EXPERIMENT NO. 2

a. Introduction to tcl programming and implementation of two node network using NS (Network Simulator)-2

Aim: To introduce tcl (tool command language) programming.

Requirements

1. Ubuntu operating system
2. Network Simulator NS2 (preferably version 2.35)

List of Programs to be conducted-

1. Print 'Hello World' using tcl programming.
2. Display age (any data) using variables using tcl programming.
3. Input data from the end user using tcl programming.
4. To create a simple two node network using tcl programming.

Introduction

The network simulator is a discrete event packet level simulator. It covers a very large number of applications of different kinds of protocols of different network types consisting of different network elements and traffic models. Network simulator includes a package of tools that simulates behavior of networks such as creating network topologies, log events that happen under any load, analyze the events and understand the network. It is mainly based on two languages, C++ and O-Tcl (object oriented extension of Tcl).

NS needs certain supportive packages such as 'Nam (network animator)'. Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet traces. It supports topology layout, packet level animation, and various data inspection tools. NS 2 and Nam have evolved substantially over the past few years. Currently, they are being developed as an open source project hosted at Sourceforge.

Common Procedure for Tcl programming,

1. Open the editor.
2. Write the program.
3. Save the program.
4. Open the terminal.
5. Run/Execute the program using the 'ns' command.

The detailed procedure to create a simple two node network using tcl is given below. Creating node includes the following steps-

i. Starting a new simulator.
ii. Creating output files.
iii. Terminating the program.
iv. Starting the simulation.
v. Defining nodes and links.
vi. Attaching agents and applications.
vii. Scheduling events.
viii. Viewing the output in nam.
ix. Tracing the output.

Algorithm

1. Start a new simulator.
2. Create the output trace file (.tr) and nam file (.nam)
3. Terminate the program.
4. End the program by calling 'finish' proc.
5. Start simulation
    a. Define nodes, links query and topology.
    b. Set agents and applications.
    c. Initiate FTP over TCP/UDP.
    d. Scheduling
    e. Run the simulation.
6. Check the out files.

Implementation

1. Open file using > gedit *expname.tcl* in the root directory (students can give other name to their files)
2. Write the program.
3. Run the program using ns *expname.tcl*
4. The output is in form of .nam and .tr file which are generated in the same directory

Observations: Take appropriate snapshots and write observations below each snapshot with a pen.

Post Experimental Exercise:

Note the observations and frame the appropriate conclusions.

1. Change the orientation of the nodes.
2. Change the bandwidth parameter and check the effect in output.
3. Change the delay parameter and check the output.

Conclusion:

EXPERIMENT NO. 2

b. Implementation of four node network topology using NS-2

Aim: To create a network consisting of four nodes in order to understand more features of Tcl programming in NS-2. This includes-

1. Formation of suitable topology
2. Marking flows and adding colors
3. Defining packet size, packet interval
4. Setting queue size of the link
5. Monitoring a queue

Requirements

Ubuntu operating system

NS2

Procedure: This program deals with data transfer between different nodes using a fixed topology. The steps to be followed are listed

1. Start a new simulator
2. Create the output files
3. Write the finish procedure
4. Create the required topology by,
    a. Defining nodes and the links
    b. Creating, attaching and connecting transport layer agents
    c. Creating and attaching application layer agent
    d. Scheduling the events
    e. Starting the simulation
5. Run the simulation
6. Observe the animated output in the 'nam' window.
7. Check the trace file

Formation of suitable topology (Ref: https://www.isi.edu/nsnam/ns/tutorial/)

When we run the ns-2 program in nam (network animator), it gives random positions to the nodes. We can give suitable initial positions to the nodes and can form a suitable topology.

```
#Give position to the nodes in nam

$ns duplex-link-op $n0 $n2 orient-right-down
```

```
$ns duplex-link-op $n1 $n2 orient-right-up
$ns simplex-link-op $n2 $n3 orient-right
$ns simplex-link-op $n3 $n2 orient-left
$ns duplex-link-op $n3 $n4 orient-right-up
$ns duplex-link-op $n3 $n5 orient-right-down
```

Marking flows and adding colors

When more than two nodes are present, it becomes necessary to associate different class to different flows and then assign a single color to each class to easily follow the different flows in nam.

```
$udp0 set class_ 1
$udp1 set class_ 2
```

The parameter 'fid_' stands for 'flow id'.

Add the following piece of code to your Tcl script, preferably at the beginning after the simulator object has been created, since this is a part of the simulator setup.

```
$ns color 1 Blue
$ns color 2 Red
```

This code allows you to set different colors for each flow id.

Defining packet size, packet interval

We can set packet size by using the command

```
$cbr set packetSize_(packetsize)
```

After defining the UDP source and UDP agent, instead of defining the rate we can define the time interval between the transmission of packets using the command

```
$cbr0 set interval_ 0.005
```

Example of setting packet size and packet interval

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

These lines create a UDP agent and attach it to the node n0, then attach a CBR traffic generator to the UDP agent. The packetSize is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second). This results in a bandwidth of 0.8 megabits per second for the link between

the two nodes. If the link bandwidth selected is less than 0.8 megabits, some packets are being discarded.

Setting queue size of the link

In NS an output queue of a node is implemented as a part of a link whose input is that node to handle the overflow at the queue. But if the buffer capacity of the output queue is exceeded then the last packet arrived is dropped and here we will use a 'DropTail' option. Many other options such as RED (Random Early Discard) mechanism, FQ (Fair Queuing), DRR (Deficit Round Robin), SFQ (Stochastic Fair Queuing) are available.

So now we will define the buffer capacity of the queue related to the above link

```
#Set queue size of the link
$ns queue-limit $n0 $n2 20
```

Monitoring a queue

Add the following line to your code to monitor the queue for the link from n2 to n3.

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

You can see the packets in the queue now, and after a while you can even see how the packets are being dropped.

Observations:

Take appropriate screenshots and write observations below each. Students are expected to take minimum of four snapshots for different periods of simulations.

Post Experimental Exercise

1. What is a ‘queue’ in computer networks? Explain different queuing mechanisms in brief.

   (Ref:PDF of Queue Network in Networking)

2. In your simulation, change the color of the packets moving from node 1 to 2 and 2 to 3 and observe the output. (take screenshot and attach as the output)

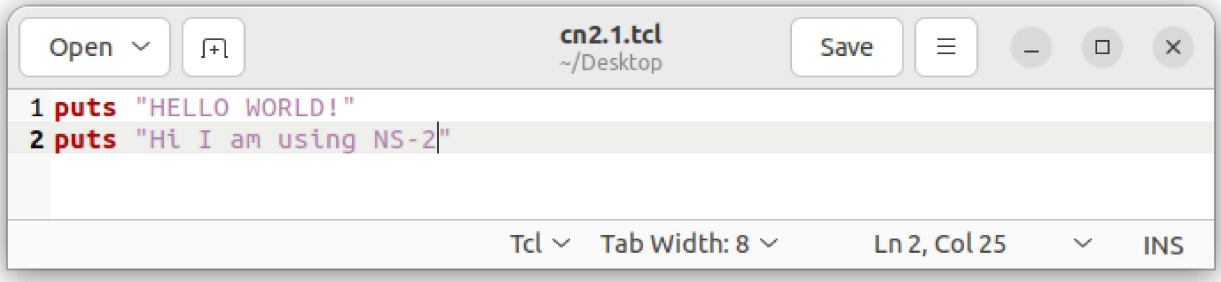
Conclusion:

EXP 2 A Introduction to tcl programming and implementation of two node network using NS (Network Simulator)-2

