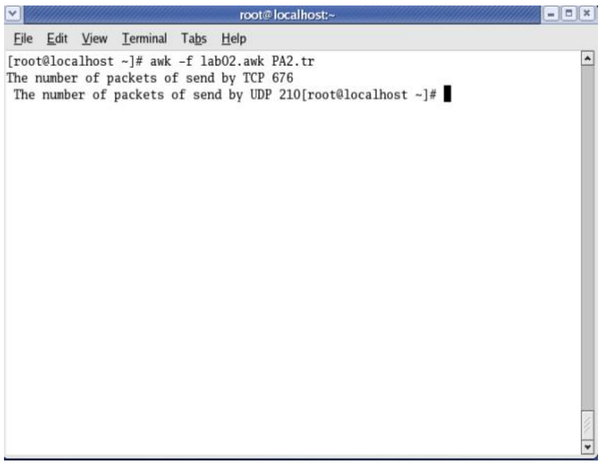
```
udp=0;
tcp=0;
}
{
if($1= = “r” && $5 = = “cbr”)
   {
           udp++;
    }
  else if($1 = = “r” && $5 = = “tcp”)
  {             tcp++;
   }
}
END{
printf(“Number of packets sent by TCP = %d\n”, tcp);
printf(“Number of packets sent by UDP=%d\n”,udp);
}
```

Steps for execution:

- *Open vi editor and type program. Program name should have the extension “ **.tcl** ”*
  [root@localhost ~]# vi lab2.tcl
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension “**.awk** ”*
  [root@localhost ~]# vi lab2.awk
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Run the simulation program **[root@localhost~]# ns lab2.tcl***
  - *Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.*
  - *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output , **[root@localhost~]# awk –f lab2.awk lab2.tr***
- *To see the trace file contents open the file as , **[root@localhost~]# vi lab2.tr***

Output:





```
[root@localhost ~]# awk -f lab02.awk PA2.tr
The number of packets of send by TCP 676
 The number of packets of send by UDP 210[root@localhost ~]#
```

3. **Simulate the study transmission of packets over Ethernet LAN and determine the number of packets drop destination.**

<u>Program:</u>

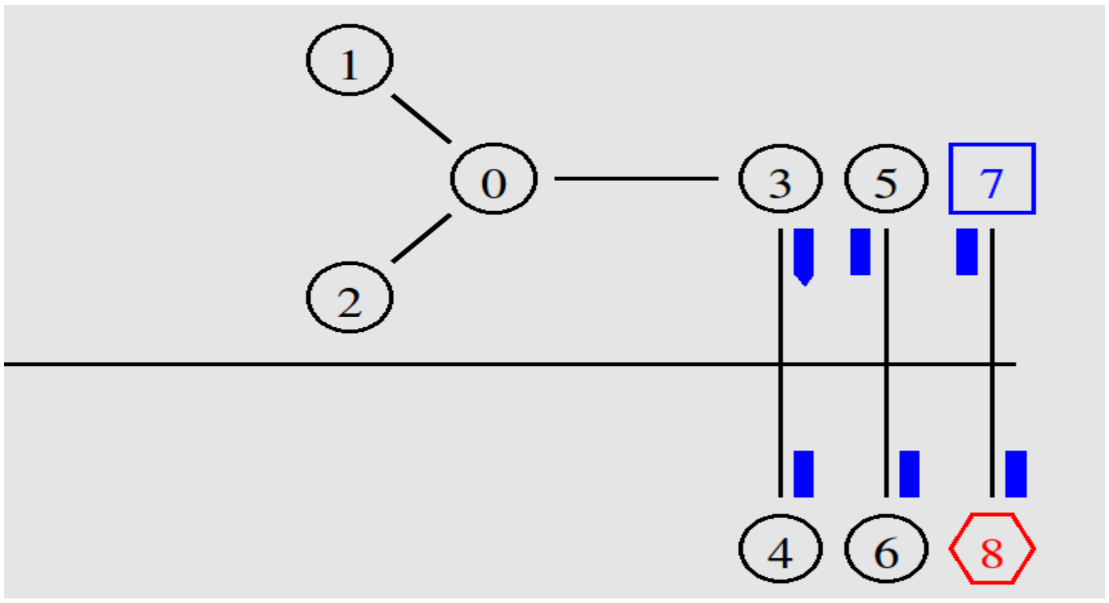
```
set ns [new Simulator]
set nf [open prog7.nam w]
$ns namtrace-all $nf
set nd [open prog7.tr w]
$ns trace-all $nd
$ns color 1 Blue
$ns color 2 Red
proc finish { } {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec nam prog7.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

$n7 shape box
$n7 color Blue
$n8 shape hexagon
$n8 color Red

$ns duplex-link $n1 $n0 2Mb 10ms DropTail
$ns duplex-link $n2 $n0 2Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 20ms DropTail

$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3

$ns duplex-link-op $n1 $n0 orient right-down
$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n3 orient right

$ns queue-limit $n0 $n3 20
```

```
set tcp1 [new Agent/TCP/Vegas]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1
$ns connect $tcp1 $sink1
$tcp1 set class_ 1
$tcp1 set packetsize_ 55

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

set tfile [open cwnd.tr w]
$tcp1 attach $tfile
$tcp1 trace cwnd_

set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n8 $sink2
$ns connect $tcp2 $sink2
$tcp2 set class_ 2
$tcp2 set packetSize_ 55

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

set tfile2 [open cwnd2.tr w]
$tcp2 attach $tfile2
$tcp2 trace cwnd_

$ns at 0.5 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp1 stop"

$ns at 5.5 "finish"
$ns run
```

