	SUBJECT TITLE: **NETWORKS LABORATORY**		
	SUBJECT CODE: CSL58	**No. of Credits:**1.5-0-0	**No. of Lecture hours per week:3**
	Exam Duration :3 hours	**Exam Marks: 50**	**Total No. of Lecture hours:**

Course Objectives:

1. To understand the fundamental concepts of simulation of communication networks

2. To evaluate the UDP, TCP protocols through simulation

3. To analyze the algorithms for congestion control, shortest path routing, error checking and correction

4. To understand and evaluate the parameters to be configured for wired and wireless communication.

5. To apply socket programming and implement client-server communication.

	PART – A
1	Simulate a three nodes point-to-point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.
2	Simulate a four node point-to-point network with the links connected as follows: n0 – n2, n1 – n2 and n2 – n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP / UDP.
3	Simulate an Ethernet LAN using n nodes (6-10), change error rate and data rate and compare throughput.
4	Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and determine collision across different nodes.
5.	Simulate simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.
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.
	PART – B
7	Implement the following in C/C++: Write a program for error detecting code using CRC-CCITT (16- bits).
8	Write a program for distance vector algorithm to find suitable path for transmission.
9	Write a program for congestion control using leaky bucket algorithm.
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Link state algorithm.
11	Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present

Course Outcomes:

CO1: Understand the simulation of communication networks and measure and evaluate the error rate, throughput, data rate, packet drop.

CO2: Understand and analyze the transport layer protocols.

CO3: Analyze the algorithms for congestion control, shortest path routing, error checking and correction.

CO4: Evaluate the parameters to be configured for wired and wireless communication

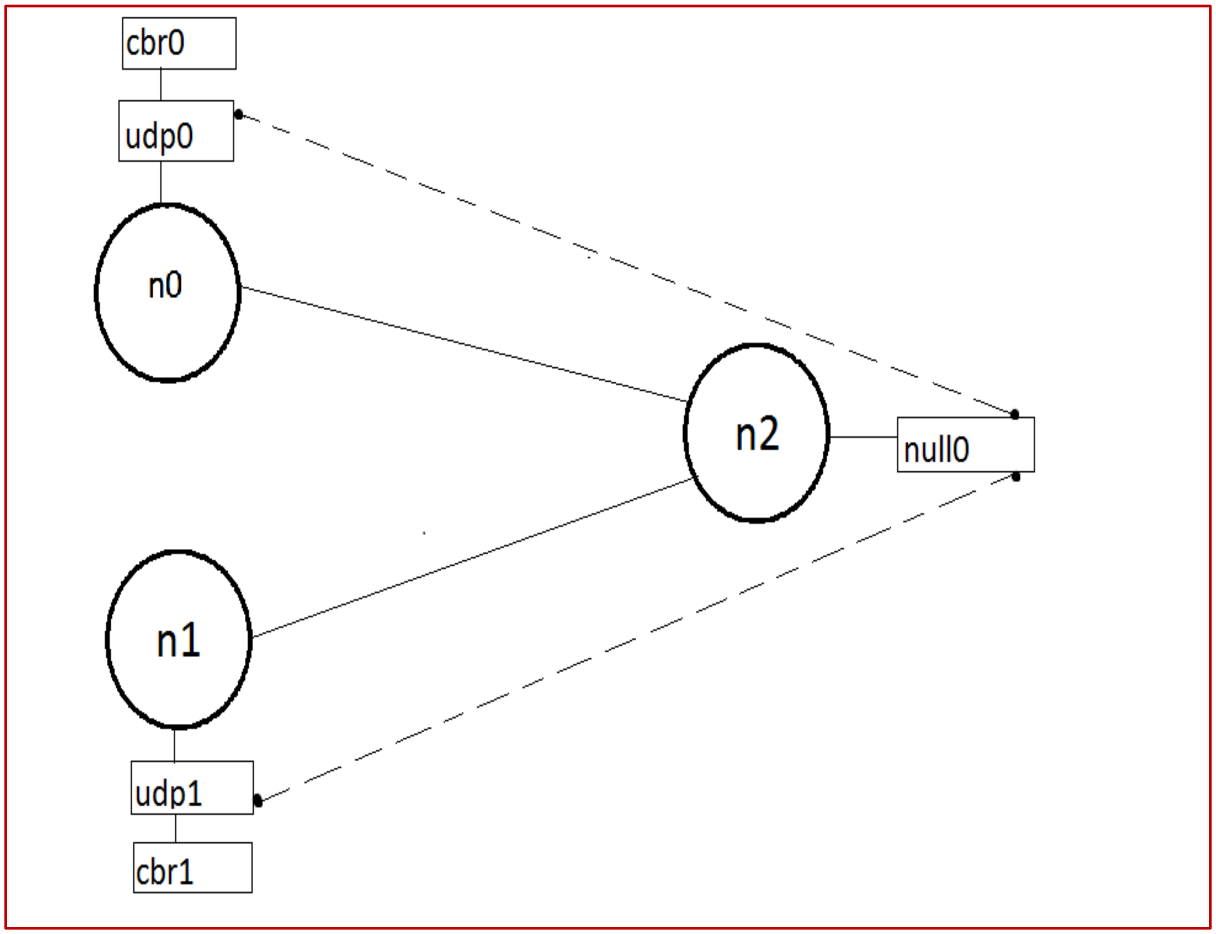
CO5: Apply the knowledge of socket programming.

Note:
Simulation Exercises:
Experiments 1 to 6 shall be conducted using either NS228/OPNET or any other suitable simulator

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

PART-A

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.



```
set ns  [ new Simulator ]

set tf  [ open lab1.tr w ]

$ns trace-all $tf
```

set nf [open lab1.nam w]

$ns namtrace-all $nf

# The below code is used to create the nodes.

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

#This is used to give color to the flow of packets.

$ns color 1 "red"

$ns color 2 "blue"

$n0 label "Source/udp0"

$n1 label "Source/udp1"

$n2 label "destination"

#providing the link

$ns duplex-link $n0 $n2 10kb 100ms DropTail

$ns duplex-link $n1 $n2 10kb 10ms DropTail

# set the queue size b/w the nodes

$ns set queue-limit $n0 $n2 5

$ns set queue-limit $n1 $n2 5

set udp0 [new Agent/UDP]

set udp1 [new Agent/UDP]

$ns attach-agent $n0 $udp0

$ns attach-agent $n1 $udp1

set cbr0 [new Application/Traffic/CBR]

set cbr1 [new Application/Traffic/CBR]

$cbr0 attach-agent $udp0

```
$cbr1 attach-agent $udp1

set null0  [new Agent/Null]

$ns attach-agent $n2 $null0
```

#The below code sets the udp0 packets to red and udp1 packets to blue color

```
$udp0 set class_ 1

$udp1 set class_ 2
```

#The below code is used to connect the agents.

```
$ns connect $udp0 $null0

$ns connect $udp1 $null0
```

#The below code is used to set the packet size to 500

```
$cbr0 set packetSize_ 500Mb

$cbr1 set packetSize_ 500Mb
```

#The below code is used to set the interval of the packets,

```
$cbr0 set interval_ 0.01

$cbr1 set interval_ 0.01

proc finish  { }  {

global  ns n f  tf

$ns flush-trace

exec nam lab1.nam &

close $tf

close $nf

exit 0

}

$ns at 0.1 "$cbr0 start"

$ns at 0.1 "$cbr1 start"
```

```
$ns at 9.5 "$cbr0 stop"

$ns at 10.0 "$cbr1 stop"

$ns at 10.0 "finish"

$ns run
```

