CMR UNIVERSITY
Private University Established in Karnataka State by Act No. 45 of 2013

SCHOOL OF ENGINEERING AND TECHNOLOGY

DCCN LAB MANUAL

Submitted in the partial fulfillment of the requirements for the course Data Communications and Computer Networks (4CSGC2061) in

Bachelor of Technology

In

Computer Science and Engineering

SoET, CMR, University, Bangalore

Prepared by,
Dr.Parameswaran T
Prof.Bhagya.K
Prof.Priyanka

Department of Computer Science and Engineering
Off Hennur-Bagalur Main Road,
Near Kempegowda International Airport, Chagalahatti,
Bangalore, Karnataka-562149

INDEX

Sl. No.	Date	Title	Signature
1		Three Nodes Point to Point Network	
2		Four Nodes Point to Point Network	
3		Congestion Monitoring	
4		Error Rate in Ethernet LAN	
5		Congestion Monitoring in LAN	
6		Routing Algorithms	
7		Error Detection Techniques	
8		Socket Programming	
9		Congestion Control	

Simulation using NS2

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

Program:

TCL file:

```
set ns [ new Simulator ]

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

set nf [ open lab1.nam w ]
$ns namtrace-all $nf



proc finish { } {
        global ns nf tf
        $ns flush-trace
        exec nam lab1.nam & close
        $tf w
        close $nf w exit 0
}

set n0 [$ns node] set n1
[$ns node] set n2 [$ns
node] set n3 [$ns node]

$ns color 1 "red"
```

```
$ns color 2 "blue"
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"


#Create link between nodes
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail


#Set queue size of links
$ns set queue-limit $n0 $n2 50
$ns set queue-limit $n1 $n2 50
$ns set queue-limit $n2 $n3 5


#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0


# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

```
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1


# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1


#Create a Null agent (a traffic sink) and attach it to node n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0


#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0
$ns connect $udp1 $null0


#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

```
#Call the finish procedure after 5 seconds of simulation time

$ns at 5.0 "finish"


#Run the simulation

$ns run
```

Awk file:

Note: *Create the file using gedit command and save it with the extension of. awk. We can name it as lab1.awk.*

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

To run the program:

1. Run the program by ns command.
2. Visualize the output through NAM.
3. Run the command: awk -f lab1.awk lab1.tr
4. It will show the number of packets dropped.

Step by step explanations of the program:

1. **Simulator Setup:**

    set ns [new Simulator]

    set tf [open lab1.tr w]

    $ns trace-all

    $tf set nf [open lab1.nam w]

    $ns namtrace-all $nf

- Creates a new network simulator object (**ns**).
- Opens a file (**lab1.tr**) for trace output and associates it with the simulator.
- Opens a file (**lab1.nam**) for NAM (Network Animator) visualization output and associates it with the simulator.

2. **Node and Link Setup:**

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

- Creates four nodes (**n0**, **n1**, **n2**, **n3**) in the simulation.

```
$ns color 1 "red"
$ns color 2 "blue"
```

- Sets colors for different classes of nodes.

```
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"
```

- Assigns labels to the nodes for better visualization in NAM.

```
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail
```

- Creates duplex links between the nodes with specified bandwidth, delay, and queuing discipline (**DropTail**).

```
$ns set queue-limit $n0 $n2 50
$ns set queue-limit $n1 $n2 50
$ns set queue-limit $n2 $n3 5
```

- Sets the queue size limits for the links.

3. **Traffic Source and Sink Setup:**

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

- Creates a UDP agent (**udp0**) and attaches it to node **n0**.
- Creates a Constant Bit Rate (CBR) traffic source (**cbr0**) and attaches it to **udp0**.

Similar setup is done for node **n1**:

```
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1

set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

- Creates a Null agent (**null0**) and attaches it to node **n3** as a traffic sink.

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

- Connects the UDP agents to the Null agent to establish the communication flow.
- A Null agent is often created and used as a "sink" or destination for traffic. The primary purpose of a Null agent is to absorb or discard incoming packets without processing them further.
- The Null agent serves as a destination or endpoint for the simulated traffic. Whenthe traffic reaches the Null agent, it effectively terminates the simulated communication.

4. **Event Scheduling:**

```
$ns at 0.5 "$cbr0 start"
```

```
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

- Schedules events to start and stop the CBR traffic sources.

5. **Finish Procedure and Simulation Execution:**

```
$ns at 5.0 "finish"
$ns run
```

- Schedules the **finish** procedure to be called after 5 seconds of simulation time.
- Executes the simulation using **$ns run**.

6. **Finish Procedure:**

```
proc finish { } {
  global ns nf tf
  $ns flush-trace
  exec nam lab1.nam &
  close $tf w
  close $nf w
  exit 0
}
```

- Clears the trace output buffer.
- Launches NAM for visualization.
- Closes the trace and NAM files.
- Exits the simulation.

This script sets up a simple network scenario with two traffic sources, a router, and a traffic sink. It uses UDP agents and CBR traffic sources to simulate data transmission through the network, and the results are visualized using NAM. The finish procedure is responsible for finalizing and terminating the simulation.

Explanation for the lab1.awk code:

1. **BEGIN Block:**

- The **BEGIN** block is executed before processing any lines from the input. In this case, it initializes a variable **count** to zero. This variable will be used to keep track of the number of dropped packets.

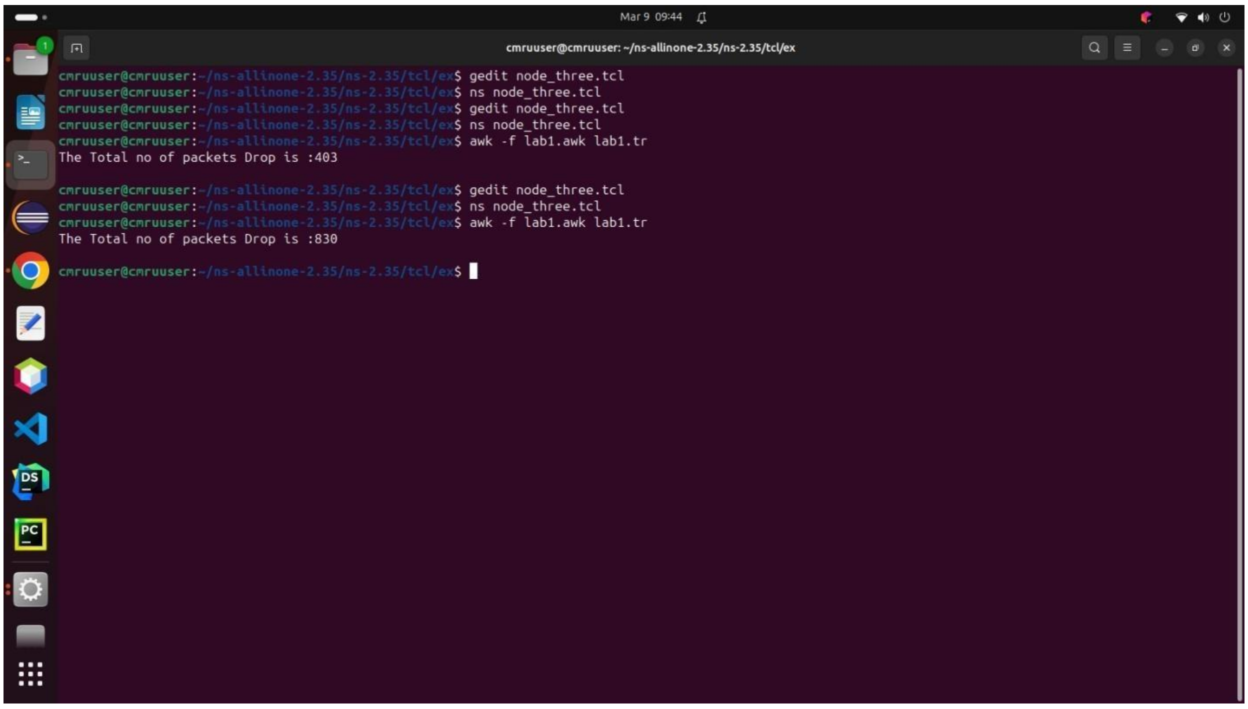
2. **Main Block:**

   - The main block processes each line of the input file. The condition **if ($1 == "d")** checks if the first field (column) of the line is equal to "d". In NS-2 trace files, the first character "d" often indicates a dropped packet.
   - If the condition is true, it increments the **count** variable, indicating the detection of a dropped packet.

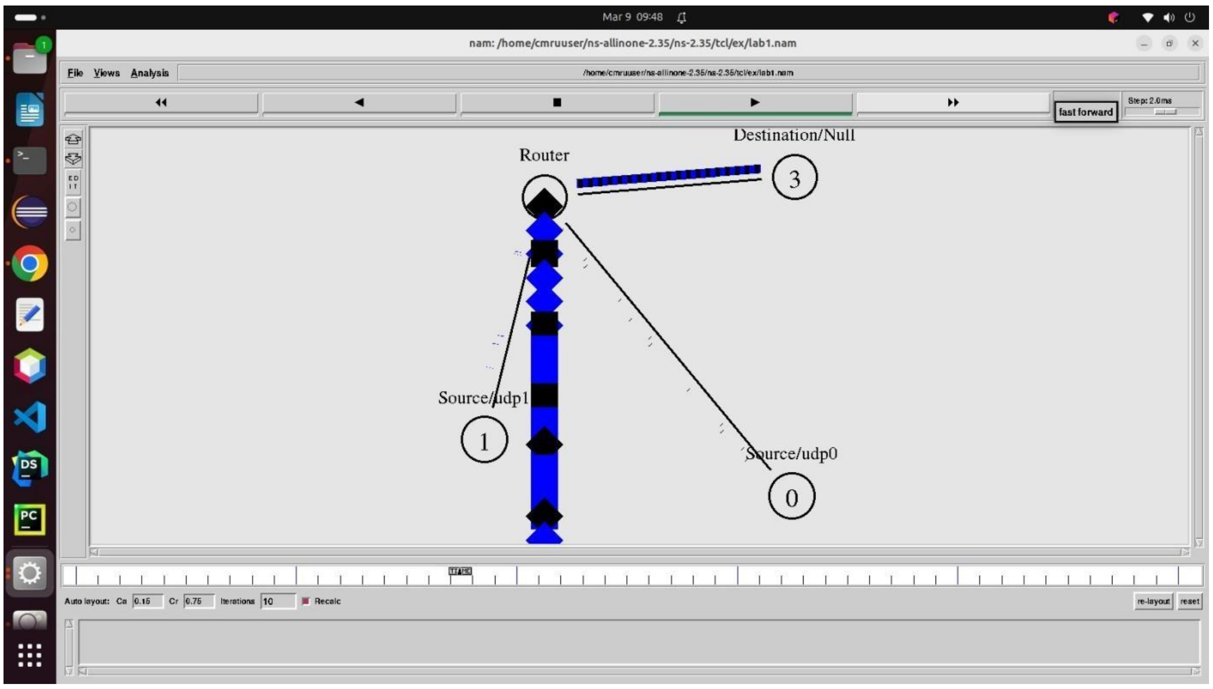
3. **END Block:**

   - The **END** block is executed after processing all lines from the input. It prints the total count of dropped packets using the **printf** statement.
   - The output message displays the total number of dropped packets detected during the simulation.

Output



```
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit node_three.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns node_three.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit node_three.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns node_three.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f lab1.awk lab1.tr
The Total no of packets Drop is :403

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit node_three.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns node_three.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f lab1.awk lab1.tr
The Total no of packets Drop is :830

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$
```



Result: The program was executed successfully and the output was obtained.

Program 2

Exercise 2: Simulate a four-node point-to-point network, and connect the links as follows:

no-n2, n1-n2 and n2-n3. Apply TCP agent between no-n3 and UDP n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets by TCP/UDP.

Program:

TCL file:

```
#Create a simulator object

set ns [new Simulator]



#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Red



set tf [open out1.tr w]

$ns trace-all Stf



#Open the NAM trace file

set nf [open out1.nam w]

Sns namtrace-all $nf



#Define a 'finish' procedure

proc finish {} {

        global ns tf nf

        $ns flush-trace

        close $tf
        #Close the NAM trace file

        close Snf

        #Execute NAM on the trace file

        exec nam out.nam &

        exit 0

}


#Create four nodes

set no [Sns node]

set n1 [Sns node]

set n2 [$ns node]

set n3 [$ns node]


#Create links between the nodes

$ns duplex-link Sn0 $n2 2Mb 10ms DropTail

$ns duplex-link Sn1 Sn2 2Mb 10ms DropTail

Sns duplex-link Sn2 Sn3 1.7Mb 20ms DropTail


#Set Queue Size of link (n2-n3) to 10

$ns queue-limit Sn2 Sn3 10


#Give node position (for NAM)

$ns duplex-link-op Sn0 Sn2 orient right-down

$ns duplex-link-op Sn1 Sn2 orient right-up

Sns duplex-link-op Sn2 Sn3 orient right
```

```
#Monitor the queue for link (n2-n3). (for NAM)
Sns duplex-link-op Sn2 Sn3 queuePos 0.5


# Setup a TCP connection
set tcp [new Agent/TCP]
Stcp set class_2
Sns attach-agent Sno Stcp
set sink [new Agent/TCPSink]
Sns attach-agent Sn3 Ssink
Sns connect Stcp $sink
Stcp set fid_1


# Setup a FTP over TCP connection
set ftp [new Application/FTP]
Sftp attach-agent Stcp
Sftp set type FTP


#Setup a UDP connection
set udp [new Agent/UDP]
Sns attach-agent Sn1 Sudp
set null [new Agent/Null]
$ns attach-agent Sn3 Snull
Sns connect Sudp $null
Sudp set fid_2
```

```
#Setup a CBR over UDP connection

set cbr [new Application/Traffic/CBR]

Scbr attach-agent Sudp

Scbr set type_ CBR

Scbr set packet_size_1000

Scbr set rate 1mb

Scbr set random_ false


#Schedule events for the CBR and FTP agents

Sns at 0.1 "Scbr start"

$ns at 1.0 "$ftp start”

Sns at 4.0 "Sftp stop"

Sns at 4.5 "Scbr stop"


#Detach tcp and sink agents (not really necessary)

Sns at 4.5 "Sns detach-agent Sno Stcp; $ns detach-agent Sn3 Ssink"


#Call the finish procedure after 5 seconds of simulation time

Sns at 5.0 "finish"


#Print CBR packet size and interval

puts "CBR packet size = [Scbr set packet_size_]"

puts "CBR interval = [Scbr set interval_]"
```

```
#Run the simulation

$ns run
```

Awk file:

Note: *Create the file using gedit command and save it with the extension of. Awk in order to find the number of packets drop in the trace file. We can name it as out1.awk.*

```
BEGIN{

count=0;

}{

if($1=="d")

count++

}

END{

printf("The Total no of Packets Drop is : %d\n\n", count)

}
```

To run the program:

1. Run the program by ns command.
2. Visualize the output through NAM.
3. Run the command: awk -f out1.awk out1.tr
4. It will show the number of packets dropped.

Step by step explanations of the program:

Step 1: Create a Simulator Object

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

This line creates a new simulator object named **ns** using the **new Simulator** command.

Step 2: Define Colors and Open Trace Files

```
# Define different colors for data flows (for NAM) $ns color 1 Blue
$ns color 2 Red
# Open the trace file
set tf [open out1.tr w]
$ns trace-all $tf
# Open the NAM trace file
set nf [open out1.nam w]
$ns namtrace-all $nf
```

Here, colors for data flows in the NAM visualizer are defined, and two trace files (out1.tr and out1.nam) are opened to store simulation data.

Step 3: Define the 'finish' Procedure

```
# Define a 'finish' procedure
proc finish {}
{ global ns tf nf
$ns flush-trace
close $tf
# Close the NAM trace file
close $nf
# Execute NAM on the trace file
exec nam out1.nam & exit 0 }
```

A procedure named finish is defined. This procedure is responsible for flushing traces, closing trace files, executing NAM on the trace file, and then exiting the simulation.

Step 4: Create Four Nodes

```
# Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

Four nodes (n0, n1, n2, and n3) are created using the $ns node command.

Step 5: Create Links Between Nodes

```
# Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
```

Links with specified characteristics (bandwidth, delay, and queuing discipline) are created between the nodes.

Step 6: Set Queue Size and Node Positions

```
# Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
# Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
# Monitor the queue for link (n2-n3) (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

These lines set the queue size for a specific link, give node positions for NAM visualization, and monitor the queue for a specific link in the visualization.

Step 7: Setup TCP Connection and FTP Application

```
# Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

# Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
```

A TCP connection is set up between nodes n0 and n3. An FTP application is attached to this TCP connection.

Step 8: Setup UDP Connection and CBR Application

```
# Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $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 type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
```

A UDP connection is set up between nodes n1 and n3.
A Constant Bit Rate (CBR) application is attached to this UDP connection.

Step 9: Schedule Events for CBR and FTP Agents

```
# Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
```

Events are scheduled to start and stop the CBR and FTP applications at specific simulation times.

Step 10: Detach TCP and Sink Agents

```
# Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
```

This line detaches the TCP and sink agents at a specific simulation time.

Step 11: Call the Finish Procedure

```
# Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
```

The finish procedure is scheduled to be called after 5 seconds of simulation time.

Step 12: Print CBR Packet Size and Interval

```
# Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"
```

The script prints the packet size and interval settings for the CBR application.

Step 13: Run the Simulation

```
# Run the simulation
$ns run
```

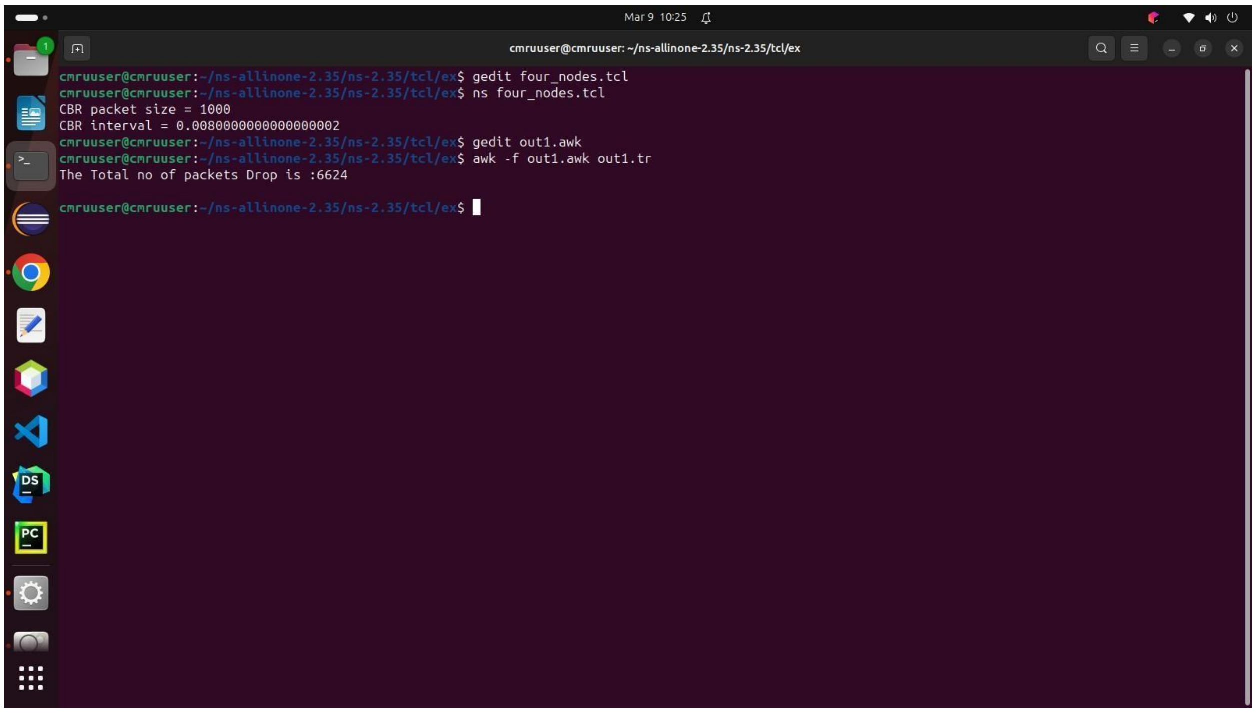
This line executes the simulation.

Step 14: Execute NAM on the Trace File

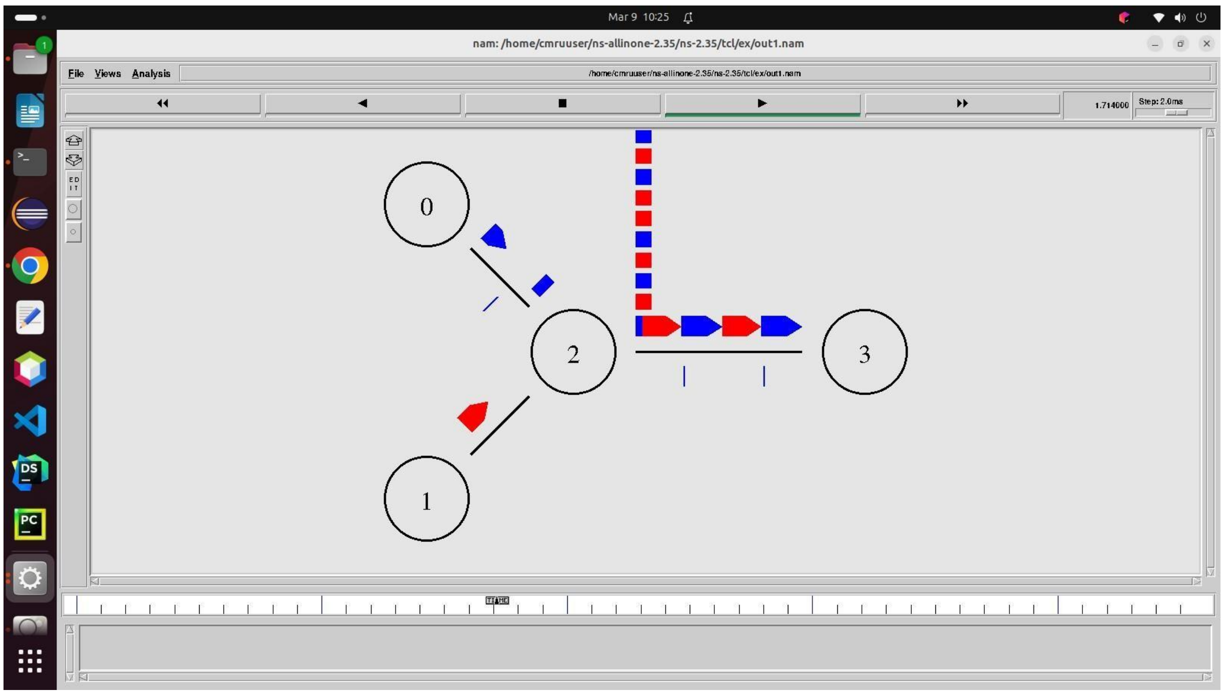
```
# Execute NAM on the trace file
exec nam out1.nam &
```

NAM is executed to visualize the simulation results.

<u>Output</u>



```
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit four_nodes.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns four_nodes.tcl
CBR packet size = 1000
CBR interval = 0.0080000000000000002
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit out1.awk
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f out1.awk out1.tr
The Total no of packets Drop is :6624

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$
```


Mar 9 10:25
nam: /home/cmruuser/ns-allinone-2.35/ns-2.35/tcl/ex/out1.nam
File
Views
Analysis
/home/cmruuser/ns-allinone-2.35/ns-2.35/tcl/ex/out1.nam
1.714000
Step: 2.0ms
0
1
2
3

Result: The program was executed successfully and the output was obtained.

Program 3

Ex.3: Simulate the transmission of ping messaged over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

Program:

TCL file:

```
#Create a simulator object
set ns [new Simulator]
set tf [open lab2.tr w]
$ns trace-all $tf

set nf [open lab2.nam w]
$ns namtrace-all $nf

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$n0 label "Ping0"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$n2 label "Router"

$ns color 1 "red"
$ns color 2 "green"
```

```
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail

$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail

$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n2 $n6 2
$ns queue-limit $n5 $n2 5

#The below code is used to connect between the ping agents to the node n0,
#n4 , n5 and n6.
set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0

set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4

set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5

set ping6 [new Agent/Ping]
$ns attach-agent $n6 $ping6

$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001
```

```
$ping5 set packetSize_ 60000
$ping5 set interval_ 0.00001

$ping0 set class_ 1
$ping5 set class_ 2

$ns connect $ping0 $ping4
$ns connect $ping5 $ping6

#Define a 'recv' function for the class 'Agent/Ping'
#The below function is executed when the ping agent receives a reply from
the destination
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts " The node [$node_ id] received an reply from $from with round trip
time of $rtt"
}

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

#Schedule events
$ns at 0.1 "$ping0 send"
$ns at 0.2 "$ping0 send"
$ns at 0.3 "$ping0 send"
$ns at 0.4 "$ping0 send"
```

```
$ns at 0.5 "$ping0 send"
$ns at 0.6 "$ping0 send"
$ns at 0.7 "$ping0 send"
$ns at 0.8 "$ping0 send"
$ns at 0.9 "$ping0 send"

$ns at 1.0 "$ping0 send"
$ns at 1.1 "$ping0 send"
$ns at 1.2 "$ping0 send"
$ns at 1.3 "$ping0 send"
$ns at 1.4 "$ping0 send"
$ns at 1.5 "$ping0 send"
$ns at 1.6 "$ping0 send"
$ns at 1.7 "$ping0 send"
$ns at 1.8 "$ping0 send"

$ns at 0.1 "$ping5 send"
$ns at 0.2 "$ping5 send"
$ns at 0.3 "$ping5 send"
$ns at 0.4 "$ping5 send"
$ns at 0.5 "$ping5 send"
$ns at 0.6 "$ping5 send"
$ns at 0.7 "$ping5 send"
$ns at 0.8 "$ping5 send"
$ns at 0.9 "$ping5 send"

$ns at 1.0 "$ping5 send"
$ns at 1.1 "$ping5 send"
$ns at 1.2 "$ping5 send"
$ns at 1.3 "$ping5 send"
```

```
$ns at 1.4 "$ping5 send"
$ns at 1.5 "$ping5 send"
$ns at 1.6 "$ping5 send"
$ns at 1.7 "$ping5 send"
$ns at 1.8 "$ping5 send"


$ns at 5.0 "finish"
$ns run
```

Awk file:

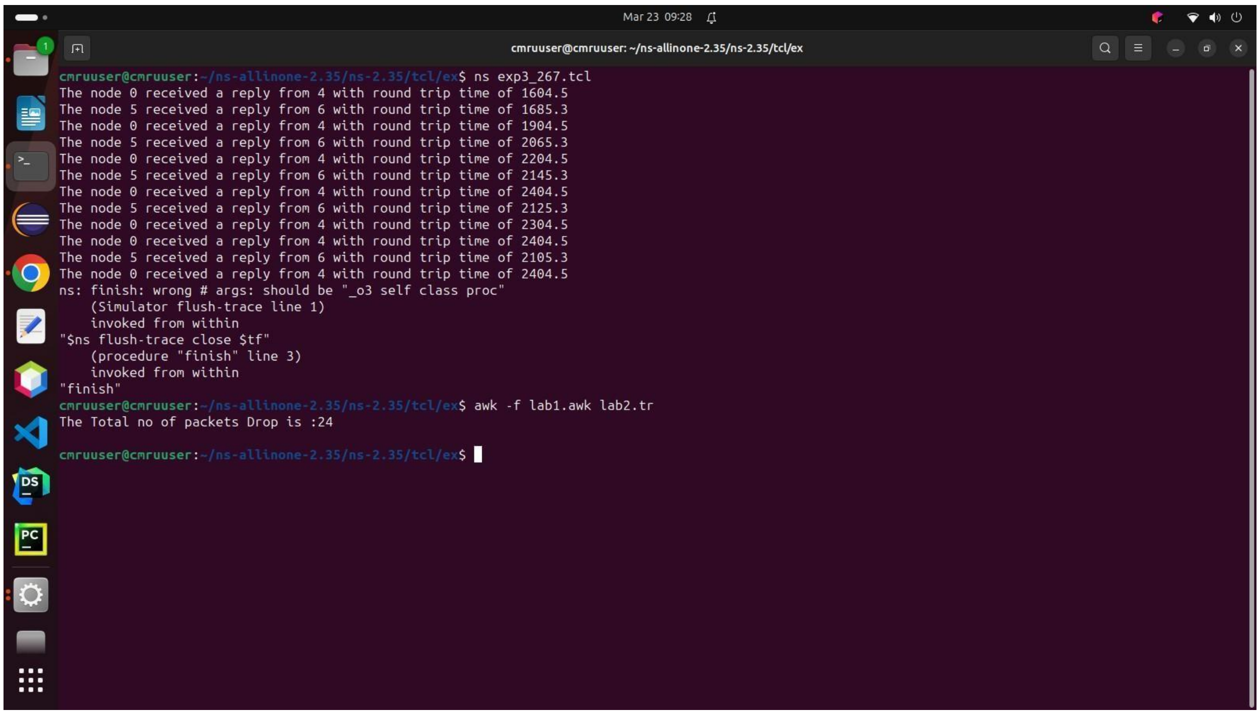
Note: *Create the file using gedit command and save it with the extension of. Awk in order to find the number of packets drop in the trace file. We can name it as out1.awk.*

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

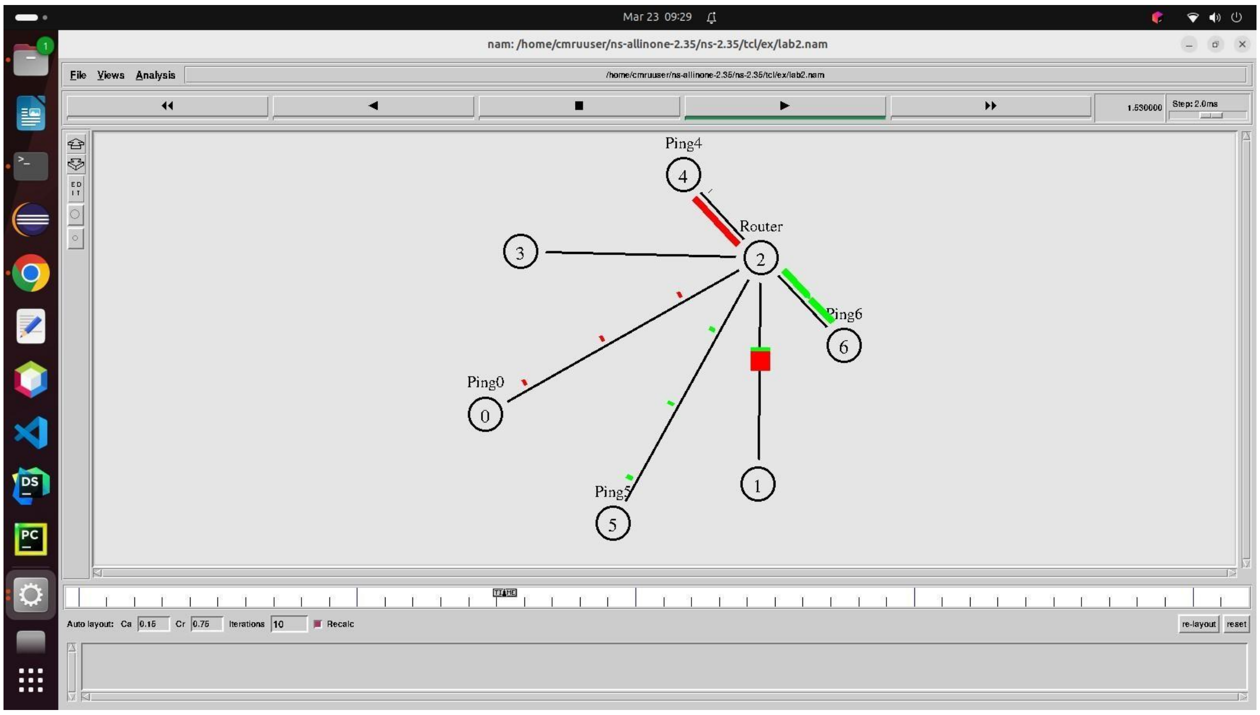
To run the program:

1. Run the program by ns command.
2. Visualize the output through NAM.
3. Run the command: awk -f lab1.awk lab2.tr
4. It will show the number of packets dropped.

Output



```
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns exp3_267.tcl
The node 0 received a reply from 4 with round trip time of 1604.5
The node 5 received a reply from 6 with round trip time of 1685.3
The node 0 received a reply from 4 with round trip time of 1904.5
The node 5 received a reply from 6 with round trip time of 2065.3
The node 0 received a reply from 4 with round trip time of 2204.5
The node 5 received a reply from 6 with round trip time of 2145.3
The node 0 received a reply from 4 with round trip time of 2404.5
The node 5 received a reply from 6 with round trip time of 2125.3
The node 0 received a reply from 4 with round trip time of 2304.5
The node 0 received a reply from 4 with round trip time of 2404.5
The node 5 received a reply from 6 with round trip time of 2105.3
The node 0 received a reply from 4 with round trip time of 2404.5
ns: finish: wrong # args: should be "_o3 self class proc"
    (Simulator flush-trace line 1)
    invoked from within
"$ns flush-trace close $tf"
    (procedure "finish" line 3)
    invoked from within
"finish"
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f lab1.awk lab2.tr
The Total no of packets Drop is :24

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ 
```


Mar 23 09:29
nam: /home/cmruuser/ns-allinone-2.35/ns-2.35/tcl/ex/lab2.nam
File
Views
Analysis
/home/cmruuser/ns-allinone-2.35/ns-2.35/tcl/ex/lab2.nam
1.530000
Step: 2.0ms
Ping4
4
Router
2
3
Ping6
6
Ping0
0
1
Ping5
5
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset

Result: The program was executed successfully and the output was obtained

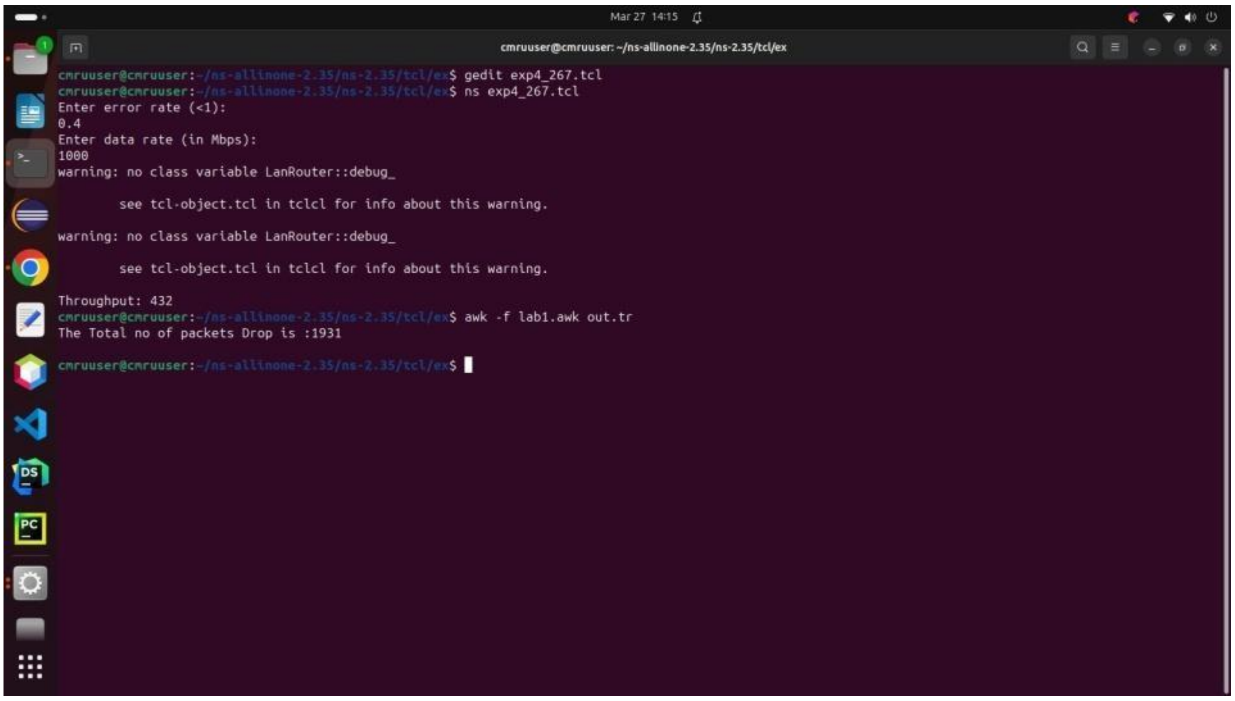
Program 4

Ex.4: Simulate an Ethernet LAN using n nodes (6-10), change errorrate and data rate and compare the throughput.

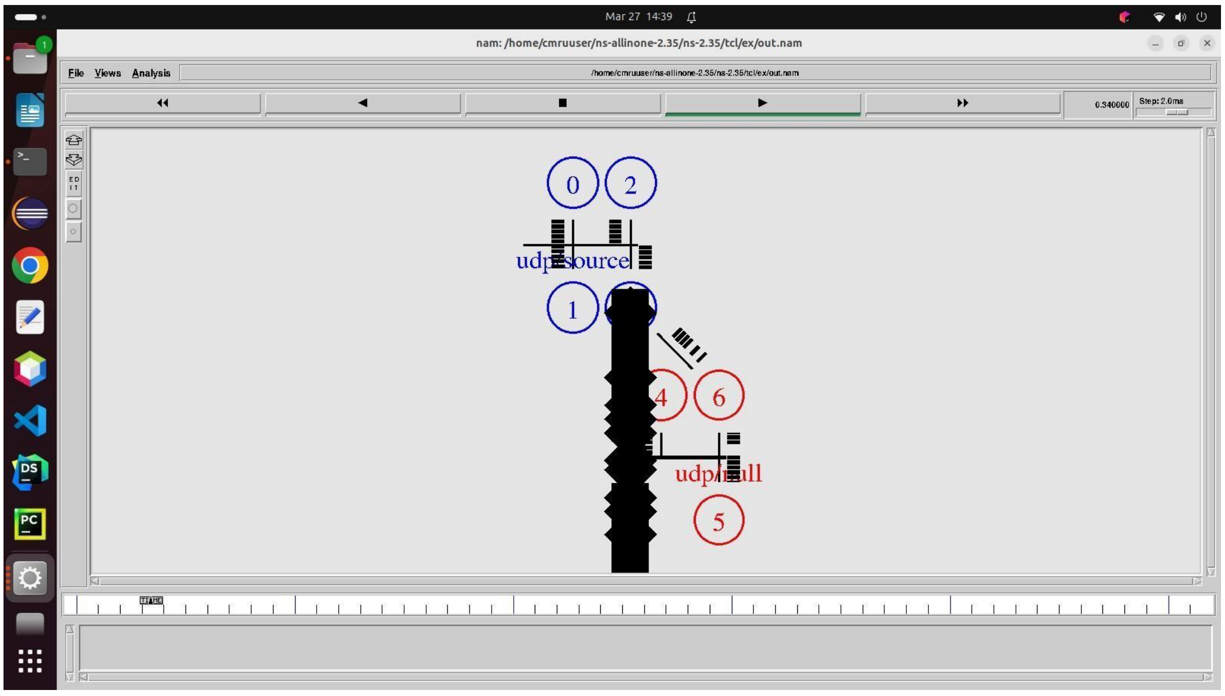
```
# Declare a new Simulatorset ns [new Simulator]

# Open nam and trace file in write modeset tf [open out.tr w]
set nf [open out.nam w]
$ns trace-all $tf
$ns namtrace-all $nf

# Take value of error rate and data rate from std inputputs "Enter error rate (<1) : "
gets stdin erate

puts "Enter data rate (in Mbps) : "gets stdin drate

# Create nodes set n0 [$ns node]set n1 [$ns node]set n2 [$ns node]set n3 [$ns node]set n4 [$ns
node]set n5 [$ns node]set n6 [$ns node]

# set label and color (OPTIONAL)
$n1 label "udp/source"
$n5 label "udp/null"
$n0 color "blue"
$n1 color "blue"
$n2 color "blue"
$n3 color "blue"
$n4 color "red"
$n5 color "red"
$n6 color "red"

# Create two lans
$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail   Mac/802_3
$ns make-lan "$n4 $n5 $n6" 10Mb 10ms LL Queue/DropTail Mac/802_3

# Setup Links
$ns duplex-link $n3 $n6 10Mb 10ms DropTail
$ns duplex-link-op $n3 $n6 orient right-down

# Declare the transport layer protocolsset udp1 [new Agent/UDP]
set null5 [new Agent/Null]
$ns attach-agent $n1 $udp1
$ns attach-agent $n5 $null5
```

