-: TABLE OF CONTENTS :-

Part – A Introduction

Introduction to NS-2:

NS-2 stands for Network Simulator Version 2. It is an open-source, event-driven simulator designed specifically for research in computer communication networks. NS-2 is one of the most widely used network simulation tools for studying the performance of network protocols, especially congestion control algorithms in TCP.

Although NS-2 was originally designed to evaluate TCP congestion control, over time it gained wide acceptance in academia and industry. It now supports the simulation of latest wired and wireless networking protocols such as TCP, UDP, Routing algorithms, FTP, and paradigms such as Mobile Ad hoc Networks (MANETs) and Vehicular Ad hoc Networks (VANETs).

A new simulator named ns-3 has gained popularity recently. It is not a sequel of NS-2.
Their APIs are not compatible, and both simulators are completely different tools.

Features of NS-2:-

- Discrete event simulator for networking research
- Supports TCP, FTP, UDP, DSR, and many other protocols
- Supports simulation of wired and wireless networks
- Primarily UNIX-based
- Uses TCL as scripting language
- Provides OTcl (Object-oriented Tcl), TclCL, and C++ linkage
- Includes a discrete event scheduler

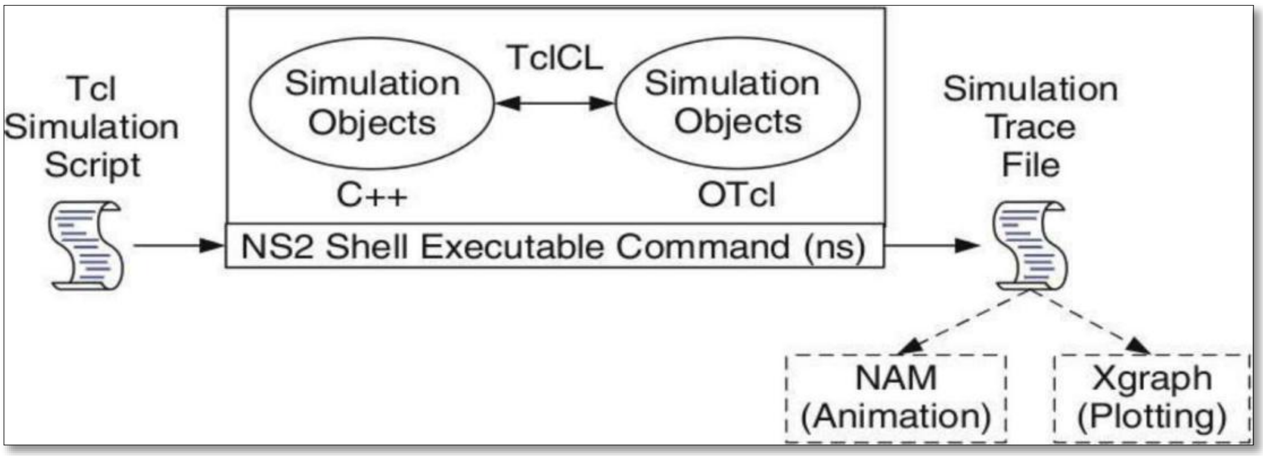


Figure 1: Basic Architecture of Network Simulator

Why Two Languages? (TCL & C++):-

NS-2 consists of two key languages:
1. C++ (Backend)
- ➢ Defines internal mechanisms
- ➢ Fast to run but slow to modify

2. OTcl (Frontend)
- ➢ Used to configure objects and schedule events
- ➢ Slow to run but fast to change

Both languages are linked using **TclCL**.

A typical NS-2 simulation includes writing a `.tcl` script (e.g., `myfirst_ns.tcl`) specifying:
- ➢ Nodes
- ➢ Links
- ➢ Applications
- ➢ Traffic generation
- ➢ Simulation time

Execution:-
ns myfirst_ns.tcl

C++ ensures efficient object creation, while OTcl is used to configure and schedule events.

Tcl Scripting:-
- ➢ Tcl is a simple, general-purpose scripting language used in NS-2.
- ➢ It runs on Unix, Windows, and Mac.
- ➢ Variables do not require prior data type declarations.

Structure of an NS-2 Program:-

1. Creating a Simulator Object
2. Setting up Trace & NAM files
3. Tracing using Trace/NAM commands
4. Closing files and launching NAM
5. Creating Node & Link Topology
6. Orientation and queue settings
7. Scheduling events

Working of NS-2:-

Users run simulations using the executable command: ns .tcl

The simulation generates:
- o Trace file (for analysis)
- o NAM file (for animation)

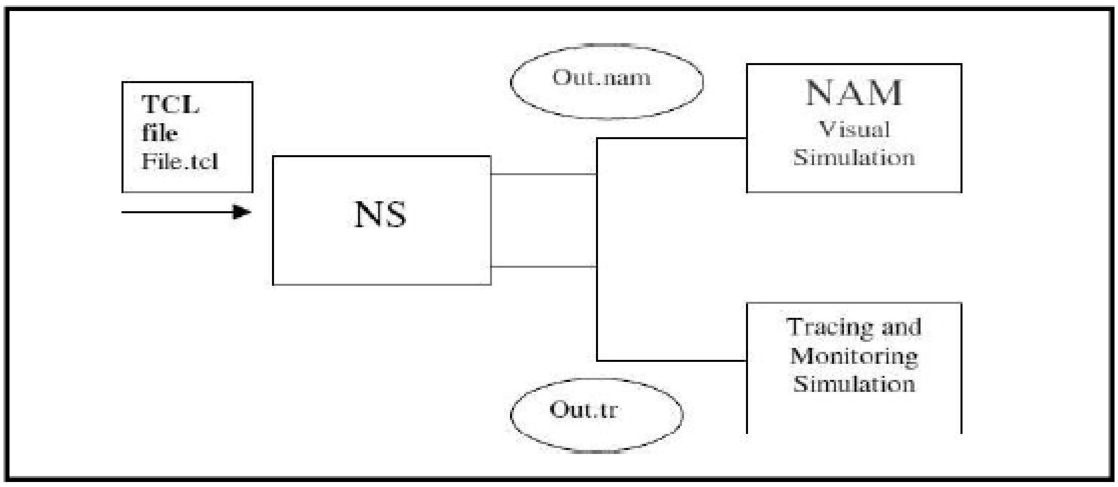


Figure 2: Working of Network Simulator 2

Trace File and NAM File

- Once the simulation is complete, we can see two files: “trace.tr” and “nam.out”.
- The trace file (trace.tr) is a standard format used by ns2.
- In ns2, each time a packet moves from one node to another, or onto a link, or into a buffer, etc., it gets recorded in this trace file.
- Each row represents one of these events and each column has its own meaning.
- Start nam with the command 'nam <nam-file>' where '<nam-file>' is the name of a nam trace file that was generated by ns.