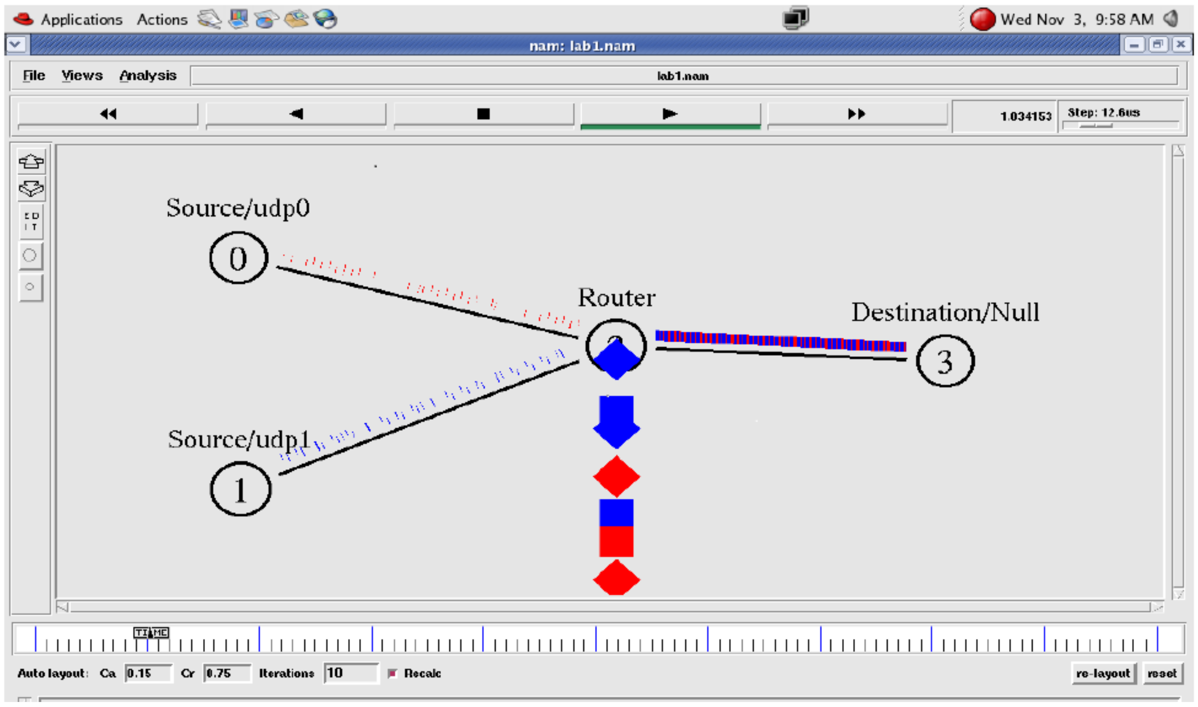
AWK Script (lab1.awk)

```
BEGIN{
count=0;
}
{
if($1=="d")
count++
}
END{
printf("The Total no of Packets Dropped due to Congestion : %d\n\n", count)
}
```

Output:

```
ns lab1.tcl
awk –f lab1.awk lab1.tr
The Total no of packets Dropped due to congestion:4560
```

Snapshot:

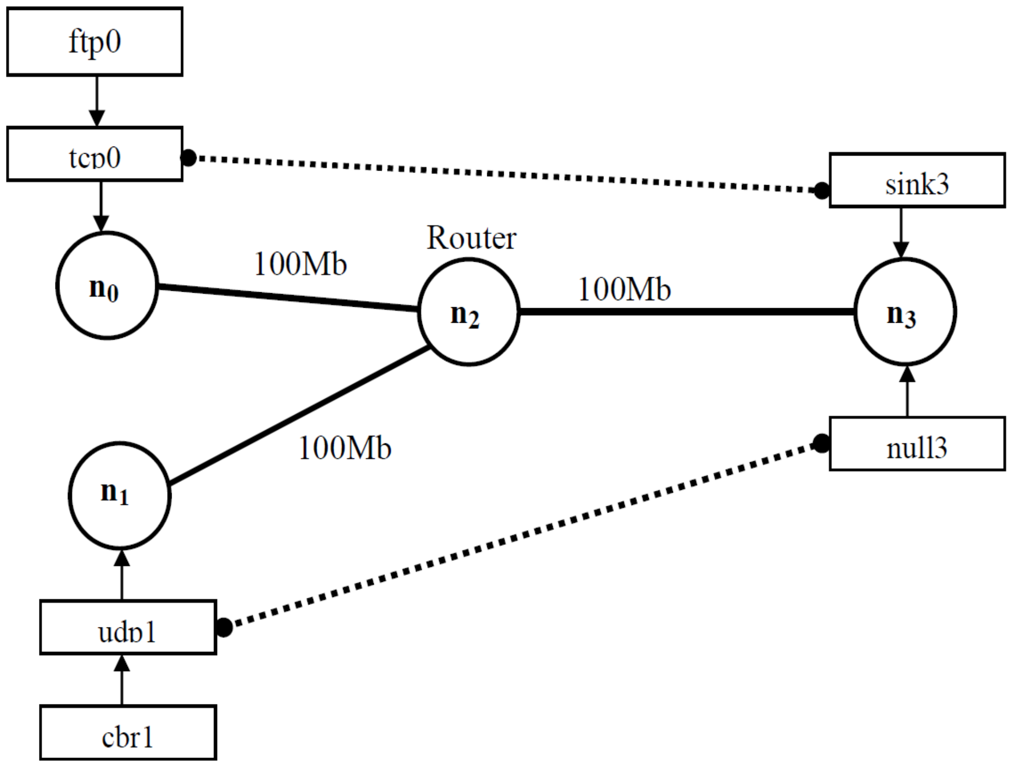


Note:

1. Find the total number of packets dropped by changing the bandwidth as shown below in the table.

BANDWIDTH	PACKETS DROPPED
10kb	
100kb	
1Mb	
10Mb	
100Mb	

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



Example:

Note:

2. Find the total number of packets sent by TCP and UDP by changing the bandwidth 100Mb, 200Mb in the topology and the data rate 0.01, 0.001 as shown below in the table.

BANDWIDTH	PACKET INTERVAL	TCP PACKETS SENT	UDP PACKETS SENT
100Mb	0.01		
100Mb	0.001		
200Mb	0.01		
200Mb	0.001		

Program
(lab2.tcl)

```
set ns  [new Simulator]
set tf  [open lab2.tr w]
$ns trace-all $tf
set  nf  [open lab2.nam w ]
$ns namtrace-all $nf
set n0  [$ns node]
set n1  [$ns node]
set n2  [$ns node]
set n3  [$ns node]

# The below code is used to set the color and name's to the nodes.
$ns color 1  "red"
$ns color 2  "blue"
$n0 label   "Source/TCP"
$n1 label   "Source/UDP"
$n2 label   "Router"
$n3 label   "Destination"
$ns  duplex-link  $n0  $n2 100Mb 1ms  DropTail
$ns  duplex-link  $n1  $n2 100Mb 1ms  DropTail
$ns  duplex-link  $n2  $n3 100Mb 1ms  DropTail

# The below code is used to set the color and labels to the links.
$ns  duplex-link-op  $n0  $n2  color  "green"
$ns  duplex-link-op  $n0  $n2  label  "from 0-2"
$ns  duplex-link-op  $n1  $n2  color  "green"
$ns  duplex-link-op  $n1  $n2  label  "from 1-2"
$ns  duplex-link-op  $n2  $n3  color  "green"
$ns  duplex-link-op  $n2  $n3  label  "from 2-3"

# The below code is used create TCP and UDP agents and the
# traffic ftp & cbr respectively.

set tcp0  [new Agent/TCP]
$ns attach-agent $n0 $tcp0
```

```
set ftp0  [new Application/FTP]
$ftp0  attach-agent  $tcp0
set sink3  [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
set udp1  [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1  [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set null3  [new Agent/Null]
$ns attach-agent $n3 $null3

#The below code is used to set the packet size of ftp and udp.
$ftp0 set packetSize_  500
$ftp0 set interval_  0.001

#The below code is used set  the data rate
$cbr1 set  packetSize_   500
$cbr1 set  interval_  0.001

#This code is used give a color red->tcp and blue ->udp.
$tcp0 set class_  1
$udp1 set class_  2

# The below code is used connect the agents.
$ns connect  $tcp0  $sink3
$ns connect  $udp1 $null3

proc finish { } {
global ns nf tf
$ns flush-trace
exec nam lab2.nam &
close $nf
close $tf
exit 0
}
$ns at 0.1 "$cbr1 start"
$ns at 0.2 "$ftp0 start"
$ns at 5.0 "finish"
$ns run
```