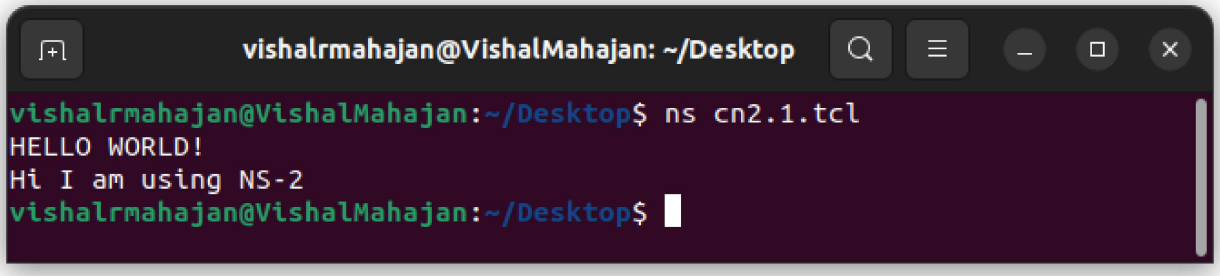
Observation:

1. Print ‘Hello World’ using tcl programming

Code:



Output:

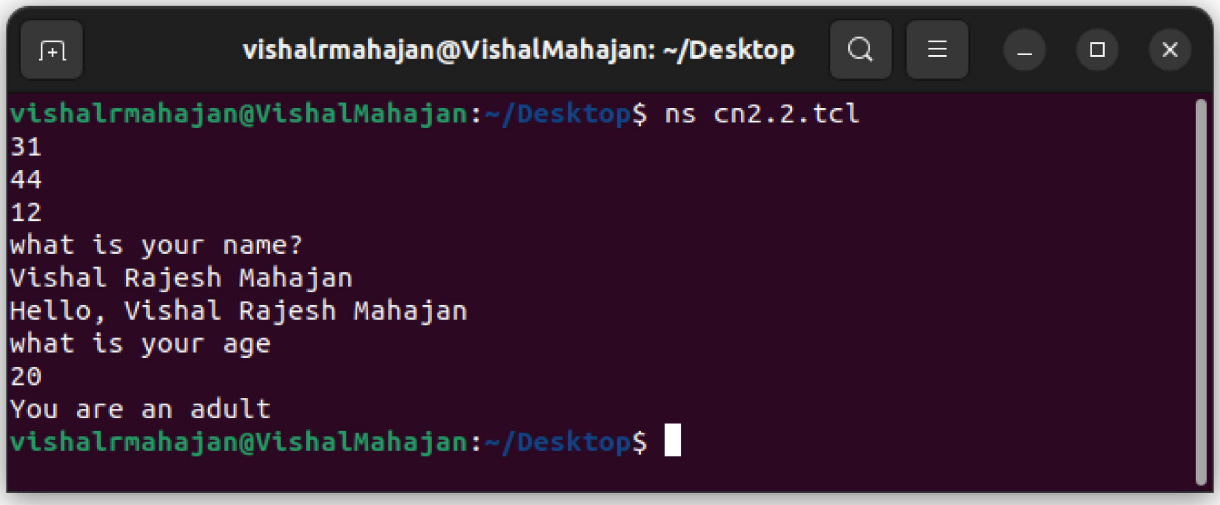


The puts command is utilized in this context to show the strings "Hello World" and "I am using NS2" in the console. This command functions similarly to printf in C or print in Python, but unlike those languages, we do not need to enclose the string in parentheses. Instead, the string is passed directly after the puts command with a whitespace in between, resembling putStr in Haskell.

Program which takes input in form of “AGE” from the user and prints whether he/she is Child,Teenager or an Adult

Program:

Output



This code determines if a user is a child, teen, or adult based on inputted name and age, utilizing variables set with set, user input with gets, and if-else statements for age categorization.

Base program of simulator object:

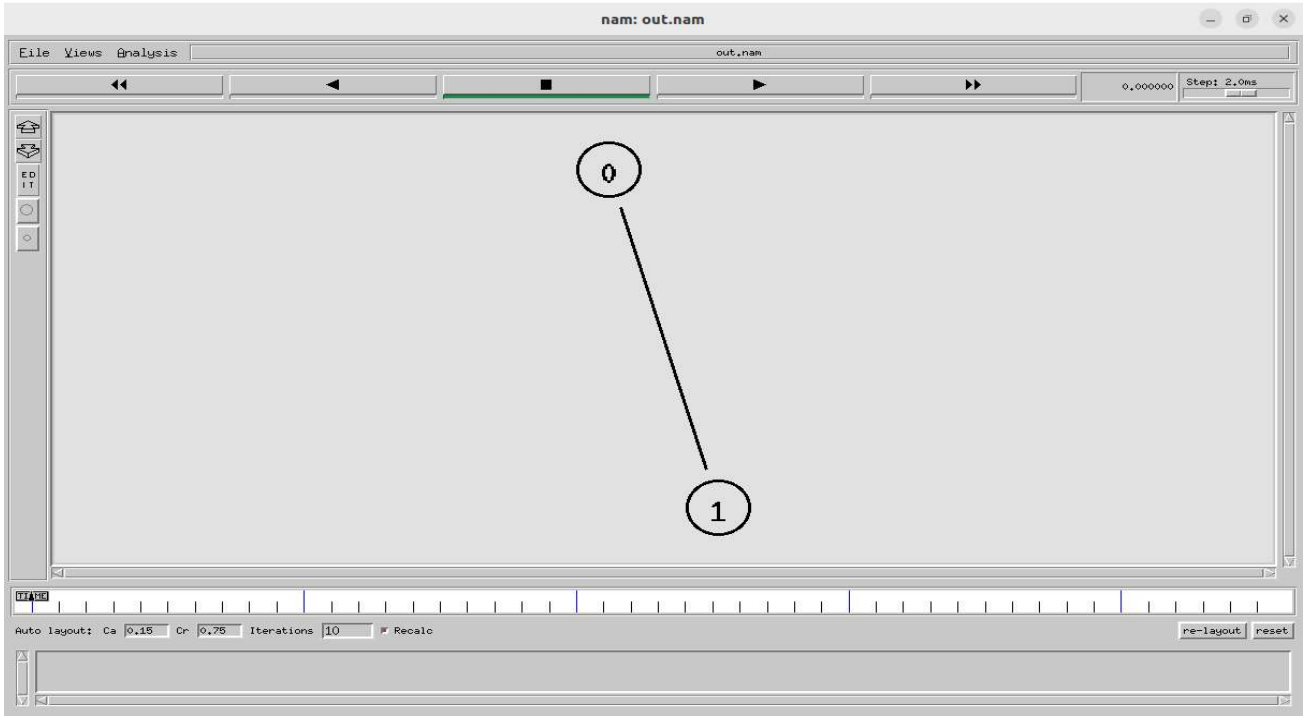
PROGRAM:

```
#Create a simulator object

set ns [new Simulator]


#Open the trace file

set tracefile [open out.tr w]

$ns trace-all $tracefile


#Open the nam trace file

set nf [open out.nam w]

$ns namtrace-all $nf


#Define a 'finish' procedure

proc finish {} {

        global ns nf

        $ns flush-trace

          #Close the trace file

        close $nf

          #Execute nam on the trace file

        exec nam out.nam &

        exit 0

}

#Create two nodes

set n0 [$ns node]
```