<u>Steps for execution:</u>

- *Open vi editor and type program. Program name should have the extension " **.tcl** "*
  [root@localhost ~]# vi lab3.tcl
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Run the simulation program **[root@localhost~]# ns lab2.tcl***
  - *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
  - *Now press the play button in the simulation window and the simulation will begins.*

<u>Output:</u>



4. **Write a TCL Script to simulate working of multicasting routing protocol and analyze the throughput of the network.**

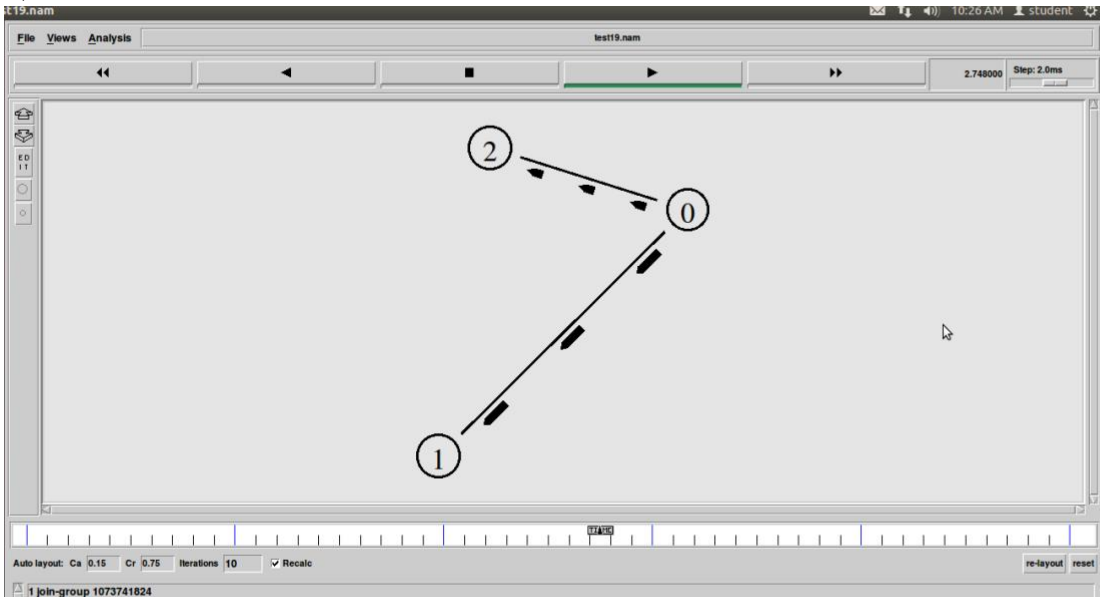
TCL Script:

```
set ns [new Simulator -multicast on] ;                    # enable multicast routing
set trace [open lab4.tr w]
$ns trace-all $trace
set namtrace [open lab4.nam w]
$ns namtrace-all $namtrace
set group [Node allocaddr] ;                      # allocate a multicast address
set node0 [$ns node] ;                            # create multicast capable nodes
set node1 [$ns node]
set node2 [$ns node]
$ns duplex-link $node0 $node1 1.5Mb 10ms DropTail
$ns duplex-link $node0 $node2 1.5Mb 10ms DropTail
set mproto DM ;                                           # configure multicast protocol
set mrthandle [$ns mrtproto $mproto] ;                    # all nodes will contain multicast protocol
agents
set udp [new Agent/UDP] ;                         # create a source agent at node 0
$ns attach-agent $node0 $udp
set src [new Application/Traffic/CBR]
$src attach-agent $udp
$udp set dst_addr_ $group
$udp set dst_port_ 0
set rcvr [new Agent/LossMonitor] ;                        # create a receiver agent at node 1
$ns attach-agent $node1 $rcvr
$ns at 0.3 "$node1 join-group $rcvr $group" ;             # join the group at simulation time 0.3 (sec)
set rcvr2 [new Agent/LossMonitor] ;               # create a receiver agent at node 1
$ns attach-agent $node2 $rcvr2
$ns at 0.3 "$node2 join-group $rcvr2 $group" ;            # join the group at simulation time 0.3 (sec)
$ns at 3.3 "$node2 leave-group $rcvr2 $group" ;           # join the group at simulation time 0.3 (sec)
$ns at 2.0 "$src start"
$ns at 5.0 "$src stop"
proc finish {} {
global ns namtrace trace
$ns flush-trace
close $namtrace ; close $trace
exec nam lab4.nam &
exit 0
}
$ns at 10.0 "finish"
$ns run
```

<u>Steps for execution:</u>

- *Open vi editor and type program. Program name should have the extension “ **.tcl** ”*
  [root@localhost ~]# vi lab4.tcl
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension “**.awk** ”*
  [root@localhost ~]# vi lab4.awk
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Run the simulation program **[root@localhost~]# ns lab4.tcl***
  - *Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.*
  - *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output , **[root@localhost~]# awk –f lab4.awk lab4.tr***
- *To see the trace file contents open the file as , **[root@localhost~]# vi lab4.tr***

OUTPUT:



```
Auto layout: Ca 0.15  Cr 0.75  Iterations 10  ✓ Recalc
1 join-group 1073741824
```

5. **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.**

TCL Script:

```
set ns [new Simulator]
#Open a new file for NAMTRACE
set nf [open lab5.nam w]
$ns namtrace-all $nf
#Open a new file to log TRACE
set tf [open lab5.tr w]
$ns trace-all $tf
#Body of the 'finish' procedure
proc finish {} {
      global ns nf tf
      $ns flush-trace
      close $nf
            close $tf
      exec nam lab5.nam &
      exec awk -f lab5.awk lab5.tr &
      exit 0
}
#Create Nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#Create Links between Nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
#Set the queue limit - default is 50 packets
$ns queue-limit $n0 $n2 50
$ns queue-limit $n1 $n2 50
$ns queue-limit $n2 $n3 50
#Create TCP Agent between node 0 and node 3
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
#Create FTP Application for TCP Agent
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
#Specify TCP packet size
Agent/TCP set packetSize_ 1000
#Create TCP Agent between node 1 and node 3
set tcp1 [new Agent/TCP]
```