AWK Script (lab2.awk)

```
BEGIN{
tcp=0;
udp=0;
}
{
if($1=="r"&&$3=="2"&&$4=="3"&& $5=="tcp")
tcp++;
if($1=="r"&&$3=="2"&&$4=="3"&&$5=="cbr")
udp++;
}
END{
printf("\n Total number of packets sent by TCP : %d\n",tcp);
printf("\n Total number of packets sent by UDP : %d\n",udp);
}
```

Execution of the program

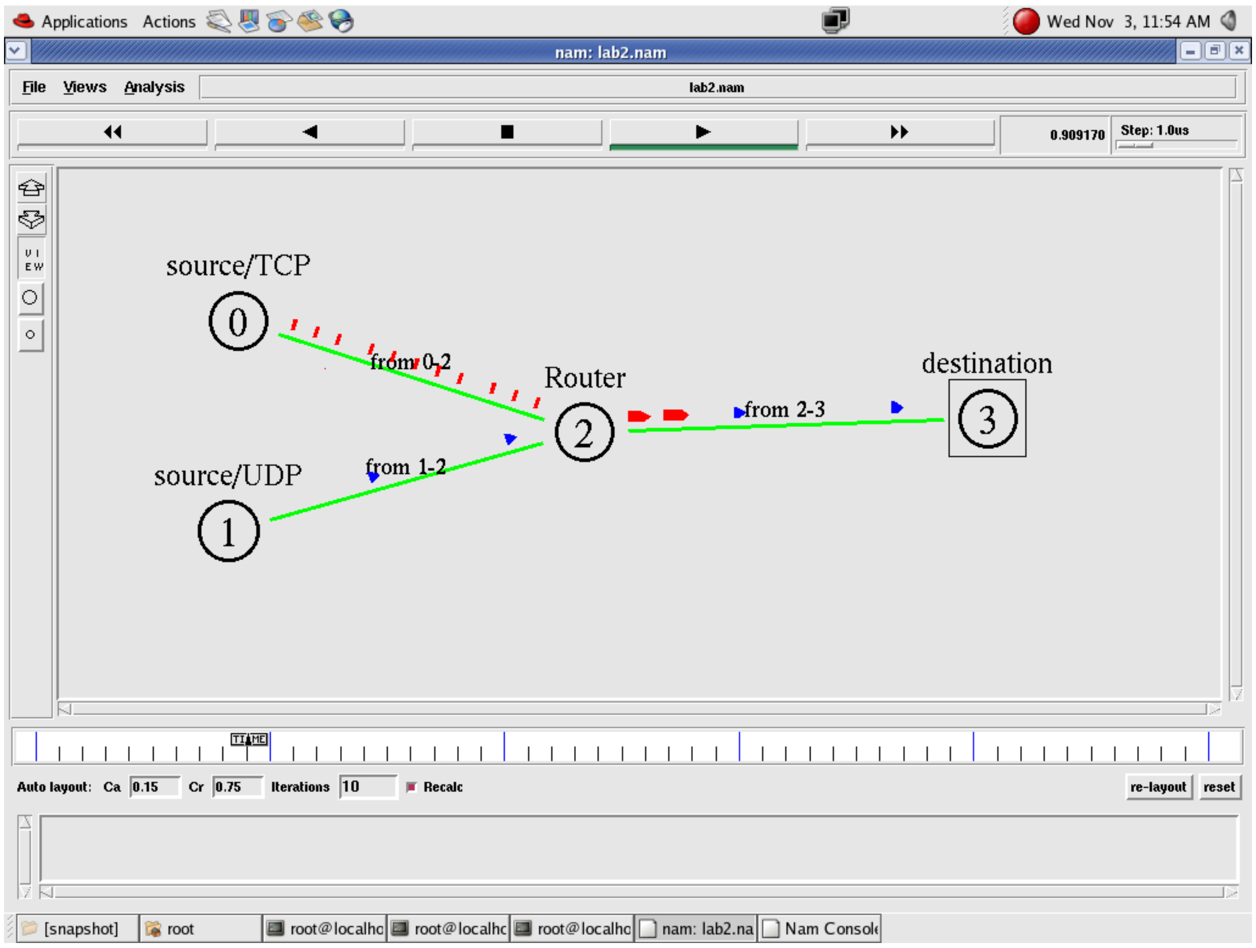
Type the following commands in the terminal window for executing the programs.

```
ns lab2.tcl
awk –f  lab2.awk  lab2.tr
```

Output:

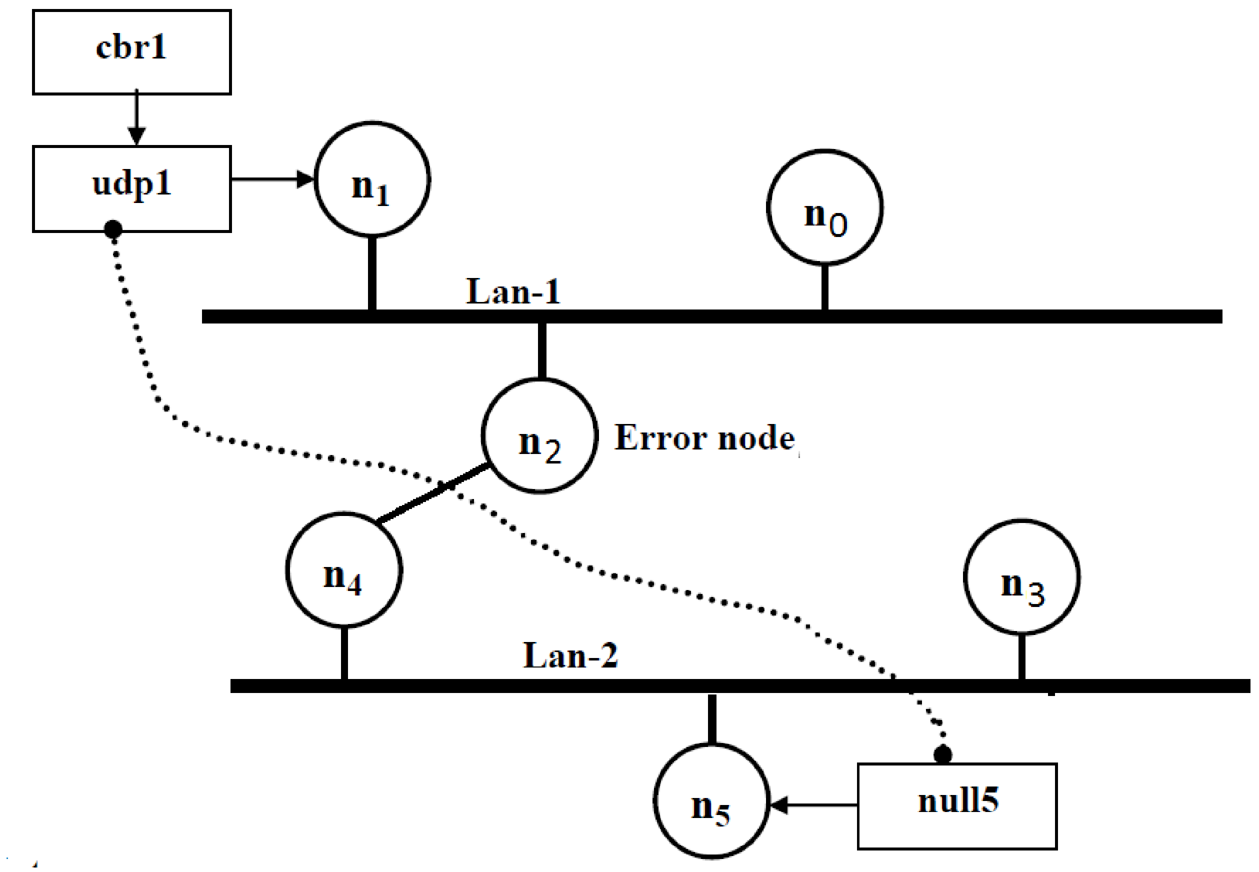
```
Total number of packets sent by TCP : 1200
Total number of packets sent by UDP : 4500
```

