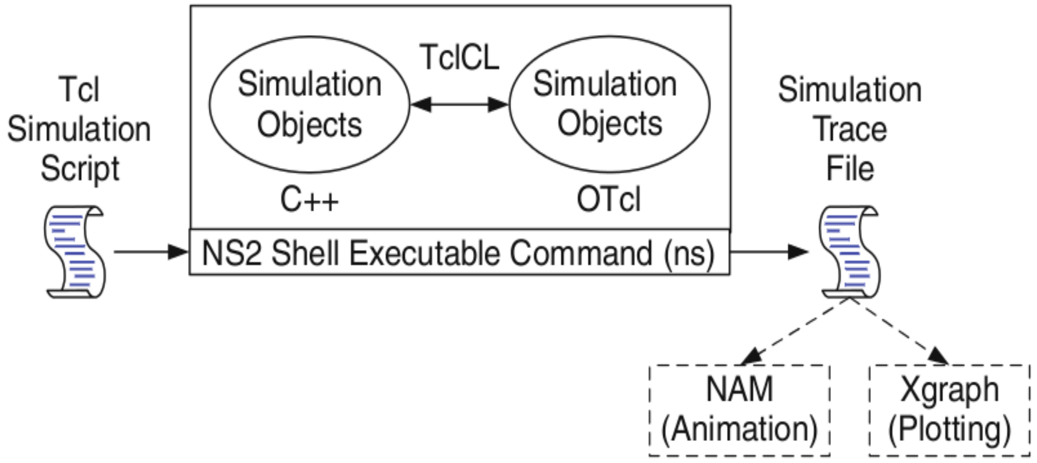
PART-A

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, and Mac.
- 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!**

```
puts stdout{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**

```
proc Diag {a b} {
set c [expr sqrt($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 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
```

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:

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

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

The command

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

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

#Setup a UDP connection

```
set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid_2
```

#setup a CBR over UDP connection

```
set cbr [new
Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

Above 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.

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 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

Part-A

Experiment No: 1 **Date:**

THREE NODE POINT TO POINT NETWORK

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

```
set ns [new Simulator]                          # Letter S is capital
set nf [open PA1.nam w]                         # open a nam trace file in write mode
$ns namtrace-all $nf                            # nf  nam filename
set tf [open PA1.tr w]                          # tf  trace filename
$ns trace-all $tf

proc finish { } {
        global ns nf tf
        $ns flush-trace                         # clears trace file contents
        close $nf
        close $tf
        exec nam PA1.nam &
        exit 0
}
set n0 [$ns node]                               # creates 3 nodes
set n2 [$ns node]
set n3 [$ns node]

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

set udp0 [new Agent/UDP]                        # attaching transport layer protocols
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]          # attaching application layer protocols
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]                      # creating sink(destination) node
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

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

<u>AWK file:</u> *(Open a new editor using "vi command" and write awk file and save with ".awk" extension)*

#immediately after BEGIN should open braces '{'

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

<u>Steps for execution</u>

- *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*

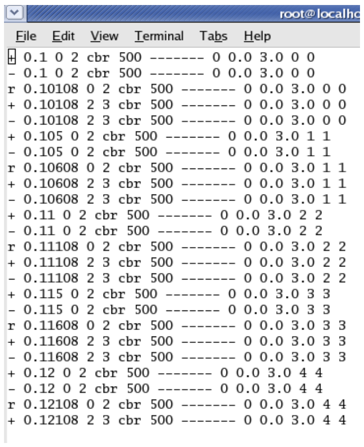
  [root@localhost~]# ns lab1.tcl
- *Here* ***"ns"*** *indicates network simulator. We get the topology shown in the snapshot.*
- *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run* ***awk file*** *to see the output ,*

  [root@localhost~]# awk –f lab1.awk lab1.tr
- *To see the trace file contents open the file as ,*

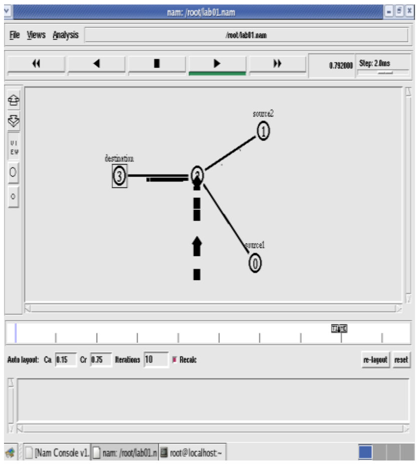
  [root@localhost~]# vi lab1.tr

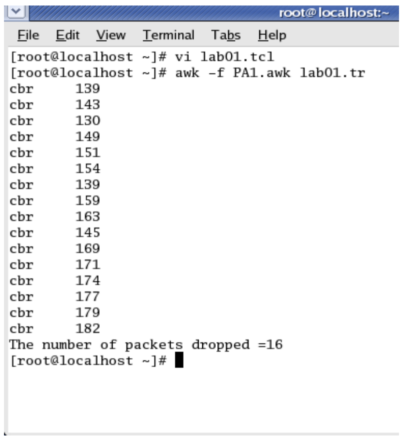
<u>Trace file contains 12 columns:</u>

Event type, Event time, From Node, To Node, Packet Type, Packet Size, Flags (indicated by --------), Flow ID, Source address, Destination address, Sequence ID, Packet ID



```
+ 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
```





```
[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 ~]# █
```

Contents of Trace File   Topology   Output

Experiment No: 2 **Date:**

FOUR NODE POINT TO POINT NETWORK

Aim: *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 agent between n1 – n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP / UDP.*

```
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 file: *(Open a new editor using "vi command" and write awk file and save with ".awk" extension)*

```
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** "*

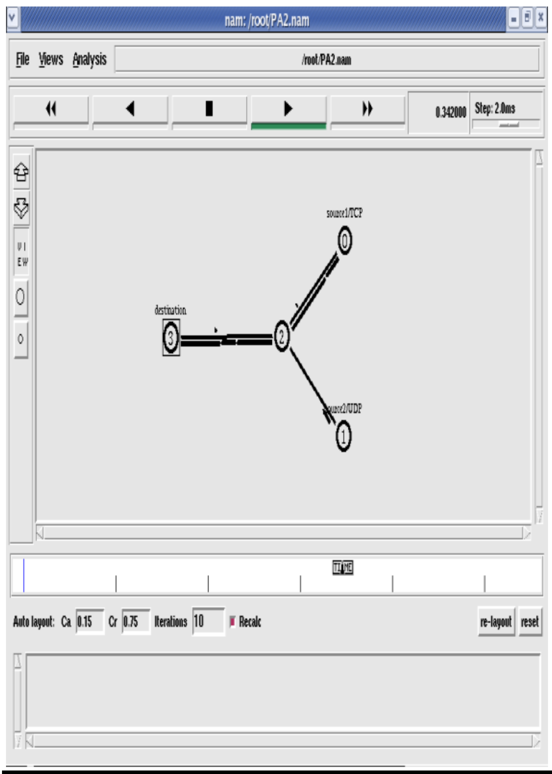
  [root@localhost ~]# vi lab2.tcl
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension "**.awk** "*

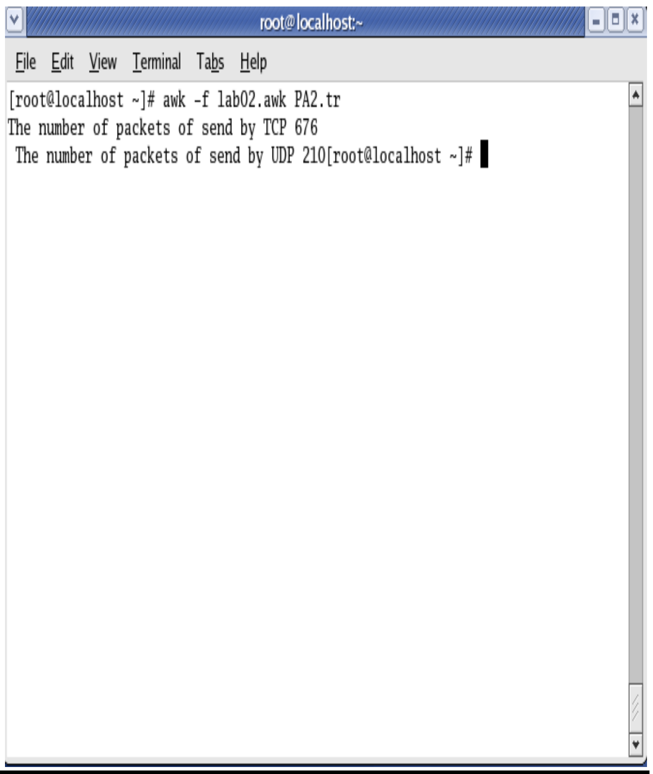
  [root@localhost ~]# vi lab2.awk
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Run the simulation program*

  [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





```
[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 ~]#
```

Topology **Output**

Experiment No: 3 **Date:**

TRANSMISSION OF PING MESSAGE

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

```
set ns [ new Simulator ]

set nf [ open lab4.nam w ]
$ns namtrace-all $nf

set tf [ open lab4.tr w ]
$ns trace-all $tf

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 duplex-link $n0 $n4 1005Mb 1ms DropTail
$ns duplex-link $n1 $n4 50Mb 1ms DropTail
$ns duplex-link $n2 $n4 2000Mb 1ms DropTail
$ns duplex-link $n3 $n4 200Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 1ms DropTail

set p1 [new Agent/Ping] # letters A and P should be capital
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001

set p2 [new Agent/Ping] # letters A and P should be capital
$ns attach-agent $n1 $p2

set p3 [new Agent/Ping] # letters A and P should be capital
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001


set p4 [new Agent/Ping] # letters A and P should be capital
$ns attach-agent $n3 $p4

set p5 [new Agent/Ping] # letters A and P should be capital
$ns attach-agent $n5 $p5
```

```
$ns queue-limit $n0 $n4 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n4 $n5 2
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id]received answer from $from with round trip time $rtt msec"
}
# please provide space between $node_ and id. No space between $ and from. No space
between and $ and rtt */

$ns connect $p1 $p5
$ns connect $p3 $p4

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab4.nam &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
$ns at 1.1 "$p1 send"
$ns at 1.2 "$p1 send"
$ns at 1.3 "$p1 send"
$ns at 1.4 "$p1 send"
$ns at 1.5 "$p1 send"
$ns at 1.6 "$p1 send"
$ns at 1.7 "$p1 send"
$ns at 1.8 "$p1 send"
$ns at 1.9 "$p1 send"
$ns at 2.0 "$p1 send"
$ns at 2.1 "$p1 send"
$ns at 2.2 "$p1 send"
$ns at 2.3 "$p1 send"
$ns at 2.4 "$p1 send"
$ns at 2.5 "$p1 send"
```

```
$ns at 2.6 "$p1 send"
$ns at 2.7 "$p1 send"
$ns at 2.8 "$p1 send"
$ns at 2.9 "$p1 send"

$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"
$ns at 1.1 "$p3 send"
$ns at 1.2 "$p3 send"
$ns at 1.3 "$p3 send"
$ns at 1.4 "$p3 send"
$ns at 1.5 "$p3 send"
$ns at 1.6 "$p3 send"
$ns at 1.7 "$p3 send"
$ns at 1.8 "$p3 send"
$ns at 1.9 "$p3 send"
$ns at 2.0 "$p3 send"
$ns at 2.1 "$p3 send"
$ns at 2.2 "$p3 send"
$ns at 2.3 "$p3 send"
$ns at 2.4 "$p3 send"
$ns at 2.5 "$p3 send"
$ns at 2.6 "$p3 send"
$ns at 2.7 "$p3 send"
$ns at 2.8 "$p3 send"
$ns at 2.9 "$p3 send"

$ns at 3.0 "finish"
$ns run
```

AWK file: *(Open a new editor using "vi command" and write awk file and save with ".awk" extension)*

```
BEGIN{
drop=0;
}
{
 if($1= ="d" )
 {
  drop++;
  }
}
END{
printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);
}
```

<u>Steps for execution:</u>

1) *Open vi editor and type program. Program name should have the extension " **.tcl** "*

   [root@localhost ~]# vi lab4.tcl

2) *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
3) *Open vi editor and type **awk** program. Program name should have the extension "**.awk** "*

   [root@localhost ~]# vi lab4.awk

4) *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
5) *Run the simulation program*

   [root@localhost~]# ns lab4.tcl

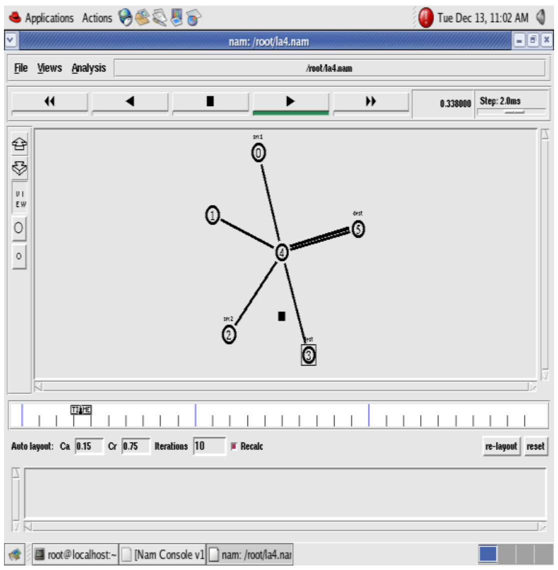
   i) *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
   ii) *Now press the play button in the simulation window and the simulation will begins.*

6) *After simulation is completed run **awk file** to see the output ,*

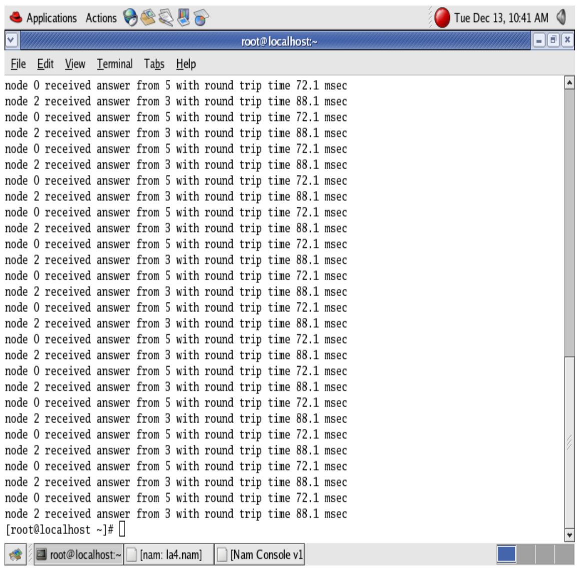
   [root@localhost~]# awk –f lab4.awk lab4.tr

7) *To see the trace file contents open the file as ,*

   [root@localhost~]# vi lab4.tr

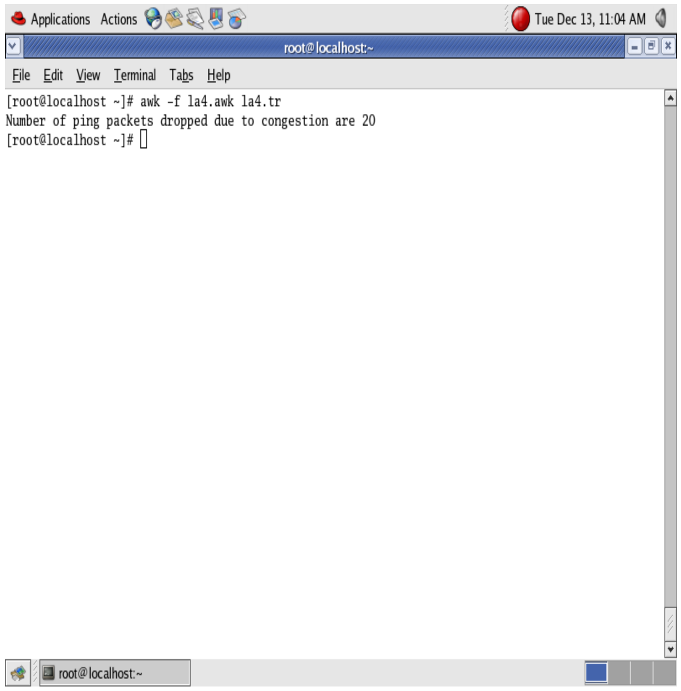


Topology



```
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
[root@localhost ~]#
```

Output



```
[root@localhost ~]# awk -f la4.awk la4.tr
Number of ping packets dropped due to congestion are 20
[root@localhost ~]#
```

Output