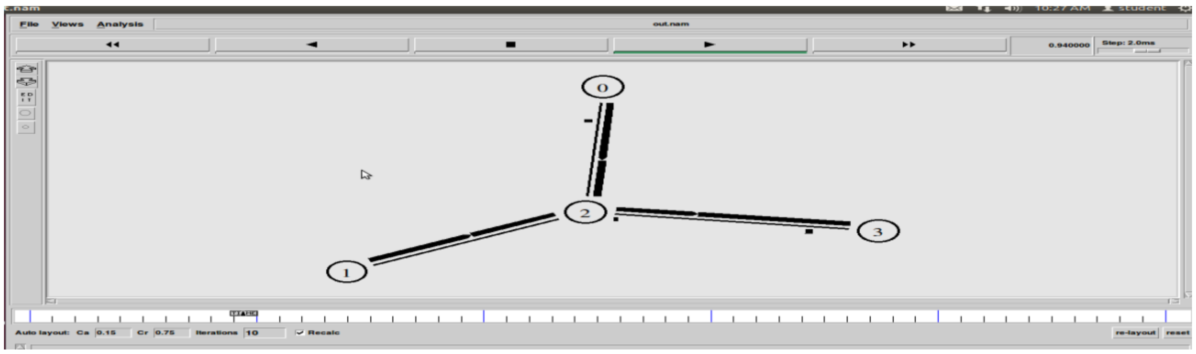
```
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
#Create Telnet Application for TCP Agent
set telnet0 [new Application/Telnet]
$telnet0 set interval_ 0.005
$telnet0 attach-agent $tcp1
#Start and Stop FTP Traffic
$ns at 0.75 "$ftp0 start"
$ns at 4.75 "$ftp0 stop"
#Start and Stop Telnet traffic
$ns at 0.5 "$telnet0 start"
$ns at 4.5 "$telnet0 stop"
#Stop the simulation
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

<u>Steps for execution:</u>

- *Open vi editor and type program. Program name should have the extension “ **.tcl** ”*
  [root@localhost ~]# vi lab5.tcl
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension “**.awk** ”*
  [root@localhost ~]# vi lab5.awk
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Run the simulation program **[root@localhost~]# ns lab5.tcl***
  - *Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.*
  - *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output , **[root@localhost~]# awk –f lab4.awk lab5.tr***

*To see the trace file contents open the file as , **[root@localhost~]# vi lab5.tr***

OUTPUT:



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

TCL Script:

```
set ns [new Simulator]
set nf [open lab6.nam w]
$ns namtrace-all $nf
set nd [open lab6.tr w]
$ns trace-all $nd
proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec nam lab6.nam &
exit 0
}
set n0 [$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]
$ns duplex-link $n1 $n0 1Mb 10ms DropTail
$ns duplex-link $n2 $n0 1Mb 10ms DropTail
$ns duplex-link $n3 $n0 1Mb 10ms DropTail
$ns duplex-link $n4 $n0 1Mb 10ms DropTail
$ns duplex-link $n5 $n0 1Mb 10ms DropTail
$ns duplex-link $n6 $n0 1Mb 10ms DropTail
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] recieved ping answer from \
$from with round-trip-time $rtt ms."
}
set p1 [new Agent/Ping]
set p2 [new Agent/Ping]
set p3 [new Agent/Ping]
set p4 [new Agent/Ping]
set p5 [new Agent/Ping]
set p6 [new Agent/Ping]
$ns attach-agent $n1 $p1
$ns attach-agent $n2 $p2
$ns attach-agent $n3 $p3
$ns attach-agent $n4 $p4
$ns attach-agent $n5 $p5
$ns attach-agent $n6 $p6
```

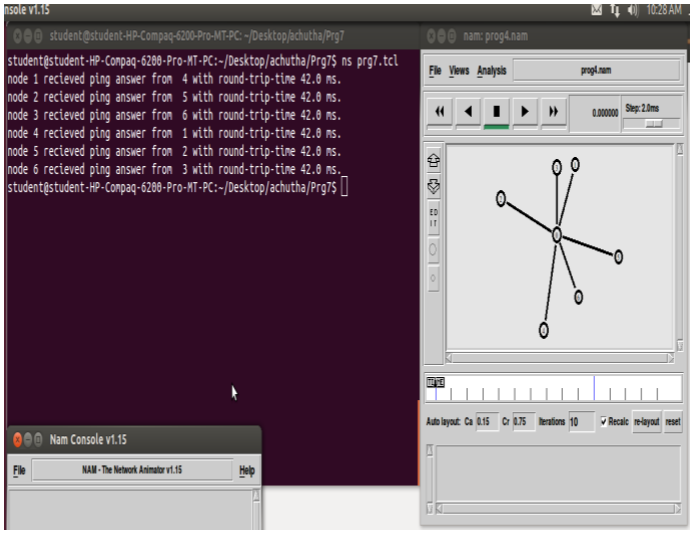
```
$ns queue-limit $n0 $n4 3
$ns queue-limit $n0 $n5 2
$ns queue-limit $n0 $n6 2
$ns connect $p1 $p4
$ns connect $p2 $p5
$ns connect $p3 $p6
$ns at 0.2 "$p1 send"
$ns at 0.4 "$p2 send"
$ns at 0.6 "$p3 send"
$ns at 1.0 "$p4 send"
$ns at 1.2 "$p5 send"
$ns at 1.4 "$p6 send"
$ns at 2.0 "finish"
$ns run
```

Steps for execution:

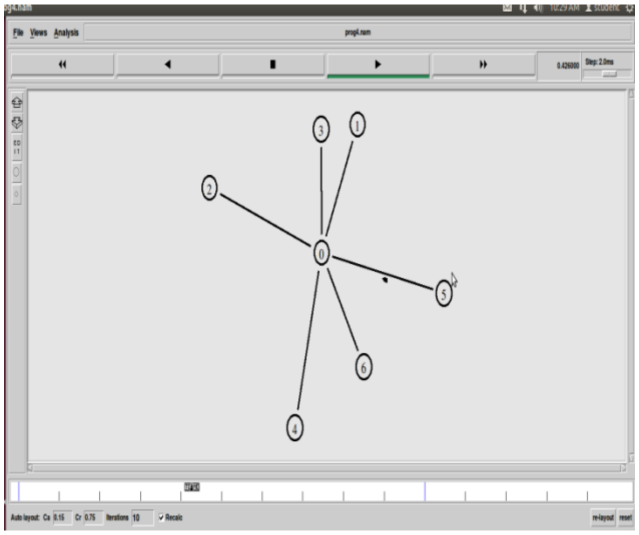
- *Open vi editor and type program. Program name should have the extension " **.tcl** "*
  [root@localhost ~]# vi lab6.tcl
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension "**.awk** "*
  [root@localhost ~]# vi lab6.awk
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Run the simulation program **[root@localhost~]# ns lab6.tcl***