Snapshot:


Applications
Actions
Wed Nov 3, 11:54 AM
nam: lab2.nam
File
Views
Analysis
lab2.nam
0.909170
Step: 1.0us
source/TCP
0
from 0-2
Router
2
destination
3
from 2-3
source/UDP
1
from 1-2
TIME
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
Recalc
re-layout
reset
[snapshot]
root
root@localho
root@localho
root@localho
nam: lab2.na
Nam Console

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

Design:



Note:

1. The lan can be created by using the command:

$ns make-lan **"**$n0 $n1 $n2**"** 100Mb 10ms LL Queue/DropTail Mac/802_3

2. The Error between the nodes n2 and n4 can be added as follows:

Set err [new ErrorModel]

$ns lossmodel $err $n2 $n4

$err set rate_ 0.1   # used to set error rate.

3. The throughput can analyzed by changing the data rate and error rate as shown below.

i. First, fix the data rate to 0.001 and vary the error rate, as below. Tabulate the throughput.
ii. Secondly, fix the error rate to 0.1 and vary the data rate, as below. Tabulate the throughput.

Example:

Error rate	Data rate	Throughput
0.1	0.001	
0.2	0.001	
0.3	0.001	
0 .4	0.001	

Error rate	Data rate	Throughput
0.1	0.1	
0.1	0.01	
0.1	0.001	
0.1	0.0001	

Program (lan.tcl)

```
set ns  [new Simulator]
set tf  [open lan.tr w]
$ns trace-all $tf
set nf  [open lan.nam w]
$ns namtrace-all $nf
set n0  [$ns node]
set n1  [$ns node]
set n2  [$ns node]
set n3  [$ns node]
set n4  [$ns node]
set n5  [$ns node]
$ns color 1 "blue"
$n1 label "Source"
$n2 label "Error node"
$n5 label "Destination"
```

#The below code is used to create two Lans (Lan1 and Lan2).

```
$ns make-lan "$n0 $n1 $n2" 10Mb 10ms LL Queue/DropTail  Mac/802_3
$ns make-lan "$n3 $n4 $n5" 10Mb 10ms LL Queue/DropTail  Mac/802_3
```

# connect node n2 and n4

```
$ns duplex-link $n2 $n4 100Mb 10ms  DropTail
$ns duplex-link-op $n2 $n4 color  "green"
set udp1  [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1  [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set null5  [new Agent/Null]
$ns attach-agent $n5 $null5
$ns connect $udp1 $null5
```

#data rate- change this to change the data rate

```
$cbr1 set packetSize_ 1000
$cbr1 set interval_ 0.001
$udp1 set class_ 1
```

# The below code is used to add an error model between the nodes n2 and n4.

```
set err  [new ErrorModel]
$ns lossmodel $err $n2 $n4
$err set rate_ 0.1
proc finish {} {
        global ns tf nf
        $ns flush-trace
        exec nam lan.nam &
        close $tf
        close $nf
        exit 0
}
$ns at 0.1 "$cbr1 start"
$ns at 6.0 "finish"
$ns run
```

AWK Script (lan.awk)

```
BEGIN{
        pkt=0;
        time=0;
}
{
        if($1=="r"&& $9=="1.0" && $10=="5.0")
        {
                pkt=pkt+$6;
                time=$2;
        }
}
END{
        printf("Throughput: %fMbps \n\n",(pkt/time)*(8/1000000));
}
```

Execution of the program

Type the following commands in the terminal window for executing the programs.

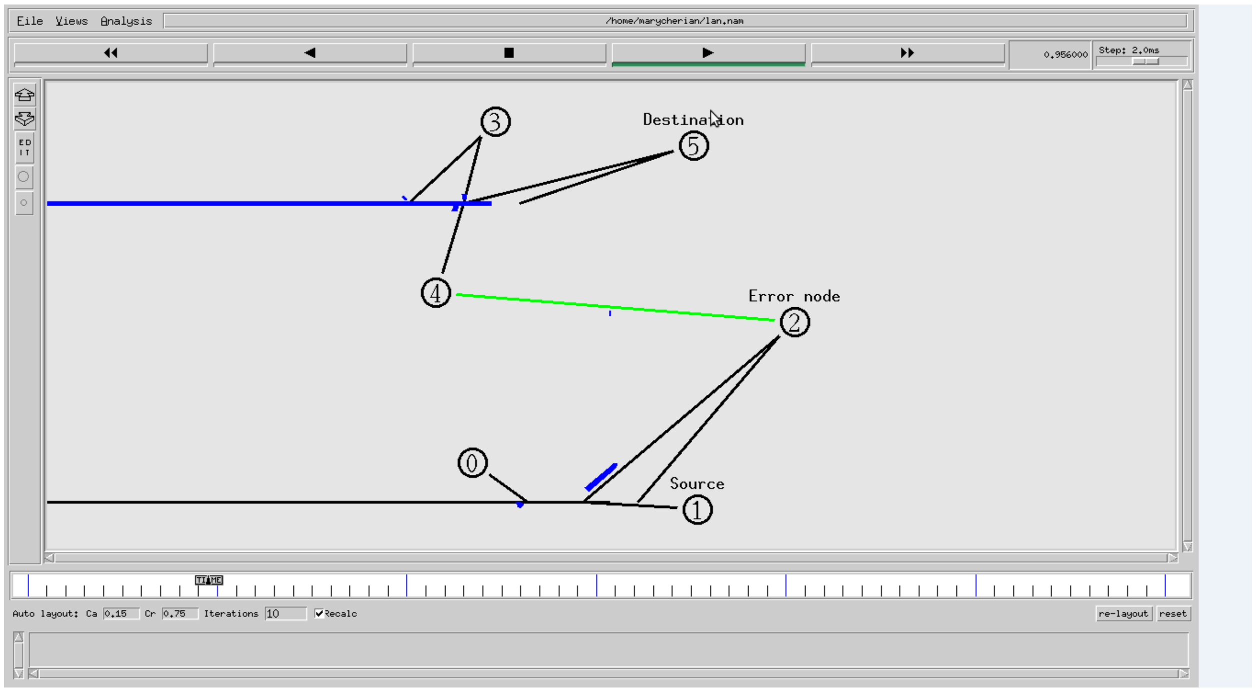
```
ns lan.tcl
awk –f lan.awk  lan.tr
```

Output:

```
Throughput : 90.3Mbps
```

Snapshot:

Snap shot


File
Views
Analysis
/home/marycherian/lan.nam
0.956000
Step: 2.0ms
Destination
Error node
Source
0
1
2
3
4
5
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset

4. Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and determine collision across different nodes.