To view NAM:
nam

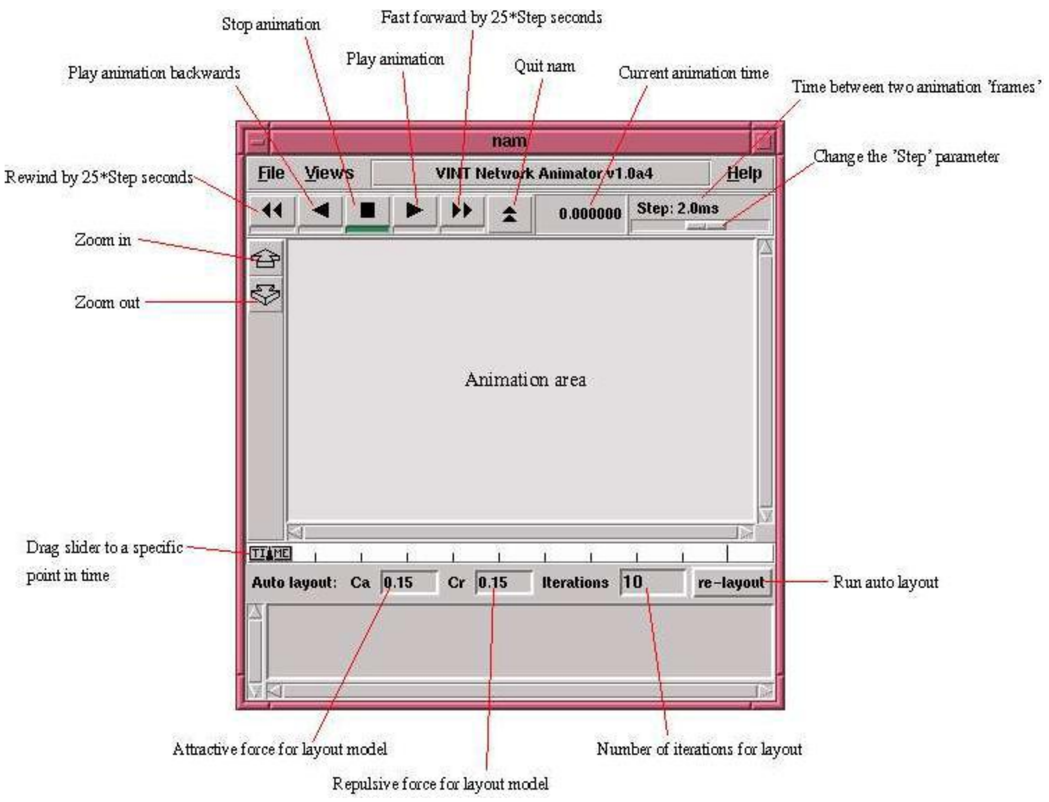

Stop animation
Fast forward by 25*Step seconds
Play animation
Quit nam
Current animation time
Time between two animation 'frames'
Play animation backwards
Change the 'Step' parameter
Rewind by 25*Step seconds
Zoom in
Zoom out
Animation area
Drag slider to a specific point in time
Run auto layout
Attractive force for layout model
Repulsive force for layout model
Number of iterations for layout
nam
File
Views
VINT Network Animator v1.0a4
Help
0.000000
Step: 2.0ms
TIME
Auto layout: Ca 0.15 Cr 0.15 Iterations 10 re-layout

Figure 3: Details of NAM Window

Advantages of NS-2:-

1. Open source
2. Complex scenarios are easy to test
3. Faster result generation
4. Supports many protocols
5. Cross-platform support
6. Modular design

Disadvantages of NS-2

1. Limited for very large networks
2. Slower compared to real-time execution
3. Does not fully reflect real-world scenarios
4. Possible statistical uncertainty

Steps to Create a Scenario File:-

Step 1: Declare Simulator and Set Output Files
Step 2: Setting Node and Link
Step 3: Setting Agents
Step 4: Setting Application
Step 5: Setting Simulation Time and Schedules
Step 6: Declare Finish

Step 1: Declare Simulator and setting

$ns [new Simulator]	#first line of tcl script. Creates ns object
get file [open out.tr w] $ns trace-all $file	#open the trace file
get namfile [open out.nam w] $ns namtrace-all $namfile	#open the nam file

.

Step 2: Setting Node and Link

$ n0 [$ns node]	setting a node
ns duplex-link $n0 $n2 3Mb 5ms DropTail	#bidirectional link between n0 and n2 is declared bandwidth 3Mbps and delay 5ms. DropTail is a waiting queue type.
$ns duplex-link-op $n0 $n2 orient right-down	#Sets positions of node and link for Nam. It does not affect to the result of simulation
$ns queue-limit $n2 $n3 20	#The length of queue on the link from n2 to n3 is 20[packets].
$ns duplex-link-op $n2 $n3 queuePos 0.5	#The position of queue is set for Nam, 0.5 is the angle between link and queue, it equals to (0.5_).

Step 3: Setting Agent

UDP Agent :To use UDP in simulation, the sender sets the Agent as UDP Agent while the receiver sets to Null Agent. Null Agents do nothing except receiving the packets.

set udp [new Agent/UDP] $ns attach-agent $n0 $udp set null [new Agent/Null] $ns attach-agent $n3 $null	#udp and null Agent are set for n0 and n3, respectively
$ns connect $udp $null	Declares the transmission between udp and null.
$udp set fid_ 0	Sets the number for data flow of udp. This number will be recorded to all packets which are sent from udp
$ns color 0 blue	Mark the color to discrete packet for showing result on NAM.

