SRI KRISHNA INSTITUTE OF TECHNOLOGY

DEPT. OF ELECTRONICS & COMMUNICATION ENGINEERING.

COMPUTER COMMUNICATION NETWORKS LABORATORY

(18ECL76)

LAB MANUAL

2023-24

Faculty Incharge

Nagaraja M

Assoc. Prof. Electronics & Communication Dept.

LABORATORY EXPERIMENTS

PART-A: Simulation experiments using NS2/ NS3/ OPNET/ NCTUNS/ NetSim/ QualNet/ Packet Tracer or any other equivalent tool.

1. Implement a point to pint network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.
2. 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.
3. Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.
4. Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.
5. Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.
6. Implementation of Link state routing algorithm.

PART-B: Implement the following in C/C++.

1. Write a program for a HLDC frame to perform the following. i) Bit stuffing ii) Character stuffing.
2. Write a program for distance vector algorithm to find suitable path for transmission.
3. Implement Dijkstra's algorithm to compute the shortest routing path.
4. For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases: i) Without error ii) With error.
5. Implementation of Stop and Wait Protocol and Sliding Window Protocol.
6. Write a program for congestion control using leaky bucket algorithm.

Course outcomes: On the completion of this laboratory course, the students will be able to:
CO1 Design and Simulate Network elements with various protocols and standards. CO2 Use the network simulator tools for learning and practice of networking algorithms. CO3 Demonstrate the working of various protocols and algorithms using C programming.

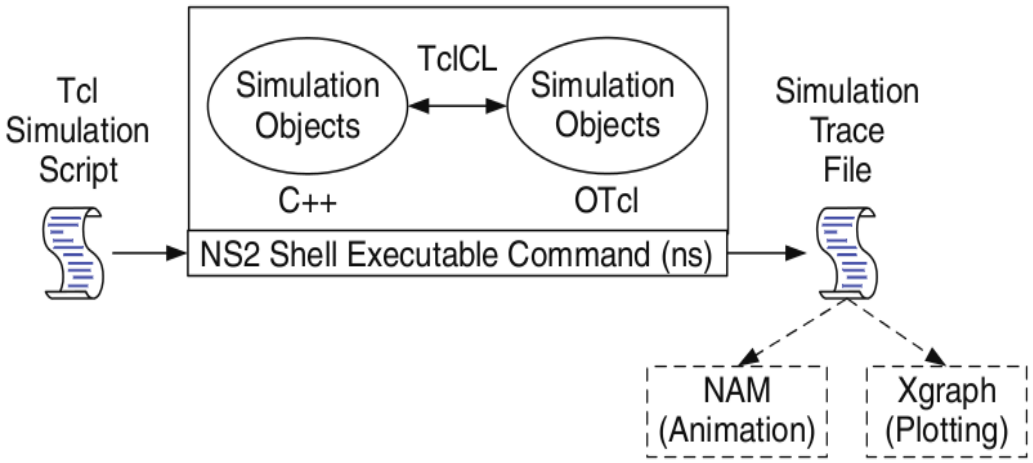
Conduct of Practical Examination:

1. All laboratory experiments are to be included for practical examination.
2. For examination one question from software and one question from hardware or only one hardware experiments based on the complexity to be set.
3. Students are allowed to pick one experiment from the lot.
4. Strictly follow the instructions as printed on the cover page of answer script for breakup of marks.
5. Change of experiment is allowed only once and 15% Marks allotted to the procedure part to be made zero.

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



Tcl scripting

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

Basics of TCL

Syntax: command arg1 arg2 arg3

⚪ **Hello World!**

putsstdout{Hello, World!}

Hello, World!

⚪ **Variables** Command Substitution

set a 5 set len [string length foobar]

set b $a set len [expr [string length foobar] + 9]

⚪ **Simple Arithmetic**

expr 7.2 / 4

⚪ **Procedures**

```
procDiag {a b} {
set c [exprsqrt($a * $a + $b * $b)]
    return $c }
puts “Diagonal of a 3, 4 right triangle is [Diag 3 4]”
```

Output: Diagonal of a 3, 4 right triangle is 5.0

⚪ **Loops**

```
while{$i < $n} {          for {set i 0} {$i < $n} {incr i} {
. . .                     . . .
}                         }
```

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 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]
```

It is 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 becauseit 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

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

The above creates a 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 namout.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"
```

It will be used to call "**finish**" at time 125sec. This 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:

```
#set Queue Size of link (n0-n2) to 20$ns
queue-limit $n0 $n2 20
```

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:

```
settcp [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

```
setudp [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.

```
setcbr [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**.

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_ 1** that assigns to the TCP connection a flow identification of "1".We shall later give the flow identification of "2" to the UDP connection.

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 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 their 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:

$ catempawk.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.awkempn.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:

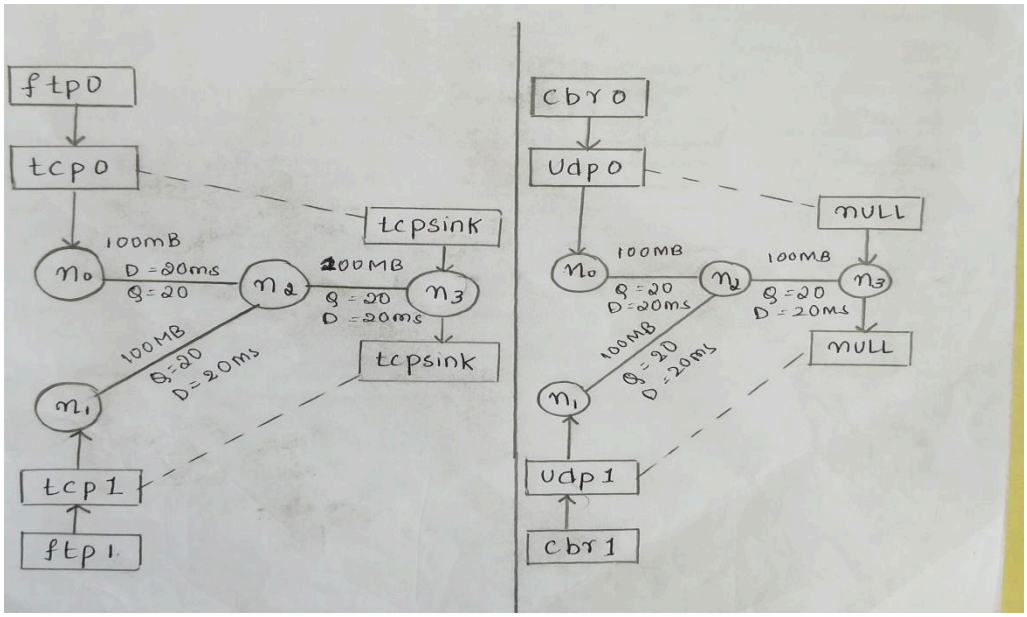
$awk 'BEGIN {FS = "|"}

NF! =6 {

Print "Record No ", NR, "has", "fields"}' empx.lst

PART-A Simulation experiments using NS2

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




```
#=====================================
#     Simulation parameters setup
#=====================================
setval(stop)   10.0;               # time of simulation end

#=====================================
#        Initialization
#=====================================
#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
settracefile [open lab1.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
setnamfile [open lab1.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
```

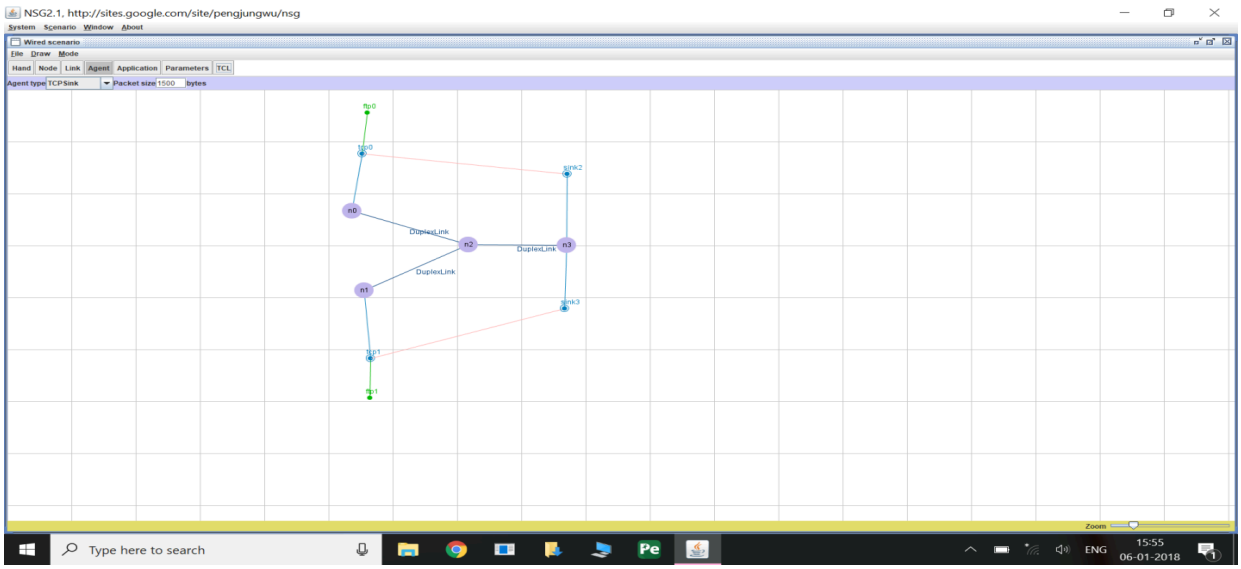
```
#===================================
#Createlinks between nodes
$ns duplex-link $n0 $n2 300.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 10
$ns duplex-link $n1 $n2 400.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 20
$ns duplex-link $n2 $n3 10.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 3

#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 sink2 [new Agent/TCPSink]
$ns attach-agent $n3 $sink2
$ns connect $tcp0 $sink2
$tcp0 set packetSize_ 1500

#Setup a TCP connection
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp1 $sink3
$tcp1 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 FTP Application over TCP connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"
$ns at 2.0 "$ftp1 stop"

#===================================
#        Termination
#===================================
#Define a 'finish' procedure
proc finish {} {
global ns tracefilenamfile
```

```
    $ns flush-trace
close $tracefile
close $namfile
execnam lab1.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
```

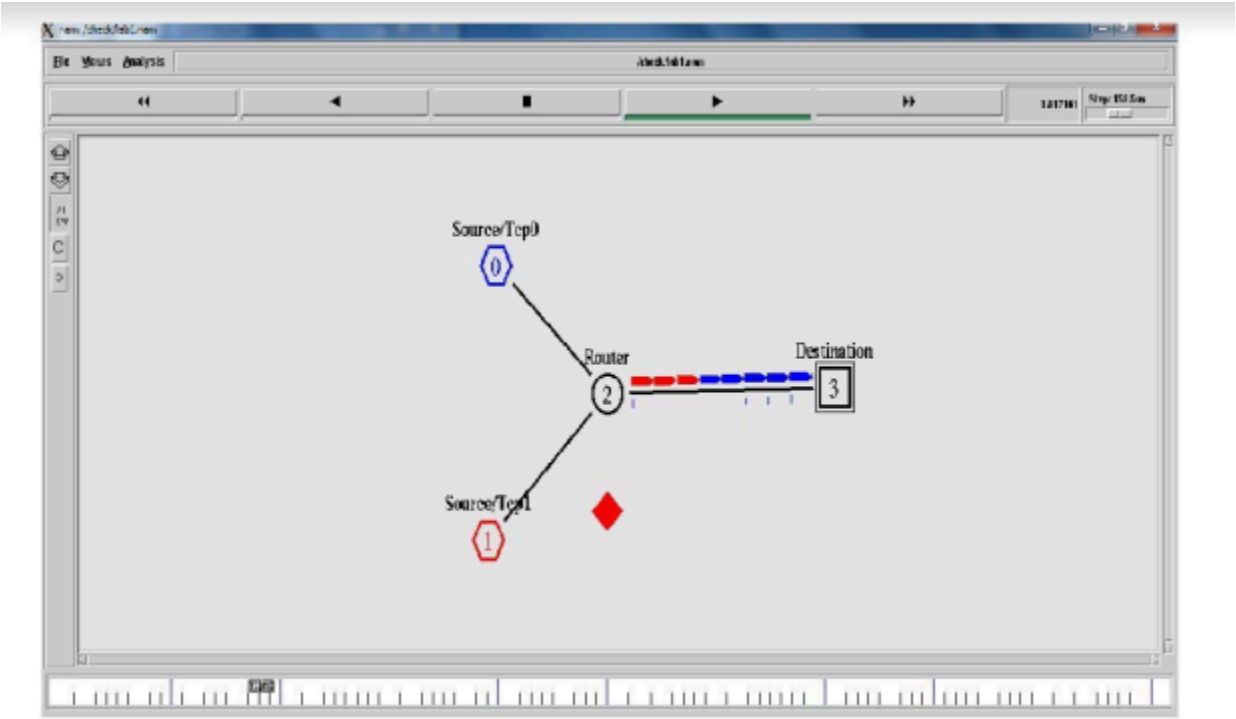
AwkScript:

```
BEGIN{
tcppack=0
tcppack1=0
}
{
if($1=="r"&&$4=="3"&&$5=="tcp"&&$6=="1540")
{
tcppack++;
}
if($1=="d"&&$3=="2"&&$4=="3"&&$5=="tcp"&&$6=="1540")
{
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++);
}
```

Output:



Network animator (lab1.nam):

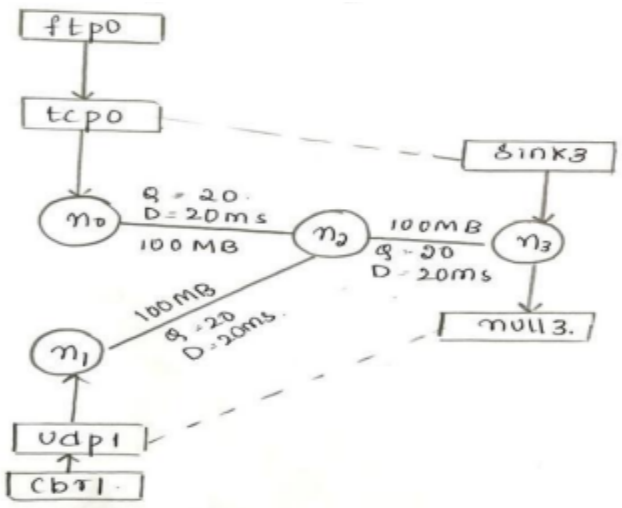


Tracefile (lab1.tr):

Results:

Bandwidth(MB) N0-N2, N1-N2, N2-N3	Queue Size Q1, Q2, Q3	Received at Node 3	Dropped at Node 2
100,100,100	20, 20,20	6820	0
100,100,1	20, 20,2	508	45
300,300,10	50, 50,3	3021	48

2. 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.




```
#==================================
#     Simulation parameters setup
#==================================
setval(stop)   10.0;# time of simulation end

#==================================
#        Initialization
#==================================
#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
settracefile [open lab2.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
setnamfile [open lab2.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]
```

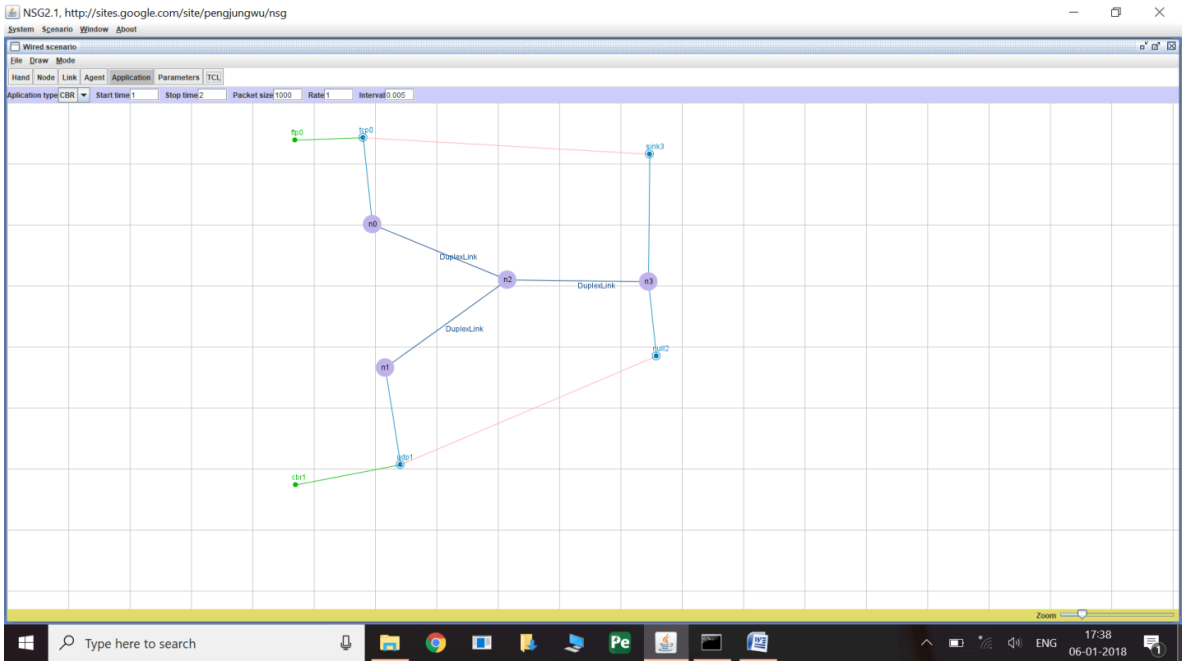
```
#==================================
#        Links Definition
#==================================
#Createlinks between nodes
$ns duplex-link $n0 $n2 200.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 50
$ns duplex-link $n2 $n3 200.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 50
$ns duplex-link $n1 $n2 200.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 50

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n1 $n2 orient right-up

#==================================
#        Agents Definition
#==================================
#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp0 $sink3
$tcp0 set packetSize_ 1000
$tcp0 set interval_ 0.1

#Setup a UDP connection
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set null2 [new Agent/Null]
$ns attach-agent $n3 $null2
$ns connect $udp1 $null2
$udp1 set packetSize_ 1100
$udp1 set interval_ 0.1
#==================================
#        Applications Definition
#==================================
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
$ns at 9.0 "$ftp0 stop"

#Setup a CBR Application over UDP connection
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSize_ 1000
$cbr1 set rate_ 1.0Mb
$cbr1 set random_ null
$ns at 1.0 "$cbr1 start"
$ns at 9.0 "$cbr1 stop"
```

```
#=================================
#        Termination
#=================================
#Define a 'finish' procedure
proc finish {} {
global ns tracefilenamfile
    $ns flush-trace
close $tracefile
close $namfile
execnam lab2.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
```

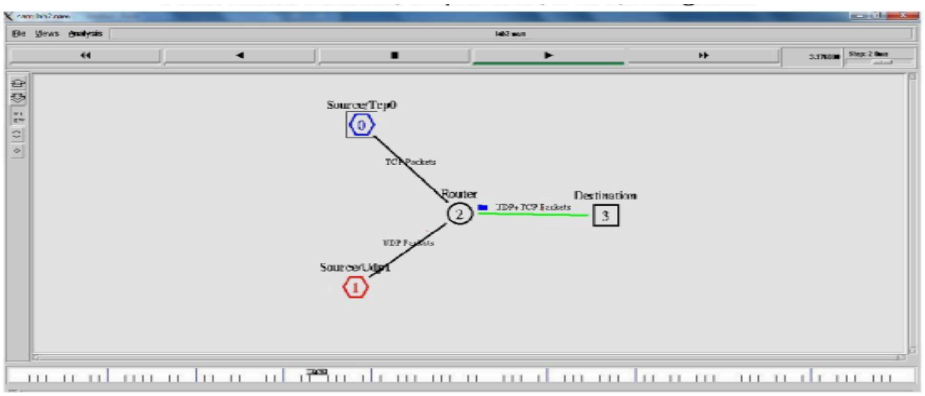
Awk Script:

```
BEGIN{
tcppack=0
tcppack1=0
}
{
if($1=="r"&&$4=="2"&&$5=="tcp"&&$6=="340")
{
tcppack++;
}
if($1=="r"&&$4=="2"&&$5=="cbr"&&$6=="300")
{
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++);
}
```

Output:



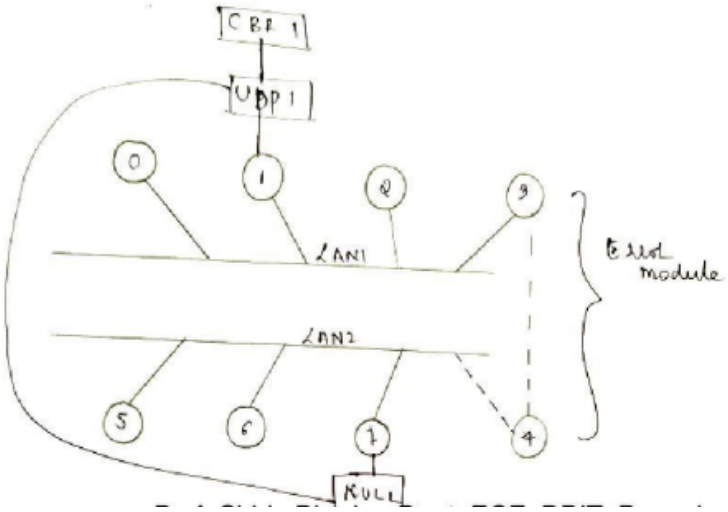
Network animator (lab2.nam):



Results:

Simulation Time (Sec)	Packet Size (Byte)	Sent at Node 2
Start Time:1 Stop Time:9	TCP:1500 UDP:1500	TCP:3930 UDP:1001
Start Time:1 Stop Time:20	TCP:1000 UDP:1500	TCP:9410 UDP:1584
Start Time:1 Stop Time:25	TCP:1300 UDP:1500	TCP:11910 UDP:2000

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




```
set ns [new Simulator]
settf [open lab3.tr w]
$ns trace-all $tf

setnf [open lab3.nam w]
$ns namtrace-all $nf

$ns color 0 blue

set n0 [$ns node]
$n0 color "red"
set n1 [$ns node]
$n1 color "red"
set n2 [$ns node]
$n2 color "red"
set n3 [$ns node]
$n3 color "red"
set n4 [$ns node]
$n4 color "magenta"
set n5 [$ns node]
$n5 color "magenta"
```

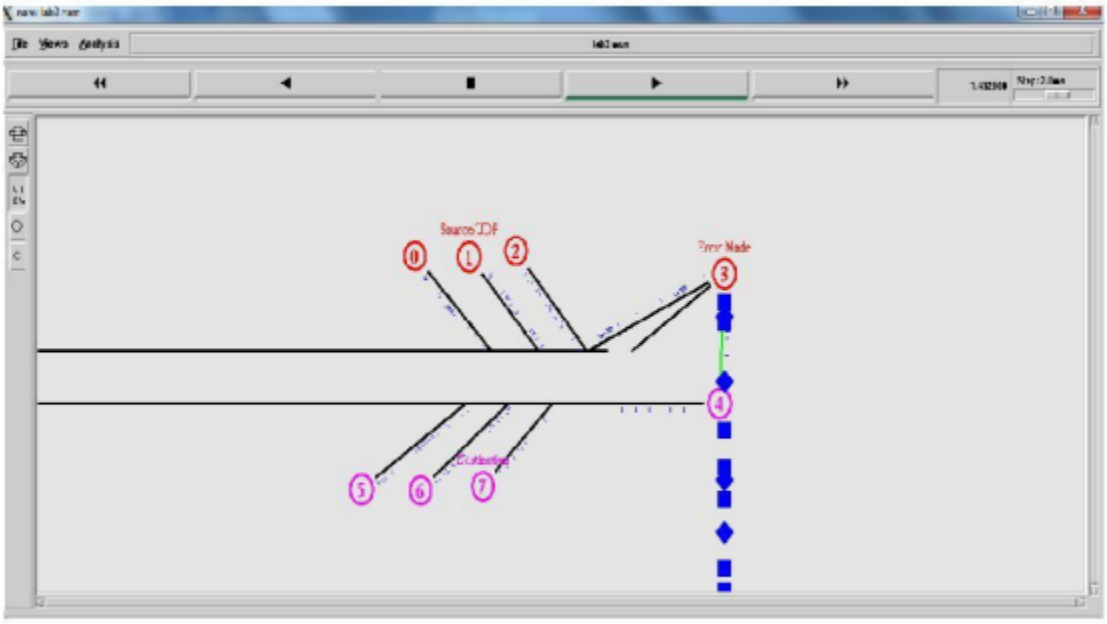
```
set n6 [$ns node]
$n6 color "magenta"
set n7 [$ns node]
$n7 color "magenta"

$n1 label "Source/UDP"
$n3 label "Error Node"
$n7 label "Destination"

$ns make-lan "$n0 $n1 $n2 $n3" 100Mb 300ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6 $n7" 100Mb 300ms LL Queue/DropTail Mac/802_3

$ns duplex-link $n3 $n4 100Mb 300ms DropTail
$ns duplex-link-op $n3 $n4 color "green"


# set error rate. Here ErrorModel is a class and it is single word and
space should not be given between Error and Model
# lossmodel is a command and it is single word. Space should not be given
between loss and model

set err [new ErrorModel]
$ns lossmodel $err $n3 $n4
$err set rate_ 0.3

# error rate should be changed for each output like 0.1,0.3,0.5…. */
setudp [new Agent/UDP]
$ns attach-agent  $n1 $udp
setcbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set fid_ 0
$cbr set packetSize_ 1000
$cbr set interval_ 0.1
set null [new Agent/Null]
$ns attach-agent $n7 $null

$ns connect $udp $null

proc finish { } {
global ns nftf
$ns flush-trace
close $nf
close $tf
execnam lab3.nam &
exit 0
}
$ns at 0.1 "$cbr start"
$ns at 3.0 "finish"
$ns run
```

Awk Script:

```
BEGIN{
tcppack=0
```

```
tcppack1=0
}
{
if($1=="r"&&$4=="7"&&$5=="cbr"&&$6=="1000")
{
tcppack++;
}
}
END{
printf("\n total number of  data packets at Node 7: %d\n", tcppack++);

}
```

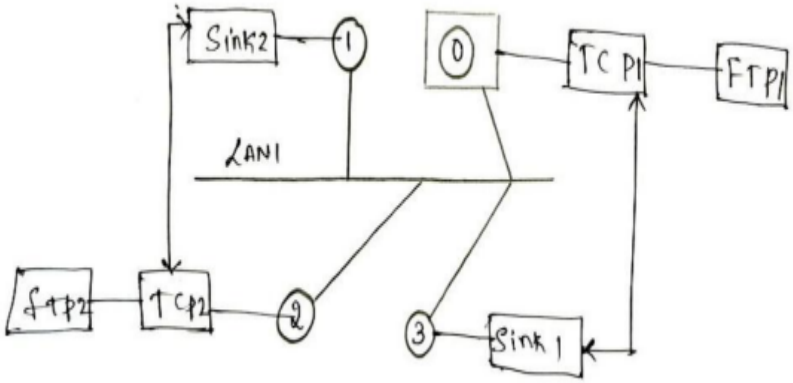
Output:



Results:

Error Rate	Data Rate	Received at Node 7
0.1	0.001	1255
0.3	0.01	95
0.5	0.1	05

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



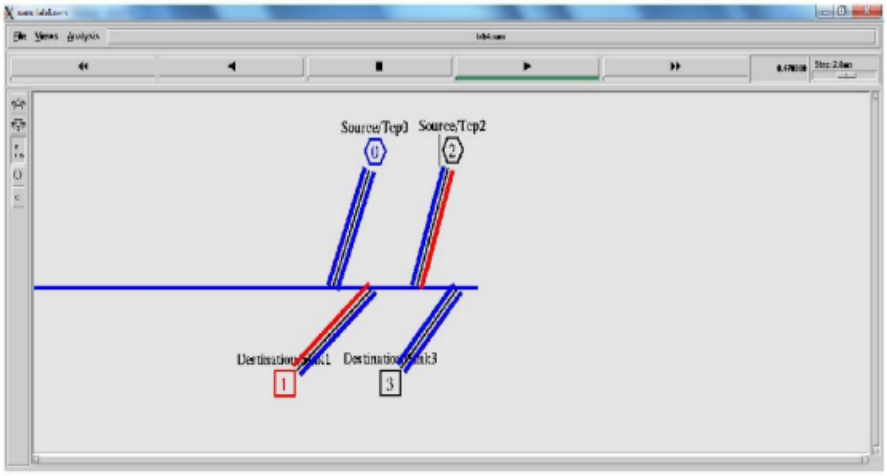

```
set ns [new Simulator]
settf [open lab4.tr w]
$ns trace-all $tf
setnf [open lab4.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns make-lan "$n0 $n1 $n2 $n3" 10mb 10ms LL Queue/DropTail Mac/802_3
set tcp0 [new Agent/TCP/Reno]
$ns attach-agent $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
######To trace the congestion window##########
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 { } {
globalnftf ns
$ns flush-trace
execnam 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
```

Awk Script:

```
BEGIN{
tcppack=0
tcppack1=0
}
{
if($1=="r"&&$4=="7"&&$5=="cbr"&&$6=="1000")
{
tcppack++;
}
}
END{
printf("\n total number of  data packets at Node 7: %d\n", tcppack++);

}
```

Output:

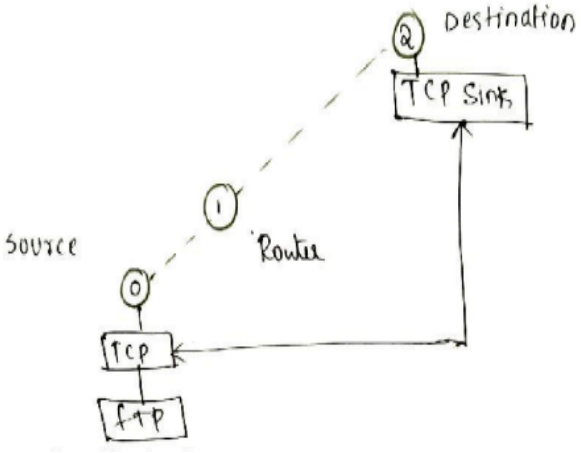


1. File1.tr is the Congestion Window (CWND) value of TCP agent.
2. File2.tr is the Congestion Window (CWND) value of TCPReno agent.

Time	TCP	TCPReno
0.5	23.02	21.91
1	29.73	29.53
1.5	34.59	34.71
2	2	41.11
2.5	26.23	2
3	32.84	24.62

Table 1: Above values are taken from file1.tr and file2.tr

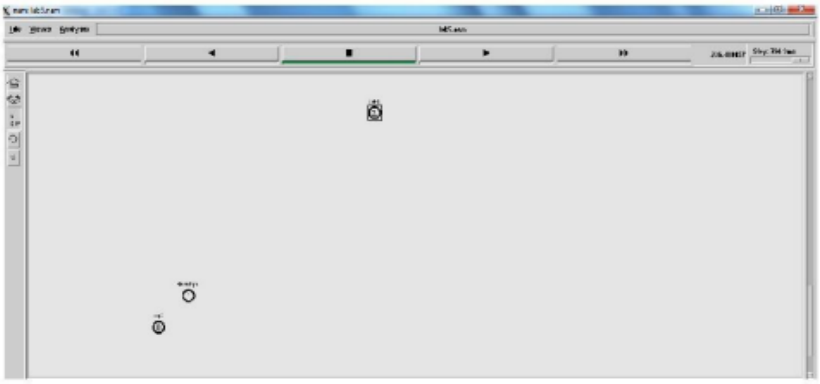
5. Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.



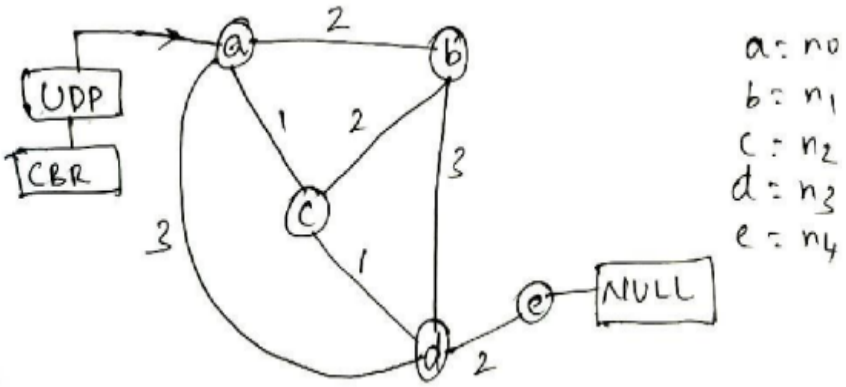

```
set ns [new Simulator]
settf [open lab5.tr w]
$ns trace-all $tf
settopo [new Topography]
$topoload_flatgrid 1000 1000
setnf [open lab5.nam w]
$ns namtrace-all-wireless $nf 1000 1000

$ns node-config -adhocRouting DSDV \
            -llType LL \
            -macType Mac/802_11 \
            -ifqType Queue/DropTail \
            -ifqLen 50 \
            -phyTypePhy/WirelessPhy \
            -channelType Channel/WirelessChannel \
            -propType Propagation/TwoRayGround \
```

```
            -antType Antenna/OmniAntenna \
            -topoInstance $topo \
            -agentTrace ON \
            -routerTrace OFF

create-god 3
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"

$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 100
$n1 set Y_ 100
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 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 $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70 15"
proc finish { } {
      global ns nftf
      $ns flush-trace
      execnam lab5.nam  &
      close $tf
      exit 0
}
$ns at 250 "finish"
$ns run
```

Output:



6. Implementation of Link state routing algorithm.

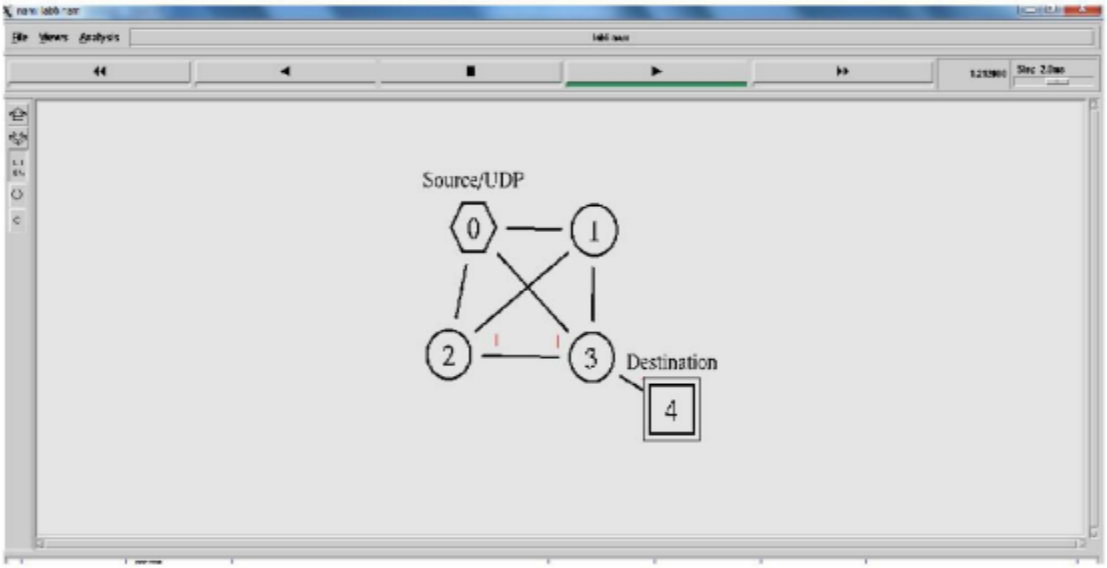



```
#==================================
#     Simulation parameters setup
#==================================
setval(stop)   10.0;# time of simulation end

#==================================
#        Initialization
#==================================
#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
settracefile [open lab6.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
setnamfile [open lab6.nam w]
```

```
$ns namtrace-all $namfile

#===================================
#        Nodes Definition
#===================================
#Create 5 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#===================================
#        Links Definition
#===================================
#Createlinks between nodes
$ns duplex-link $n0 $n1 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n1 50
$ns duplex-link $n0 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 50
$ns duplex-link $n2 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 50
$ns duplex-link $n1 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n3 50
$ns duplex-link $n3 $n4 100.0Mb 10ms DropTail
$ns queue-limit $n3 $n4 50
$ns duplex-link $n0 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n3 50
$ns duplex-link $n1 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 50

#Give node position (for NAM)
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n1 $n3 orient left-down
$ns duplex-link-op $n3 $n4 orient left-down
$ns duplex-link-op $n0 $n3 orient right-down
$ns duplex-link-op $n1 $n2 orient left-down

#Set the link costs. All link costs are symmetric

$ns cost $n0 $n1 2
$ns cost $n0 $n2 1
$ns cost $n0 $n3 3

$ns cost $n1 $n0 2
$ns cost $n1 $n2 2
$ns cost $n1 $n3 3

$ns cost $n2 $n1 2
$ns cost $n2 $n0 1
$ns cost $n2 $n3 1
```

```
$ns cost $n3 $n2 1
$ns cost $n3 $n1 3
$ns cost $n3 $n0 3
$ns cost $n3 $n4 2


$ns cost $n4 $n3 2


#=================================
#        Agents Definition
#=================================
#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set null1 [new Agent/Null]
$ns attach-agent $n4 $null1
$ns connect $udp0 $null1
$udp0 set packetSize_ 1500

#=================================
#        Applications Definition
#=================================
#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"
$ns at 5.0 "$cbr0 stop"
$ns rtproto LS

#=================================
#        Termination
#=================================
#Define a 'finish' procedure
proc finish {} {
global ns tracefilenamfile
    $ns flush-trace
close $tracefile
close $namfile
execnam lab6.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
```

Output:



PART-B:

Implement the

following in C/C++.

1. **Write a program for a HLDC frame to perform the following. i) Bit stuffing ii) Character stuffing.**

i) Bit stuffing:

```
#include<string.h>
#include<stdio.h>
main()
 {
   char a[20],fs[50]="",t[6],r[5];
inti,j,p=0,q=0;
printf("enter bit string : ");
scanf("%s",a);
strcat(fs,"01111110");
```

```
if(strlen(a)<6)
    {
strcat(fs,a);
    }
else
    {
for(i=0;i<strlen(a)-5;i++)
            {
            for(j=i;j<i+6;j++)
               {
            t[p++]=a[j];
               }
            t[p]='\0';
            if(strcmp(t,"011111")==0)
               {
            strcat(fs,"0111110");
                 i=j-1;
               }
            else
               {
            r[0]=a[i];
            r[1]='\0';
            strcat(fs,r);
               }
             p=0;
            }
for(q=i;q<strlen(a);q++)
```

```
        {
            t[p++]=a[q];
        }
    t[p]='\0';
    strcat(fs,t);
     }
    strcat(fs,"01111110");
    printf("After stuffing : %s",fs);
    getch();
     }
```

Output:

ii) Character stuffing:

```
#include<string.h>
#include<stdio.h>
main()
{
 char a[30],fs[50]="",t[3],sd,ed,x[3],s[3],d[3],y[3];
inti,j,p=0,q=0;
printf("Enter characters to be stuffed : ");
scanf("%s",a);
printf("\nEnter a character that represents starting delimiter : ");
scanf(" %c",&sd);
printf("\nEnter a character that represents ending delimiter   : ");
```

```
scanf(" %c",&ed);
x[0]=s[0]=s[1]=sd;
x[1]=s[2]='\0';
y[0]=d[0]=d[1]=ed;
d[2]=y[1]='\0';
strcat(fs,x);
for(i=0;i<strlen(a);i++)
{
t[0]=a[i];
t[1]='\0';
if(t[0]==sd)
strcat(fs,s);
else
if(t[0]==ed)
strcat(fs,d);
else
strcat(fs,t);
 }
strcat(fs,y);
printf("\nAfter stuffing : %s",fs);

getch();
 }
```

Output:

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

```
#include<stdio.h>
#include<stdlib.h>
#define nul 1000
#define nodes 10
int no;
struct node
{
int a[nodes][3];
}
router[nodes];
```

```
voidinit(int r)
{
int i;
for(i=1;i<=no;i++)
{
router[r].a[i][1]=i;
router[r].a[i][2]=999;
router[r].a[i][3]=nul;
}
router[r].a[r][2]=0;
router[r].a[r][3]=r;
}
voidinp(int r)
{
int i;
printf("\n enter dist to the node %d to other nodes",r);
printf("\n please enter 999 if there is no direct route\n");
for(i=1;i<=no;i++)
{
if(i!=r)
{
printf("\n enter dist to the node %d:",i);
scanf("%d",&router[r].a[i][2]);
router[r].a[i][3]=i;
}
}
}
```

```
void display(int r)
{
int i;
printf("\n\n the routimg table for node %d is as follows",r);
for(i=1;i<=no;i++)
{
if(router[r].a[i][2]>=999)
printf("\n\t\t\t %d \t no link \t no hop",router[r].a[i][1]);
else
printf("\n\t\t\t %d \t %d \t\t %d",router[r].a[i][1],router[r].a[i][2],router[r].a[i][3]);
}
}
voiddv_algo(int r)
{
inti,j,z;
for(i=1;i<=no;i++)
{
if(router[r].a[i][2]!=999 && router[r].a[i][2]!=0)
{
for(j=1;j<=no;j++)
{
z=router[r].a[i][2]+router[i].a[j][2];
if(router[r].a[j][2]>z)
{
router[r].a[j][2]=z;
router[r].a[j][3]=i;
}
```

```
}
}
}
}
void find(intx,int y)
{
if(router[x].a[y][3]!=y)
{
find(x,router[x].a[y][3]);
printf("%d-->",router[x].a[y][3]);
find(router[x].a[y][3],y);
return;
}
}
int main()
{
inti,j,x,y;
int choice;
printf("enter the no of nodes required(less than 10 pls):");
scanf("%d",&no);
for(i=1;i<=no;i++)
{
init(i);
inp(i);
}
printf("\n the configuration of the nodes after initilization is as follows:");
for(i=i;i<=no;i++)
```

```
display(i);
for(j=1;j<=no;j++)
for(i=1;i<=no;i++)
dv_algo(i);
printf("\n the configuration of the nodes after computation of path is as follows:");
for(i=1;i<=no;i++)
display(i);
while(1)
{
printf("\n\n enter 1 to continue 0 to quit:");
scanf("\%d",&choice);
if(choice!=1)
break;
printf("\n enter the nodes btn which shortest path is to be found:\n");
scanf("%d%d",&x,&y);
printf("\n the shortest path is:");
printf("%d-->",x);
find(x,y);
printf("%d",y);
printf("\n the length of the shortest path is%d",router[x].a[y][2]);
}
return 0;
}
```

Output:

3. Implement Dijkstra's algorithm to compute the shortest routing path.

```
#include<stdio.h>

void sort(void);

staticintdsp[10][10],nodes;
struct{
charsrc;
chardest;
int length;
}stemp,permanent[10]={' ',' ',0},temp[10]={' ',' ',-1};
staticintperm,tem;

void main()
{
inti,j,k,l,m,n=0,point;
charinitial,dest,path[10]={' '};

printf("\t\t Shortest Path (Dijkstra's algorithm)");
printf("\n**************************************************");
printf("\nEnter the number of nodes:");
```

```
scanf("%d",&nodes);
printf("\nEnter the adjacency matrix for the graph:\n");

for(i=0;i<nodes;i++)
  {
for(j=0;j<nodes;j++)
                    scanf("%d",&dsp[i][j]);
  }
fflush(stdin);
printf("\n enter the source node:");
scanf("%c",&initial);fflush(stdin);
printf("\n Enter the destination node:");
scanf("%c",&dest);
permanent[perm].src=initial;
permanent[perm].dest=initial;
permanent[perm++].length=0;
  i=permanent[perm-1].dest-97;

for(j=0;j<nodes;j++)
  {
if(i!=j)
     {
if(dsp[i][j]>0)
       {
temp[tem].src=permanent[perm-1].src;
temp[tem].dest=j+97;
temp[tem++].length=dsp[i][j];
       }
    }
  }
sort();

while(tem>=0)
  {
j=permanent[perm-1].dest-97;
for(i=0;i<nodes;i++)
    {
if(i!=initial-97)
       {
if(dsp[j][i]>0)
           {
  l=-1;
for(k=0;k<perm;k++)
            {
          if(permanent[k].dest==(i+97))
l=k;
```

```
        }
for(k=0;k<=tem;k++)
            {
                if(temp[k].dest==(i+97))
 l=k;
        }
                if(l<0)
        {
                        temp[tem].src=j+97;
                temp[tem].dest=i+97;
                        for(m=0;m<perm;m++)
                        {
                if(permanent[m].dest==temp[tem].src)
                      n=permanent[m].length;
                        }
                temp[tem++].length=dsp[j][i]+n;
                  }

        else
          {
                for(m=0;m<perm;m++)
                    {
                        if(permanent[m].dest==j+97)
                        {
n=permanent[m].length+dsp[j][i];break;
}
                        else
                                n=dsp[j][i];
                    }
        if((n<temp[l].length))
                    {
temp[l].length=n;
                temp[l].src=j+97;
                        temp[l].dest=i+97;
               }
                  }
               }
      }
   }
sort();
 }
printf("\nShortest path:\n");
printf("From %c to %c is:",initial,dest);

for(i=0;i<perm-1;i++)
  {
```

```
if(permanent[i].dest==dest)
     {
          point=i;n=i;      break;
     }
  }   i=0;

for(j=perm;j>0;j--)
  {
if(permanent[j-1].dest==permanent[point].src)
            {
path[i++]=permanent[point].dest;
point=j-1;
  }
  }
path[i]=initial;
for(j=i;j>=0;j--)
printf("%c ",path[j]);
printf("\t length=%d",permanent[n].length);
getch();
 }

void sort()
{
inti,j,k;
          for(i=0;i<=tem;i++)
{
    k=1;
for(j=0;j<=tem;j++)
               {
                    if((temp[j].length <= temp[j+1].length))
                    {
          stemp=temp[j];
          temp[j]=temp[j+1];
          temp[j+1]=stemp;         k=0;
                    }
               }
if(k)
                    break;
          }

}
```

Output:

4. **For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases: i) Without error ii) With error.**

```
#include<stdio.h>
int a[100],b[100],i,j,len,k,count=0;

//Generator Polynomial:g(x)=x^16+x^12+x^5+1
intgp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,};

int main()
{
        void div();
        printf("\nEnter the length of Data Frame :");
        scanf("%d",&len);
        printf("\nEnter the Message :");
        for(i=0;i<len;i++)
                scanf("%d",&a[i]);

        //Append r(16) degree Zeros to Msg bits
        for(i=0;i<16;i++)
                a[len++]=0;
```

```
//Xr.M(x)  (ie. Msg+16 Zeros)
        for(i=0;i<len;i++)
                b[i]=a[i];

        //No of times to be divided ie.Msg Length
        k=len-16;
        div();
        for(i=0;i<len;i++)
                b[i]=b[i]^a[i];  //MOD 2 Substraction
        printf("\nData to be transmitted : ");
        for(i=0;i<len;i++)
                printf("%2d",b[i]);

        printf("\n\nEnter the Reveived Data : ");
        for(i=0;i<len;i++)
                scanf("%d",&a[i]);

        div();

        for(i=0;i<len;i++)
                if(a[i]!=0)
                {
                        printf("\nERROR in Recived Data");
                        return 0;
                }
        printf("\nDataRecived is ERROR FREE");
}

void div()
{
        for(i=0;i<k;i++)
        {
                if(a[i]==gp[0])
                {
                        for(j=i;j<17+i;j++)
                                a[j]=a[j]^gp[count++];
                }
                count=0;
                                                                }
}
```

Output:

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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
        int i,j,noframes,x,x1=10,x2;
        noframes=10;
        i=1;
        j=1;
                printf("number of frames is %d ",noframes);

        getch();
        while(noframes>0)
        {
                printf("\nsending frames is %d",i);
                x=rand()%10;
                if(x%10==0)
                {
                        for(x2=1;x2<2;x2++)
                        {
```

```
                                printf("\n waiting for %d seconds\n",x2);
                                sleep(x2);
                        }
                        printf("\n sending frames %d\n",i);
                        x=rand()%10;
                }
                printf("\n ack for frame %d\n",j);
                noframes=noframes-1;
i++;
j++;
        }
        printf("\n end of stop and wait protocol\n");

}
```

Output:

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

```
#include<stdio.h>
#include<stdlib.h>
#define MIN(x,y) (x>y)?y:x
int main()
{
intorate,drop=0,cap,x,count=0,inp[10]={0},i=0,nsec,ch;
printf("\n enter bucket size : ");
scanf("%d",&cap);
printf("\n enter output rate :");
scanf("%d",&orate);
do
{
printf("\n enter number of packets coming at second %d :",i+1);
```

```
scanf("%d",&inp[i]);
i++;
printf("\n enter 1 to contiue or 0 to quit.....");
scanf("%d",&ch);
}
while(ch);
nsec=i;
printf("\n second \t recieved \t sent \t dropped \t remained \n");
for(i=0;count || i<nsec;i++)
 {
printf("%d",i+1);
printf(" \t%d\t ",inp[i]);
printf(" \t %d\t ",MIN((inp[i]+count),orate));
if((x=inp[i]+count-orate)>0)
  {
if(x>cap)
{
count=cap;
drop=x-cap;
}
else
{
count=x;
drop=0;
}
}
else
{
drop=0;
count=0;
}
```

```
printf(" \t %d          \t %d \n",drop,count);
}
return 0;
}
```

Output:

VIVA QUESTIONS

1. What is FIFO? How to implement Multiple FIFOs? Disadvantages of FIFO.
2. What is congestion? Explain leaky bucket and token bucket algorithm? What are its disadvantages of each?
3. List other congestion control algorithm.
4. What are the disadvantages of RSA Algorithm?
5. What is the need for routing protocols? List some routing protocols. Which are common routing algorithms used by routing protocols?
6. Disadvantages of Distance vector routing. Explain the method to overcome such disadvantage.
7. What is CRC? What is the need for CRC? Which Layer it is implemented?
8. Can CRC be employed for error correction if so how.
9. Explain steps of error correction and error detection. What is encoding and decoding, why it is required?

10. What is Hamming code? Can it be used for error detection and correction how?
11. What are linear block codes? Explain their advantages.
12. What is protocol? List some protocols. How to modify existing protocol?
13. Difference between TCP/IP and OSI Layer Model?
14. Difference between Router, Bridge, Gateway channel. In which layer it is used?
15. Difference between hub and switch. Which is more compatible and reliable?
16. Explain various Ethernet standards.
17. Difference between IP and MAC address.
18. In which layer ATM protocol is used? Explain the purpose.
19. In Which layer or protocol which converts IP to MAC Address.
20. Which are the different collision detection protocols?
21. Which are the different channelization protocol and explain their needs?
22. Explain term error rate, bandwidth, throughput transmission delay, BER, Interarrivaltime. What is unit of bandwidth?

CONTENT BEYOND SYLLABUS

1. What is congestion/contention Window? How it is constructed and managed and by which layer?
2. List and explain different link types. Differentiate its characteristics.
3. List TCP and UDP Properties?
4. Why TCP is more reliable than UDP? Is data rate and bandwidth are same?
5. Explain various factors that results in congestion and collision.
6. Explain QOS and different measures taken for implementing QOS in network.
7. List different N/W topologies.
8. Difference between Data rate versus Throughput.
9. List the meaning and term used for data in different layers of TCP/IP protocol suite.
10. Difference between intranet, internet and Ethernet. What are the relationships between these network types?