Program

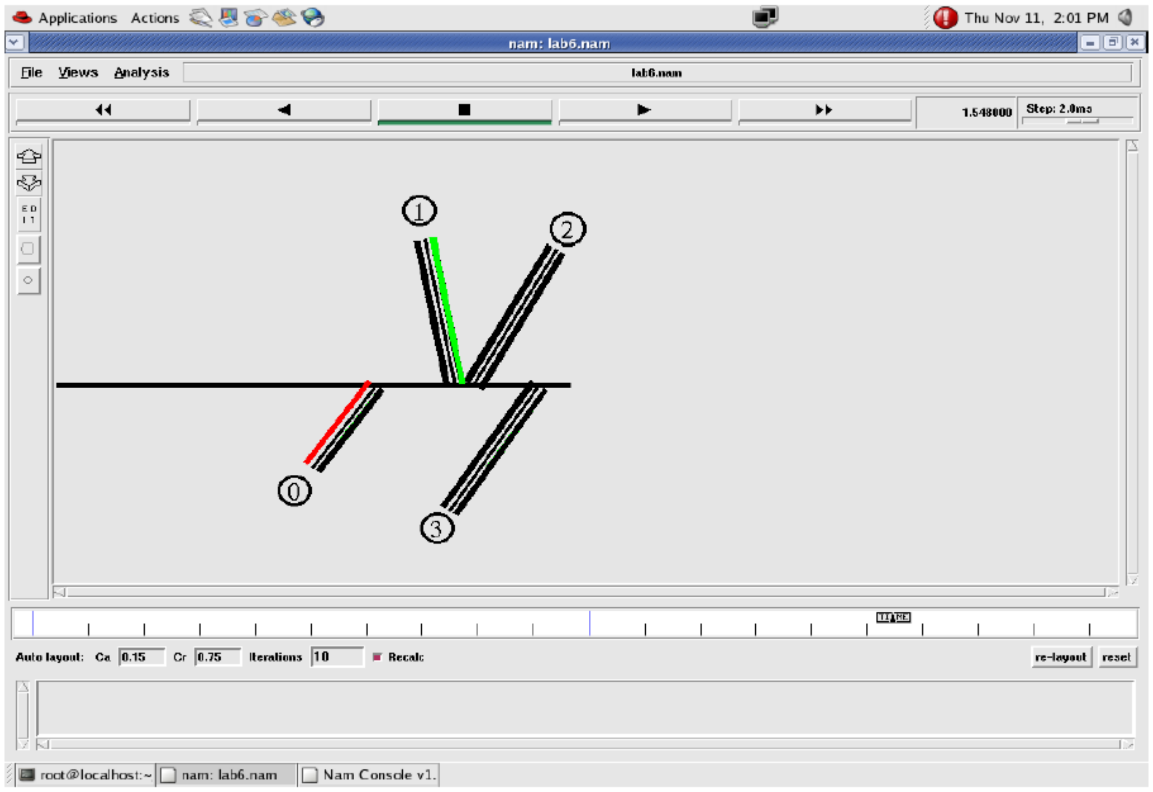
```
set ns  [new Simulator]
set tf  [open lab6.tr w]
$ns trace-all $tf
set nf  [open lab6.nam w]
$ns namtrace-all $nf
set n0  [$ns node]
set n1  [$ns node]
set n2  [$ns node]
set n3  [$ns node]
set n4  [$ns node]
#The below code is used to create the Lan

$ns make-lan   -trace on "$n0 $n1 $n2 $n3 $n4" 100mb 10ms LL Queue/DropTail  Mac/802_3
$ns color 1 "red"
$ns color 2 "blue"
$ns color 3 "green"
set tcp0  [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0  [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink2  [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp0 $sink2
set udp2  [new Agent/UDP]
$ns attach-agent $n2 $udp2
set cbr2  [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
set null1   [new Agent/Null]
$ns attach-agent $n1 $null1
$ns connect $udp2 $null1
$udp2 set class_ 1
$tcp0 set class_ 2
set tcp1  [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1  [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink3  [new Agent/ TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp1 $sink3
$tcp1 set class_ 3
$ftp0 set interval_ 0.001
```

```
$cbr2 set interval_ 0.001
$ftp1 set interval_ 0.01

proc finish  {} {
global ns nf tf
$ns flush-trace
exec nam lab6.nam &
close $tf
close $nf
exit 0
}


$ns at 0.1 "$cbr2 start"
$ns at 1.2 "$ftp1 start"
$ns at 1.3 "$ftp0 start"
$ns at 5.0 "finish"
$ns run
```

AWK Script (lab6.awk)

```
BEGIN{

count=0

}

{

if($1=="c")

count++

}

END{

printf("\nThe Total Packet Collision %d\n ",count);

}
```

SnapShot:



Execution of the program

Type the following commands in the terminal window for executing the programs.

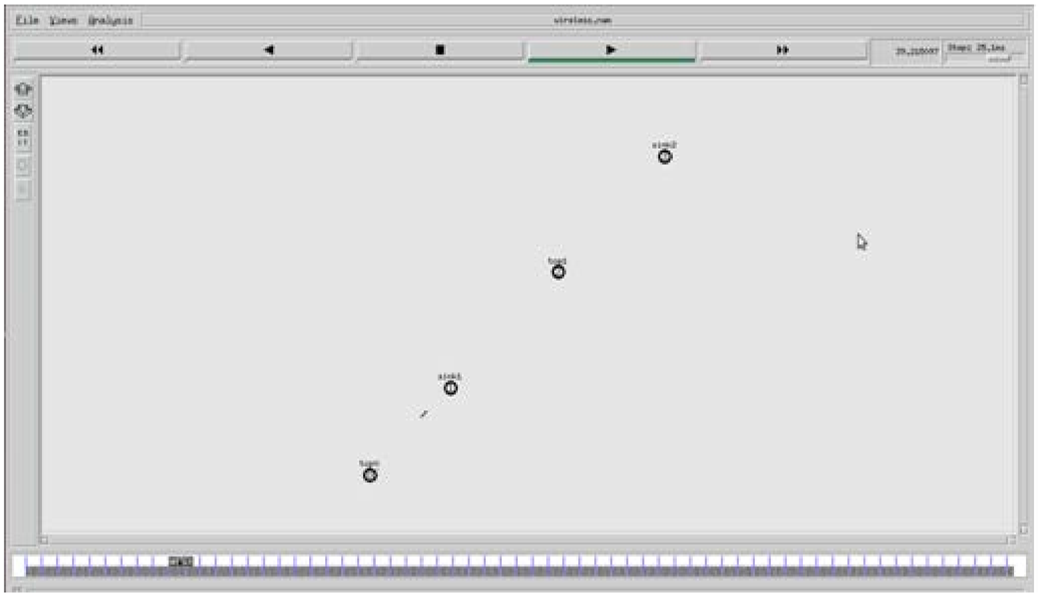
```
ns  lab6.tcl
awk  –f  lab6.awk lab6.tr
```

Output:

```
The Total Packet Collision: 3450
```

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

Design:



Simulation Parameters:

Area **:** 700m *700m
Simulation Time **:** 250 simulation sec
Wireless nodes **:** 4
Routing Protocol **:** DSDV (Destination Sequenced Distance vector)
Interface queue Type **:** Queue/DropTail
MAC **:** 802.11

Application **:** FTP
Antenna **:** Omni antenna

Program

```
set ns  [new Simulator]
set tf  [open wireless.tr w]
$ns trace-all $tf
set topo  [new Topography]
$topo load_flatgrid 700 700
set nf  [open wireless.nam w]
$ns namtrace-all-wireless $nf 700 700

$ns node-config -adhocRouting DSDV\
                -llType LL\
                -ifqType Queue/DropTail\
                -macType Mac/802_11\
                -ifqLen 50 \
                -phyType Phy/WirelessPhy\
                -channelType Channel/WirelessChannel\
                -propType Propagation/TwoRayGround\
                -antType Antenna/OmniAntenna\
                -topoInstance $topo\
                -agentTrace ON\
                -routerTrace ON
create-god 4
set n0  [$ns node]
set n1  [$ns node]
set n2  [$ns node]
set n3  [$ns node]
$n0 label  "tcp0"
$n1 label  "sink1"
$n2 label  "tcp1"
$n3 label  "sink2"
```

#The below code is used to give the initial node positions.

```
$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 200
$n1 set Y_ 200
$n1 set Z_ 0
$n2 set X_ 400
$n2 set Y_ 400
```

```
$n2 set Z_ 0
$n3 set X_ 600
$n3 set Y_ 600

$n3 set Z_ 0
$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1  setdest 200 200 25"
$ns at 0.1 "$n2 setdest 400 400 25"
$ns at 0.1 "$n3 setdest 600 600 25"
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n3 $sink2
$ns connect $tcp1 $sink2
$ns at 5 "$ftp0 start"
$ns at 10 "$ftp1 start"
```

#The below code is used to give node movements .

```
$ns  at 100  "$n2 setdest 500 500 25"

proc finish {}  {
global nf ns tf
$ns flush-trace
exec nam wireless.nam &
close $tf
exit 0
}
$ns at 250 "finish"
$ns run
```

AWK Script (wireless.awk)

```
BEGIN{

        count1=0
        count2=0
        pack1=0
        pack2=0
        time1=0
        time2=0
}
{
        if($1=="r" && $3=="_1_" && $4=="AGT")
        {
                count1++
                pack1=pack1+$8
                time1=$2
        }
        if($1=="r" && $3=="_3_" && $4=="AGT")
        {
                count2++
                pack2=pack2+$8
                time2=$2
        }
}
END{
        printf(" The Throughput from n0 to n1:
%fMbps\n",((count1*pack1*8)/(time1*1000000)))


        printf(" The Throughput from n2 to n3:
%fMbps\n",((count2*pack2*8)/(time2*1000000)))
}
```

Execution of the program

Type the following commands in the terminal window for executing the programs.

```
ns wireless.tcl
awk –f wireless.awk  wireless.tr
```

Output:

```
The Throughput from n0 to n1: 5444Mbps
The Throughput from n2 to n3: 345Mbps
```

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.

Program

```
set ns [new Simulator ]
set tf  [open labping.tr w]
$ns trace-all $tf
set nf  [open labping.nam w]
$ns namtrace-all $nf
set n0  [$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]
$n0 label "Ping0"
$n2 label "Router"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$ns color 1 "red"
$ns color 2 "blue"
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4  1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n6 2
$ns queue-limit $n2 $n4 3
$ns queue-limit $n5 $n2 5
set ping0  [new Agent/Ping ]
$ns attach-agent $n0 $ping0
set ping4  [new Agent/Ping ]
$ns attach-agent $n4 $ping4
set ping5  [new Agent/Ping ]
$ns attach-agent $n5 $ping5
```

```
set ping6  [new Agent/Ping ]
$ns attach-agent $n6 $ping6
$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001
$ping5 set packetSize_ 50000
$ping5 set interval_ 0.0001
$ping0 set class_ 1
$ping5 set class_ 2
$ns connect $ping0 $ping4
$ns connect $ping5 $ping6
Agent/Ping instproc recv {from rtt} {
        $self instvar node_
puts "The node [$node_ id] received a reply from $from with the round trip time of $rtt"
}
$ns at 0.1 "$ping0 send"
$ns at 0.2 "$ping0 send"
$ns at 0.3 "$ping0 send"
$ns at 0.4 "$ping0 send"
$ns at 0.5 "$ping0 send"
$ns at 0.6 "$ping0 send"
$ns at 0.7 "$ping0 send"
$ns at 0.8 "$ping0 send"
$ns at 0.9 "$ping0 send"
$ns at 1.0 "$ping0 send"
$ns at 1.1 "$ping0 send"
$ns at 1.2 "$ping0 send"
$ns at 1.3 "$ping0 send"
$ns at 1.4 "$ping0 send"
$ns at 1.5 "$ping0 send"
$ns at 1.6 "$ping0 send"
$ns at 1.7 "$ping0 send"
$ns at 1.8 "$ping0 send"
$ns at 1.9 "$ping0 send"
$ns at 0.1 "$ping5 send"
$ns at 0.2 "$ping5 send"
$ns at 0.3 "$ping5 send"
$ns at 0.4 "$ping5 send"
$ns at 0.5 "$ping5 send"
$ns at 0.6 "$ping5 send"
$ns at 0.7 "$ping5 send"
$ns at 0.8 "$ping5 send"
$ns at 0.9 "$ping5 send"
$ns at 1.0 "$ping5 send"
$ns at 1.1 "$ping5 send"
$ns at 1.2 "$ping5 send"
$ns at 1.3 "$ping5 send"
```

```
$ns at 1.4 "$ping5 send"
$ns at 1.5 "$ping5 send"
$ns at 1.6 "$ping5 send"
$ns at 1.7 "$ping5 send"
$ns at 1.8 "$ping5 send"
$ns at 1.9 "$ping5 send"
$ns at 0.1 "$ping5 send"
$ns at 0.2 "$ping5 send"
$ns at 0.3 "$ping5 send"
$ns at 0.4 "$ping5 send"
$ns at 0.5 "$ping5 send"
$ns at 0.6 "$ping5 send"
$ns at 0.7 "$ping5 send"
$ns at 0.8 "$ping5 send"
$ns at 0.9 "$ping5 send"
$ns at 1.0 "$ping5 send"
$ns at 1.1 "$ping5 send"
$ns at 1.2 "$ping5 send"
$ns at 1.3 "$ping5 send"
$ns at 1.4 "$ping5 send"
$ns at 1.5 "$ping5 send"
$ns at 1.6 "$ping5 send"
$ns at 1.7 "$ping5 send"
$ns at 1.8 "$ping5 send"
$ns at 1.9 "$ping5 send"


proc finish {} {
global ns nf tf
exec nam labping.nam &
$ns flush-trace
close $tf
close $nf
exit 0
}
$ns at 5.0 "finish"
$ns run
```

AWK Script (labping.awk)

```
BEGIN{

count=0;
}

  {
if($1=="d")
        count++
}

  END{
  printf("The Total no of Packets Dropped due to Congestion:%d", count);
}
```