TCP Agent: To use TCP in simulation, the sender sets the Agent as TCP Agent while the receiver sets to TCPSink Agent. When receiving a packet, TCPSink Agent will reply an acknowledgment packet (ACK). Setting Agent for TCP is similar to UDP.

set tcp [new Agent/TCP] $ns attach-agent $n1 $tcp set sink [new Agent/TCPSink] $ns attach-agent $n3 $sink	# tcp and sink Agent are set for n1 and n3, respectively
$ns connect $tcp $sink	#declares the transmission between tcp and sink.
$tcp set fid_ 1	#sets the number for data flow of tcp. This number will be recorded to all packet which are sent from tcp
$ns color 1 red	#mark the color to discrete packet for showing result on Nam.

Step 4: Setting Application

In general, UDP Agent uses CBR Application while TCP Agent uses FTP Application.

set cbr [new Application/Traffic/CBR] $cbr attach-agent $udp
set ftp [new Application/FTP] $ftp attach-agent $tcp

Step 5: Setting time schedule for simulation

Time schedule of a simulation is set as below:

$ns at 1.0 "$cbr start" $ns at 3.5 "$cbr stop"	cbr transmits data from 1.0[sec] to 3.5[sec]
$ns at 1.5 "$ftp start" $ns at 3.0 "$ftp stop"	ftp transmits data from 1.5[sec] to 3.0[sec].

Step 6: Declare finish

After finish setting, declaration of finish is written at the end of file.

$ns at 4.0 "finish"	

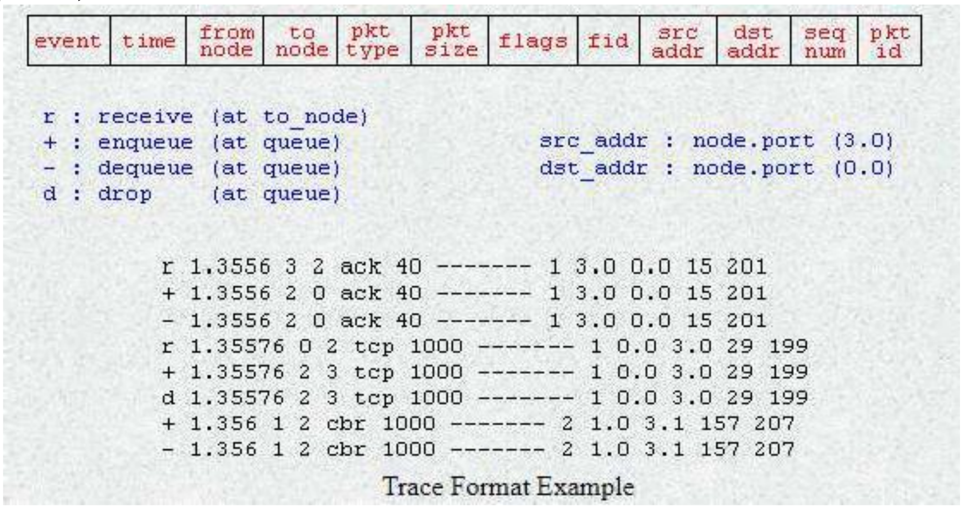
proc finish {} {
global ns file namfile tcpfile
$ns flush-trace
close $file
close $namfile
close $tcpfile
exit 0
}
	The finish function is used to output data file at the end of simulation.

Execute Simulation and start Nam:-

By executing below command line, simulation will be started and shows the animation of simulation.

```
ns sample.tcl
nam out.nam
```

View trace file (out.tr):-



event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)              src_addr : node.port (3.0)
- : dequeue (at queue)              dst_addr : node.port (0.0)
d : drop    (at queue)

        r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
        + 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
        - 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
        r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
        + 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
        d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
        + 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
        - 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Trace Format Example

Figure 4: Details of Trace Window

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of "node:port".
10. This is the destination address, given in the same form.
11. This is the transport layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes
12. The last field shows the Unique id of the packet.

AWK File Overview:-

- The basic function of awk is to search files for lines (or other units of text) that contain certain patterns. When a line matches one of the patterns, awk performs specified actions on that line. awk keeps processing input lines in this way until the end of the input files are reached.

- Programs in awk are different from programs in most other languages, because awk programs are data-driven; that is, we describe the data to work with, and then what to do when we find it. Most other languages are procedural. When working with procedural languages, it is usually much harder to clearly describe the data of our program will process.

- For this reason, awk programs are often refreshingly easy to both write and read. When we run awk, we can specify an awk program that tells awk what to do. The program consists of a series of rules. (It may also contain function definitions, an advanced feature which we will ignore for now.)

- Each rule specifies one pattern to search for, and one action to perform when that pattern is found). Syntactically, a rule consists of a pattern followed by an action. The action is enclosed in curly braces to separate it from the pattern. Rules are usually separated by newlines. Therefore, an awk program looks like this:

```
pattern { action }                              pattern { action }
```

- Since we are dealing with column oriented data, AWK is probably the easiest tool we can use to format our data. AWK is a simple scripting language that scans through a file line by line. It allows to access any column in the current line by using special variables $1, $2, $3, etc. for the first, second and third columns. The definition of each column of the trace file is shown above, so we can use the AWK script to check the value of each column and collect the data we need.

- The BEGIN and END sections are only executed once (before and after the file has been processed). The middle section is executed for each line of the file. The AWK script keeps three variables to store the throughput in Mb/s for flow 1, flow 2, and the total. For each line of the trace file, it checks if an TCP packet (SS == “tcp”) is received (“r”) at node 3 (since n3 is our destination node). If so, it increments the count for the appropriate flow, using the size of that particular packet (in bytes) from column 6. After each second, it prints its total while converting bytes to Mb.

```
BEGIN { print "START" }
{ print }
END { print "STOP" }
```

To run AWK script:
awk -f <filename.awk> <inputfile> > <outputfile>

<u>**XGRAPH:**</u> A tool for plotting output data.

- Plotting purposes.
- Comes together with NS2 installation package.
- Running graph

Usage:
xgraph <file1> <file2> -bg <color> -t <title> -x <xlabel> -y <ylabel>

NS Simulation Example Script:-

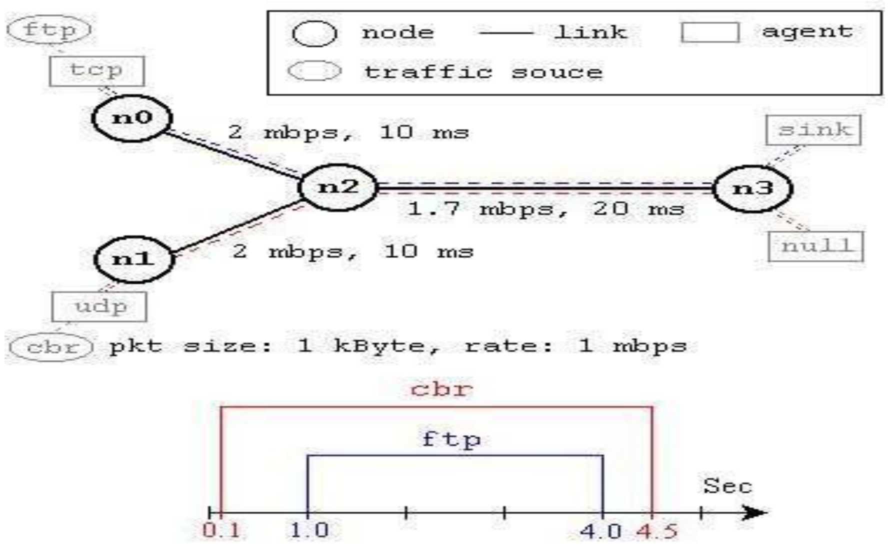


Figure 5: Example Network Topology

This network consists of 4 nodes (n0, n1, n2, n3) as shown in **Fig. 5: Network Topology**. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. Each node uses a DropTail queue, of which the maximum size is 10. A "tcp" agent is attached to n0, and a connection is established to a tcp "sink" agent attached to n3. As default, the maximum size of a packet that a "tcp" agent can generate is 1KByte.

A tcp "sink" agent generates and sends ACK packets to the sender (tcp agent) and frees the received packets. A "udp" agent that is attached to n1 is connected to a "null" agent attached to n3. A "null" agent just frees the packets received. A "ftp" and a "cbr" traffic generator are attached to "tcp" and "udp" agents respectively, and the "cbr" is configured to generate 1KByte packets at the rate of 1 Mbps.

The "cbr" is set to start at 0.1 sec and stop at 4.5 sec, and "ftp" is set to start at 1.0 sec and stop at 4.0 sec.

Explanation:

- Nodes: n0, n1, n2, n3
- Links: 2 Mbps, 10 ms (n0–n2, n1–n2)
- Link: 1.7 Mbps, 20 ms (n2–n3)
- Queue limit between n2 and n3: 10

An Example Script

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

#Define colors for flows
$ns color 1 Blue
$ns color 2 Red

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

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

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

#Create links
$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

#Queue settings
$ns queue-limit $n2 $n3 10

#Orientation 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

#Queue position for NAM
$ns duplex-link-op $n2 $n3 queuePos 0.5

#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

#FTP application
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#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

#CBR application
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

#Scheduling events
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
$ns at 5.0 "finish"

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

#Run simulation
$ns run
```

Program No. 1 :-

Implement 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 Objective:- To understand the implementation of duplex links in a network and analyze packet drops by varying bandwidth and queue size.

Theory

1. Create a simulator object.
2. Open the trace file for storing simulation data.
3. Create nodes in the simulator.
4. Attach agents (UDP/Null) to the nodes.
5. Attach CBR applications over the UDP agents.
6. Connect the agents for communication.
7. Set up queue size, bandwidth, and delay.
8. Run the simulation and generate the trace file.
9. Use an AWK script to count packet drops.
10. View results in NAM and trace files.

TCL Program: "lab1.tcl"

```
# Create Simulator
set ns [new Simulator]

# Open trace & NAM files
set tf [open lab1.tr w]
$ns trace-all $tf
set nf [open lab1.nam w]
$ns namtrace-all $nf

# Create Nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# Set colors
$ns color 1 "red"
$ns color 2 "blue"

# Label nodes
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"

# Create duplex links (vary bandwidth here)
$ns duplex-link $n0 $n2 10Mb 300ms DropTail
$ns duplex-link $n1 $n2 10Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail

# Set Queue Sizes
$ns queue-limit $n0 $n2 10
```

```
$ns queue-limit $n1 $n2 10
$ns queue-limit $n2 $n3 5
# Attach UDP Agents
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set null [new Agent/Null]
$ns attach-agent $n3 $null

# Attach CBR Applications
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

# Set packet color
$udp0 set class_ 1
$udp1 set class_ 2

# Connect agents
$ns connect $udp0 $null
$ns connect $udp1 $null

# Packet size (corrected)
$cbr1 set packetSize_ 500

# Set interval (data rate)
$cbr1 set interval_ 0.005
$cbr0 set interval_ 0.005

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

# Start traffic
$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"

# End simulation
$ns at 10.0 "finish"

$ns run
```

AWK Program: lab1.awk

```
BEGIN {
   count = 0;
}
{
   # The first field "d" indicates packet drop
   if ($1 == "d")
       count++;
}
END {
   printf("The Total No. of Packets Dropped due to Congestion : %d\n\n", count);
}
```

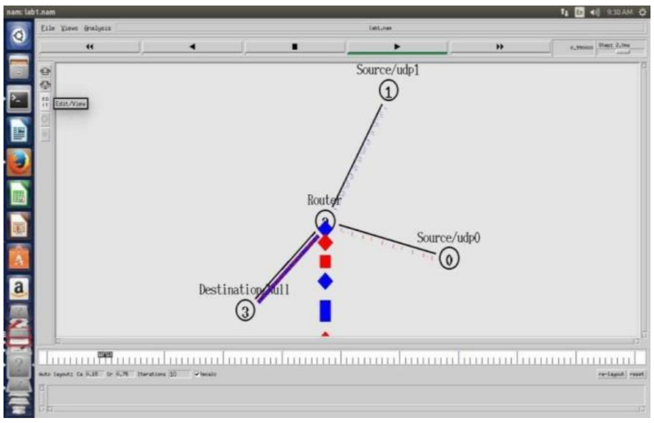
Steps for Execution

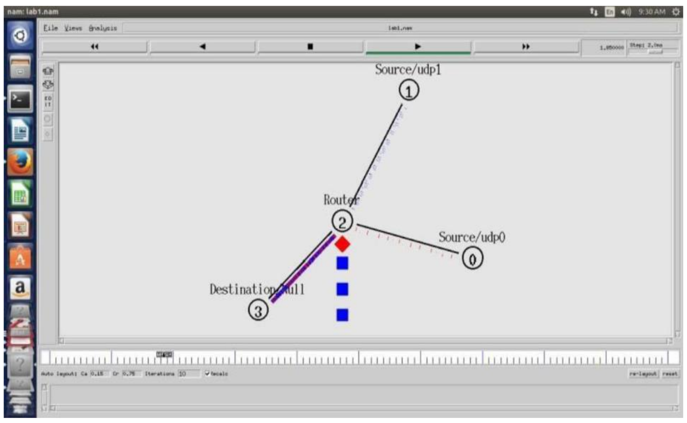
1. Open gedit and type the TCL program:
2. gedit lab1.tcl
3. Save the file.
4. Open gedit and write the AWK file:
5. gedit lab1.awk
6. Save the file.
7. Run the simulation:
8. ns lab1.tcl
9. The NAM window opens—press **Play** to view simulation.
10. After simulation, run the AWK file to count packet drops:
11. awk -f lab1.awk lab1.tr
12. To view trace file:
13. gedit lab1.tr

Output

```
The Total No. of Packets Dropped due to Congestion : 456
```

Topology:





Program No. 2 :-
Implement transmission of ping messages/trace route over a 6-node network topology and find the number of packets dropped due to congestion.

Program Objective:-
To understand the implementation of network topology with multiple nodes and analyze packet drops due to congestion using ping messages.

TCL Program: "lab2.tcl"

```
# Create Simulator
set ns [new Simulator]

# Open NAM & Trace files
set nf [open lab2.nam w]
$ns namtrace-all $nf

set nd [open lab2.tr w]
$ns trace-all $nd

# Finish procedure
proc finish {} {
   global ns nf nd
   $ns flush-trace
   close $nf
   close $nd
   exec nam lab2.nam &
   exit 0
}

# Create nodes (1 central + 6 spokes)
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]

# Create duplex links from all peripheral nodes to central node
$ns duplex-link $n1 $n0 1Mb 10ms DropTail
$ns duplex-link $n2 $n0 1Mb 10ms DropTail
$ns duplex-link $n3 $n0 1Mb 10ms DropTail
$ns duplex-link $n4 $n0 1Mb 10ms DropTail
$ns duplex-link $n5 $n0 1Mb 10ms DropTail
$ns duplex-link $n6 $n0 1Mb 10ms DropTail
```

```
# Override Ping agent receive procedure for printing RTT
Agent/Ping instproc recv {from rtt} {
   $self instvar node_
   puts "Node [$node_ id] received ping reply from $from with RTT = $rtt ms."
}

# Create Ping agents
set p1 [new Agent/Ping]
set p2 [new Agent/Ping]
set p3 [new Agent/Ping]
set p4 [new Agent/Ping]
set p5 [new Agent/Ping]
set p6 [new Agent/Ping]

# Attach Ping agents to nodes
$ns attach-agent $n1 $p1
$ns attach-agent $n2 $p2
$ns attach-agent $n3 $p3
$ns attach-agent $n4 $p4
$ns attach-agent $n5 $p5
$ns attach-agent $n6 $p6

# Set Queue limits (introducing congestion)
$ns queue-limit $n0 $n4 3
$ns queue-limit $n0 $n5 2
$ns queue-limit $n0 $n6 2

# Connect ping agents (source → destination)
$ns connect $p1 $p4
$ns connect $p2 $p5
$ns connect $p3 $p6

# Schedule ping messages
$ns at 0.2 "$p1 send"
$ns at 0.4 "$p2 send"
$ns at 0.6 "$p3 send"
$ns at 1.0 "$p4 send"
$ns at 1.2 "$p5 send"
$ns at 1.4 "$p6 send"

# End simulation
$ns at 2.0 "finish"
$ns run
```

AWK Program: lab2.awk

```
BEGIN {
   count = 0;
}
{
   event = $1;
   if (event == "d")   # packet drop symbol
       count++;
}
END {
   printf("No. of packets dropped : %d\n", count);
}
```

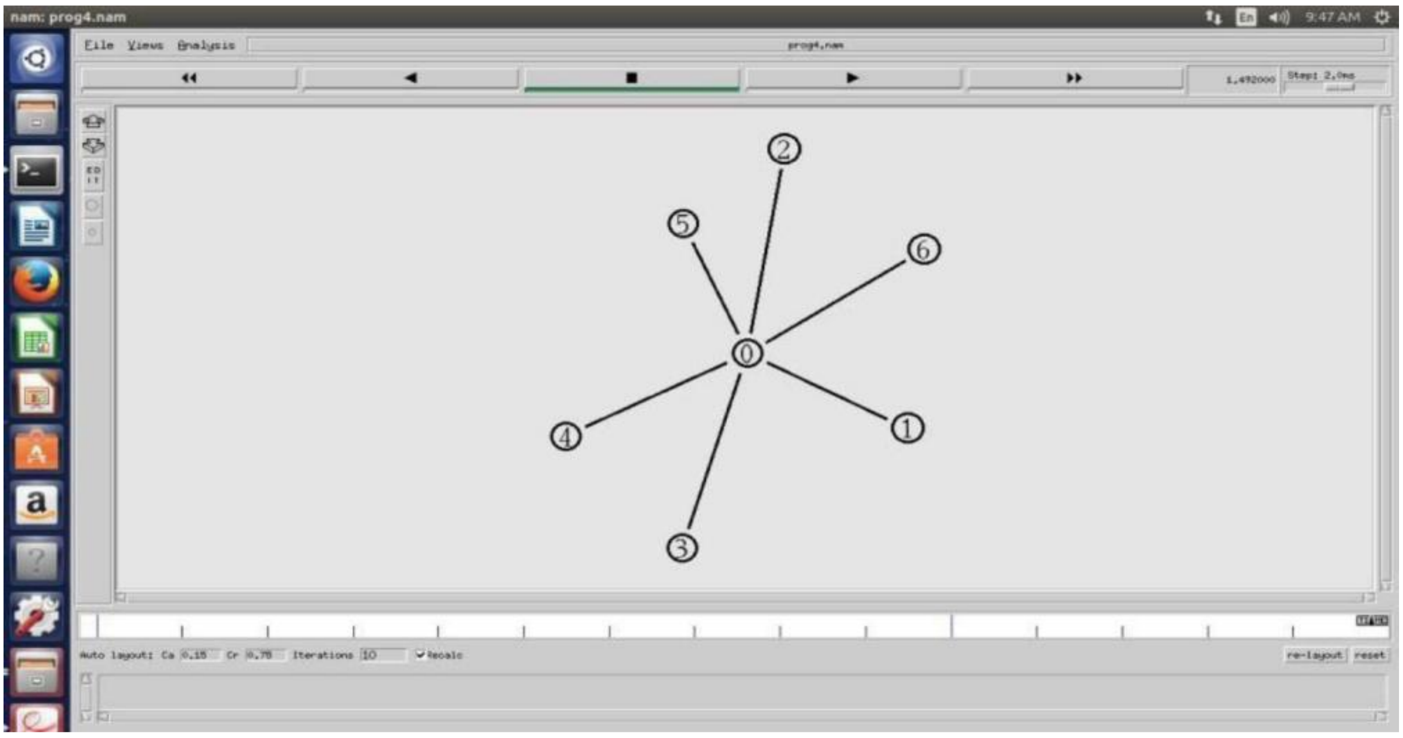
Execution Commands:-

```
[root@localhost ~]# ns lab2.tcl
[root@localhost ~]# awk -f lab2.awk lab2.tr
```

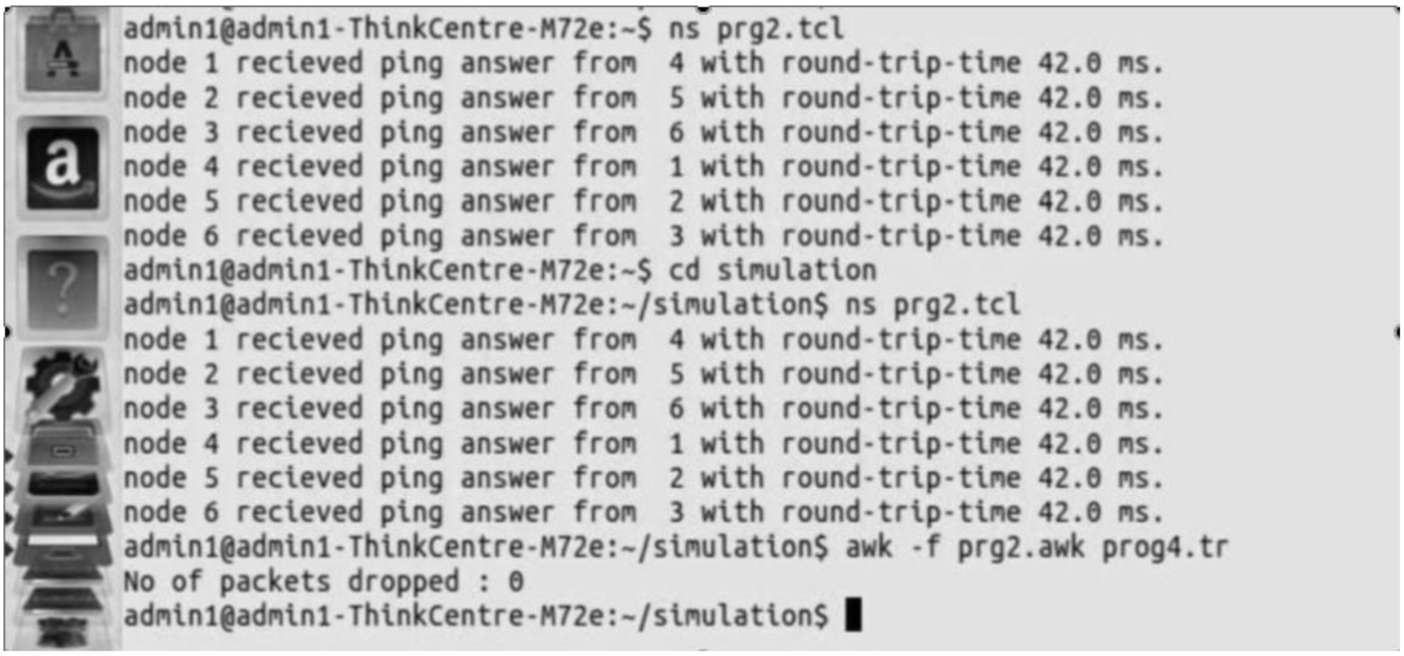
Output:-

```
The Total no. of packets dropped due to congestion: 6
```

Snapshot 1:-



Snapshot 2:-



```
admin1@admin1-ThinkCentre-M72e:~$ ns prg2.tcl
node 1 recieved ping answer from  4 with round-trip-time 42.0 ms.
node 2 recieved ping answer from  5 with round-trip-time 42.0 ms.
node 3 recieved ping answer from  6 with round-trip-time 42.0 ms.
node 4 recieved ping answer from  1 with round-trip-time 42.0 ms.
node 5 recieved ping answer from  2 with round-trip-time 42.0 ms.
node 6 recieved ping answer from  3 with round-trip-time 42.0 ms.
admin1@admin1-ThinkCentre-M72e:~$ cd simulation
admin1@admin1-ThinkCentre-M72e:~/simulation$ ns prg2.tcl
node 1 recieved ping answer from  4 with round-trip-time 42.0 ms.
node 2 recieved ping answer from  5 with round-trip-time 42.0 ms.
node 3 recieved ping answer from  6 with round-trip-time 42.0 ms.
node 4 recieved ping answer from  1 with round-trip-time 42.0 ms.
node 5 recieved ping answer from  2 with round-trip-time 42.0 ms.
node 6 recieved ping answer from  3 with round-trip-time 42.0 ms.
admin1@admin1-ThinkCentre-M72e:~/simulation$ awk -f prg2.awk prog4.tr
No of packets dropped : 0
admin1@admin1-ThinkCentre-M72e:~/simulation$ █
```

Program No. 3 :-

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

Program Objective:-

To understand Ethernet LAN topology creation using NS2 and analyze congestion behavior by generating TCP congestion window curves for multiple simultaneous TCP flows.

TCL Program : "lab3.tcl"

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

# Open NAM & trace files
set nf [open lab3.nam w]
$ns namtrace-all $nf

set nd [open lab3.tr w]
$ns trace-all $nd

# Colors for tcp flows
$ns color 1 Blue
$ns color 2 Red

# Finish procedure
proc finish { } {
   global ns nf nd
   $ns flush-trace
   close $nf
   close $nd
   exec nam lab3.nam &
   exit 0
}

# 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 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

# Shapes for visualization
$n7 shape box
$n7 color Blue
```

```
$n8 shape hexagon
$n8 color Red

# Create duplex links
$ns duplex-link $n1 $n0 2Mb 10ms DropTail
$ns duplex-link $n2 $n0 2Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 20ms DropTail

# Create Ethernet LAN connected to n3
$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3

# Orientation of links
$ns duplex-link-op $n1 $n0 orient right-down
$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n3 orient right

# Set queue size
$ns queue-limit $n0 $n3 20

# FLOW 1  (tcp1 → sink1)
set tcp1 [new Agent/TCP/Vegas]
$tcp1 set class_ 1
$tcp1 set packetSize_ 55
$ns attach-agent $n1 $tcp1

set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1

$ns connect $tcp1 $sink1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

# cwnd trace for flow 1
set file1 [open file1.tr w]
$tcp1 attach $file1
$tcp1 trace cwnd_

# FLOW 2  (tcp2 → sink2)
set tcp2 [new Agent/TCP/Vegas]
$tcp2 set class_ 2
$tcp2 set packetSize_ 55
$ns attach-agent $n2 $tcp2

set sink2 [new Agent/TCPSink]
$ns attach-agent $n8 $sink2

$ns connect $tcp2 $sink2
```

```
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

# cwnd trace for flow 2
set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp2 trace cwnd_

# Start / Stop Traffic
$ns at 0.5 "$ftp1 start"
$ns at 1.0 "$ftp2 start"

$ns at 5.0 "$ftp1 stop"
$ns at 5.0 "$ftp2 stop"

# End simulation
$ns at 5.5 "finish"
$ns run
```

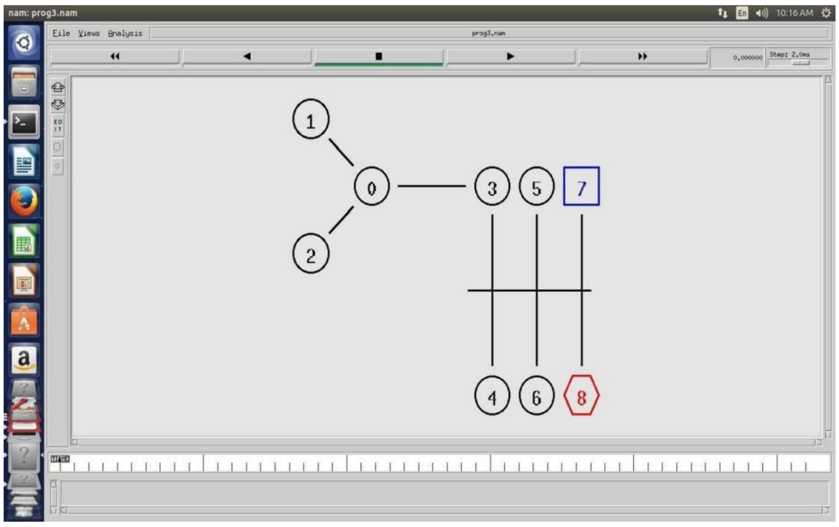
AWK Program — lab3.awk

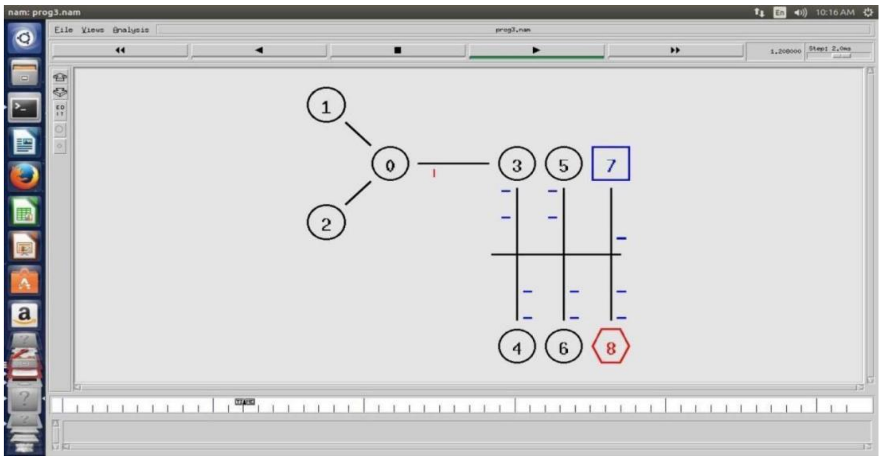
```
BEGIN { }

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

END { }
```