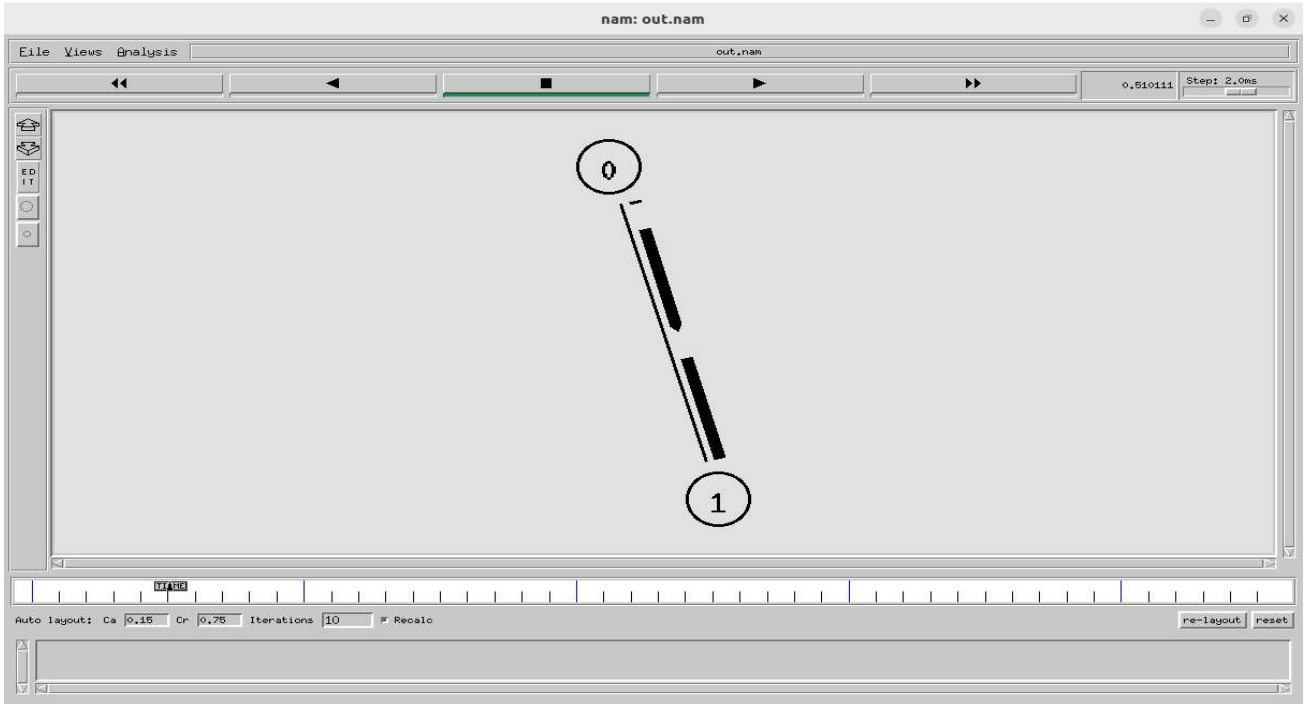
```
set n1 [$ns node]
#Create a simplex link between the nodes
$ns simplex-link $n0 $n1 1Mb 10ms DropTail
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0

#Schedule events for the CBR agent
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

Output:

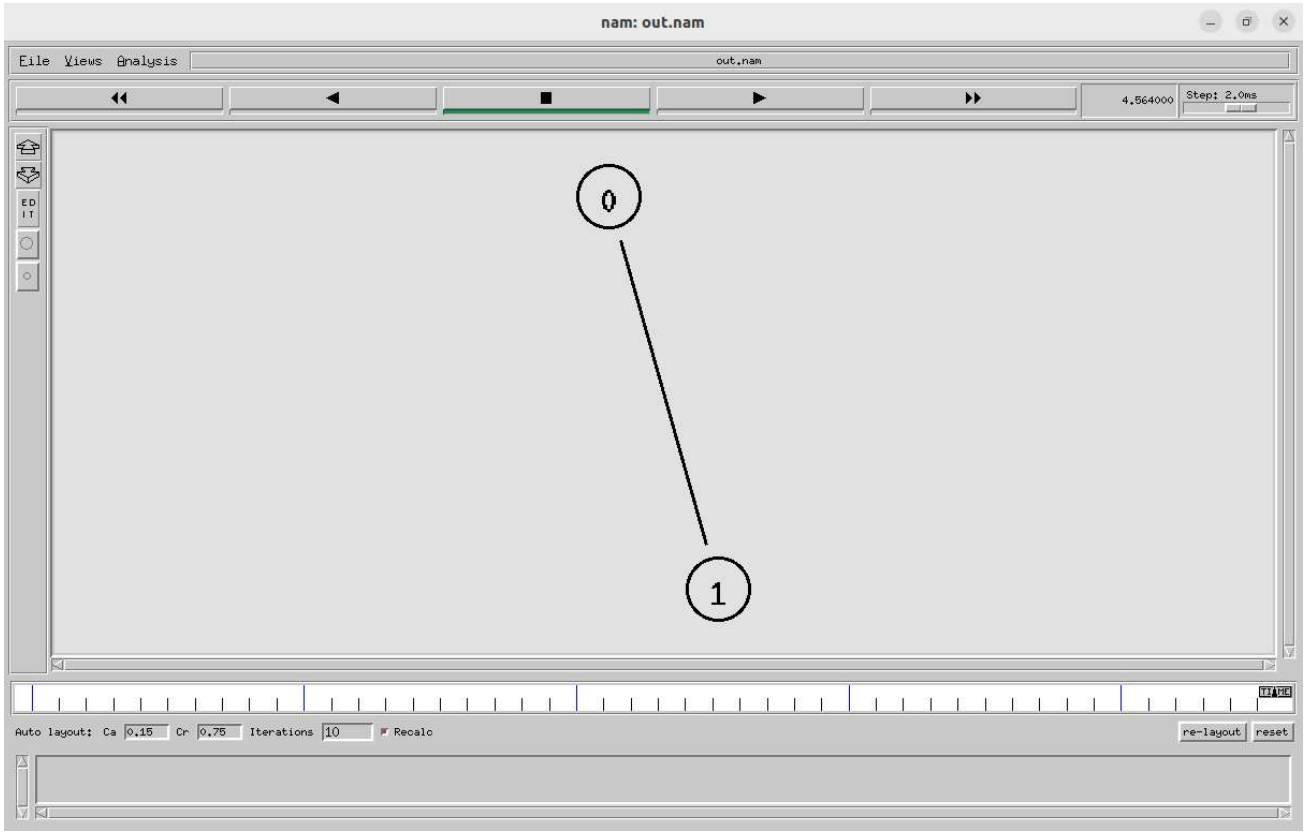
1.We've developed a simulation featuring two interconnected nodes with a simple duplex connection. At 0 seconds, there is no packet flow, as specified in the code.



2.At time interval 0.5ms ,flow of packets starts from node 0 to node 1 and the flow is continue till 4.5 sec as shown in the figure below

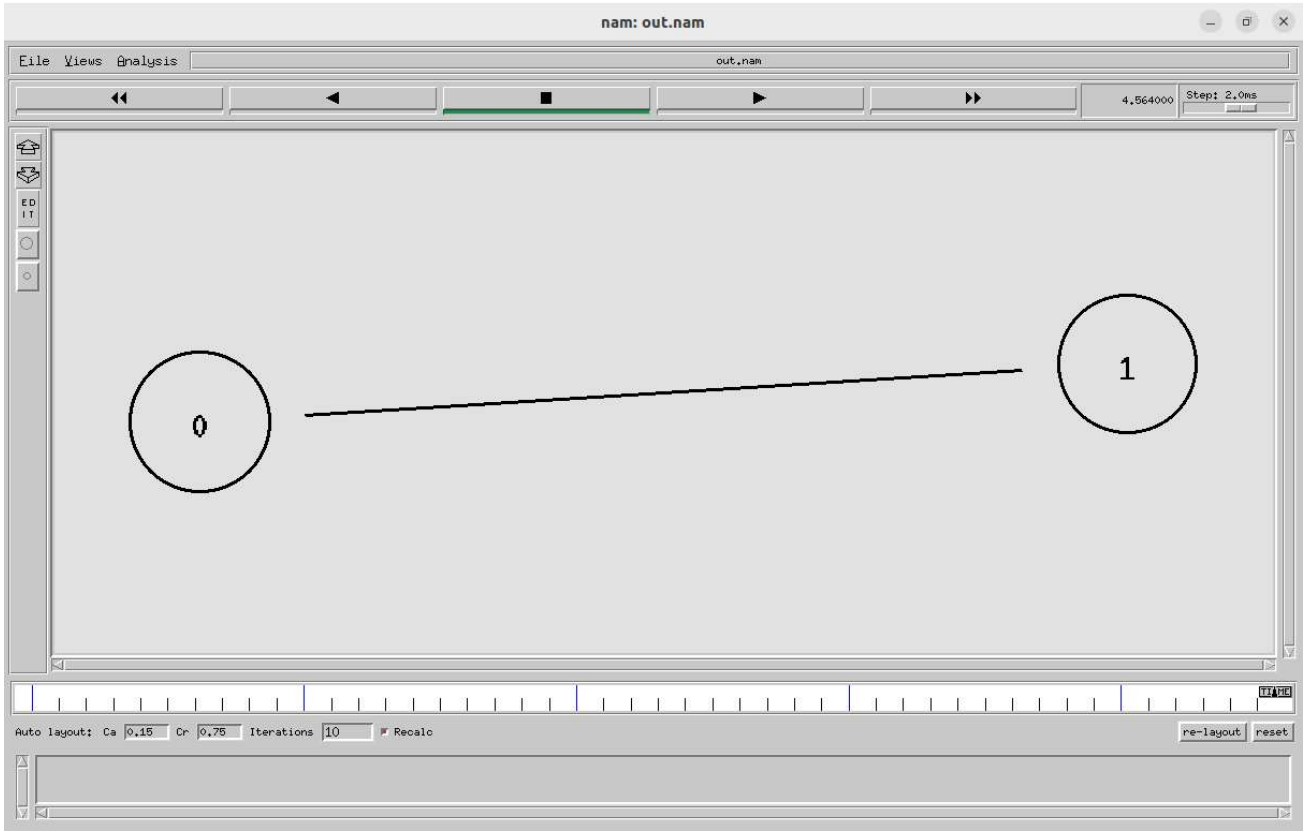


After 4.5 seconds the flow of packets stops as it calls the finish procedure and no flow of the packets are there between the two nodes.



POST-EXPERIMENT

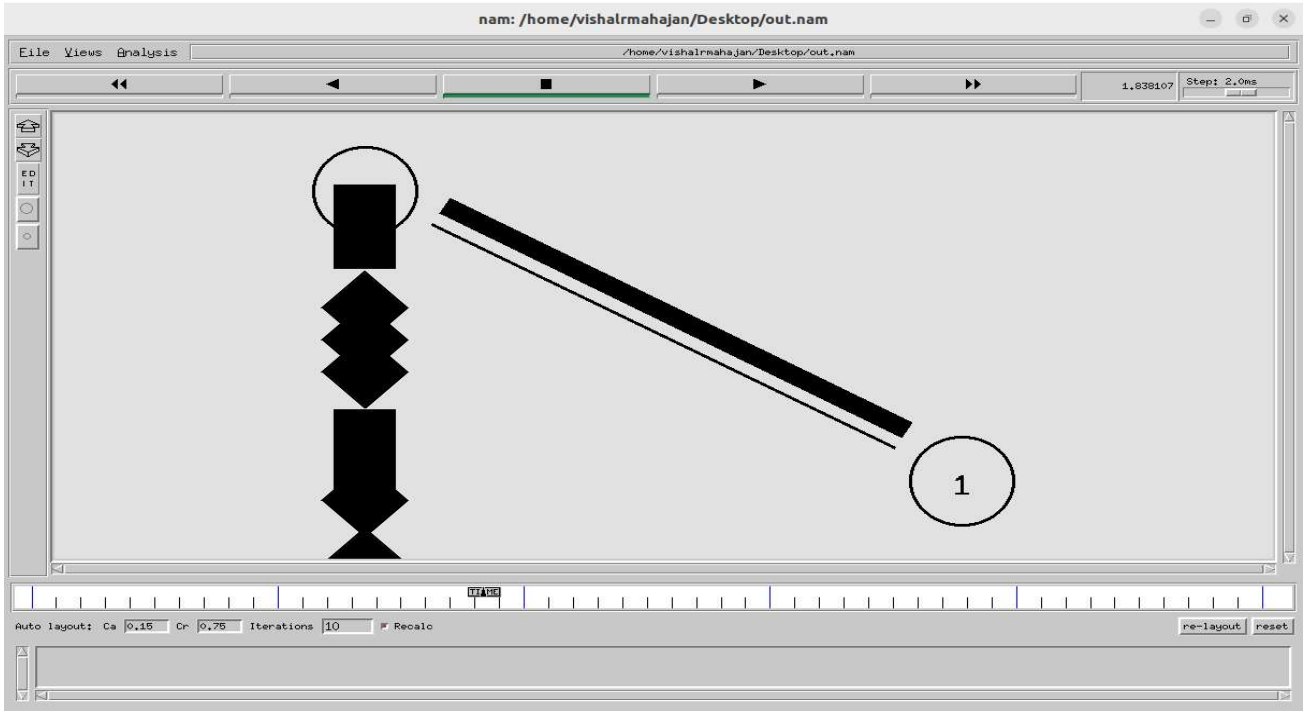
Q.1] Change the Orientation of the nodes.



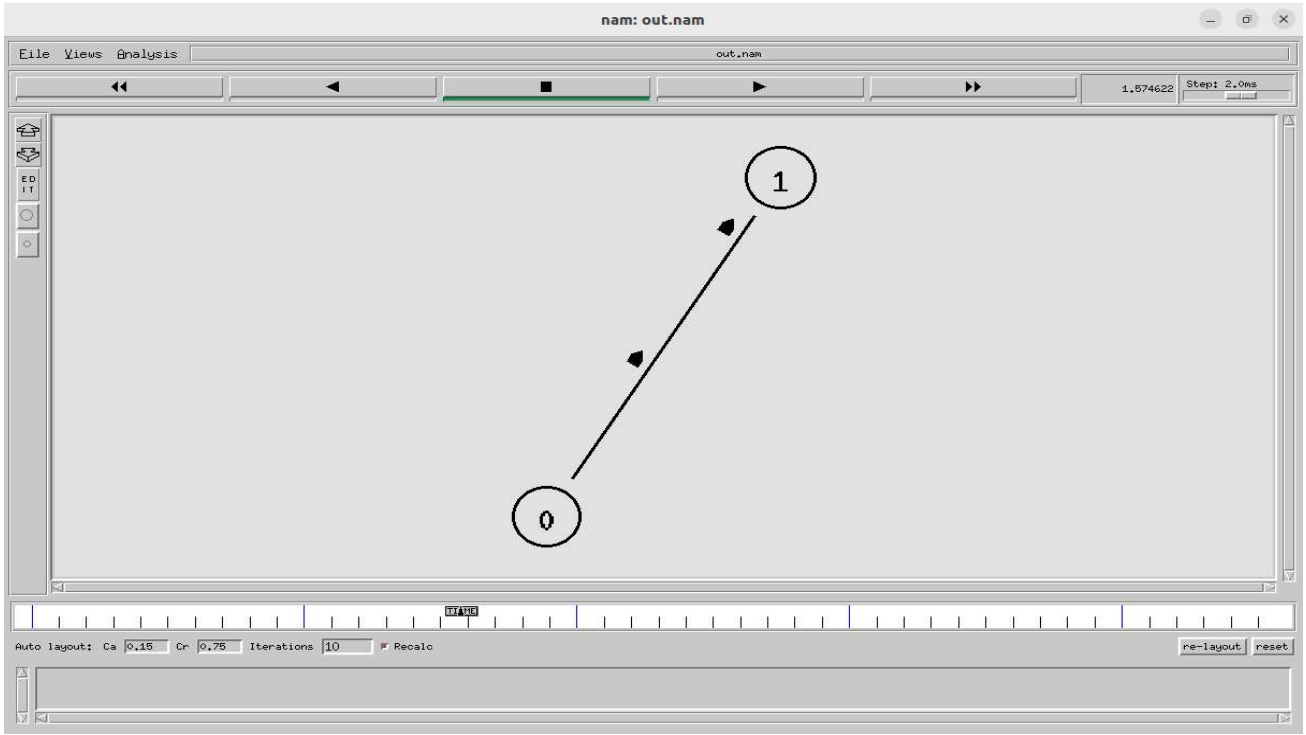
Using the re-layout and reset button we can change the layout and orientation of the nodes in the simulation. The new layout is random. To set a specific layout we need to write that in the code.

Q.2] Change the bandwidth parameter and check the effect in output.

Change the bandwidth parameter and check the effect in output. From 1Mb to BW 0.2Mb : **$ns simplex-link n0 $n1 0.2Mb 10ms DropTail** decreasing bandwidth resulting in decreased throughput and increased packet loss.

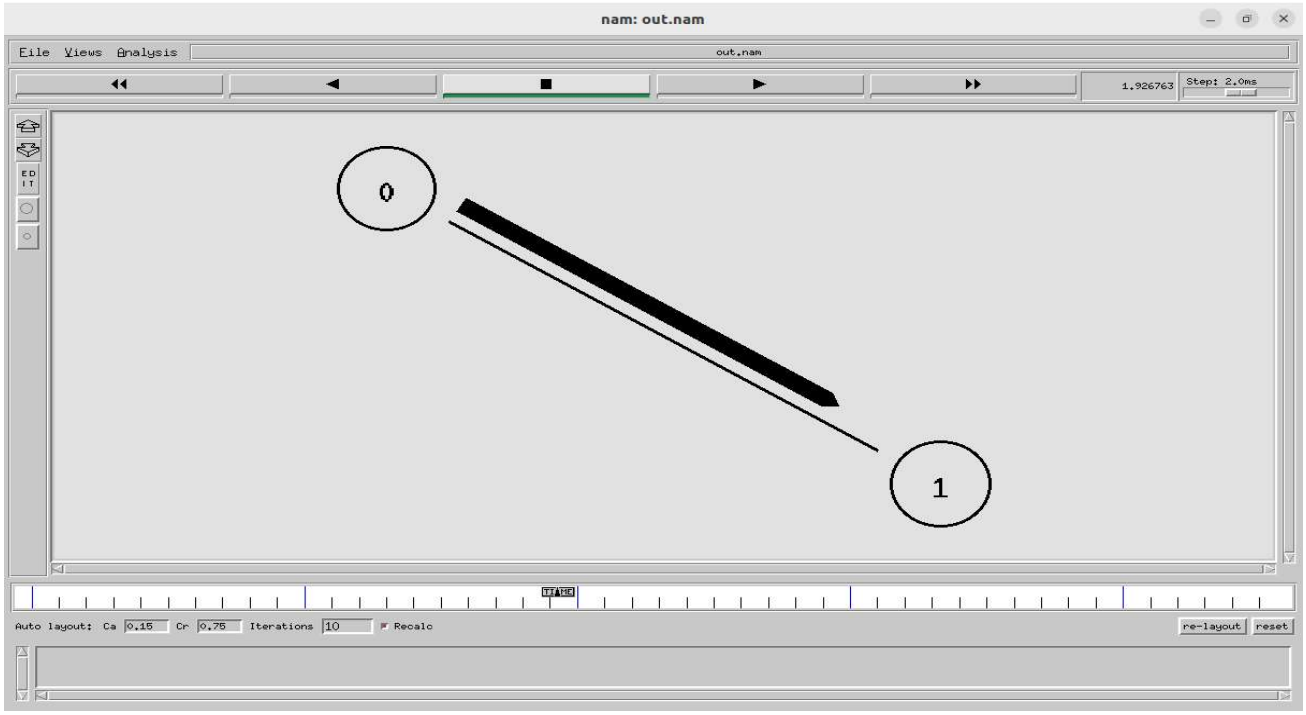


When Bandwidth is 7Mb From 1Mb to BW 7Mb : **$ns simplex-link n0 $n1 7Mb 10ms DropTail** increasing bandwidth resulting in increased throughput and decreased packet loss

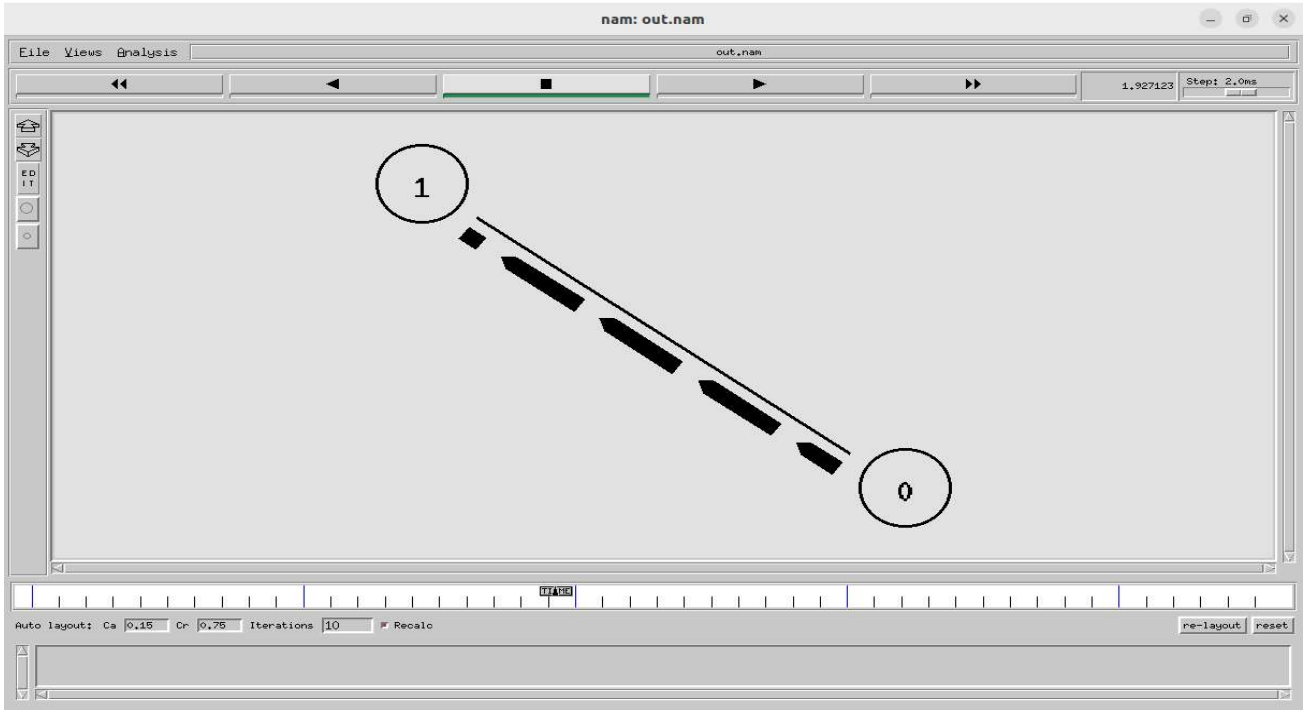


Q.3] Change the delay parameter and check the output.

From 10ms to 2ms Delay : **$ns simplex-link $n0 $n1 1Mb 2ms DropTail**. decreased delay may decrease end-to-end delay and increase throughput



From 10ms to 19ms Delay : **$ns simplex-link $n0 $n1 1Mb 19ms DropTail**. Increased delay may increase end-to-end delay and decrease throughput

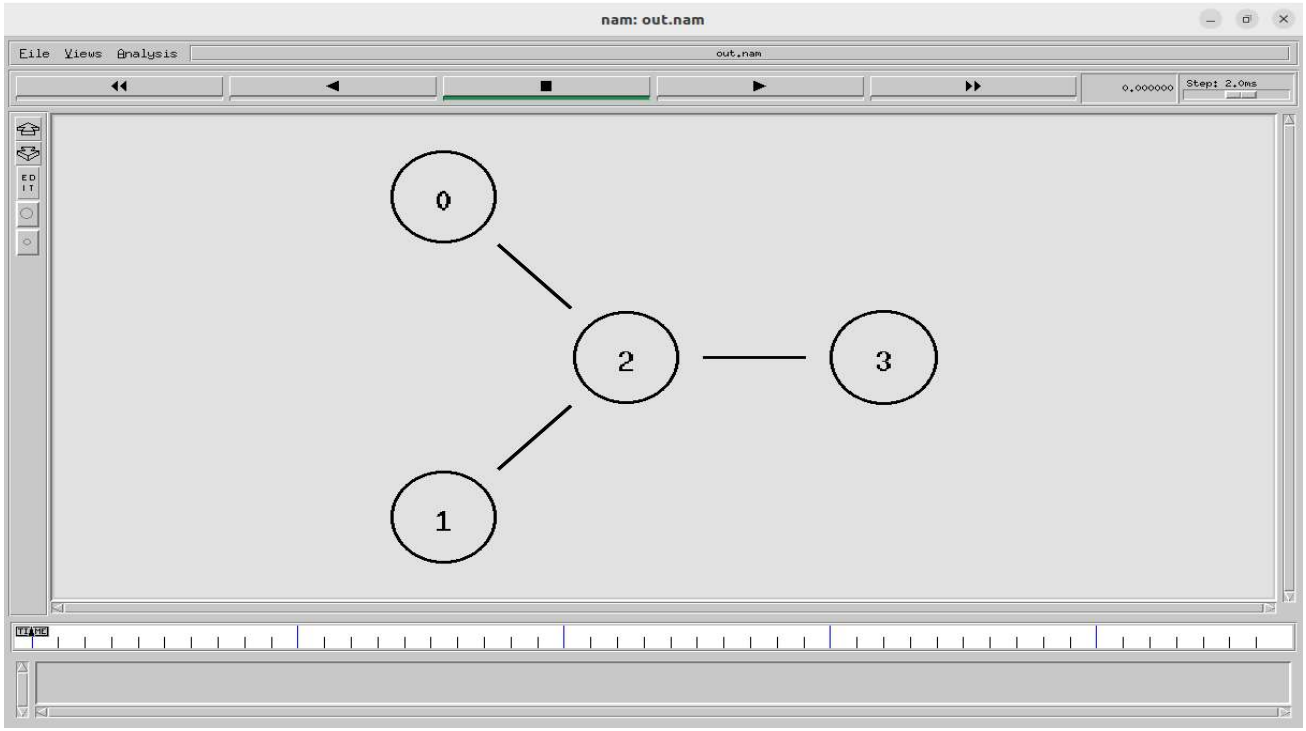


Code to simulate a simple duplex connection between two nodes:

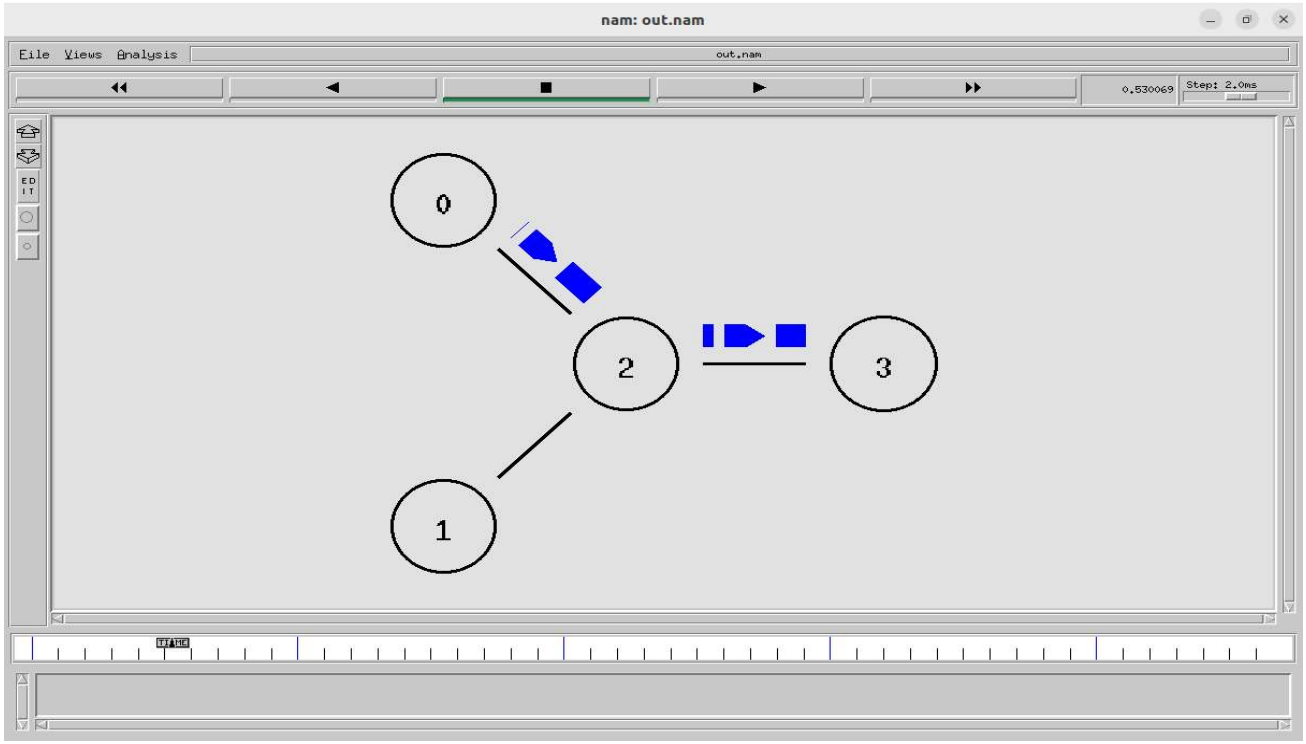
```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows
$ns color 1 darkmagenta
$ns color 2 Orange

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

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

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

```
#Create links between the nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail

$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 the link between node 2 and node 3
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 set class_ 1
$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
```

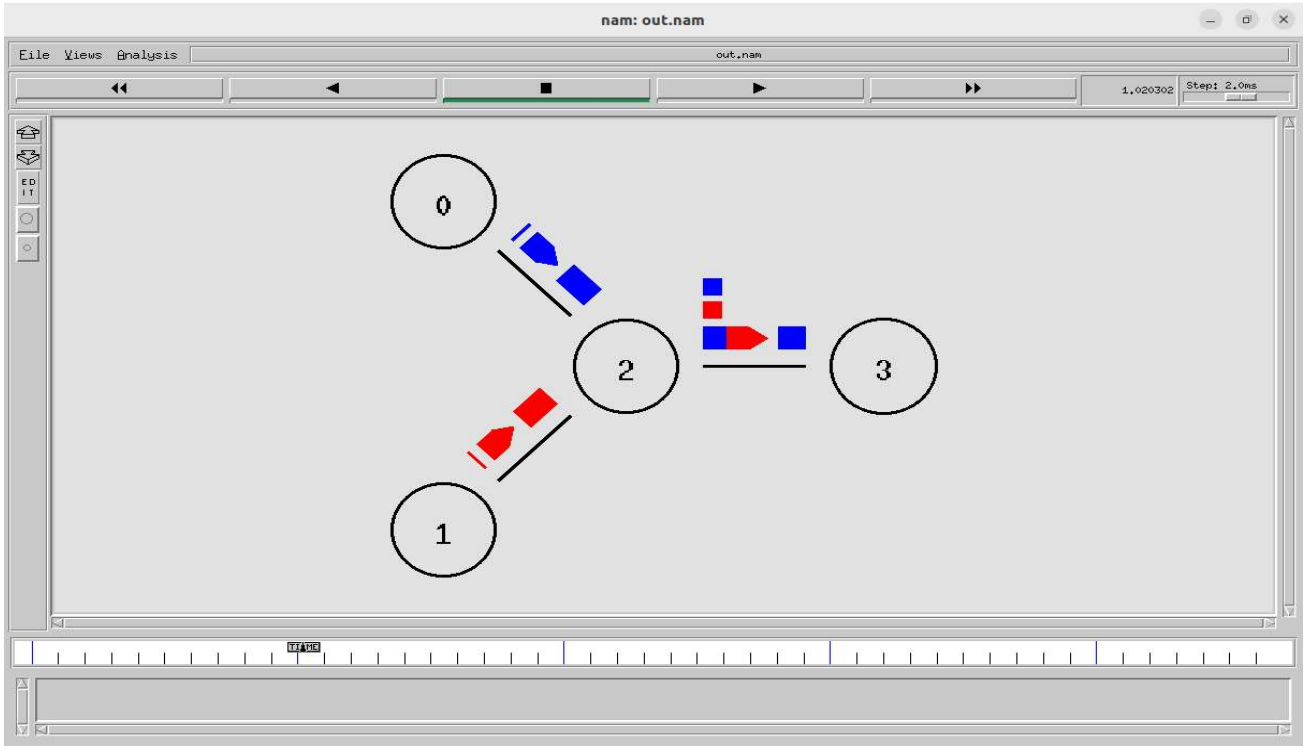
```
# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#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
```



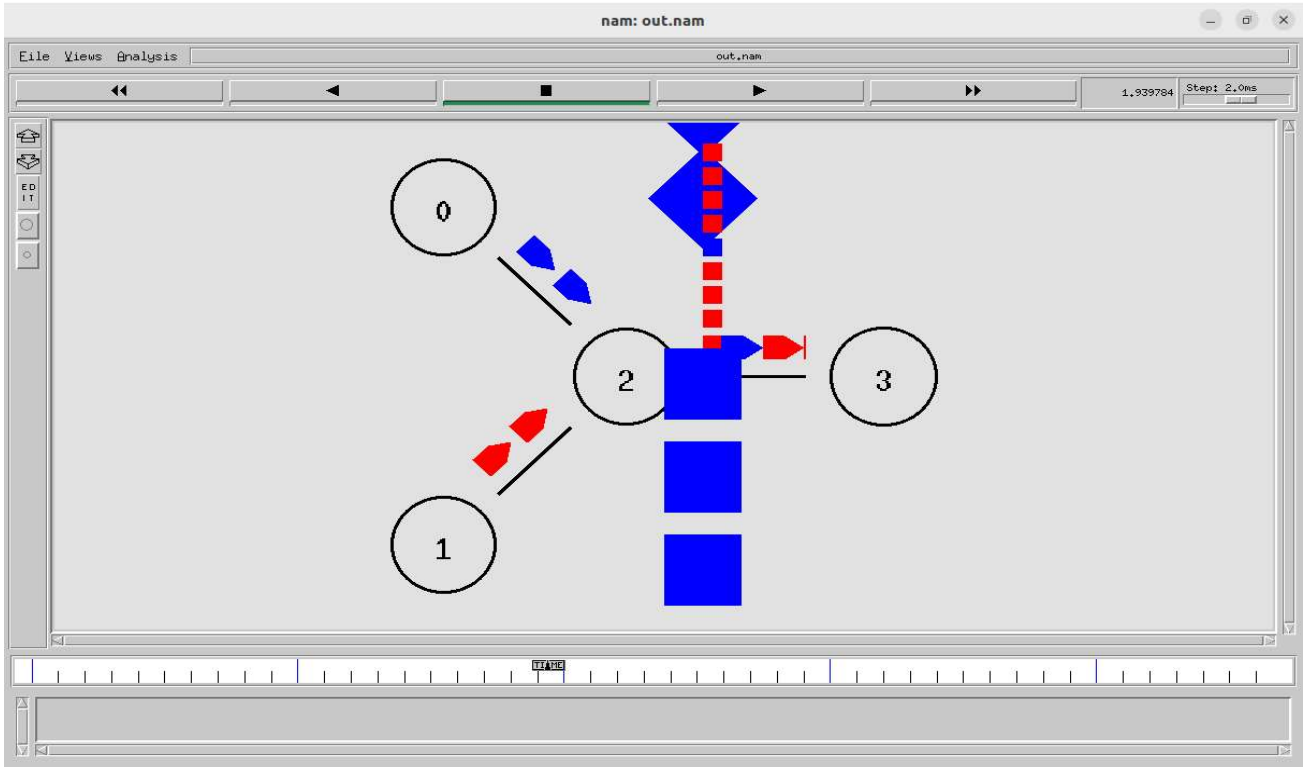
We created a star topology connection using nodes 0,1,2 and 3. node 0 and 1 are the sender nodes and node 3 is the receiver node (traffic sink).



The node 0 starts sending packets at 0.5sec. The data passes from node 0 through node 2 and reaches the receiver node 3

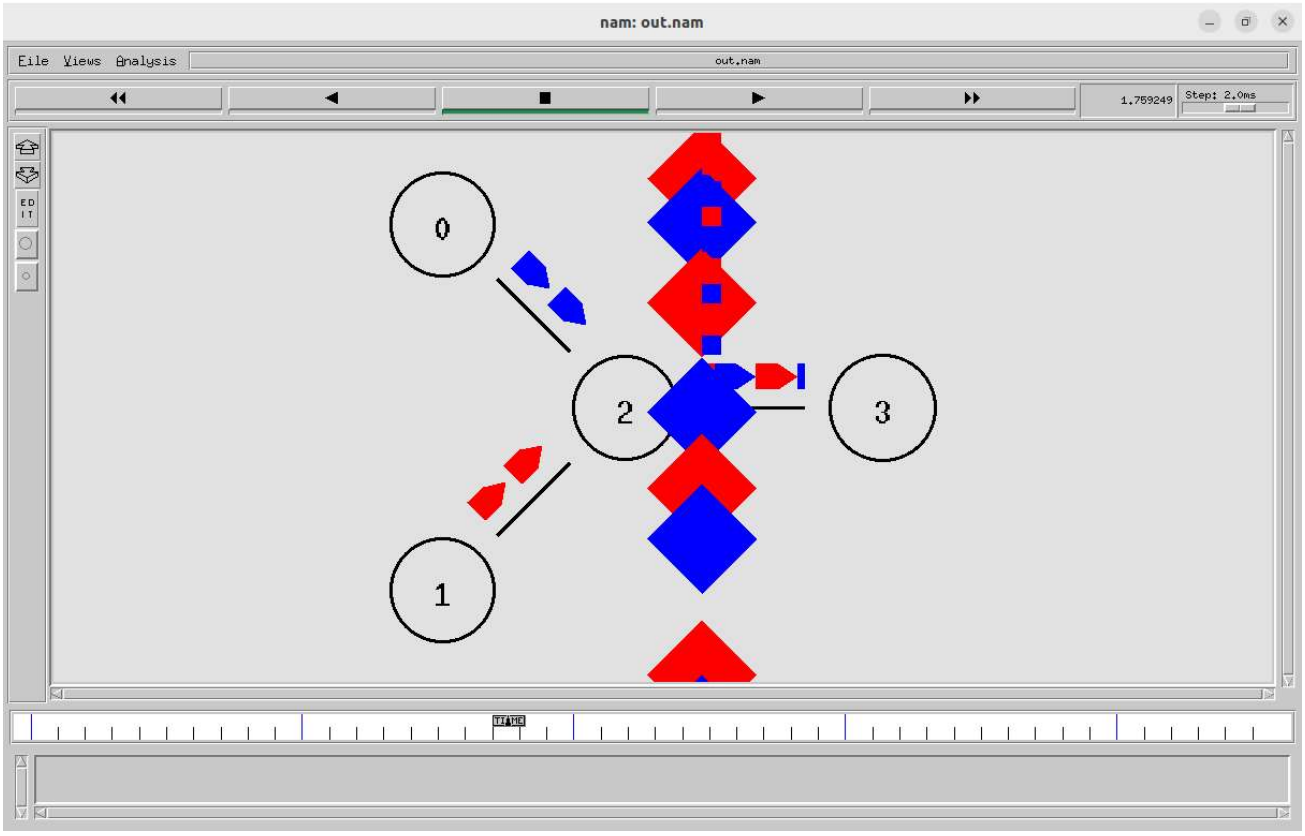


Node 1 started sending packets at 1 sec. The excess data is getting stored in the queue.

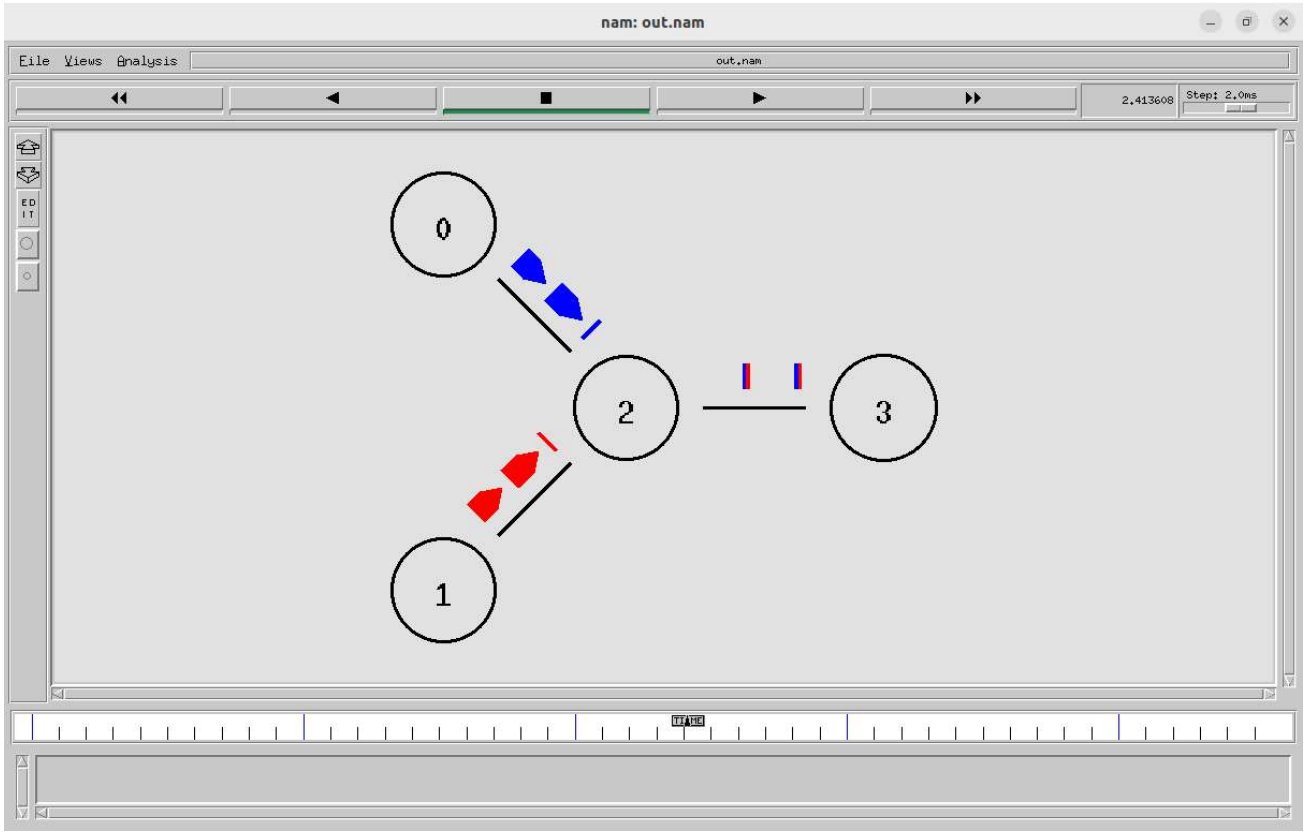


As the queue is getting full the excess data is getting dropped out of the queue. Only the blue nodes getting overflowed as the Droptail queue mechanism is unfair.

```
Open        *cn2.4.tcl                    Save
            ~/Desktop
28
29 #Create links between the nodes
30 $ns duplex-link $n0 $n2 1Mb 10ms SFQ
31 $ns duplex-link $n1 $n2 1Mb 10ms SFQ
32 $ns duplex-link $n3 $n2 1Mb 10ms SFQ
Tcl    Tab Width: 8    Ln 32, Col 37    INS
```



Changed bandwidth to 1mb and the Queue to SFQ (stochastic fair queueing). This particular queue is providing fair chance to drop



When the bandwidth of link 2 to 3 is increased we see no data getting dropped out as there is enough bandwidth to transmit the packets.

POST EXPERIMENT:

In your simulation, change the color of the packets moving from node 1 to 2 and 2 to 3 and observe the output. (take a screenshot and attach as the output).

the color of packets are orange and darkmagenta