Execution of the program

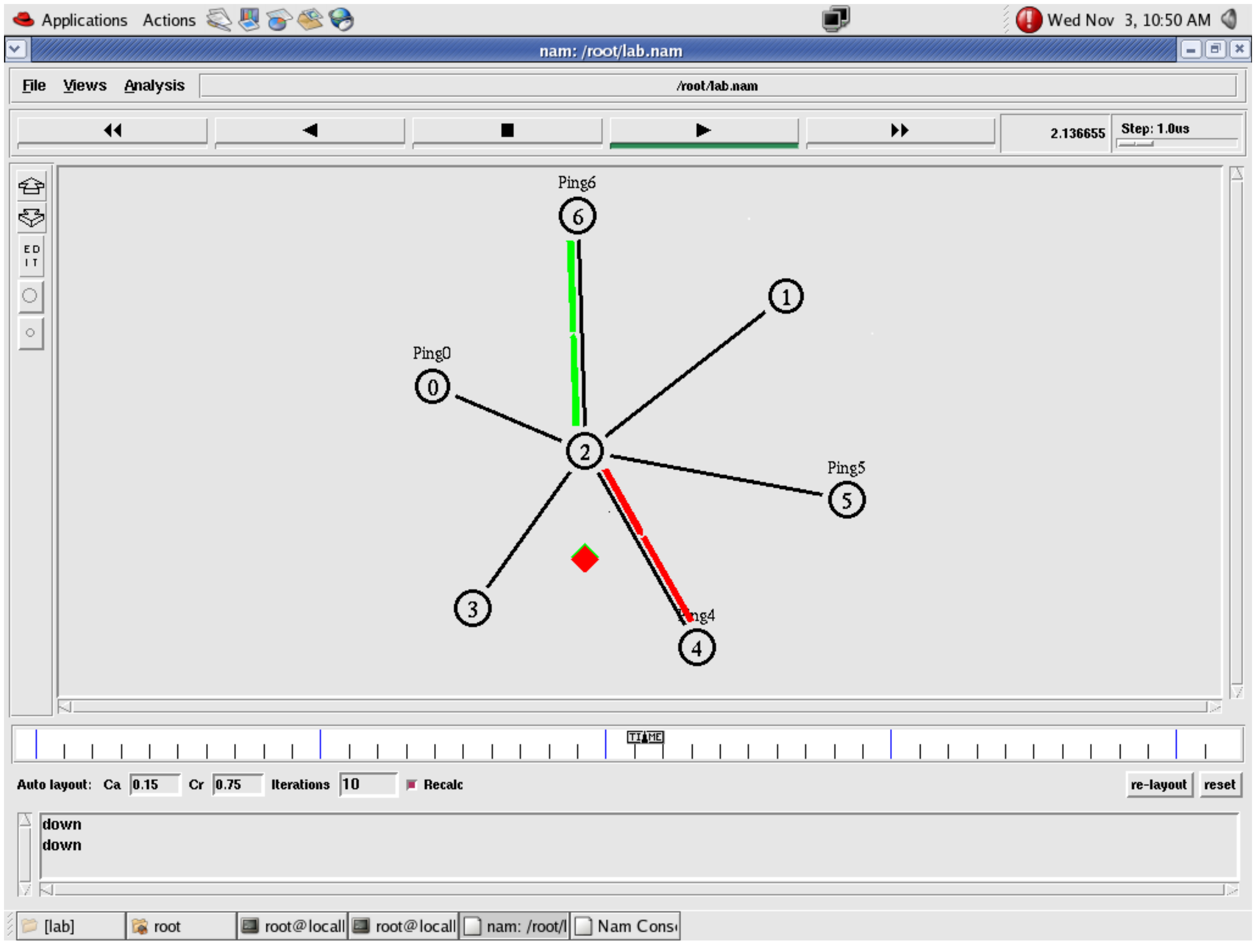
Type the following commands in the terminal window for executing the programs

```
ns labping.tcl

awk –f  labping.awk  labping.tr
```

Output:

```
The Total no of packets dropped due to congestion:  345
```

Snap Shot:



```
down
down
```

Part B Programs

CRC

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

Theory

It does error checking via polynomial division. In general, a bit string

$$b_{n-1}b_{n-2}b_{n-3}\dots b_2b_1b_0$$

As

$$b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + b_{n-3}X^{n-3} + \dots b_2X^2 + b_1X^1 + b_0$$

Ex: -

10010101110

As

$$X^{10} + X^7 + X^5 + X^3 + X^2 + X^1$$

All computations are done in modulo 2

Algorithm:-

1. Given a bit string, append 0^S to the end of it (the number of 0^s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.
2. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.
3. Define T(x) = B(x) –R(x)

   (T(x)/G(x) => remainder 0)

4. Transmit T, the bit string corresponding to T(x).

5. Let T’ represent the bit stream the receiver gets and T’(x) the associated polynomial. The receiver divides $T^1(x)$ by G(x). If there is a 0 remainder, the receiver concludes T = T’ and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

Program

```
#include<stdio.h>

#include<string.h>

#include<conio.h>

#define N strlen(g)


char t[128], cs[128], g[]="10001000000100001";

int a, e, c;


void xor() {

  for(c=1;c<N;c++) cs[c]=((cs[c]==g[c])?'0':'1');

}


void crc() {

  for(e=0;e<N;e++) cs[e]=t[e];

  do {

    if(cs[0]=='1') xor();

    for(c=0;c<N-1;c++) cs[c]=cs[c+1];

    cs[c]=t[e++];

  }while(e<=a+N-1);

}
```

```
void main() {
 clrscr();
 printf("\nEnter poly : "); scanf("%s",t);
 printf("\nGenerating Polynomial is : %s",g);
 a=strlen(t);
 for(e=a;e<a+N-1;e++) t[e]='0';
 printf("\nModified t[u] is :  %s",t);
 crc();
 printf("\nChecksum is : %s",cs);
 for (e=a;e<a+N-1;e++) t[e]=cs[e-a];
 printf("\nFinal Codeword is : %s",t);
 printf("\nTest Error detection 0(yes) 1(no) ? : ");
 scanf("%d",&e);
 if (e==0) {
   printf("Enter position where error is to inserted : ");
   scanf("%d",&e);
   t[e]=(t[e]=='0')?'1':'0';
   printf("Errorneous data   : %s\n",t);
 }
 crc();
 for  (e=0;(e<N-1)&&(cs[e]!='1');e++);
 if (e<N-1) printf("Error detected.");
 else printf("No Error Detected.");
 getch();
}
```

Output

```
Enter poly : 1011101

Generating Polynomial is : 10001000000100001

Modified t[u] is :  101110100000000000000000

Final Codeword is : 10111011000101101011000

Test Error detection 0(yes) 1(no) ? : 0

Enter position where you want to insert error : 3

Errorneous data   : 10101011000101101011000

Error detected.


Enter poly : 1011101

Generating Polynomial is : 10001000000100001

Modified t[u] is :  101110100000000000000000

Checksum is : 1000101101011000

Final Codeword is : 10111011000101101011000

Test Error detection 0(yes) 1(no) ? : 1

No Error Detected.
```

Distance Vector Routing

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

Theory

Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on.

Routing algorithms can be grouped into two major classes: adaptive and nonadaptive. Nonadaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ΔT sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which path to get there. These tables are updated by exchanging information with the neighbors. The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it. In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred out going line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, or the total number of packets queued along the path. The router is assumed to know the "distance" to each of its neighbor.

Program

```
#include<stdio.h>

struct node

{

unsigned dist[20];

unsigned from[20];

}rt[10];
```

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

```
if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
{
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].from[j]=k;

count++;
}
}
}while(count!=0);
for(i=0;i<n;i++)
{
printf("\n state value for router %d is\n",i+1);
for(j=0;j<n;j++)
{
printf("\t \n to node %d via %d,distance=%d\n",j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}
printf("\n\n");
return 0;
}
```

<u>Output</u>

```
enter the no of nodes
4
enter the cost matrix:
```

```
0 1 2 999

1 0 999 3

2 999 0 4

999 3 4 0


 state value for router 1 is

          to node 1 via 1,distance=0

          to node 2 via 2,distance=1

          to node 3 via 3,distance=2

          to node 4 via 2,distance=4

 state value for router 2 is

          to node 1 via 1,distance=1

          to node 2 via 2,distance=0

          to node 3 via 1,distance=3

          to node 4 via 4,distance=3

 state value for router 3 is

          to node 1 via 1,distance=2

          to node 2 via 1,distance=3

          to node 3 via 3,distance=0

          to node 4 via 4,distance=4

 state value for router 4 is
```

```
to node 1 via 2,distance=4

to node 2 via 2,distance=3

to node 3 via 3,distance=4

to node 4 via 4,distance=0
```

Leaky Bucket

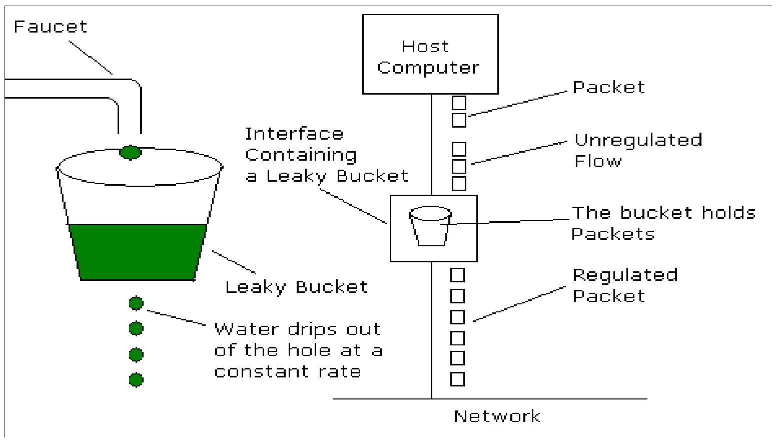
7. Write a program for congestion control using Leaky bucket algorithm.

Theory

The congestion control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back. The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is called **traffic shaping**. The leaky bucket algorithm is used for traffic shaping. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.



Program

```
#include<stdio.h>

#include<stdlib.h>

#define bktsz 700

int c=0;

int count=0;

void bktip(int a,int b,int i)

{

if(a>bktsz-c)
```

```
{
printf("bucket overflow and non-conforming packet\n");


}
else
{
sleep(1);
while(a>b)
{
printf("%d bytes outputted\n",b);
a=a-b;
sleep(1);
count++;
}
if(a>0)
{
c=c+a;
if(count==0)
{
printf("last %d bytes outputted\n",c);
c=0;
printf("%d bytes stored\n",c);
}
else{
```

```
if(c==b)
{
printf("%d bytes stored is outputted\n",b);
c=0;
}if(c>b)
{
printf("%d bytes stored is outputted\n",b);
c=c-b;
printf("%d bytes stored\n",c);
}else
printf("%d bytes stored\n",c);}
printf("conforming packet\n");
}
}
if((i==5)&&(c!=0))
{
printf("%d bytes stored is outputted\n",c);
}
}
int main()
{
int opr,psz,i;
printf("output rate:\n");
scanf("%d",&opr);
for(i=1;i<=5;i++)
{
```

```
printf("enter the packet size:\n");
scanf("%d",&psz);
printf("packet number %d   packet size %d\n",i,psz);
bktip(psz,opr,i);
}
}
```

OUTPUT:

```
output rate:
100
enter packet size:
33
packet number 1  packet size 33
last 33 bytes outputted
0 bytes stored
conforming packet
enter the packet size:
120
packet number 2  packet size 120
100 bytes outputted
20 bytes stored
conforming packet
enter the packet size:
140
packet number 3  packet size 140
```

```
100 bytes outputted

60 bytes stored

conforming packet

enter the packet size:

120

packet number 4  packet size 345

100 bytes outputted

100 bytes outputted

100 bytes outputted

100 bytes stored is outputted

5 bytes stored

conforming packet

enter the packet size:

900

packet number 5  packet size 900

bucket overflow

5 bytes stored is outputted
```

TCP Socket

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

Algorithm (Client Side)

1. Start.

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

Algorithm (Server Side)

1. Start.
2. Create a socket using socket() system call.
3. Bind the socket to an address using bind() system call.
4. Listen to the connection using listen() system call.
5. accept connection using accept()
6. Receive filename and transfer contents of file with client.
7. Stop.

******Client program******

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<netdb.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<fcntl.h>
#define PORT 6490
int main()
{
int i=0,sockfd,len;
char buf1[40],buf2[20000];
FILE* fp;
struct sockaddr_in their_addr;
```

```
if((sockfd=socket(AF_INET, SOCK_STREAM,0))==-1)
{
perror("socket");
exit(1);
}
their_addr.sin_family=AF_INET;
their_addr.sin_port=htons(PORT);
their_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
memset(&(their_addr.sin_zero), '\0', 8);
if(connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr))==-1)
{
perror("connect");
exit(1);
}
printf("CLIENT is online!\n");
printf("CLIENT:Enter the filename to be displayed: ");
scanf("%s",buf1);

send(sockfd,buf1,sizeof(buf1),0);
if(recv(sockfd,buf2,sizeof(buf2),0)==1)
{
perror("recv");
exit(1);
}
else
{
printf("Displyaing the contents of %s",buf1);
printf("\n%s\n",buf2);
}
close(sockfd);
return 0;
```

```
}
```

******Server program******

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/wait.h>
#include<signal.h>
#define MYPORT 6490
#define BACKLOG 10
int main(void)
{
int sockfd,fp,new_fd;
struct sockaddr_in my_addr,their_addr;
int sin_size,i=0;
int yes=1;
char buf1[20],buf2[20000];
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
perror("socket");
exit(1);
}
```

```
if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))==-1)
{
perror("setsockopt");
exit(1);
}
my_addr.sin_family=AF_INET;
my_addr.sin_port=htons(MYPORT);
my_addr.sin_addr.s_addr=INADDR_ANY;
memset(&(my_addr.sin_zero), '\0', 8);
if(bind(sockfd,(struct sockaddr *)&my_addr, sizeof(struct sockaddr)) ==-1)
{
perror("Bind");
exit(1);
}
if(listen(sockfd, BACKLOG) == -1)
{
perror("listen");
exit(1);
}
printf("\n SERVER is online! \n SERVER: Waiting for the client........\n");
sin_size=sizeof(struct sockaddr_in);
if((new_fd=accept(sockfd,(struct sockaddr *)&their_addr, &sin_size))==-1)
{
perror("Accept");
exit(0);
}
printf("\n SERVER: Got connection from %s \n", inet_ntoa(their_addr.sin_addr));
recv(new_fd,&buf1,sizeof(buf1),0);
printf("File requested is %s\n", buf1);
if((fp=open(buf1,O_RDONLY))<0)
{
```

```
printf("File not found\n");
strcpy(buf2,"File not found");
}

else
{
printf("SERVER: %s found and ready to transfer.\n",buf1);
read(fp,&buf2,20000);
close(fp);
}
send(new_fd,&buf2,sizeof(buf2),0);
close(new_fd);
close(sockfd);
printf("Transfer success \n");
printf("\n");
return 0;
}
```

Output (Server)

```
SERVER is online!
 SERVER: Waiting for the client........
SERVER: Got connection from
File requested
SERVER:  file found and ready to transfer
Transfer Success
```

Output (Client)

```
CLIENT is online!
CLIENT: Enter the filename to be displayed:
```

```
file

Displyaing the contents of file


The contents of file are :-

This a demo of client server using Sockets

Just for trial.

Now End of file


EOF
```