  - *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
  - *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output , **[root@localhost~]# awk –f lab4.awk lab6.tr***

*To see the trace file contents open the file as , **[root@localhost~]# vi lab6.tr***

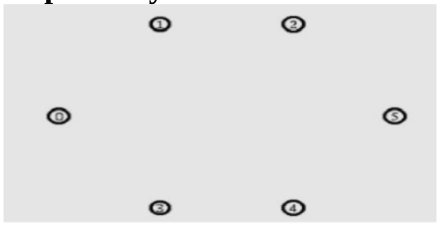
OUTPUT:



```
student@student-HP-Compaq-6200-Pro-MT-PC:~/Desktop/achutha/Prg7$ ns prg7.tcl
node 1 recieved ping answer from  4 with round-trip-time 42.0 ms.
node 2 recieved ping answer from  5 with round-trip-time 42.0 ms.
node 3 recieved ping answer from  6 with round-trip-time 42.0 ms.
node 4 recieved ping answer from  1 with round-trip-time 42.0 ms.
node 5 recieved ping answer from  2 with round-trip-time 42.0 ms.
node 6 recieved ping answer from  3 with round-trip-time 42.0 ms.
student@student-HP-Compaq-6200-Pro-MT-PC:~/Desktop/achutha/Prg7$
```



7. **Write a TCL script to simulate the following scenario with ns2 simulator. Consider six nodes, (as shown in the figure below) moving within a flat topology of 700m x 700m. The initial positions of nodes are 0 (150,300) ,1 (300,500),2 (500,500),3 (300,100),4(500,100) and 5(650,300) respectively.**



A TCP connection is initiated between node 0 (source) and node 5 (destination) through node 3 and node 4 i.e the route is 0-3-4-5. At time t = 3 seconds the FTP application runs over it. After time t=4.0 sec, node 3 (300,100) moves towards node 1 (300,500) with a speed of 5.0m/sec and after some time the path break, then the data transmit with a new path via node 1 and node 2 i.e the new route 0-1-2-5. The simulation lasts for 60 secs. In the above said case both the route has equal cost. Use DSR as the routing protocol and the IEEE 802.11 MAC protocol.

TCL Script:

```
#           Setting the Default Parameters                  #
set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)                    Mac/802_11
set val(ifq)            CMUPriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(x)              700
set val(y)              700
set val(ifqlen)         50
set val(nn)             6
set val(stop)           60.0
set val(rp)                     DSR
#           Creating New Instance of a Scheduler            #
set ns_         [new Simulator]
#               Creating Trace files                        #
set tracefd     [open ex_07.tr w]
$ns_ trace-all $tracefd
#             Creating NAM Trace files                      #
set namtrace [open ex_07.nam w]
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
set prop        [new $val(prop)]
set topo        [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)
#               Node Configuration                          #
```

```
    $ns_ node-config -adhocRouting $val(rp) \
                     -llType $val(ll) \
                     -macType $val(mac) \
                     -ifqType $val(ifq) \
                     -ifqLen $val(ifqlen) \
                     -antType $val(ant) \
                     -propType $val(prop) \
                     -phyType $val(netif) \
                     -channelType $val(chan) \
                     -topoInstance $topo \
                     -agentTrace ON \
                     -routerTrace ON \
                     -macTrace ON

#           Creating Nodes                          #
for {set i 0} {$i < $val(nn)} {incr i} {
   set node_($i) [$ns_ node]
   $node_($i) random-motion 0
}
$node_(0) set X_ 150.0
$node_(0) set Y_ 300.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 300.0
$node_(1) set Y_ 500.0
$node_(1) set Z_ 0.0
$node_(2) set X_ 500.0
$node_(2) set Y_ 500.0
$node_(2) set Z_ 0.0
$node_(3) set X_ 300.0
$node_(3) set Y_ 100.0
$node_(3) set Z_ 0.0
$node_(4) set X_ 500.0
$node_(4) set Y_ 100.0
$node_(4) set Z_ 0.0
$node_(5) set X_ 650.0
$node_(5) set Y_ 300.0
$node_(5) set Z_ 0.0
#           Initial Positions of Nodes          #
for {set i 0} {$i < $val(nn)} {incr i} {
      $ns_ initial_node_pos $node_($i) 60  ;# initial node position and size of the node
}
#           Topology Design                         #
$ns_ at 4.0 "$node_(3) setdest 300.0 500.0 5.0"
#           Generating Traffic                  #
set tcp0 [new Agent/TCP]
set sink0 [new Agent/TCPSink]
```

```
$ns_ attach-agent $node_(0) $tcp0
$ns_ attach-agent $node_(5) $sink0
$ns_ connect $tcp0 $sink0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns_ at 3.0 "$ftp0 start"
$ns_ at 59.0 "$ftp0 stop"
#                    Simulation Termination                                        #
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at $val(stop) "$node_($i) reset";
}
$ns_ at $val(stop) "puts \"NS EXITING...\" ; $ns_ halt"
puts "Starting Simulation..."
$ns_ run
exit(0)
```

<u>Steps for execution:</u>

- *Open vi editor and type program. Program name should have the extension “ **.tcl** ”*
  [root@localhost ~]# vi lab7.tcl
- *Save the program by pressing **“ESC key”** first, followed by **“Shift and :”** keys simultaneously and type **“wq”** and press **Enter key**.*
- *Run the simulation program **[root@localhost~]# ns lab7.tcl***
  - *Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.*
  - *Now press the play button in the simulation window and the simulation will begins.*

OUTPUT: