Program 1

Implement Three nodes point – to – point network with duplex links between them for different topologies. 1Set the queue size, vary the bandwidth, and find the number of packets dropped for various iterations.

```
set ns [new Simulator]        /* Letter S is capital */
set nf [open lab1.nam w] /* open a nam trace file in write mode */

$ns namtrace-all $nf           /* nf – nam file */

set tf [open lab1.tr w] /* tf- trace file */
$ns trace-all $tf

proc finish { } {        /* provide space b/w proc and finish and all are in small
case */global ns nf tf
$ns flush-trace         /* clears trace file
contents */close $nf
close $tf
exec nam
lab1.nam &
exit 0
}

set n0 [$ns node] /* creates 4
nodes */set n1 [$ns node]
set n2
[$ns
node]
set n3
[$ns
node]

$ns duplex-link $n0 $n2 200Mb 10ms DropTail /*Letter M is capital Mb*/
$ns duplex-link $n1 $n2 100Mb 5ms DropTail /*D and T are capital*/
$ns duplex-link $n2 $n3 1Mb 1000ms DropTail

$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10

set udp0 [new Agent/UDP] /* Letters A,U,D and P are capital */
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR] /* A,T,C,B and R are capital*/
$cbr0 set packetSize_ 500 /*S is capital, space after underscore*/
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
```

```
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2

set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2

set null0 [new Agent/Null] /* A and N are capital */
$ns attach-agent $n3 $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0

$ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1 start"
$ns at 1.0 "finish"

$ns run
```

AWK file (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);
  }
 }
```

/*immediately after END should open braces '{'

```
END{
    printf("The number of packets dropped =%d\n",c);
}
```

Steps for execution

1) Open vi editor and type program. Program name should have the extension " **.tcl** "
   [root@localhost ~]# vi lab1.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 lab1.awk
```

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

```
[root@localhost~]# ns lab1.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 willbegins.

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

```
[root@localhost~]# awk –f lab1.awk lab1.tr
```

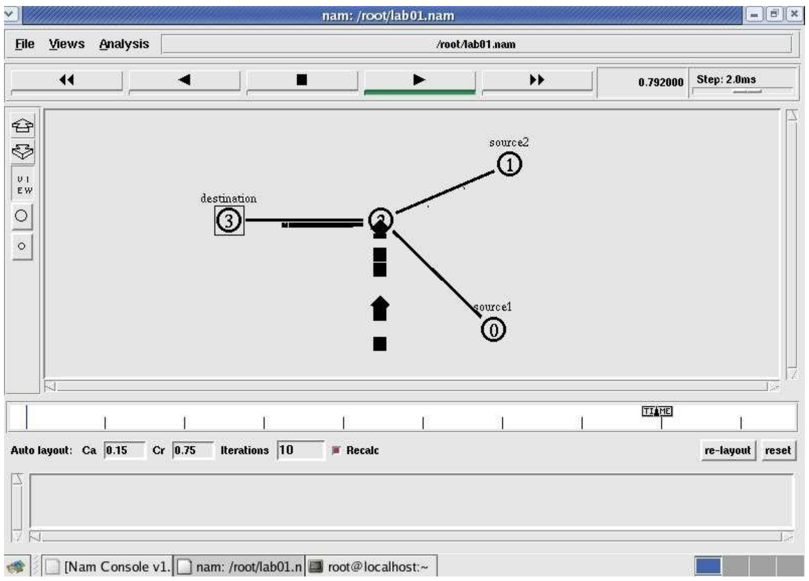
7) To see the trace file contents open the file as ,

```
[root@localhost~]# vi lab1.tr
```

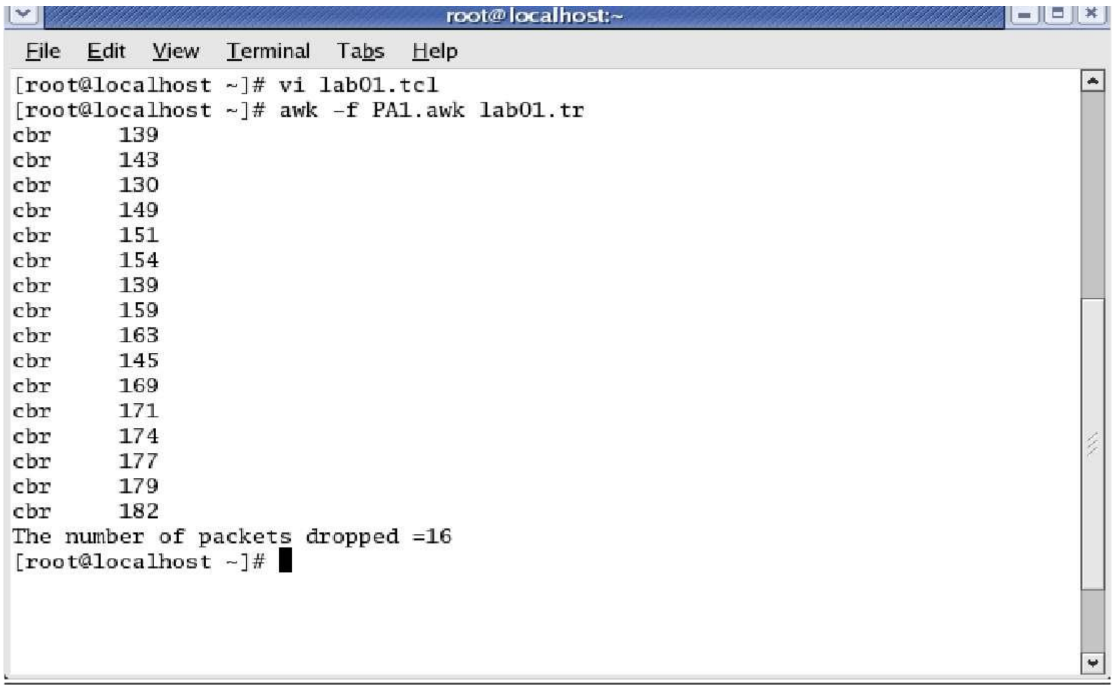
<u>Trace file contains 12 columns:-</u>

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

Topology



Output



```
[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 ~]#
```

Note:

1. Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10 and from n2-n3 as 5.
   Syntax: To set the queue size
   $ns set queue-limit <from> <to> <size> Eg:
   $ns set queue-limit $n0 $n2 10
2. Go on varying the bandwidth from 10, 20 30 . . and find the number of packets dropped at the node

Explanation of the code:

Sl.no	Code	Explanation
1	set ns [new Simulator] /* Letter S is capital */	This line creates a new simulator object named `ns`.
	set nf [open lab1.nam w] /* open a nam trace file in write mode */	A new file named `lab1.nam` is opened in write mode, and the file handle is stored in the variable `nf`. This file will be used for NAM (Network Animator) trace.
3	$ns namtrace-all $nf /* nf – nam file */	This line enables NAM tracing for all events in the simulator (`$ns`) and directs the output to the file represented by the file handle `nf`.
4	set tf [open lab1.tr w] /* tf- trace file */	Similar to the NAM trace file, a

		new file named `lab1.tr` is opened in write mode, and the file handle is stored in the variable `tf`. This file will be used for general trace output.
5	$ns trace-all $tf	This line enables general tracing for all events in the simulator (`$ns`) and directs the output to the file represented by the file handle `tf`.

6	proc finish { } { /* provide space b/w proc and finish and all are in small case */
	  global ns nf tf
	  $ns flush-trace/* clears trace file contents */
	  close $nf
	  close $tf
	  exec nam lab1.nam & exit 0
	}
		This block defines a procedure named `finish`. It flushes the traces, closes the NAM and general trace files, and then executes the NAM animator with the `lab1.nam` file. The `& exit 0` ensures that the script exits after executing NAM.

7	set n0 [$ns node] /* creates 4 nodes */ set n1 [$ns node] set n2 [$ns node] set n3 [$ns node]	These lines create four nodes (`n0`, `n1`, `n2`, and `n3`) in the network simulation.
8	$ns duplex-link $n0 $n2 200Mb 10ms DropTail /*Letter M is capital Mb*/ $ns duplex-link $n1 $n2 100Mb 5ms DropTail /*D and T are capital*/ $ns duplex-link $n2 $n3 1Mb 1000ms DropTail	These lines create duplex links between nodes with specified bandwidth, delay, and queuing mechanism. The third and fourth arguments represent bandwidth and delay in each case.
9	$ns queue-limit $n0 $n2 10 $ns queue-limit $n1 $n2 10	These lines set the queue limit for the links between nodes.
	set udp0 [new Agent/UDP] /* Letters A,U,D and P are capital */ $ns attach-agent $n0 $udp0	Here, an Agent/UDP object named `udp0` is created, and it is attached to node `n0`.
	set cbr0 [new Application/Traffic/CBR] /* A,T,C,B and R are capital*/ $cbr0 set packetSize_ 500 /*S is capital, space after underscore*/ $cbr0 set interval_ 0.005 $cbr0 attach-agent $udp0	This block creates a CBR traffic source (`cbr0`) attached to `udp0`, and sets its packet size and interval.
	$ns connect $udp0 $null0 $ns connect $udp1 $null0	These lines connect the UDP agents to a Null agent, effectively directing their traffic to a "black hole."

	$ns at 0.1 "$cbr0 start" $ns at 0.2 "$cbr1 start" $ns at 1.0 "finish"	These lines schedule the start of CBR traffic from `cbr0` and `cbr1` at simulation time 0.1 and 0.2, respectively. The `finish` procedure is scheduled to run at simulation time 1.0.
	$ns run	This line starts the simulation.
	AWK file	

1	BEGIN
	{
	   C=0;
	}
		This block is the BEGIN block in AWK, which is executed before processing any lines from the input. In this block, a variable `C` is initialized to 0.

2	{
	   if ($1 == "d")
	   {
	      C++;
	      printf("%s\t%s\n", $5, $11);
	   }
	}
		This block is executed for each line in the input. It checks if the first field (`$1`) is equal to "d". If true, it increments the variable `C` by 1 and prints the 5th and 11th fields separated by a tab.

3	END
	{
	   printf("The number of packets dropped = %d\n", C);
	}
		This block is the END block in AWK, which is executed after processing all lines from the input. It prints a message indicating the number of packets dropped, using the value stored in the variable `C`.

AWK file:

AWK, a text processing and pattern scanning language.

Logic Explanation:

1. The `BEGIN` block initializes a counter variable `C` to 0.

2. The main block checks each line of input. If the first field is "d", it increments the counter `C` and prints the 5th and 11th fields.

3. The `END` block prints the total number of packets dropped based on the value of the counter `C`.

Program 2:

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

```
set ns [new Simulator]
set tf [open lab8.tr w]
$ns trace-all $tf
set topo [new Topography]
$topoload_flatgrid 1000 1000
set nf [open lab8.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 \
-prrootype Propagation/TwoRayGround \
-antType Antenna/OmniAntenna \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON
create-god 3
set n0 [$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
exec nam lab8.nam &
close $tf
exit 0
}
$ns at 250 "finish"
$ns run
```

This code is a script written in the Tcl scripting language for the Network Simulator (NS-2). NS-2 is a discrete event simulator used for simulating network protocols and scenarios.

Sl.no	Command	Function
1	set ns [new Simulator]	Creates a new simulation object (ns) of the Simulator class.
2	set tf [open lab8.tr w]	Opens a new file named "lab8.tr" for writing. This file will be used to store the simulation trace.
3	`$ns trace-all $tf	Configures the simulator to trace all events and write the trace to the "lab8.tr" file.
4	set topo [new Topography]	Creates a new topography object (topo) to represent the simulated environment.
5	$topoload_flatgrid 1000 1000	Loads a flat grid of size 1000x1000 in the topography
6	set nf [open lab8.nam w]	Opens a new file named "lab8.nam" for writing. This file will be used for network animation (using the Network Animator - Nam).
7	$ns namtrace-all-wireless $nf 1000 1000	Configures the simulator to trace all wireless events and write the trace to the

		"lab8.nam" file. The simulation area is set to 1000x1000.
8	$ns node-config -adhocRouting DSDV \ -llType LL \ -macType Mac/802_11 \ -ifqType Queue/DropTail \ -ifqLen 50 \ -phyTypePhy/WirelessPhy \ -channelType Channel/WirelessChannel \ -prrootype Propagation/TwoRayGround \ -antType Antenna/OmniAntenna \ -topoInstance $topo \ -agentTrace ON \ -routerTrace ON	Configures the nodes in the simulation with various parameters, such as ad-hoc routing protocol (DSDV), link layer type, MAC type, queue type, physical layer type, etc.
9	create-god 3	Creates a God object to manage global information in the simulation for 3 nodes
10	set n0 [$ns node] set n1 [$ns node] set n2 [$ns node]	Creates three nodes (n0, n1, and n2) in the simulation.
11	$n0 label "tcp0" $n1 label "sink1/tcp1" $n2 label "sink2"	Labels the nodes for better identification in the simulation.
12	$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	Setting initial positions for the nodes
13	$ns at 0.1 "$n0 setdest 50 50 15" $ns at 0.1 "$n1 setdest 100 100 25" $ns at 0.1 "$n2 setdest 600 600 25"	Sets the initial destination for node n0 to (50, 50, 15) at simulation time 0.1.
14	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	Agents and applications are configured for communication between nodes using TCP and FTP **Detailed Explanation at Table 2**

	$ns at 5 "$ftp0 start" $ns at 5 "$ftp1 start"	
15	$ns at 100 "$n1 setdest 550 550 15" $ns at 190 "$n1 setdest 70 70 15"	behavior of node n1 during the course of the simulation. The destinations specified in these lines represent the new positions to which node n1will move at the specified simulation times. 1. $ns at 100 "$n1 setdest 550 550 15": At simulation time 100, it sets the destination of node n1to (550, 550, 15). This means that at time 100, node n1 will change its destination to (550, 550, 15). 2. $ns at 190 "$n1 setdest 70 70 15": At simulation time 190, it sets the destination of node n1 to (70, 70, 15). This means that at time 190, node n1 will change its destination to (70, 70, 15).

16	proc finish { } {
	global ns nftf
	$ns flush-trace
	exec nam lab8.nam &
	close $tf
	exit 0
	}
		Defines a procedure called finis` to perform cleanup tasks when the simulation ends.

17	$ns at 250 "finish"	Schedules thefinishprocedure to be executed at simulation time 250
18	$ns run	Initiates the simulation.

The code sets up a wireless ad-hoc network with three nodes, configures their initial positions, defines communication between nodes using TCP and FTP, and schedules the simulation to run until time 250, at which point the `finish` procedure is called to perform cleanup tasks. The simulation trace is written to "lab8.tr," and the network animation trace is written to "lab8.nam."

Table 2:

Sl. no	Lines of code	Detailed Explaination
1	-llType LL \	-Specifies the link layer type. In

	-macType Mac/802_11 \\ -ifqType Queue/DropTail \\ -ifqLen 50 \\ -phyTypePhy/WirelessPhy \\ -channelType Channel/WirelessChannel \\ -prrootype Propagation/TwoRayGround \\ -antType Antenna/OmniAntenna \\ -topoInstance $topo \\ -agentTrace ON \\ -routerTrace ON	this case, it's set to "LL" (Link Layer). - Specifies the MAC (Medium Access Control) layer type. It's set to "Mac/802_11," indicating the use of the IEEE 802.11 standard for wireless LANs. - Specifies the type of interface queue. In this case, it's set to a drop-tail queue. - Sets the length of the interface queue to 50. This parameter defines the maximum number of packets that can be enqueued at the interface. - Specifies the physical layer type. It's set to "Phy/WirelessPhy," indicating the use of a wireless physical layer. - Specifies the type of channel. It's set to "Channel/WirelessChannel," indicating the use of a wireless channel. - Specifies the type of propagation model. In this case, it's set to "Propagation/TwoRayGround," which is a commonly used propagation model for wireless communication. - Specifies the type of antenna. It's set to "Antenna/OmniAntenna," indicating the use of an omnidirectional antenna. - Specifies the topology instance to be used. It uses the previously created topography object \`$topo\` for the simulation. - Enables tracing for agents. This means that information about agent activities will be recorded in the simulation trace - Enables tracing for routers. This means that information

		about router activities will be recorded in the simulation trace.
2	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"	**set up TCP agents, FTP applications, and TCP sinks for communication between nodes** 1. set tcp0 [new Agent/TCP]: Creates a new TCP agent object namedtcp0 2. Attaches the TCP agent tcp0 to node n0. This means that node n0 will use this TCP agent for communication. 3. Creates a new FTP application object named ftp0. 4. This means that the FTP application will use the TCP agent for sending data. 5. Creates a new TCP sink agent object named sink1. A TCP sink is used to receive data. 6. This means that node n1 will use this TCP sink for receiving data. 7. This establishes the communication link between node n0 (sending data) and node n1 (receiving data). 8. Creates a new TCP agent object named tcp1 Attaches the TCP agent tcp1 to node n1. Creates a new FTP application object named ftp1. Attaches the TCP agent tcp1 to the FTP application ftp1. Creates a new TCP sink agent object named sink2. Attaches the TCP sink agent

		sink2 to node n2. Connects the TCP agent tcp1to the TCP sink sink2. 9. Schedules the FTP application ftp0 to start sending data at simulation time 5. 10. Schedules the FTP application ftp1 to start sending data at simulation time 5. These lines essentially set up two TCP connections with FTP applications, where tcp0 and ftp0 are associated with the communication between nodes n0 and n1, and tcp1 and ftp1 are associated with the communication between nodes n1 and n2. The data transmission is scheduled to start at simulation time 5.

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

```
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= ="_2_" && $4= ="AGT")
{ count2++

pack2=pack2+$8
time2=$2 }
}
END{
printf("The Throughput from n0 to n1: %f Mbps \n”,
((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps", ((count2*pack2*8)/(time2*1000000)));
}
```

Output:

<u>**Steps for execution**</u>

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

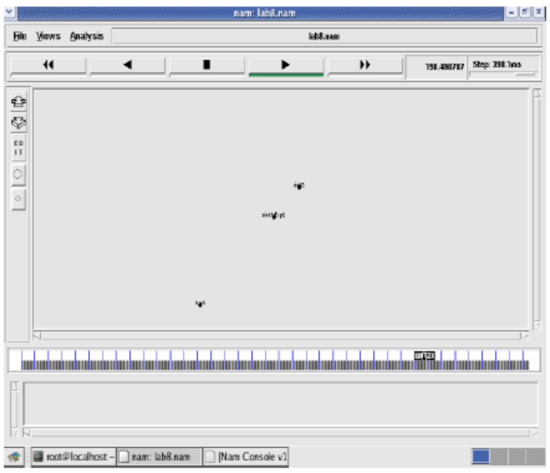
  [root@localhost ~]# vi lab8.tcl
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension "**.awk** "*

  [root@localhost ~]# vi lab8.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*

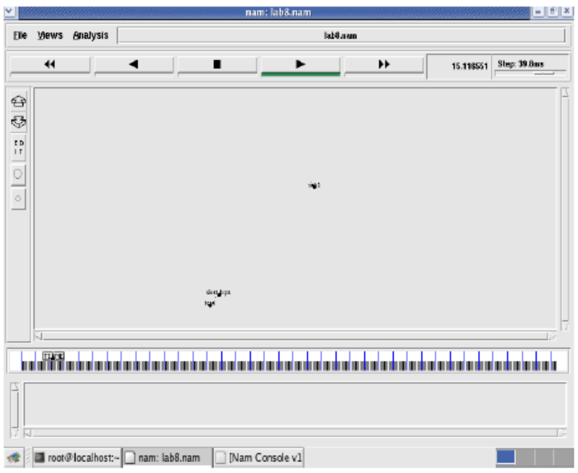
  [root@localhost~]# ns lab8.tcl
  - *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
  - *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output ,*

  [root@localhost~]# awk –f lab8.awk lab8.tr
- *To see the trace file contents open the file as ,*

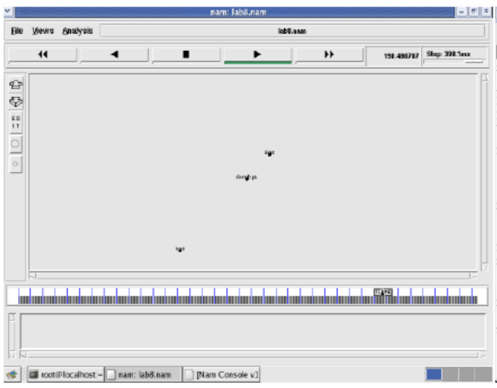
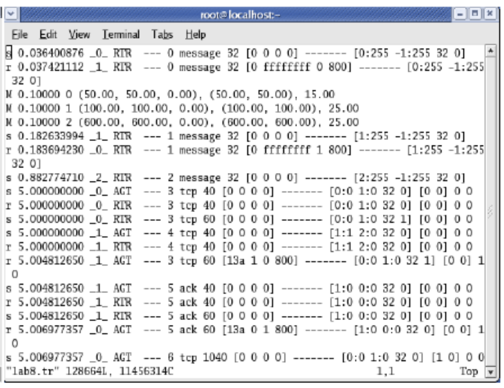
  [root@localhost~]# vi lab8.tr



Node 1 and 2 are communicating



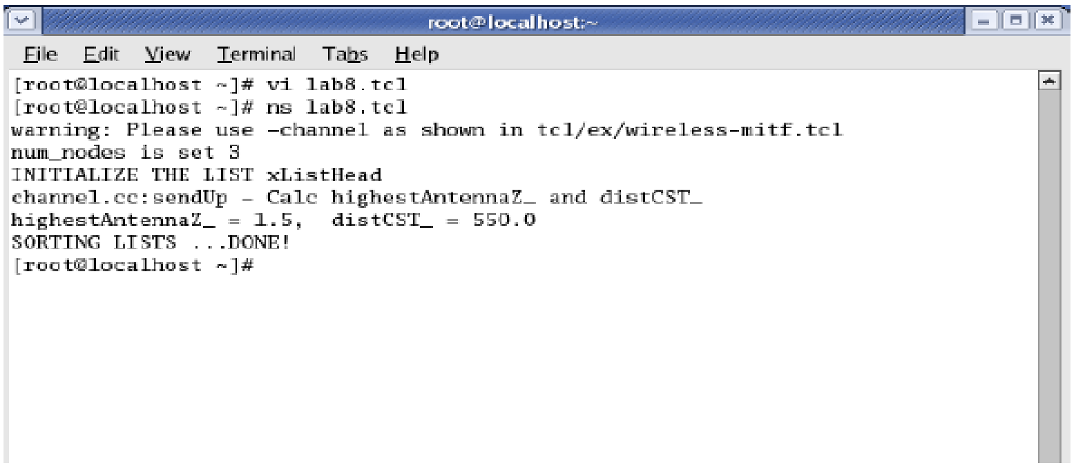
Node 2 is moving towards node 3

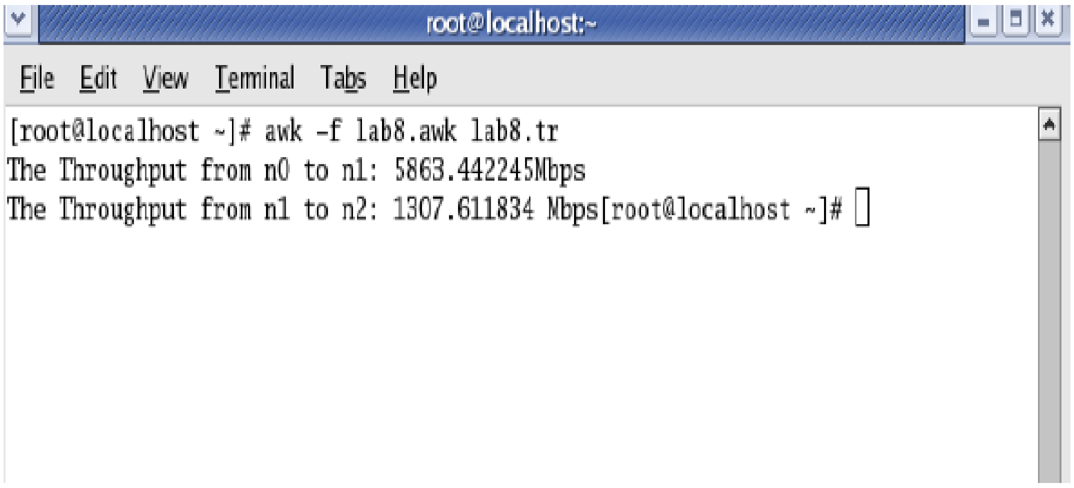
```
s 0.036400876 _0_ RTR  --- 0 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
r 0.037421112 _1_ RTR  --- 0 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255
 32 0]
M 0.10000 0 (50.00, 50.00, 0.00), (50.00, 50.00), 15.00
M 0.10000 1 (100.00, 100.00, 0.00), (100.00, 100.00), 25.00
M 0.10000 2 (600.00, 600.00, 0.00), (600.00, 600.00), 25.00
s 0.182633994 _1_ RTR  --- 1 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
r 0.183694230 _0_ RTR  --- 1 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255
 32 0]
s 0.882774710 _2_ RTR  --- 2 message 32 [0 0 0 0] ------- [2:255 -1:255 32 0]
s 5.000000000 _0_ AGT  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 5.000000000 _0_ RTR  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 5.000000000 _0_ RTR  --- 3 tcp 60 [0 0 0 0] ------- [0:0 1:0 32 1] [0 0] 0 0
s 5.000000000 _1_ AGT  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.000000000 _1_ RTR  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.004812650 _1_ AGT  --- 3 tcp 60 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1
 0
s 5.004812650 _1_ AGT  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.004812650 _1_ RTR  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
s 5.004812650 _1_ RTR  --- 5 ack 60 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.006977357 _0_ AGT  --- 5 ack 60 [13a 0 1 800] ------- [1:0 0:0 32 0] [0 0] 1
 0
s 5.006977357 _0_ AGT  --- 6 tcp 1040 [0 0 0 0] ------- [0:0 1:0 32 0] [1 0] 0 0
"lab8.tr" 128664L, 11456314C                              1,1           Top
```

Node 2 is coming back from node 3 towards node1 Trace File

Here **"M"** indicates mobile nodes, **"AGT"** indicates Agent Trace, **"RTR"** indicates Router Trace



```
[root@localhost ~]# vi lab8.tcl
[root@localhost ~]# ns lab8.tcl
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
num_nodes is set 3
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
[root@localhost ~]#
```



```
[root@localhost ~]# awk -f lab8.awk lab8.tr
The Throughput from n0 to n1: 5863.442245Mbps
The Throughput from n1 to n2: 1307.611834 Mbps[root@localhost ~]#
```

Program 3

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

Java is a general-purpose computer programming language that is simple, concurrent, class-based, object-oriented language. The compiled Java code can run on all platforms that support Java without the need for recompilation hence Java is called as "write once, run anywhere" (WORA).The Java compiled intermediate output called “byte-code” that can run on any Java virtual machine (JVM) regardless of computer architecture. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.
In Linux operating system Java libraries are preinstalled. It’s very easy and convenient to compile and run Java programs in Linux environment. To compile and run Java Program is a two-step process:
1. Compile Java Program from Command Prompt
[root@host ~]# javac Filename.java
The Java compiler (Javac) compiles java program and generates a byte-code with the same file name and .class extension.
2. Run Java program from Command Prompt
[root@host ~]# java Filename The java interpreter (Java) runs the byte-code and gives the respective output. It is important to note that in above command we have omitted the .class suffix of the byte- code (Filename.class).

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The CRC calculation or cyclic redundancy check was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used. The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation—by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "Lammert"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient. All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an exclusive or operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

• The message bits are appended with c zero bits; this augmented message is the dividend
• A predetermined c+1-bit binary sequence, called the generator polynomial, is the divisor
• The checksum is the c-bit remainder that results from the division operation

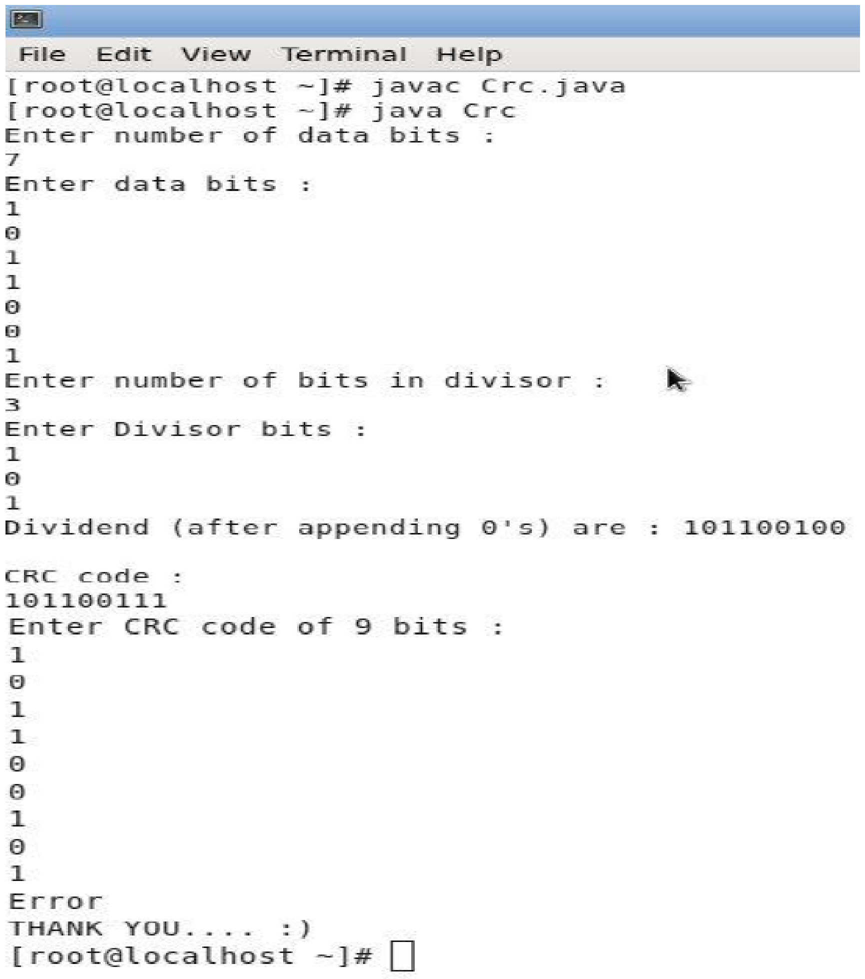
Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

```
import java.io.*;
class Crc
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[ ] data;
int[ ]div;
int[ ]divisor;
int[ ]rem;
int[ ] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());
/* System.out.print("Data bits are : ");
for(int i=0; i< data_bits; i++)
System.out.print(data[i]);
System.out.println();
System.out.print("divisor bits are : ");
for(int i=0; i< divisor_bits; i++)
System.out.print(divisor[i]);
System.out.println();
*/ tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
/*------------------ CRC GENERATION-----------------------*/
for(int i=0;i<data.length;i++)
div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
```

```
for(int i=0; i< div.length; i++)
System.out.print(div[i]);
System.out.println();
for(int j=0; j<div.length; j++){
rem[j] = div[j];
}
rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++) //append dividend and ramainder
{
crc[i]=(div[i]^rem[i]);
}
System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);
/*-------------------ERROR DETECTION---------------------*/
System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
crc[i]=Integer.parseInt(br.readLine());
/* System.out.print("crc bits are : ");
for(int i=0; i< crc.length; i++)
System.out.print(crc[i]);
System.out.println();
*/
for(int j=0; j<crc.length; j++){
rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error");
break;
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
}
static int[] divide(int div[],int divisor[], int rem[])
{
int cur=0;
while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);
while(rem[cur]==0 && cur!=rem.length-1)
```

```
cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}
```

Output:



```
[root@localhost ~]# javac Crc.java
[root@localhost ~]# java Crc
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 101100100

CRC code :
101100111
Enter CRC code of 9 bits :
1
0
1
1
0
0
1
0
1
Error
THANK YOU.... :)
[root@localhost ~]#
```

Code Explanation

```
import java.io.*;
class Crc {
```

This line imports the necessary Java input/output classes and defines a class named `Crc`.

```
public static void main(String args[]) throws IOException {
```

This line declares the main method. The `throws IOException` indicates that the method may throw an IOException, so it needs to be handled or declared.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Here, a BufferedReader object `br` is created to read input from the console.

```
int[] data;
int[] div;
int[] divisor;
int[] rem;
int[] crc;
```

These lines declare arrays to store data, divisor, remainder, and CRC values.

```
int data_bits, divisor_bits, tot_length;
```

Variables `data_bits`, `divisor_bits`, and `tot_length` are declared to store the number of data bits, number of bits in the divisor, and the total length.

```
System.out.println("Enter number of data bits : ");
data_bits = Integer.parseInt(br.readLine());
```

The program prompts the user to enter the number of data bits and reads the input from the console.

```
data = new int[data_bits];
System.out.println("Enter data bits : ");
for (int i = 0; i < data_bits; i++)
    data[i] = Integer.parseInt(br.readLine());
```

An array `data` is created to store the data bits, and the program prompts the user to enter the data bits, which are then read and stored in the array.

```
System.out.println("Enter number of bits in divisor : ");
divisor_bits = Integer.parseInt(br.readLine());
```

The program prompts the user to enter the number of bits in the divisor and reads the input.

```
divisor = new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for (int i = 0; i < divisor_bits; i++)
    divisor[i] = Integer.parseInt(br.readLine());
```

An array `divisor` is created to store the divisor bits, and the program prompts the user to enter the divisor bits, which are then read and stored in the array.

```
tot_length = data_bits + divisor_bits - 1;
div = new int[tot_length];
rem = new int[tot_length];
crc = new int[tot_length];
```

The total length is calculated, and arrays `div`, `rem`, and `crc` are initialized with the calculated length.

```
/*------------------ CRC GENERATION-----------------------*/
for (int i = 0; i < data.length; i++)
    div[i] = data[i];
```

The data bits are copied to the `div` array.

```
System.out.print("Dividend (after appending 0's) are : ");
for (int i = 0; i < div.length; i++)
    System.out.print(div[i]);
System.out.println();
```

The program prints the dividend (data bits after appending 0's) to the console.

```
for (int j = 0; j < div.length; j++) {
    rem[j] = div[j];
}
```

The remainder array is initialized with the values of the dividend.

```
rem = divide(div, divisor, rem);
```

The `divide` method is called to perform the division.

```
for (int i = 0; i < div.length; i++) {
    crc[i] = (div[i] ^ rem[i]);
}
```

The CRC code is generated by XORing the original data bits with the remainder.

```
System.out.println();
System.out.println("CRC code : ");
for (int i = 0; i < crc.length; i++)
    System.out.print(crc[i]);
```

The generated CRC code is printed to the console.

```
/*-------------------ERROR DETECTION---------------------*/
System.out.println();
System.out.println("Enter CRC code of " + tot_length + " bits : ");
for (int i = 0; i < crc.length; i++)
    crc[i] = Integer.parseInt(br.readLine());
```

The program prompts the user to enter the CRC code for error detection.

```
for (int j = 0; j < crc.length; j++) {
    rem[j] = crc[j];
}
rem = divide(crc, divisor, rem);
```

The remainder array is initialized with the entered CRC code, and the `divide` method is called to perform the division.

```
for (int i = 0; i < rem.length; i++) {
    if (rem[i] != 0) {
        System.out.println("Error");
        break;
    }
    if (i == rem.length - 1)
        System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
```

The program checks for errors in the remainder and prints whether an error is detected or not. Finally, a thank you message is printed.

```
static int[] divide(int div[], int divisor[], int rem[]) {
    int cur = 0;
    while (true) {
        for (int i = 0; i < divisor.length; i++)
            rem[cur + i] = (rem[cur + i] ^ divisor[i]);
        while (rem[cur] == 0 && cur != rem.length - 1)
            cur++;
        if ((rem.length - cur) < divisor.length)
            break;
    }
    return rem;
}
```

The `divide` method performs the polynomial division and returns the remainder. The XOR operation is used to update the remainder during the division process. The method is called with the dividend, divisor, and initial remainder arrays.

Program 4:
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 nf [ open lab2.nam w ]
$ns namtrace-all $nf
set tf [ open lab2.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 lab2.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);
}
```

Steps for execution

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

```
[root@localhost ~]# vi lab2.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 lab2.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

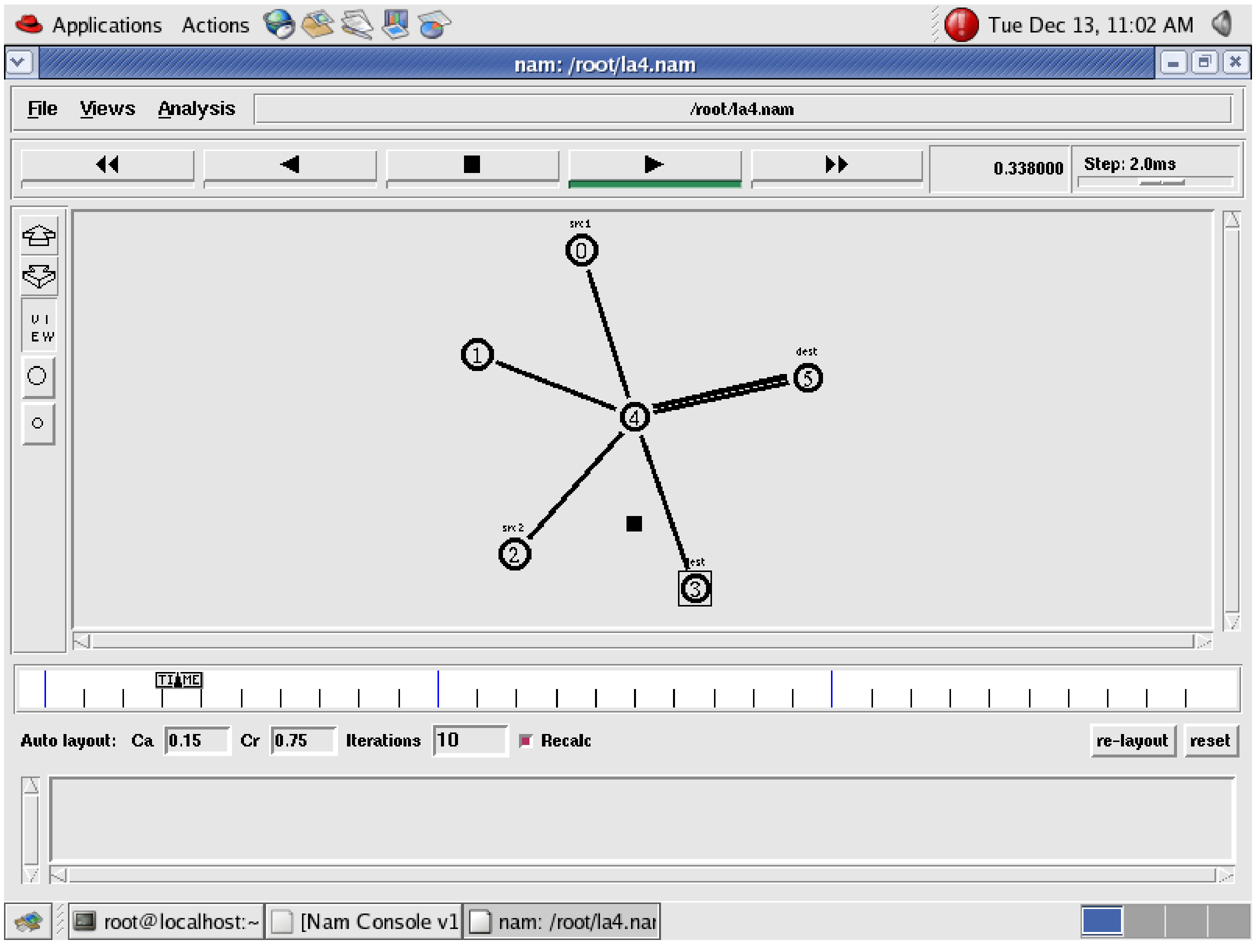
```
[root@localhost~]# ns lab2.tcl
```

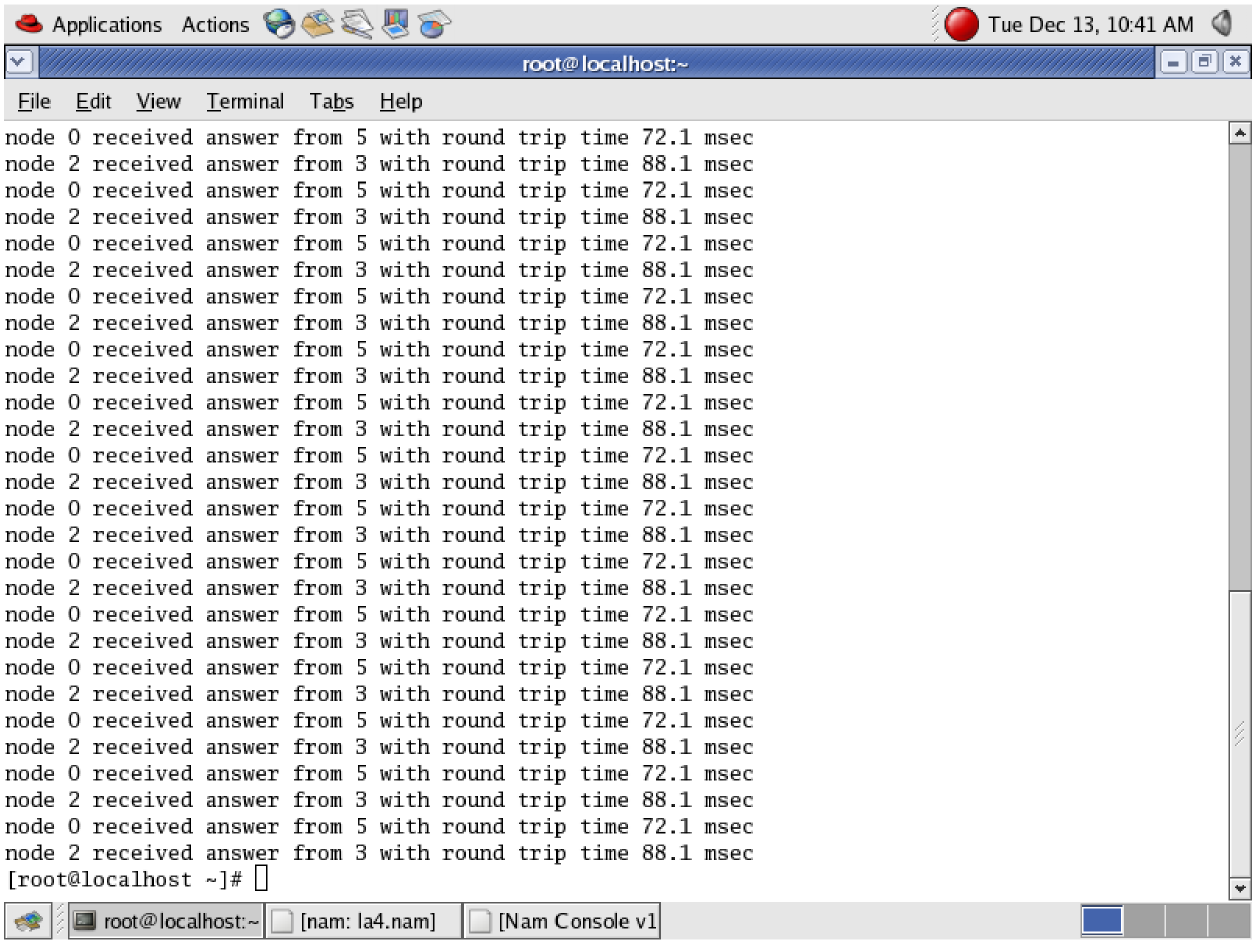
*i) Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.*

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 ,*

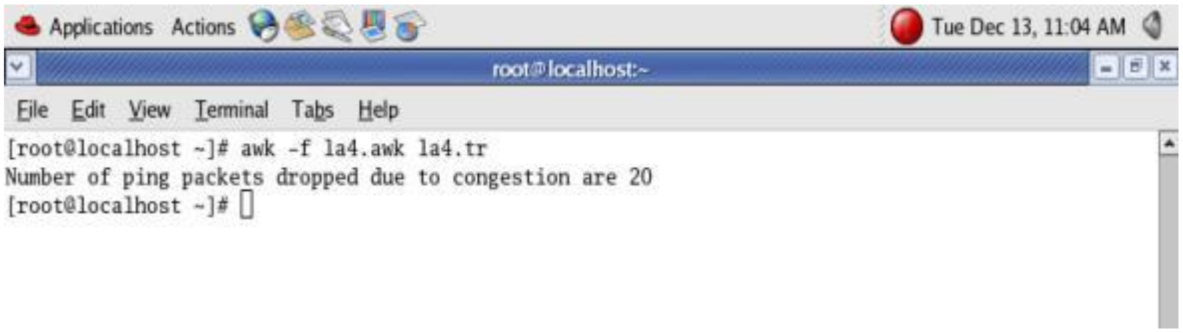
[root@localhost~]# awk –f lab2.awk lab2.tr
7) To see the trace file contents open the file as ,
[root@localhost~]# vi lab2.tr

Output:





```
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 ~]#
```



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

Explanation for the code:

Certainly, let's go through the code line by line:

set ns [new Simulator]

- This line creates a new ns-2 simulator object and assigns it to the variable `ns`.

set nf [open lab2.nam w]

- Opens a file named "lab2.nam" for writing and assigns the file identifier to the variable `nf`. This file will be used for network animation.

```
$ns namtrace-all $nf
```

- Enables tracing of all events in the simulator and directs the output to the file specified by `nf` (lab2.nam).

```
set tf [ open lab2.tr w ]
```

- Opens a file named "lab2.tr" for writing and assigns the file identifier to the variable `tf`. This file will store the simulation trace.

```
$ns trace-all $tf
```

- Enables tracing of all events in the simulator and directs the output to the file specified by `tf` (lab2.tr).

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
```

- Creates six nodes (`n0` to `n5`) in the network.

```
$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
```

- Sets up duplex links between nodes with specified characteristics (bandwidth, delay, and queuing discipline).

```
set p1 [new Agent/Ping]
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001
```

- Creates a Ping agent `p1`, attaches it to node `n0`, and configures its packet size and interval.

```
set p2 [new Agent/Ping]
$ns attach-agent $n1 $p2
```

- Creates another Ping agent `p2`, attaches it to node `n1`.

```
set p3 [new Agent/Ping]
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001
```

- Creates a third Ping agent `p3`, attaches it to node `n2`, and configures its packet size and interval.

```
set p4 [new Agent/Ping]
$ns attach-agent $n3 $p4
```

- Creates another Ping agent `p4`, attaches it to node `n3`.

```
set p5 [new Agent/Ping]
$ns attach-agent $n5 $p5
```

- Creates the last Ping agent `p5` and attaches it to node `n5`.

```
$ns queue-limit $n0 $n4 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n4 $n5 2
```

- Sets queue limits for specific links.

```
Agent/Ping instproc recv {from rtt} {
   $self instvar node_
   puts "node [$node_ id] received answer from $from with round trip time $rtt msec"
}
```

- Defines a procedure named `recv` for the Ping agent to handle received responses. It prints information about the received packets, including the source node and round-trip time.

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

- Connects Ping agents to each other.

```
proc finish { } {
    global ns nf tf
    $ns flush-trace
    close $nf
    close $tf
    exec nam lab2.nam &
    exit 0
}
```

- Defines a procedure named `finish` to be executed when the simulation is complete. It flushes the trace, closes the animation and trace files, and then invokes the Network Animator (nam) to visualize the network animation.

```
$ns at 0.1 "$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"
```

- Schedules Ping messages to be sent at specific time intervals using `$ns at`.

```
$ns run
```

- Initiates the simulation by running the ns-2 simulator.

The code sets up a network simulation with nodes exchanging Ping messages, configures various parameters, schedules events, and defines procedures to handle events and finish the simulation. The results are then visualized using the Network Animator (nam).

Program 5:
Write a program to find the shortest path between vertices using bellman-ford algorithm.

program:

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

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

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstras algorithm for the same problem, but more versatile, as it is capable of handling graphs in whichsome of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, thenthere is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence

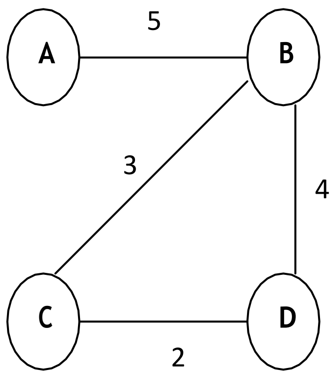
Source code:

```
import java.util.Scanner;

public class BellmanFord
{
private int D[];
private int num_ver;
public static final int MAX_VALUE = 999;

public BellmanFord(int num_ver)
{
                this.num_ver = num_ver;
                D = new int[num_ver + 1];
}
```

```
    public void BellmanFordEvaluation(int source, int A[][])
    {
            for (int node = 1; node <= num_ver; node++)
            {
                    D[node] = MAX_VALUE;

            }
            D[source] = 0;
            for (int node = 1; node <= num_ver - 1; node++)
            {
                    for (int sn = 1; sn <= num_ver; sn++)
                    {
                            for (int dn = 1; dn <= num_ver; dn++)
                            {
                                    if (A[sn][dn] != MAX_VALUE)
                                    {
                                            if (D[dn] > D[sn]+ A[sn][dn])
                                                    D[dn] = D[sn] + A[sn][dn];
                                     }
                            }
                     }
             }
             for (int sn = 1; sn <= num_ver; sn++)
            {
                     for (int dn = 1; dn <= num_ver; dn++)
                    {
                             if (A[sn][dn] != MAX_VALUE)
                            {
                                     if (D[dn] > D[sn]+ A[sn][dn])
                                     System.out.println("The Graph contains negative
                                     egde cycle");
                            }
                     }
              }
           for (int vertex = 1; vertex <= num_ver; vertex++)
           {
                     System.out.println("distance of source " + source + " to
                     "+ vertex + "is " + D[vertex]);
            }
}

public static void main(String[ ] args)
{
        int num_ver = 0;
        int source;
```

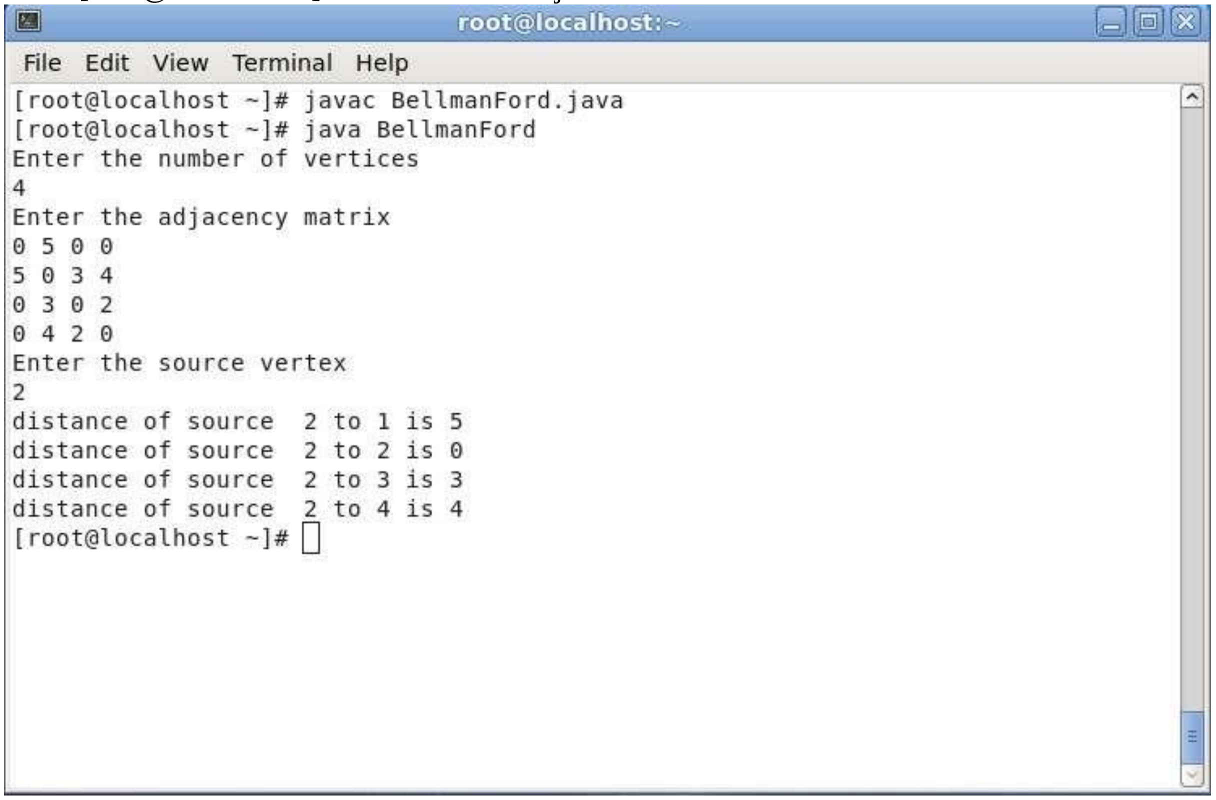
```
                  Scanner scanner = new Scanner(System.in);
                  System.out.println("Enter the number of vertices");
                  num_ver = scanner.nextInt();
                  int A[][] = new int[num_ver + 1][num_ver + 1];
                  System.out.println("Enter the adjacency matrix");
                  for (int sn = 1; sn <= num_ver; sn++)
                    {
                       for (int dn = 1; dn <= num_ver; dn++)
                          {
                                    A[sn][dn] = scanner.nextInt();
                                     if (sn == dn)
                                       {
                                          A[sn][dn] = 0;
                                          continue;
                                       }
                                      if (A[sn][dn] == 0)
                                       {
                                         A[sn][dn] = MAX_VALUE;
                                        }
                              }
                        }
                       System.out.println("Enter the source vertex");
                       source = scanner.nextInt();
                       BellmanFord b = new BellmanFord (num_ver);
                       b.BellmanFordEvaluation(source, A);
                      scanner.close();
              }
          }
```

Input graph:



Output:

[root@localhost ~]# vi BellmanFord.java



```
[root@localhost ~]# javac BellmanFord.java
[root@localhost ~]# java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix
0 5 0 0
5 0 3 4
0 3 0 2
0 4 2 0
Enter the source vertex
2
distance of source  2 to 1 is 5
distance of source  2 to 2 is 0
distance of source  2 to 3 is 3
distance of source  2 to 4 is 4
[root@localhost ~]# 
```

Explanation for the code:

1. import java.util.Scanner;: Imports the Scanner class from the `java.util` package, which is used for reading input from the user.

2. public class BellmanFord: Defines a class named `BellmanFord`.

3. private int D[];: Declares an integer array `D` to store the distances from the source to each vertex.

4. private int num_ver;: Declares an integer variable `num_ver` to store the number of vertices in the graph.

5. public static final int MAX_VALUE = 999;: Declares a constant integer variable `MAX_VALUE` and initializes it to 999. It represents the maximum value for distances, used as a placeholder for infinity.

6. public BellmanFord(int num_ver): Constructor for the `BellmanFord` class, which takes the number of vertices as an argument and initializes the `num_ver` and `D` array.

7. public void BellmanFordEvaluation(int source, int A[][]): Method to perform the Bellman-Ford algorithm. It takes the source vertex and the adjacency matrix `A` as parameters.

8. for (int node = 1; node <= num_ver; node++): Initializes the distance array `D` with maximum values for all vertices.

9. D[source] = 0;: Sets the distance from the source vertex to itself to 0.

10. for (int node = 1; node <= num_ver - 1; node++): Outer loop for the relaxation process, iterating for `num_ver - 1` times.

11. for (int sn = 1; sn <= num_ver; sn++): Inner loop for the source vertex.

12. for (int dn = 1; dn <= num_ver; dn++): Innermost loop for the destination vertex.

13. if (A[sn][dn] != MAX_VALUE): Checks if there is an edge between the source and destination.

14. if (D[dn] > D[sn] + A[sn][dn]): Checks if the distance to the destination can be reduced by going through the source.

15. D[dn] = D[sn] + A[sn][dn];: Updates the distance if a shorter path is found.

16. The next block of code (lines 19-27) checks for negative edge cycles in the graph.

17. for (int vertex = 1; vertex <= num_ver; vertex++): Prints the distances from the source to all vertices after the Bellman-Ford algorithm is executed.

18. public static void main(String[] args): The main method where the execution of the program starts.

19. int num_ver = 0;`: Initializes the variable `num_ver.

20. int source;: Declares the variable `source` to store the source vertex.

21-25. Takes input from the user for the number of vertices and the adjacency matrix.

26. source = scanner.nextInt();: Takes input for the source vertex.

27. Creates an instance of the `BellmanFord` class and calls the `BellmanFordEvaluation` method with the source and adjacency matrix as parameters.

28. scanner.close(); : Closes the Scanner to release resources.

Program 6: Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

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

set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1" set n1 [$ns node] set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2" set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2" set n4 [$ns node]
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"

$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
/* should come in single line */
$ns duplex-link $n4 $n5 1Mb 1ms DropTail

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5

$ns connect $tcp0 $sink5

set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
```

```
$ns attach-agent $n3 $sink3

$ns connect $tcp2 $sink3set file1 [open file1.tr w]

$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp2 attach $file2

$tcp0 trace cwnd_ /* must put underscore ( _ ) after cwnd and no space between them*/
$tcp2 trace cwnd_

proc finish { } { global ns nf tf
$ns flush-trace close $tf
close $nf
exec nam lab3.nam & exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
```

AWK file (Open a new editor using "vi command" and write awk file and save with ".awk" extension)
cwnd:- means congestion window

```
BEGIN {
}
{
if($6 =="cwnd_") /* don't leave space after writing cwnd_ */
printf("%f\t%f\t\n",$1,$7); /* you must put \n in printf */
} END {
}
```

<u>Steps for execution</u>

1) Open vi editor and type program. Program name should have the extension " **.tcl** "
   [root@localhost ~]# vi lab3.tcl
2) Save the program by pressing **"ESC key"** first, followed by **"Shift**

and :” keyssimultaneously 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 lab3.awk
```

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

```
[root@localhost~]# ns lab3.tcl
```

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

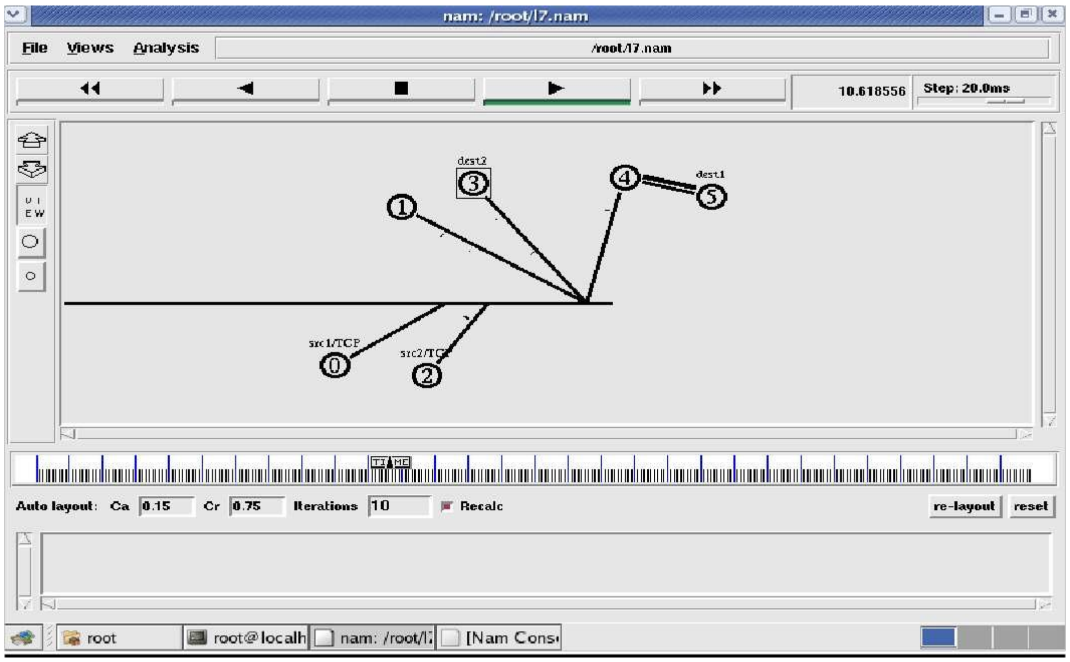
```
i.   [root@localhost~]# awk  –f  lab3.awk  file1.tr  >  a1
ii.  [root@localhost~]# awk  –f  lab3.awk  file2.tr  >  a2
iii. [root@localhost~]# xgraph a1 a2
```

7) Here we are using the congestion window trace files i.e**. file1.tr and file2.tr and weare redirecting the contents of those files to new files say a1** and **a2** using **output** redirection operator (>).

8) To see the trace file contents open the file as ,

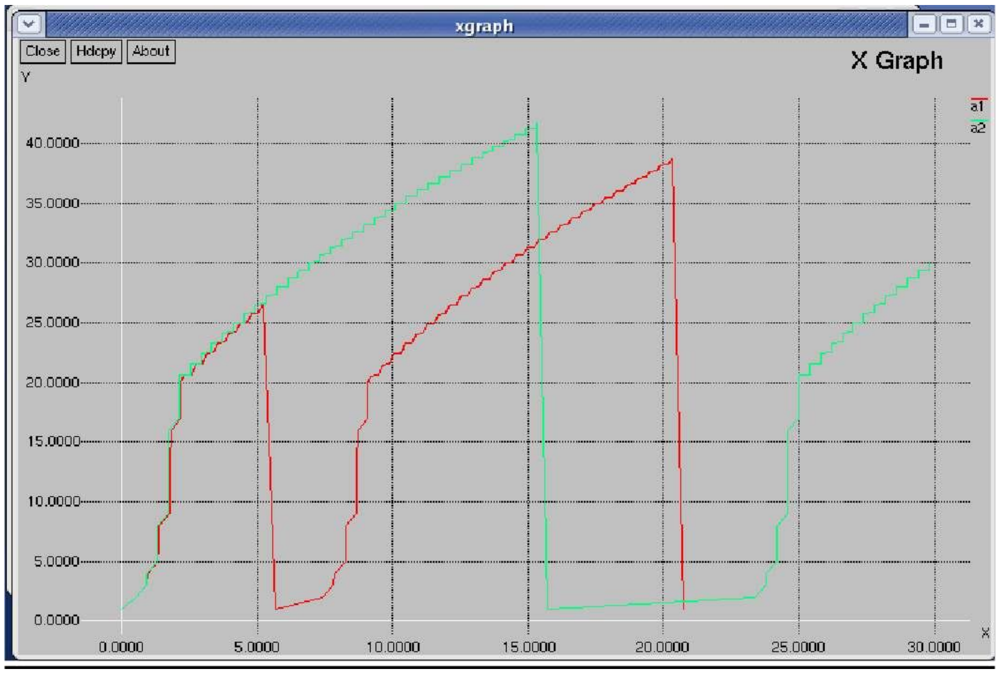
```
[root@localhost~]# vi lab3.tr
```

Topology



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc

Output:



Explanation of the code:

This code is a simulation script written in the ns-2 network simulator language. It simulates a simple network scenario with FTP (File Transfer Protocol) applications running between two source nodes (src1 and src2) and two destination nodes (dest1 and dest2). The nodes are connected through a LAN and a duplex link. The simulation records various parameters like congestion window size (cwnd) and creates trace files for analysis.

code line by line:

1. set ns [new Simulator]: Creates a new network simulation object.

2. set tf [open lab3.tr w]: Opens a trace file (lab3.tr) for writing. The trace file will store simulation events.

3. $ns trace-all $tf: Enables tracing of all events in the simulation and directs the output to the trace file.

4. set nf [open lab3.nam w]: Opens another trace file (lab3.nam) for the Nam network animator.

5. $ns namtrace-all $nf: Enables tracing of all events for Nam and directs the output to the Nam trace file.

6. Node creation and configuration:
1 Nodes n0, n1, n2, n3, n4, and n5 are created.
2Colors and labels are assigned to nodes.
3A LAN is created using make-lan with a specified bandwidth, delay, link layer, and queue type.
4 A duplex link is created between nodes n4 and n5.

7. TCP agents and applications setup:
1 TCP agents (Agent/TCP) and FTP applications (Application/FTP) are created for source nodes (n0 and n2).
1 TCP sink agents (Agent/TCPSink) are created for destination nodes (n5 and n3).
2 Connections are established between TCP agents and sink agents.

8. Trace setup:
1 Trace files (file1.tr and file2.tr) are opened for writing.
2 Trace events related to congestion window size (cwnd_) are traced for both TCP agents.

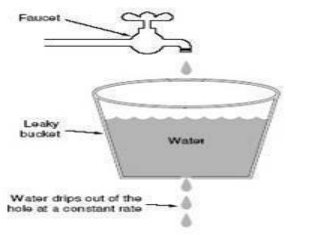
9. Event scheduling using $ns at:
1 Events are scheduled at specific simulation times to start and stop FTP applications.
2 The **finish** procedure is scheduled to run at simulation time 16.

10. proc finish { }: Defines a procedure named `finish` that is executed when called.
1 Flushes and closes the trace file for the simulator (lab3.tr).
2 Closes the Nam trace file (lab3.nam).
3 Executes the Nam animator to visualize the simulation.
4 Exits the simulation.

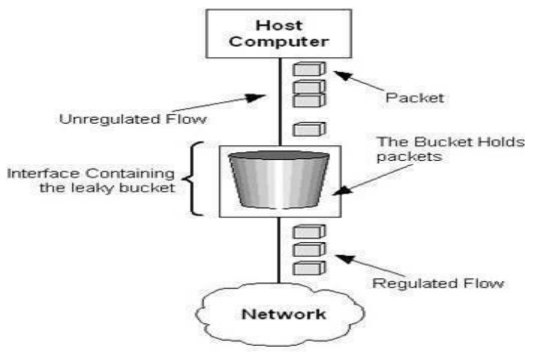
11. $ns run: Initiates the simulation.

Program 7: Write a program for congestion control using leaky bucket algorithm.

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).



(a) A leaky bucket with water



(b) A leaky bucket with packets

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

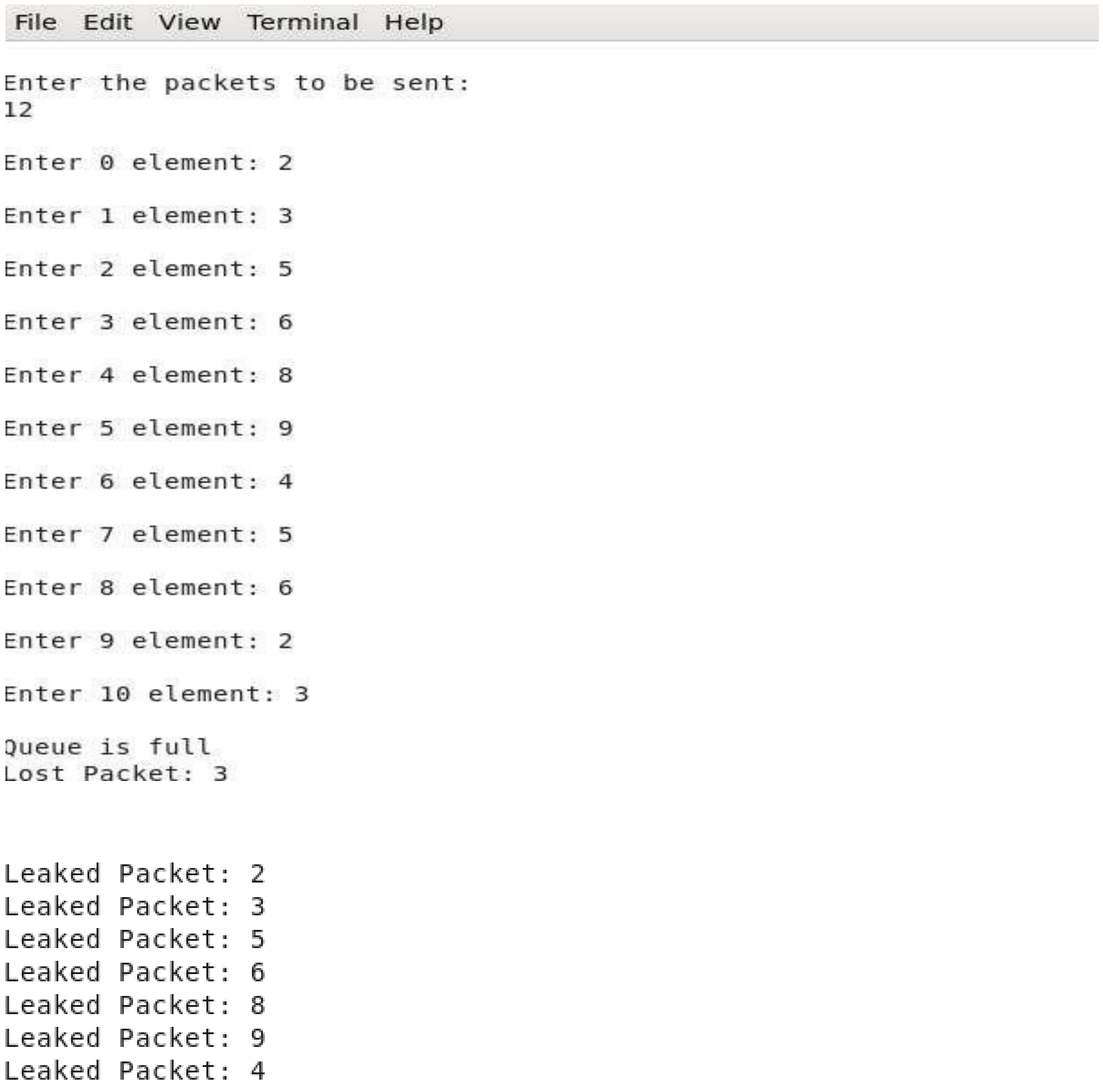
Source Code:

```
import java.io.*;
import java.util.*;
class Queue
{
        int q[],f=0,r=0,size;
        void insert(int n)
        {
                Scanner in = new Scanner(System.in);
                q=new int[10];
                for(int i=0;i<n;i++)
                {
                        System.out.print("\nEnter " + i + " element: ");
                        int ele=in.nextInt();
                        if(r+1>10)
                        {
```

```
                    System.out.println("\nQueue is full \nLost Packet: "+ele); break;

                    }
                    else
                    {
                            r++;
                           q[i]=ele;


                     }
               }
  }

void delete()
{
            Scanner in = new Scanner(System.in);
            Thread t=new Thread();
             if(r==0)
                    System.out.print("\nQueue empty ");
          else
             {
             for(int i=f;i<r;i++)
                    {
                         try
                        {
                           t.sleep(1000);
                           }
                          catch(Exception e){}
              System.out.print("\nLeaked Packet: "+q[i]);
             f++;
             }
      }
      System.out.println();
    }
}
class Leaky extends Thread
{
            public static void main(String ar[]) throws Exception
            {
                    Queue q=new Queue();
                     Scanner src=new Scanner(System.in);
                     System.out.println("\nEnter the packets to be sent:");
                     int size=src.nextInt();
                     q.insert(size);
```

```
                        q.delete();
                }
    }
```

Output:



```
Enter the packets to be sent:
12

Enter 0 element: 2

Enter 1 element: 3

Enter 2 element: 5

Enter 3 element: 6

Enter 4 element: 8

Enter 5 element: 9

Enter 6 element: 4

Enter 7 element: 5

Enter 8 element: 6

Enter 9 element: 2

Enter 10 element: 3

Queue is full
Lost Packet: 3


Leaked Packet: 2
Leaked Packet: 3
Leaked Packet: 5
Leaked Packet: 6
Leaked Packet: 8
Leaked Packet: 9
Leaked Packet: 4
```

Explanation:

import java.io.*; Imports necessary classes for input/output operations.

import java.util.*; Imports the Scanner class from the java.util package for user input.

class Queue { Defines a class named Queue to represent a simple queue.

int q[], f = 0, r = 0, size; Declares instance variables for the queue array (q), front (f), rear (r), and the size of the queue.

void insert(int n) { Defines a method insert to insert elements into the queue. It takes the number of elements to be inserted (n) as a parameter.

Scanner in = new Scanner(System.in); Creates a Scanner object for user input.

q = new int[10]; Initializes the queue array with a size of 10.

for (int i = 0; i < n; i++) { Loop to insert n elements into the queue.

System.out.print("\nEnter " + i + " element: "); Prompts the user to enter the i-th element.

int ele = in.nextInt(); Reads the user input as the element to be inserted.

if (r + 1 > 10) { Checks if the queue is full.

System.out.println("\nQueue is full \nLost Packet: " + ele); break; Prints a message indicating that the queue is full and a packet is lost. Exits the loop.

else { If the queue is not full:

r++; Increments the rear pointer.

q[i] = ele; Inserts the element into the queue.

void delete() { Defines a method delete to delete (leak) packets from the queue.

Thread t = new Thread(); Creates a Thread object for simulating a delay.

if (r == 0) System.out.print("\nQueue empty "); Checks if the queue is empty.

else { If the queue is not empty:

for (int i = f; i < r; i++) { Loop to iterate through elements in the queue.

try { t.sleep(1000); } catch (Exception e) {} Introduces a delay of 1000 milliseconds (1 second) between printing each leaked packet.

System.out.print("\nLeaked Packet: " + q[i]); Prints the leaked packet.

f++;Increments the front pointer.

System.out.println(); Prints a newline after leaking all packets.

class Leaky extends Thread {: Defines a class named Leaky that extends Thread.

public static void main(String ar[]) throws Exception {: The main method.

Queue q = new Queue();: Creates an instance of the Queue class.

Scanner src = new Scanner(System.in);: Creates a Scanner object for user input.

System.out.println("\nEnter the packets to be sent:");: Prompts the user to enter the number of packets to be sent.

int size = src.nextInt();: Reads the user input as the number of packets to be sent.

q.insert(size);: This method is responsible for inserting elements into the queue based on the specified size.

q.delete();
This method simulates the process of leaking packets from the queue and prints the leaked packets.