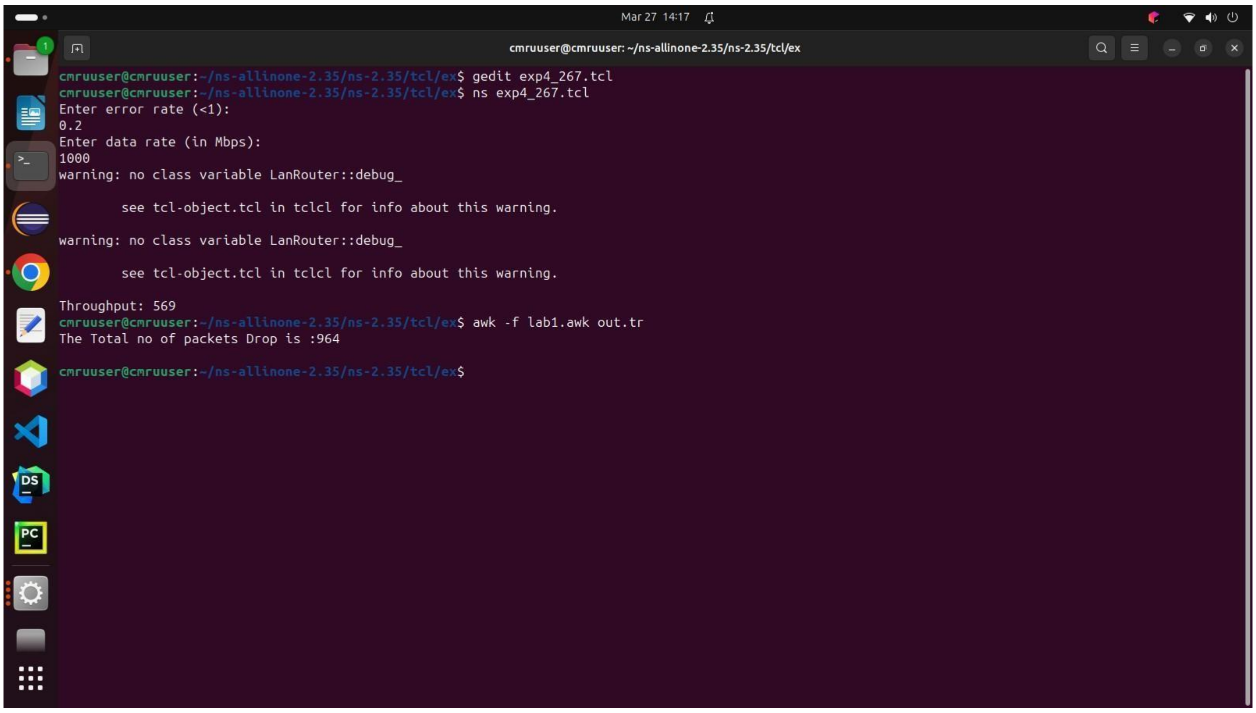
```
# Declare the application layer protocolset cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

# Connect the source and destination
$ns connect $udp1 $null5

# Create error model
set err [new ErrorModel]
$ns lossmodel $err $n3 $n6
$err set rate_ $erate

# Define the data rate
$cbr1 set packetSize_ $drate.Mb
$cbr1 set interval_ 0.001# Define procedure
procfinish { } {
global ns nf tf
$ns flush-trace exec nam out.nam&close$nf
close $tf

set count 0
set tr [open out.tr r]
while {[gets $tr line] != -1} {
# 8 denotes LAN at destination side and 5 denotes destinationnode
if {[string match "* 8 5 *" $line]} {set count [expr $count+1]
}

}

set thr [expr $count/7]puts "Throughput : $thr"exit 0
}

$ns at 0.1 "$cbr1 start"
$ns at 5.1 "finish"
$ns run
```

Output

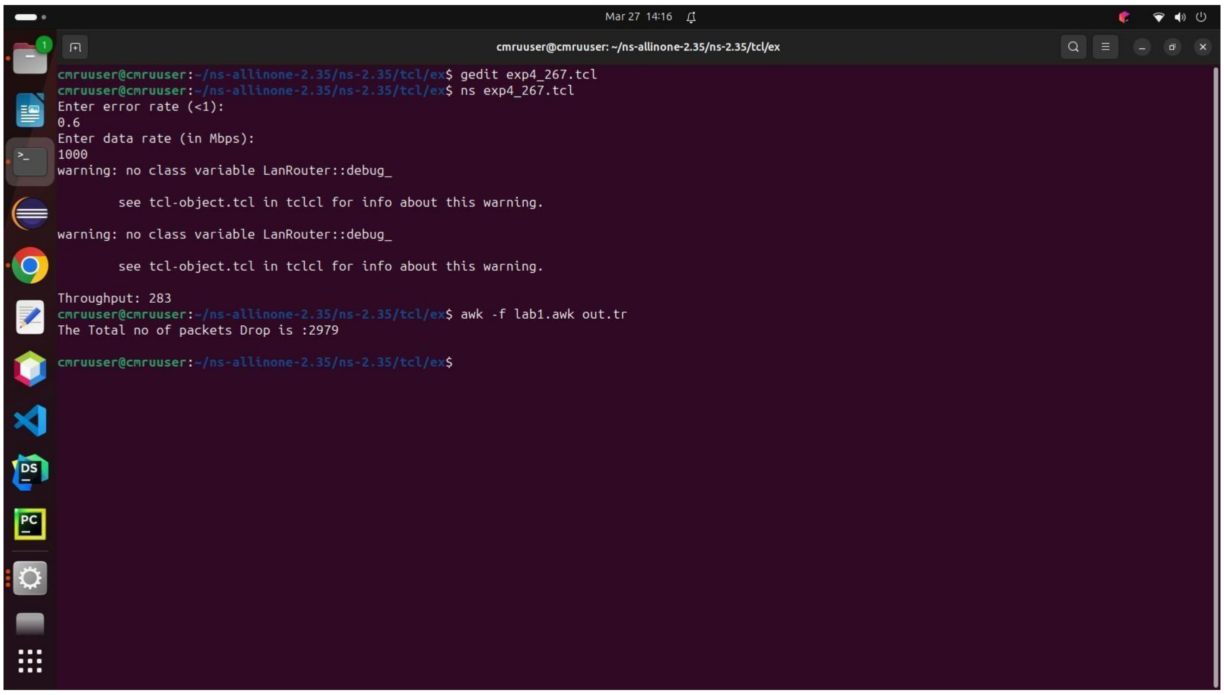


```
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit exp4_267.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns exp4_267.tcl
Enter error rate (<1):
0.4
Enter data rate (in Mbps):
1000
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

Throughput: 432
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f lab1.awk out.tr
The Total no of packets Drop is :1931

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$
```


nam: /home/cmruuser/ns-allinone-2.35/ns-2.35/tcl/ex/out.nam
File Views Analysis
/home/cmruuser/ns-allinone-2.35/ns-2.35/tcl/ex/out.nam
0.340000
Step: 2.0ms
0
2
udp/source
1
4
6
udp/null
5



```
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit exp4_267.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns exp4_267.tcl
Enter error rate (<1):
0.2
Enter data rate (in Mbps):
1000
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

Throughput: 569
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f lab1.awk out.tr
The Total no of packets Drop is :964

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$
```



```
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit exp4_267.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns exp4_267.tcl
Enter error rate (<1):
0.6
Enter data rate (in Mbps):
1000
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

Throughput: 283
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f lab1.awk out.tr
The Total no of packets Drop is :2979

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$
```

Result: The program was executed successfully and the output was obtained.

Program 5

Ex.5: Congestion Monitoring in LAN: Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and plot congestion window for different source/destination.

```
#Make a NS simulator
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf

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

# Create the nodes,color and label
set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1"
set n1 [$ns node]
$n1 color "red"
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"
set n4 [$ns node]
$n4 shape square
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"

#Creates a lan from a set of nodes given by <nodelist>. Bandwidth, delay
#characteristics along with the link-layer, Interface queue, Mac layer and #channel type for the
lan also needs to be defined.
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 50Mb 100ms LL Queue/DropTail Mac/802_3

# Create the link
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
# Create the node position
$ns duplex-link-op $n4 $n5 orient right

# Add a TCP sending module to node n0
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
# Setup a FTP traffic generator on "tcp0"
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
```

```
# Add a TCP receiving module to node n5
set sink0 [new Agent/TCPSink]
$ns attach-agent $n5 $sink0
# Direct traffic from "tcp0" to "sink1"
$ns connect $tcp0 $sink0

# Add a TCP sending module to node n2
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1

# Setup a FTP traffic generator on "tcp1"
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set packetSize_ 600
$ftp1 set interval_ 0.001

# Add a TCP receiving module to node n3
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1

# Direct traffic from "tcp1" to "sink1"
$ns connect $tcp1 $sink1
set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp1 attach $file2
$tcp0 trace cwnd_
$tcp1 trace cwnd_

# Define a 'finish' procedure
proc finish { } {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam lab3.nam &
exit 0
}

# Schedule start/stop times
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp1 start"
$ns at 8 "$ftp1 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp1 start"
$ns at 15 "$ftp1 stop"

# Set simulation end time
$ns at 16 "finish"
$ns run
```

AWK code:

```
BEGIN {
}
{
if($6=="cwnd_")
printf("%f\t%f\t\n",$1,$7);
}
END {
}
```

How to run:

1. Create separate file .awk as extension
2. Command to run awk script

```
awk -f filename.awk file1.tr>a1
awk -f filename.awk file2.tr>a2
xgraph a1 a2
```

Note: TCL and awk filename should be same

Explanation of the program:

This TCL script simulates an Ethernet LAN with multiple nodes and sets up TCP traffic between them to monitor congestion windows. Let's break down the script line by line:

1. set ns [new Simulator]: This line initializes the NS (Network Simulator) object.
2. set tf [open lab3.tr w]: This line opens a file named lab3.tr in write mode for storing trace information.
3. $ns trace-all $tf: This line instructs the simulator to trace all events and output them to the lab3.tr file.
4. set nf [open lab3.nam w]: This line opens a file named lab3.nam in write mode for generating network animation (nam) trace.
5. $ns namtrace-all $nf: This line instructs the simulator to trace all events for network animation and output them to the lab3.nam file.
6. Node creation and configuration:
   - Nodes are created using $ns node command, and their properties like color and label are set.
   - Each node represents a source or destination in the simulated LAN.
7. make-lan command:
   - $ns make-lan "$n0 $n1 $n2 $n3 $n4" 50Mb 100ms LL Queue/DropTail Mac/802_3: This command creates a LAN topology with specified nodes, bandwidth (50Mb), delay (100ms), link-layer type (LL), queue type (Queue/DropTail), and MAC layer type (Mac/802_3).
8. duplex-link and duplex-link-op commands:
   - These commands create duplex links between nodes n4 and n5 with specified bandwidth, delay, and queue type. duplex-link-op sets the orientation of the link for visualization.
9. Traffic generation and routing setup:
   - TCP agents and applications (FTP) are attached to nodes to generate traffic.
   - Traffic between nodes n0 and n5 is set up using TCP.
   - Similarly, traffic between nodes n2 and n3 is set up using TCP.

10. Trace setup:
    - Tracing of congestion window (cwnd) for each TCP agent is set up using the trace command.
11. finish procedure:
    - This procedure is called at the end of the simulation to flush traces, close files, and launch the network animator (nam).
12. Event scheduling:
    - Events are scheduled using $ns at command to start and stop traffic at specific times.
13. run command:
    - Initiates the simulation run.

AWK script:

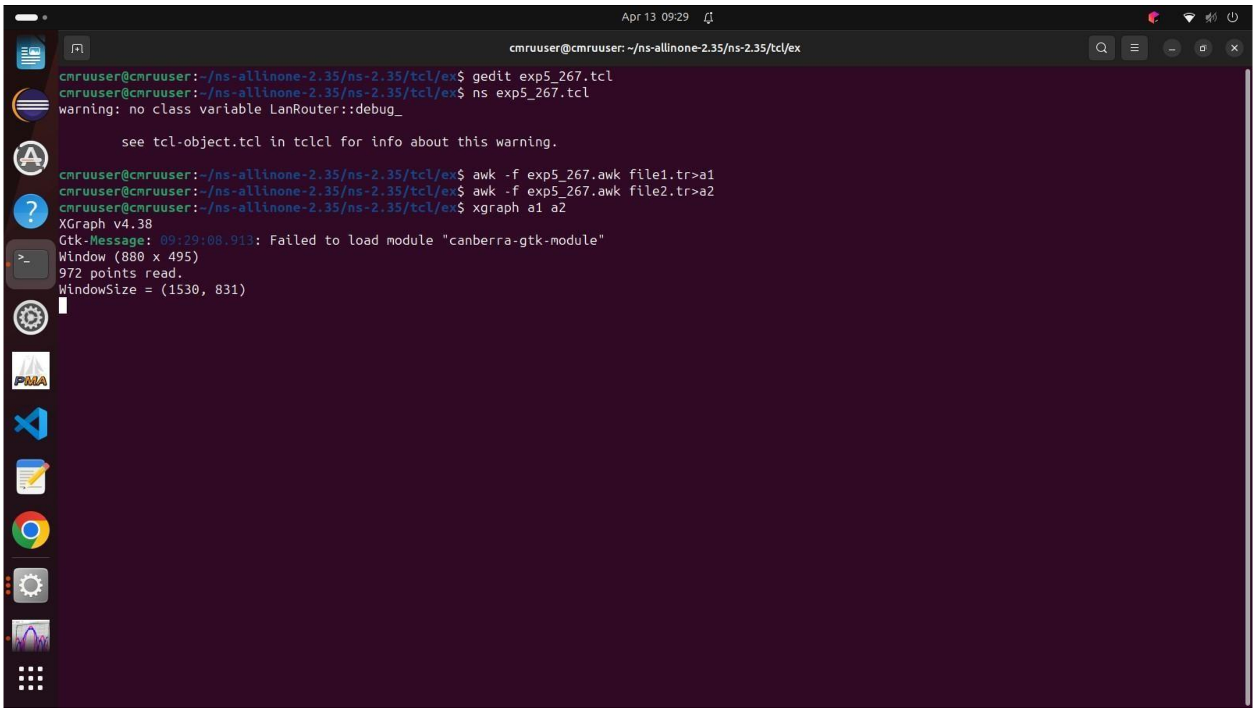
This script extracts congestion window (cwnd) information from the trace files generated during the simulation (file1.tr and file2.tr), and formats it for plotting.

Running the script:

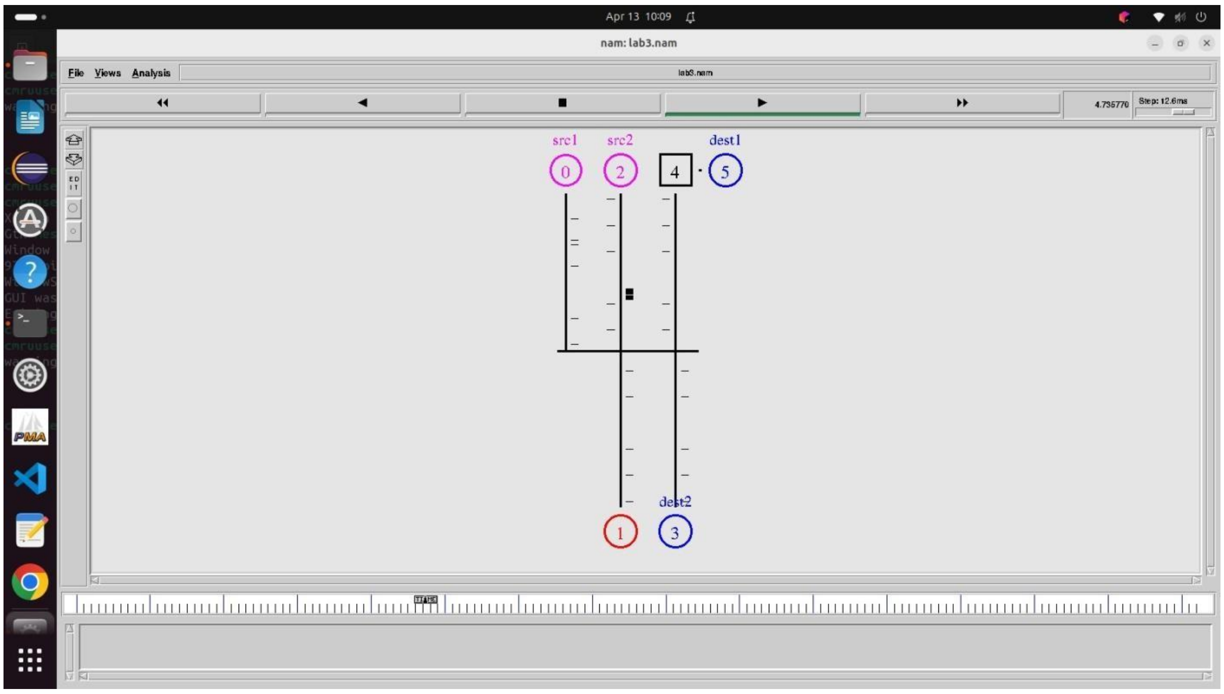
1. Save the AWK script with the same filename and. awk extension.
2. Run the AWK script using awk -f filename.awk file1.tr > a1 and awk -f filename.awk file2.tr > a2 to process the trace files and redirect the output to files a1 and a2.
3. Finally, plot the congestion windows using xgraph with a1 and a2 files as inputs.

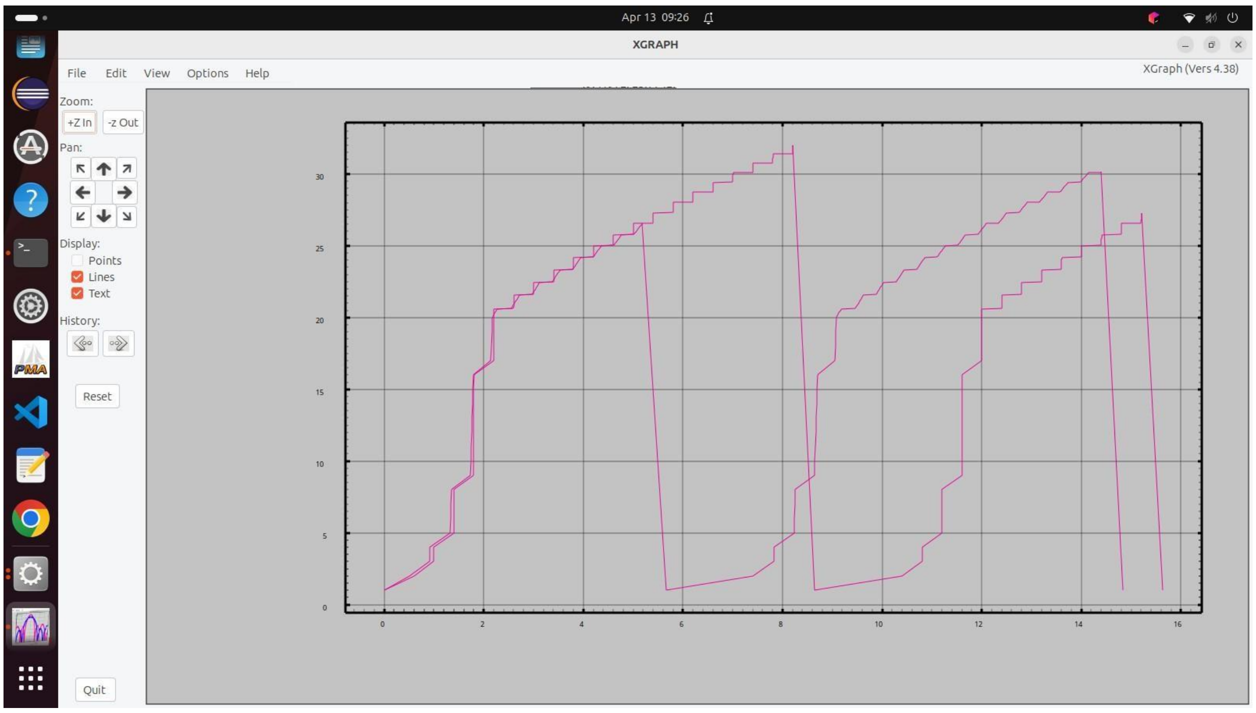
This setup allows you to simulate LAN congestion and monitor congestion window dynamics between different source-destination pairs in the network.

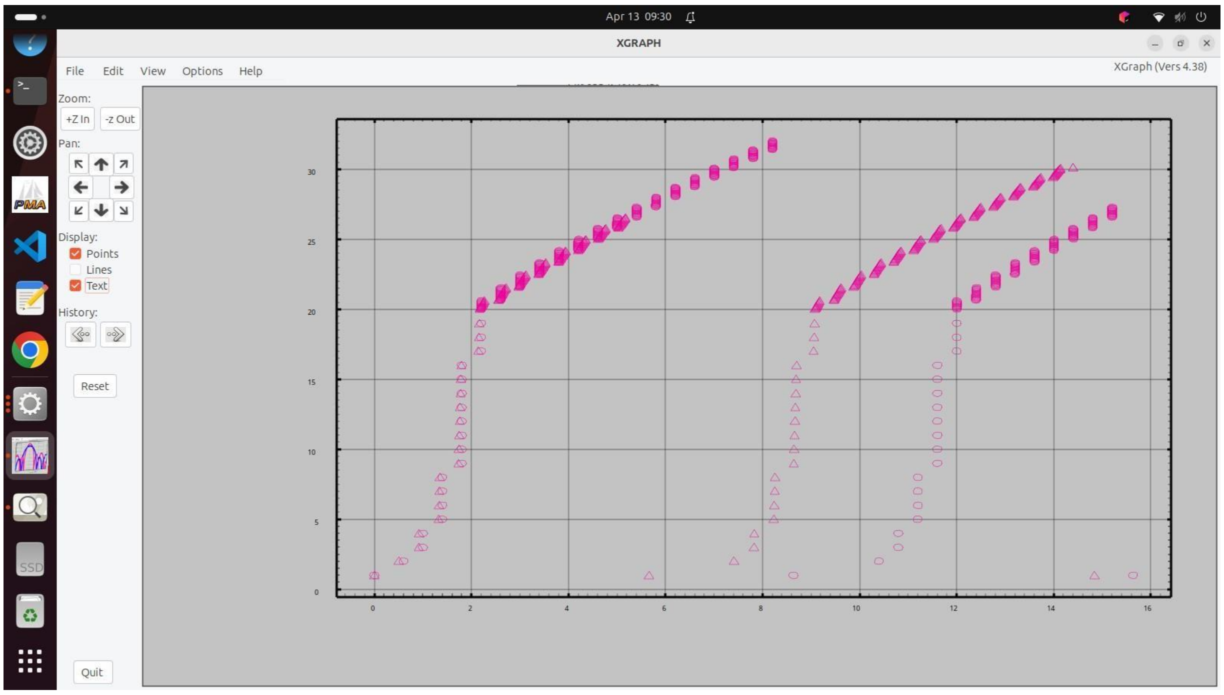
Output



```
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ gedit exp5_267.tcl
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns exp5_267.tcl
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f exp5_267.awk file1.tr>a1
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ awk -f exp5_267.awk file2.tr>a2
cmruuser@cmruuser:~/ns-allinone-2.35/ns-2.35/tcl/ex$ xgraph a1 a2
XGraph v4.38
Gtk-Message: 09:29:08.913: Failed to load module "canberra-gtk-module"
Window (880 x 495)
972 points read.
WindowSize = (1530, 831)
```







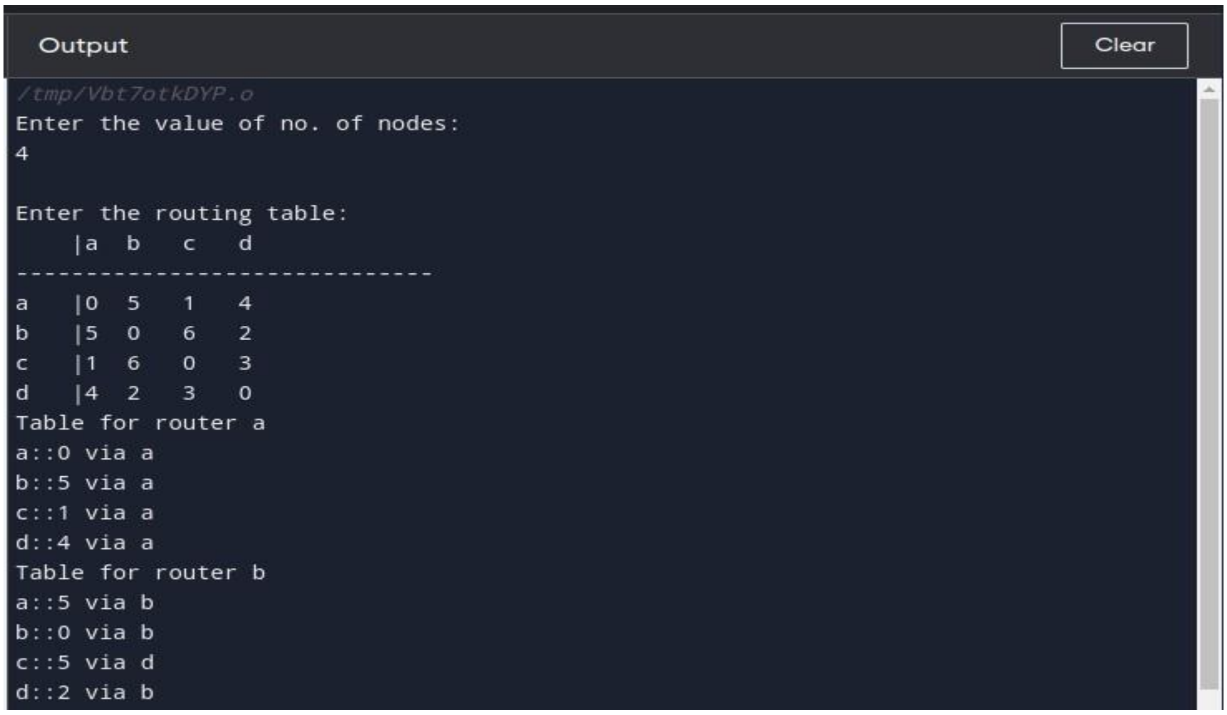
Result: The program was executed successfully and the output was obtained.

Program 6

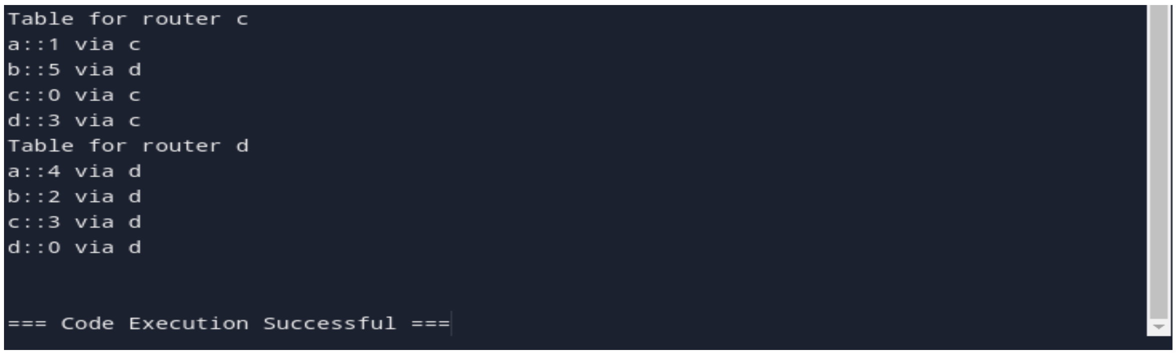
Ex. 6: **Routing Algorithms: Write a program for distance vector algorithm to find suitable path for transmission.**

```
#include<stdio.h>
#include<stdlib.h>
void rout_table();
int d[10][10],via[10][10];
int i,j,k,l,m,n,g[10][10],temp[10][10],ch,cost;

int main()
{
        printf("enter the value of no. of nodes\n");
        scanf("%d",&n);
        rout_table();
        for(i=0;i<n;i++)
        for(j=0;j<n;j++)
                temp[i][j]=g[i][j];
        for(i=0;i<n;i++)
        for(j=0;j<n;j++)
                via[i][j]=i;
        while(1)
        {
                for(i=0;i<n;i++) for(j=0;j<n;j++) if(d[i][j])
                for(k=0;k<n;k++)
                        if(gli+g][k]<g[i][k])
                        {
                                g[i][k]=g[i][j]+gj][k];
                                via[i][k]-j;
                        }
                for(i=0;i<n;i++)
                {
                        printf("table for router %c\n",i+97);
                        for(j=0;j<n;j++)
                                printf("%c:: %d via %c\n",j+97,g[i][j],via[i][j]+97);

                        }
                break;
                }
        }

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

                for(j=0;j<n;j++)
                {
                        scanf("%d",&g[i]]);
                         if(g[i][j]!=999)
                                d[i][j]=1;
                }
        }
}
```

Output



```
/tmp/Vbt7otkDYP.o
Enter the value of no. of nodes:
4

Enter the routing table:
    |a  b   c   d
------------------------------
a   |0  5   1   4
b   |5  0   6   2
c   |1  6   0   3
d   |4  2   3   0
Table for router a
a::0 via a
b::5 via a
c::1 via a
d::4 via a
Table for router b
a::5 via b
b::0 via b
c::5 via d
d::2 via b
```



```
Table for router c
a::1 via c
b::5 via d
c::0 via c
d::3 via c
Table for router d
a::4 via d
b::2 via d
c::3 via d
d::0 via d


=== Code Execution Successful ===
```

Result: The program was executed successfully and the output was obtained.

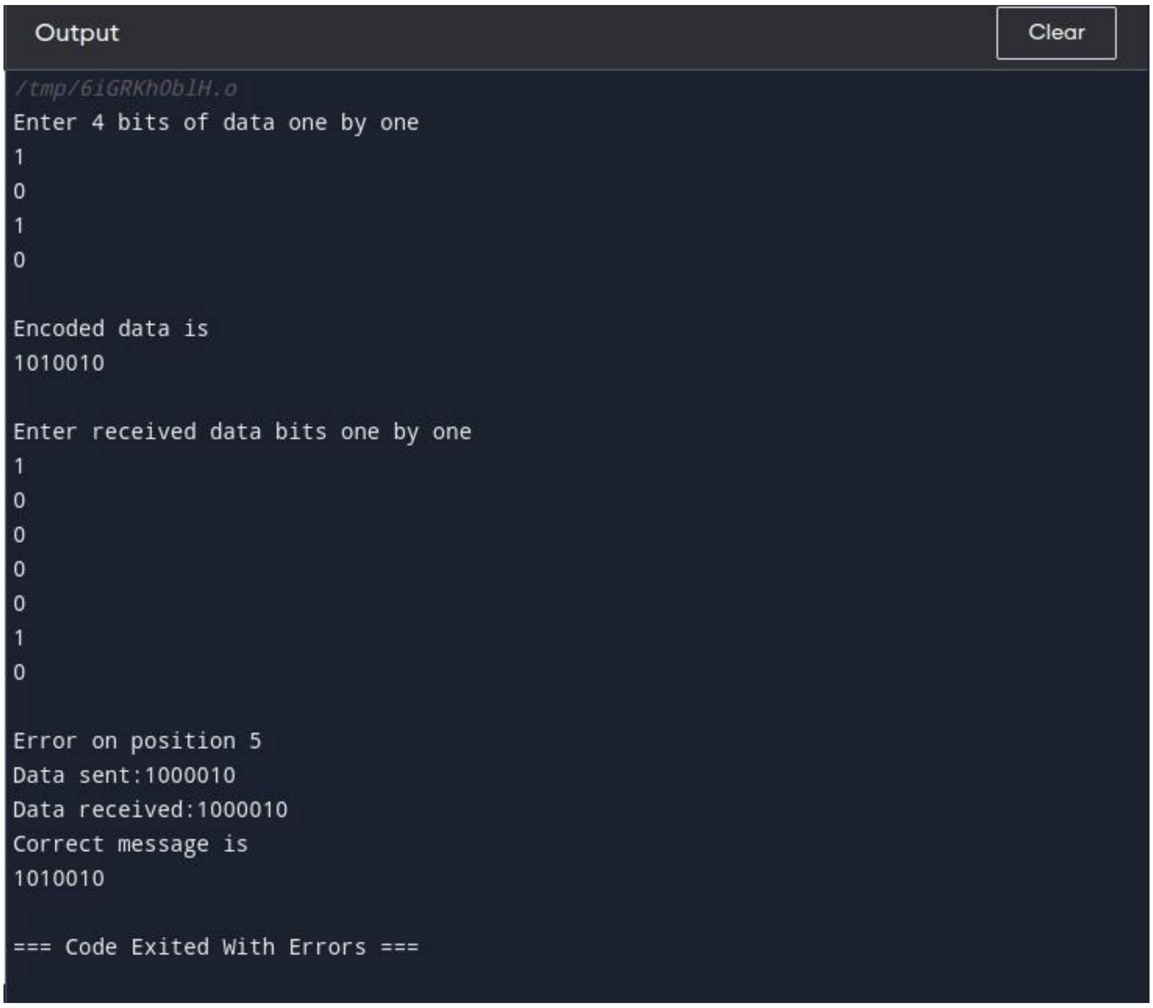
Program 7

Ex. 7. **Error Detection Techniques: Write a program for Hamming code/CRC**.

Program for Hamming Code in C

```
#include<stdio.h>

void main() {
        int data[10];
        int dataatrec[10],c,c1,c2,c3,i;

        printf("Enter 4 bits of data one by one\n");
        scanf("%d",&data[0]);
        scanf("%d",&data[1]);
        scanf("%d",&data[2]);
        scanf("%d",&data[4]);

        //Calculation of even parity
        data[6]=data[0]^data[2]^data[4];
        data[5]=data[0]^data[1]^data[4];
        data[3]=data[0]^data[1]^data[2];

        printf("\nEncoded data is\n");
        for(i=0;i<7;i++)
                printf("%d", data[i]);

        printf("\n\nEnter received data bits one by one\n");
        for(i=0;i<7;i++)
                scanf("%d",&dataatrec[i]);

        c1=dataatrec[6]^dataatrec[4]^dataatrec[2]  dataatrec[0];
        c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
        c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
        c=c3*4+c2*2+cl;

        if(c=0) {
                printf("\nNo error while transmission of data\n");
        }
        else {
                printf("\nError on position %d",c);

                printf("\nData sent: ");
                for(i=0;i<7;i++)
                        printf("%d", data[i]);

                printf("\nData received: ");
                for(i=0;i<7;i++)
                        printf("%d", dataatrec[i]);
                printf("\nCorrect message is\n");

                //if errorneous bit is 0 we complement it else vice versa
                if(dataatrec 7-c]=0)
                        dataatrec[7-c]=1;
                else
                        dataatrec[7-c]=0;

                for (i=0;i<7;i++) {
                        printf("%d", dataatrec[i]);
                }
        }
}
```

Output



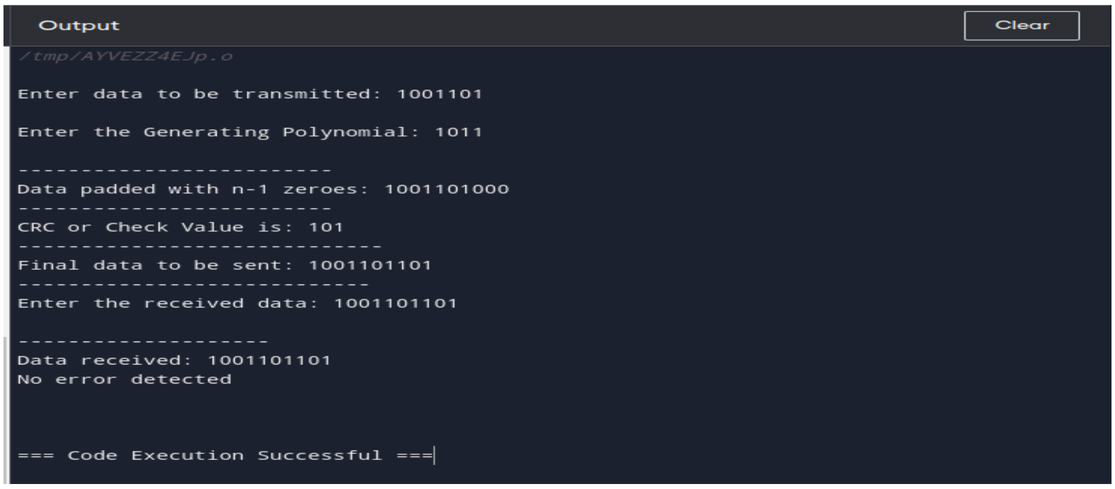
```
/tmp/6iGRKhOblH.o
Enter 4 bits of data one by one
1
0
1
0

Encoded data is
1010010

Enter received data bits one by one
1
0
0
0
0
1
0

Error on position 5
Data sent:1000010
Data received:1000010
Correct message is
1010010

=== Code Exited With Errors ===
```

Program for Cyclic Redundancy Check(CRC) in C

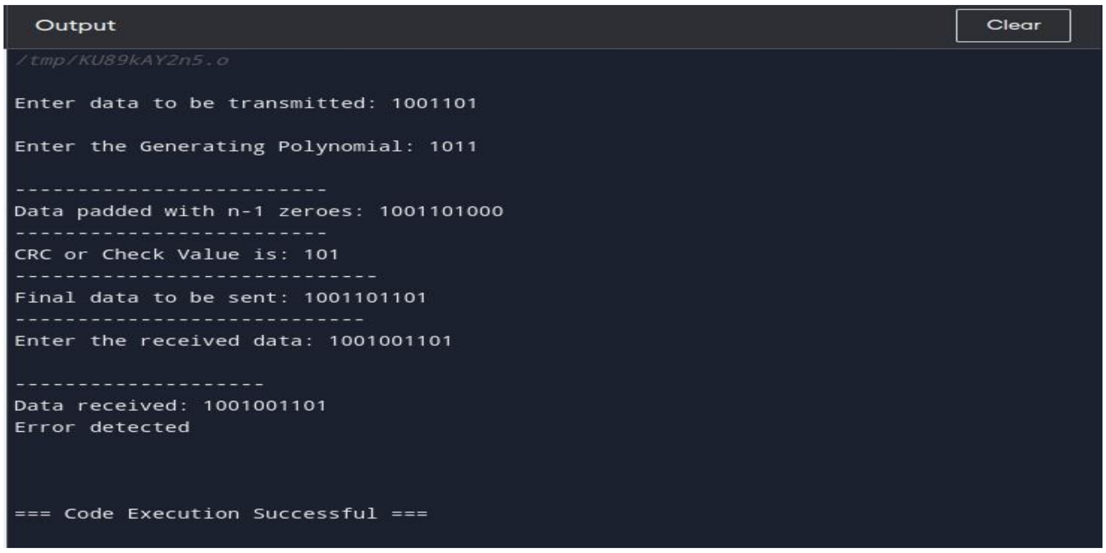
```
//  Include  headers
#include<stdio.h>
#include<string.h>

// length of the generator polynomial
#define N strlen(gen_poly)

// data to be transmitted and received
char data[28];

// CRC value
char check_value[28];

// generator polynomial
char gen_poly[10];

// variables
int data_length,i,j;

// function that performs XOR operation
void XOR(){

        // if both bits are the same, the output is 0
        // if the bits are different the output is 1
        for(j=1;j<N; j++)
                check_value[j] = ((check_value[j] == gen_poly[j])?'0':'1');
}

// Function to check for errors on the receiver side
void receiver(){

        // get the received data
        printf("Enter the received data: ");
        scanf("%s", data);
        printf("\n-----------------------------\n");
        printf("Data received: %s", data);

        // Cyclic Redundancy Check
        crc();

        // Check if the remainder is zero to find the error
        for(i=0;(i<N-1) && (check_value[i]!='1');i++);
                if(i<N-1)
                        printf("\nError detected\n\n");
                else
                        printf("\nNo error detected\n\n");
}
```

```
void crc(){

        // initializing check_value
        for(i=0;i<N;i++)
                check_value[i]=data[i];
        do{

                // check if the first bit is 1 and calls XOR function
                if(check_value[0]=='1')
                        XOR();

                // Move the bits by 1 position for the next computation
                for(j=0;j<N-1;j++)
                        check_value[j]=check_value[j+1];

                // appending a bit from data
                check_value[j]=data[i++];

        } while(i<=data_length+N-1);

        // loop until the data ends
}

int main()
{

        // get the data to be transmitted
        printf("\nEnter data to be transmitted: ");
        scanf("%s", data);

        printf("\n Enter the Generating polynomial: ");
        // get the generator polynomial
        scanf("%s",gen_poly);

        // find the length of data
        data_length=strlen(data);

        // appending n-1 zeros to the data
        for(i=data_length;i<data_length+N-1;i++)
                data[i]='0';
        printf("\n----------------------------------------");

        // print the data with padded zeros
        printf("\n Data padded with n-1 zeros: %s", data);
        printf("\n----------------------------------------");

        // Cyclic Redundancy Check crc();
        // print the computed check value
        printf("\nCRC or Check value is: %s", check_value);
        // Append data with check_value(CRC)
        for(i=data_length;i<data_length+N-1;i++)
                data[i]=check_value[i-data_length];
        printf("\n----------------------------------------");

        // printing the final data to be sent
        printf("\n Final data to be sent: %s", data);
        printf("\n----------------------------------------\n");

        // Calling the receiver function to check errors
        receiver();
        return 0;

}
```

Output



```
Output                                                    Clear
/tmp/AYVEZZ4EJp.o

Enter data to be transmitted: 1001101

Enter the Generating Polynomial: 1011

-------------------------
Data padded with n-1 zeroes: 1001101000
-------------------------
CRC or Check Value is: 101
-----------------------------
Final data to be sent: 1001101101
----------------------------
Enter the received data: 1001101101

--------------------
Data received: 1001101101
No error detected


=== Code Execution Successful ===
```



```
Output                                                    Clear
/tmp/KU89kAY2n5.o

Enter data to be transmitted: 1001101

Enter the Generating Polynomial: 1011

-------------------------
Data padded with n-1 zeroes: 1001101000
-------------------------
CRC or Check Value is: 101
-----------------------------
Final data to be sent: 1001101101
----------------------------
Enter the received data: 1001001101

--------------------
Data received: 1001001101
Error detected


=== Code Execution Successful ===
```

Result: The program was executed successfully and the output was obtained.

Socket Programming

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

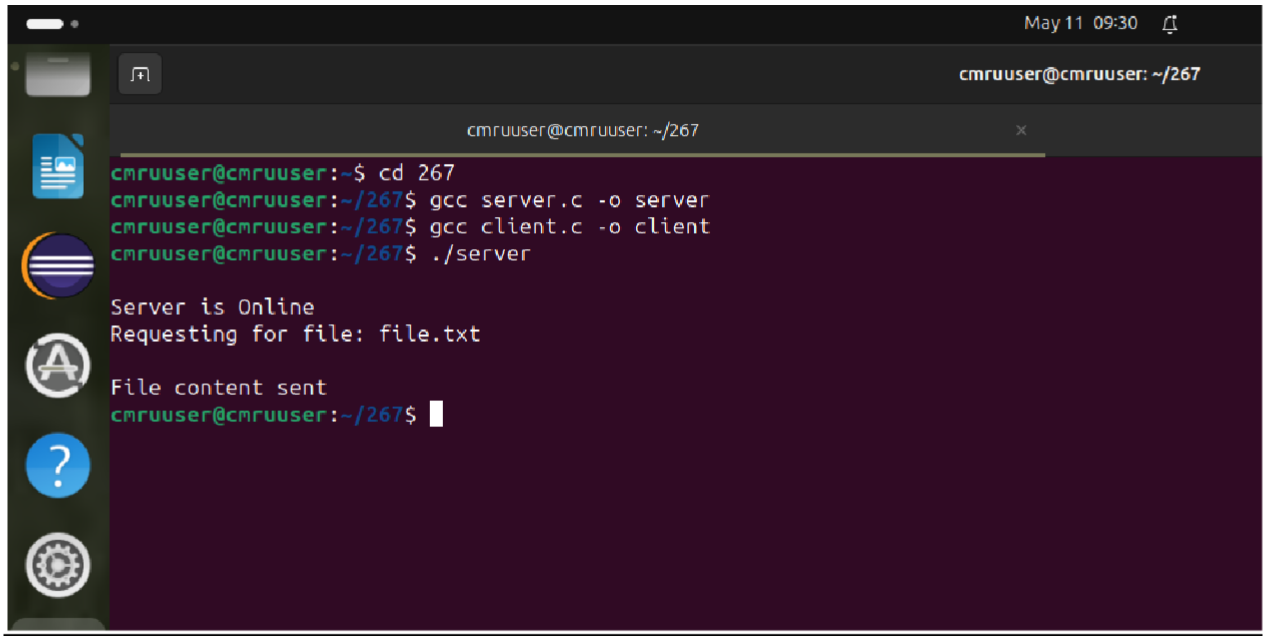
Client Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
        int sock, n;
        char buffer[1024], fname[50];

        sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in addr = {AF_INET, htons (1234), inet_addr("127.0.0.1") };

        /* keep trying to esatablish connection with server */
        while(connect(sock, (struct sockaddr *) &addr, sizeof(addr)));
        printf("\nClient is connected to Server");

        /* send the filename to the server */
        printf("\nEnter file name: ");
        scanf("%s", fname);

        send(sock, fname, sizeof(fname), 0);

        printf("\nRecieved file data\n");

        printf("- - - - - - - - - - -\n");

        /* keep printing any data received from the server */
        while ((n = recv(sock, buffer, sizeof(buffer), 0)) > 0)
        {
                buffer[n] = '\0';
                printf("%s", buffer);
        }

        printf(" -----------\n");

        return 0;

}
```

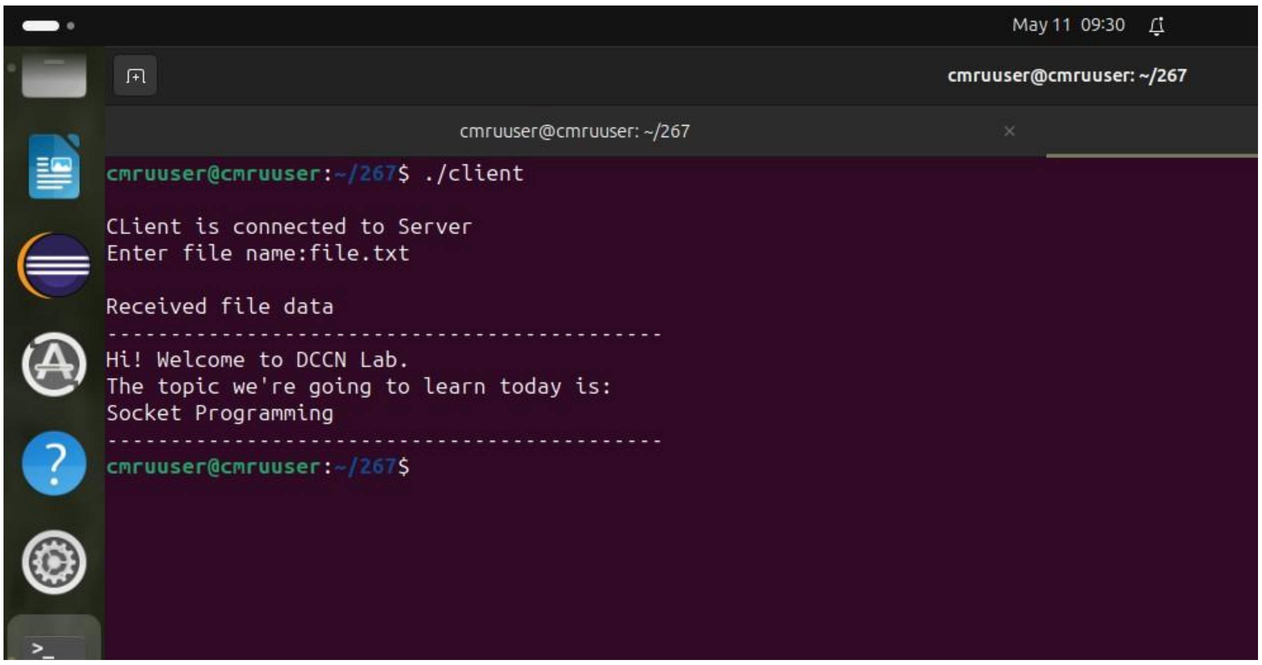
Server Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
        int sersock, sock, fd, n, reuse = 1;
        char buffer[1024], fname[50];

        /* sockfd = socket(domain, type, protocol) */
        sersock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in addr = {AF_INET, htons (1234), inet_addr("127.0.0.1") };

        // Forcefully connecting to same port everytime
        setsockopt(sersock, SOL_SOCKET, SO_REUSEADDR, (char *)&reuse,
        sizeof(reuse));

        /* attaching socket to port */
        bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
        printf("\nServer is Online");
        listen(sersock, 5); // listen(int sockfd, int backlog)
        sock = accept(sersock, NULL, NULL);

        /* receive the filename */
        recv(sock, fname, 50, 0);
        printf("\nRequesting for file: %s\n", fname);

        /* open the file in read-only mode */
        fd = open(fname, O_RDONLY);
        if (fd < 0)
        {
                send(sock, "\nFile not found\n", 15, 0);        // strlen(\nFile not found)=1
        }

        else
        {
                while ((n = read(fd, buffer, sizeof(buffer))) > 0)
                {
                        send(sock, buffer, n, 0);
                }
        }

        printf("\nFile content sent\n");
        close(fd);
        return 0;

}
```

Output



```
cmruuser@cmruuser:~$ cd 267
cmruuser@cmruuser:~/267$ gcc server.c -o server
cmruuser@cmruuser:~/267$ gcc client.c -o client
cmruuser@cmruuser:~/267$ ./server

Server is Online
Requesting for file: file.txt

File content sent
cmruuser@cmruuser:~/267$
```



```
cmruuser@cmruuser:~/267$ ./client

CLient is connected to Server
Enter file name:file.txt

Received file data
-------------------------------------------
Hi! Welcome to DCCN Lab.
The topic we're going to learn today is:
Socket Programming
-------------------------------------------
cmruuser@cmruuser:~/267$
```

Result: The program was executed successfully and the output was obtained.

Congestion Control

Ex.9: Write a program for congestion control using leaky bucket algorithm

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

#define NOF_PACKETS 10

int my_rand(int a) {
        int n = (random() % 10) % a;
        return rn == 0 ? 1 : rn;
}

int main() {

        int packet_sz[NOF_PACKETS], i, clk, b_size, o_rate, p_sz_rm = 0, p_sz, p_time, op;

        for (i = 0; i < NOF_PACKETS; ++i)
                packet_sz[i] = my_rand(6) *10;

        for (i = 0; i < NOF_PACKETS; ++i)
                printf("\npacket[%d]: %d bytes\t", i, packet_sz[i]);

        printf("\nEnter the Output rate:");
        scanf("%d", &o_rate);

        printf("Enter the Bucket Size:");
        scanf("%d", &b_size);

        for (i = 0; i < NOF_PACKETS; ++i) {

                if ((packet_sz[i] + p_sz_rm) > b_size)
                if (packet_sz[i] > b_size)

                        /*compare the packet siz with bucket size*/
                        printf("\n\nIncoming packet size (%dbytes) is Greater than bucket capacity
                (%dbytes)-PACKET REJECTED", packet_sz[i], b_size);

                else
                        printf("\n\nBucket capacity exceeded-PACKETS REJECTED!!");

                else {
                        p_sz_rm += packet_sz[i];
                        printf("\n\nIncoming Packet size: %d", packet_sz[i]);
                        printf("\nBytes remaining to Transmit: %d", p_sz_m);
                        p_time = my_rand(4)*10;
                        printf("\nTime left for transmission: %d units", p_time);
                        for (clk = 10; clk <= p_time; clk += 10) {
                                sleep(1);

                                if (p_sz_rm) {
                                        if (p_sz_rm <= o_rate)

                                        /*packet size remaining comparing with output rate*/
                                        op = p_sz_rm, p_sz_rm= 0;

                                else
                                        op = o_rate, p_sz_rm -= o_rate;

                                printf("\nPacket of size %d Transmitted", op);
                                printf(" --- Bytes Remaining to Transmit: %d", p_sz_rm);
                                }

                                else {

                                printf("\nTime left for transmission: %d units", p_time - clk);
                                printf("\nNo packets to transmit!!");
                                }
                        }
                }
        }
}
```

Explanation:
What is Leaky-Bucket algorithm?

The leaky bucket algorithm is a method of managing traffic in a network by smoothing the rate at which data is transmitted. It is a congestion control mechanism used to regulate the flow of data and to prevent bursts of traffic from overwhelming a network.

The concept of the leaky bucket can be understood through an analogy with a physical bucket that has a leak at the bottom. Here's how it works:

1. **Bucket**: Imagine a bucket that can hold a certain amount of water. This bucket represents a buffer or queue in the network.

2. **Water**: Incoming data packets are represented as water pouring into the bucket.

3. **Leak**: The bucket has a small hole at the bottom, causing water to leak out at a constant rate. This leak represents the maximum rate at which data can be transmitted from the buffer onto the network.

4. **Overflow**: If water pours into the bucket too quickly, it will eventually overflow. Similarly, if data packets arrive at a rate faster than the leak rate, they will overflow the buffer and may be dropped or delayed.

5. **Smoothed Traffic**: By limiting the rate at which data can be transmitted (the leak rate), the leaky bucket algorithm ensures that the traffic leaving the bucket is smooth and controlled, even if the incoming traffic is bursty.

In summary, the leaky bucket algorithm provides a way to regulate the flow of data in a network by controlling the rate at which data is transmitted. It helps in managing congestion and ensuring that the network operates efficiently without being overwhelmed by sudden bursts of traffic.

Explanation of the program:

1. **Include Libraries**: The program includes necessary header files such as **<stdio.h>**, **<stdlib.h>**, and **<unistd.h>** for input/output, standard library functions, and system calls respectively.

2. **Define Constants**: It defines a constant **NOF_PACKETS** which represents the number of packets to be processed.

3. **Custom rand Function**: There's a custom rand function defined which takes an integer argument **a** and generates a random number between 0 and **a-1**.

4. **Main Function**:

   - **Packet Size Initialization**: It initializes an array **packet_sz** with random sizes for each packet. Packet sizes are multiples of 10 and generated using the custom **rand** function.

   - **Print Packet Sizes**: It prints the size of each packet.

   - **Input Parameters**: It prompts the user to input the output rate (**o_rate**) and bucket size (**b_size**) for the token bucket algorithm.

   - **Packet Processing Loop**: It iterates through each packet to process them.

     - **Check Bucket Capacity**: It checks if adding the size of the current packet to the remaining size in the bucket exceeds the bucket size. If so, it rejects the packet.

     - **Packet Transmission**: If the bucket capacity is not exceeded, it proceeds with packet transmission.

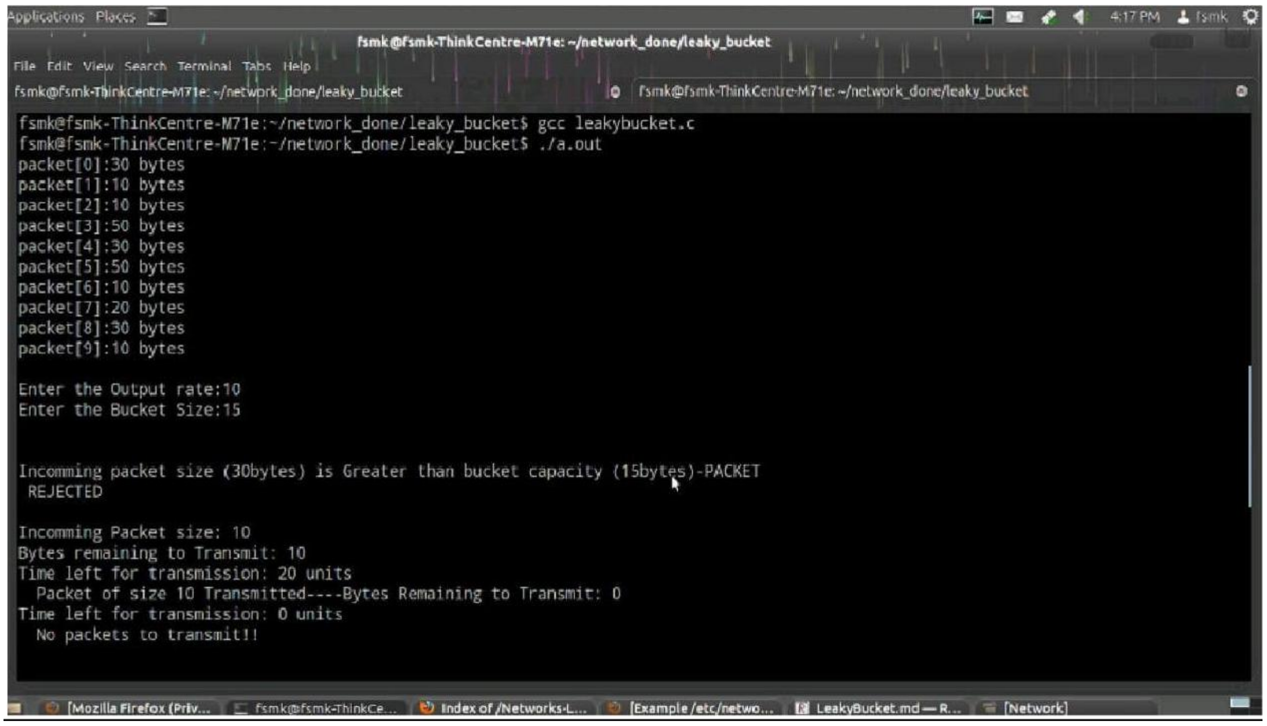
       - **Update Bucket Size**: It updates the remaining size in the bucket after adding the current packet size.

       - **Calculate Transmission Time**: It calculates a random transmission time for the packet.

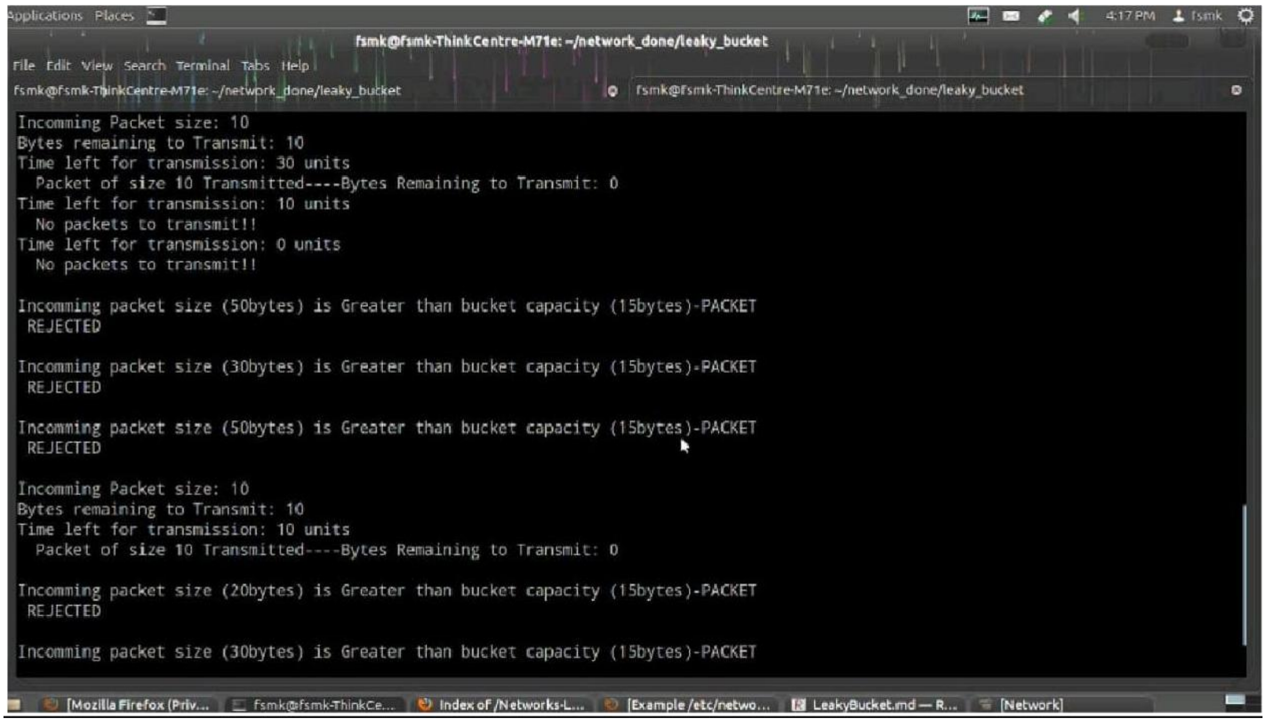
       - **Transmission Loop**: It simulates the transmission process by decrementing the transmission time in intervals of 10 units. During each interval, it checks if there are packets to transmit and if so, transmits them based on the output rate. It also updates the remaining packet size in the bucket after transmission.

This program helps to understand how token bucket algorithm works for controlling the rate of data transmission in a network by regulating the flow of packets. It demonstrates the concept of token bucket algorithm in a simplified manner.

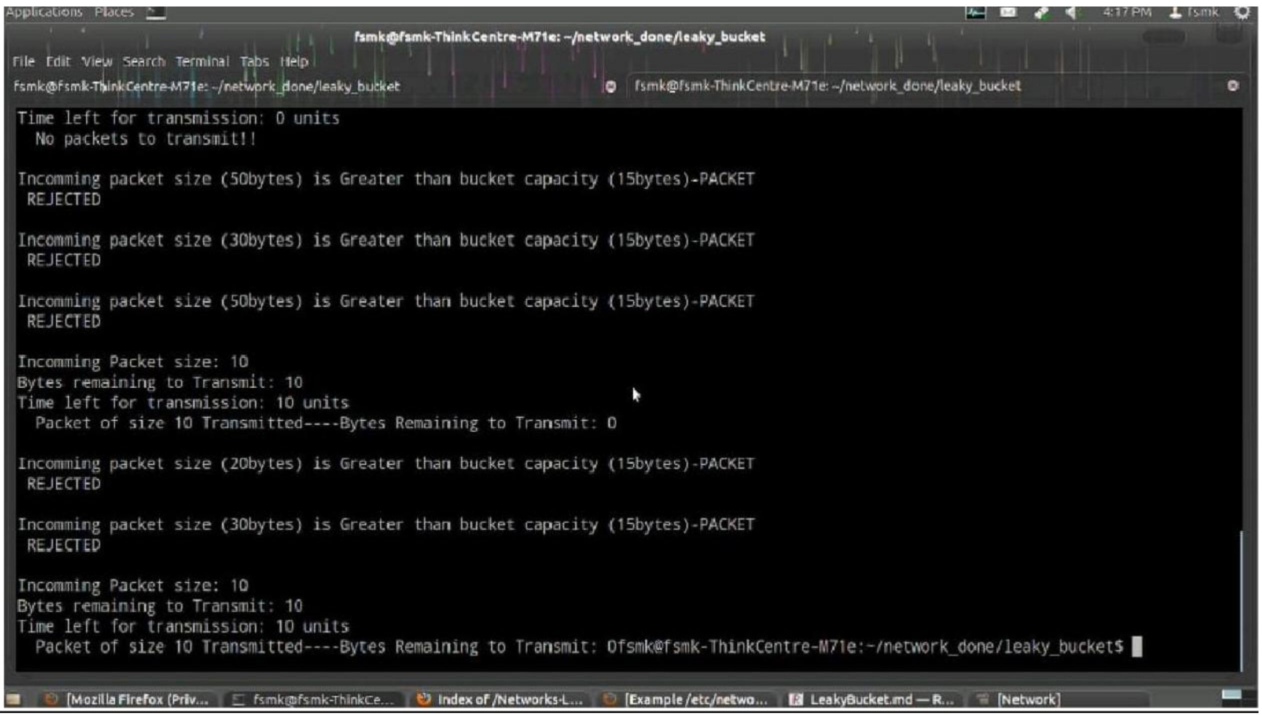
Output


Applications Places
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
File Edit View Search Terminal Tabs Help
[Mozilla Firefox (Priv... fsmk@fsmk-ThinkCe... Index of /Networks-L... [Example /etc/netwo... LeakyBucket.md — R... [Network]

```
fsmk@fsmk-ThinkCentre-M71e:~/network_done/leaky_bucket$ gcc leakybucket.c
fsmk@fsmk-ThinkCentre-M71e:~/network_done/leaky_bucket$ ./a.out
packet[0]:30 bytes
packet[1]:10 bytes
packet[2]:10 bytes
packet[3]:50 bytes
packet[4]:30 bytes
packet[5]:50 bytes
packet[6]:10 bytes
packet[7]:20 bytes
packet[8]:30 bytes
packet[9]:10 bytes

Enter the Output rate:10
Enter the Bucket Size:15


Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 20 units
  Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0
Time left for transmission: 0 units
  No packets to transmit!!
```


Applications Places
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
File Edit View Search Terminal Tabs Help
[Mozilla Firefox (Priv... fsmk@fsmk-ThinkCe... Index of /Networks-L... [Example /etc/netwo... LeakyBucket.md — R... [Network]

```
Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 30 units
  Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0
Time left for transmission: 10 units
  No packets to transmit!!
Time left for transmission: 0 units
  No packets to transmit!!

Incomming packet size (50bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming packet size (50bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
  Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0

Incomming packet size (20bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
```



```
Time left for transmission: 0 units
  No packets to transmit!!

Incomming packet size (50bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming packet size (50bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
  Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0

Incomming packet size (20bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
 REJECTED

Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
  Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0fsmk@fsmk-ThinkCentre-M71e:~/network_done/leaky_bucket$
```

Result: The program was executed successfully and the output was obtained.