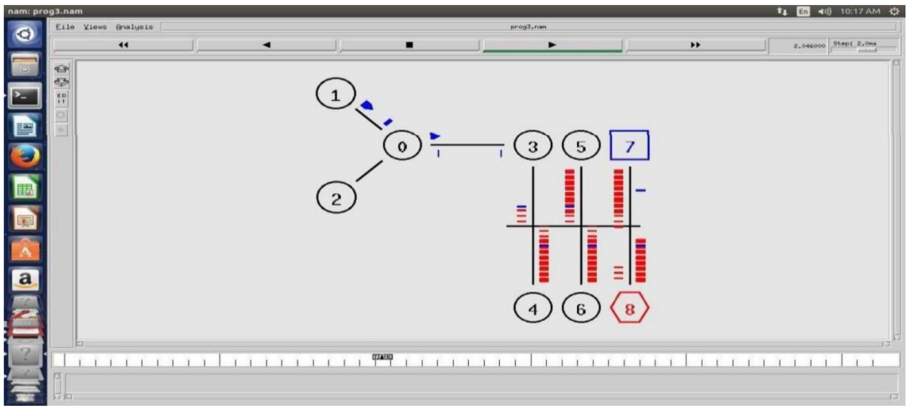
Execution Commands

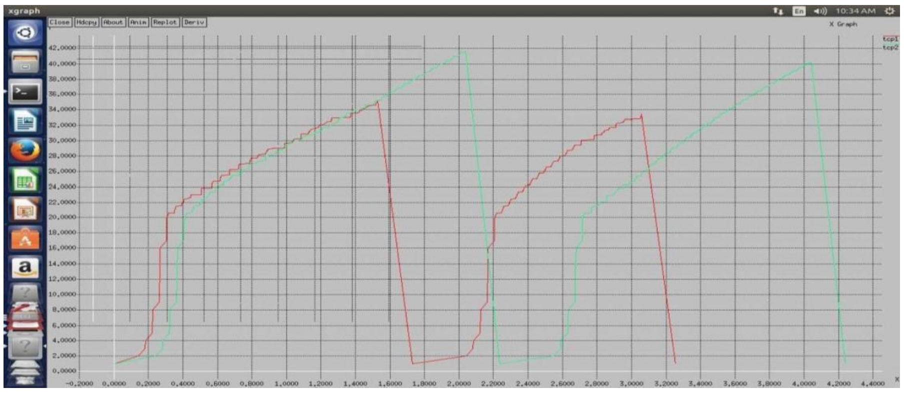
```
ns lab3.tcl
awk -f lab3.awk file1.tr > tcp1
awk -f lab3.awk file2.tr > tcp2
xgraph -x "time" -y "cwnd value" tcp1 tcp2
```

Output:-


nam: prog3.nam
File Views Analysis
prog3.nam
0.000000
Step: 2.0ms
1
0
2
3
5
7
4
6
8


nam: prog3.nam
File Views Analysis
prog3.nam
1.200000
Step: 2.0ms
1
0
2
3
5
7
4
6
8


nam: prog3.nam
File Views Analysis
prog3.nam
2.046000
Step: 2.0ms
1
0
2
3
5
7
4
6
8


xgraph
Close Hdcpy About Anim Replot Deriv
X Graph
tcp1
tcp2

PART-B Introduction

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).

Here is a clean, well-organized explanation of **Apache NetBeans**, followed by its **advantages and disadvantages** — suitable for direct use in a **Computer Networks Lab Manual (BCS502)**.

Apache NetBeans – Overview:-

Apache NetBeans is a free, open-source, cross-platform Integrated Development Environment (IDE) used widely for developing applications in **Java**, **C/C++**, **PHP**, **HTML5**, and more. It is officially supported by the **Apache Software Foundation**. NetBeans provides an easy-to-use graphical interface that simplifies the tasks of writing, debugging, and executing programs.

NetBeans is particularly popular in academic environments because it provides built-in support for Java development, project templates, automatic code completion, debugging tools, and integrated version control systems.

It is available for:

- **Windows**
- **Linux**
- **macOS**

NetBeans also includes tools for:

- GUI creation using Swing/AWT (Drag-and-Drop interface)
- Database connectivity (MySQL, Oracle, SQLite, etc.)
- Web application development (Servlets, JSP, HTML, CSS, JavaScript)
- Enterprise applications (Java EE)

Key Features of Apache NetBeans

- **Open Source & Free** – Anyone can download and use it without licensing restrictions.
- **Cross-platform** – Works on Windows, Linux, and macOS.

- **Powerful Code Editor** – Syntax highlighting, auto-completion, suggestions, and live error detection.
- **Modular Architecture** – Additional functionalities can be added using plugins.
- **Integrated Debugger** – Helps detect runtime errors quickly.
- **Version Control Support** – Works with Git, Subversion, and Mercurial.
- **GUI Builder (Matisse)** – Simplifies Java Swing application creation visually.
- **Support for multiple languages** – Java, C, C++, PHP, HTML5, JavaScript, and more.

Advantages of Apache NetBeans

1. Free and Open Source

NetBeans is completely free and actively developed by the Apache community.

2. Powerful Code Editor

Automatic indentation, syntax highlighting, auto-complete, suggestions, and real-time error checking.

3. Easy Project Management

Project Explorer helps manage source files, libraries, and configurations intuitively.

4. Built-in GUI Builder

The Matisse GUI Builder allows drag-and-drop design of desktop applications.

5. Debugging Tools

Includes breakpoints, watches, thread monitoring, and step-by-step execution.

6. Cross-Platform Support

Runs on Java Virtual Machine (JVM) — works on all major operating systems.

7. Plugin Support

Many community plugins are available to extend functionality (e.g., Maven, Gradle, JavaFX).

8. Great for Beginners

Ideal for students learning Java, as installation and usage are straightforward.

Disadvantages of Apache NetBeans

1. Heavy on System Resources

Requires more RAM and CPU compared to lightweight editors (e.g., VS Code, Sublime Text).

2. Slower Startup Time

NetBeans may start slower, especially on older systems.

3. Limited Enterprise Tools

Compared to Eclipse or IntelliJ IDEA Ultimate, some enterprise features are limited.

4. Plugin Availability

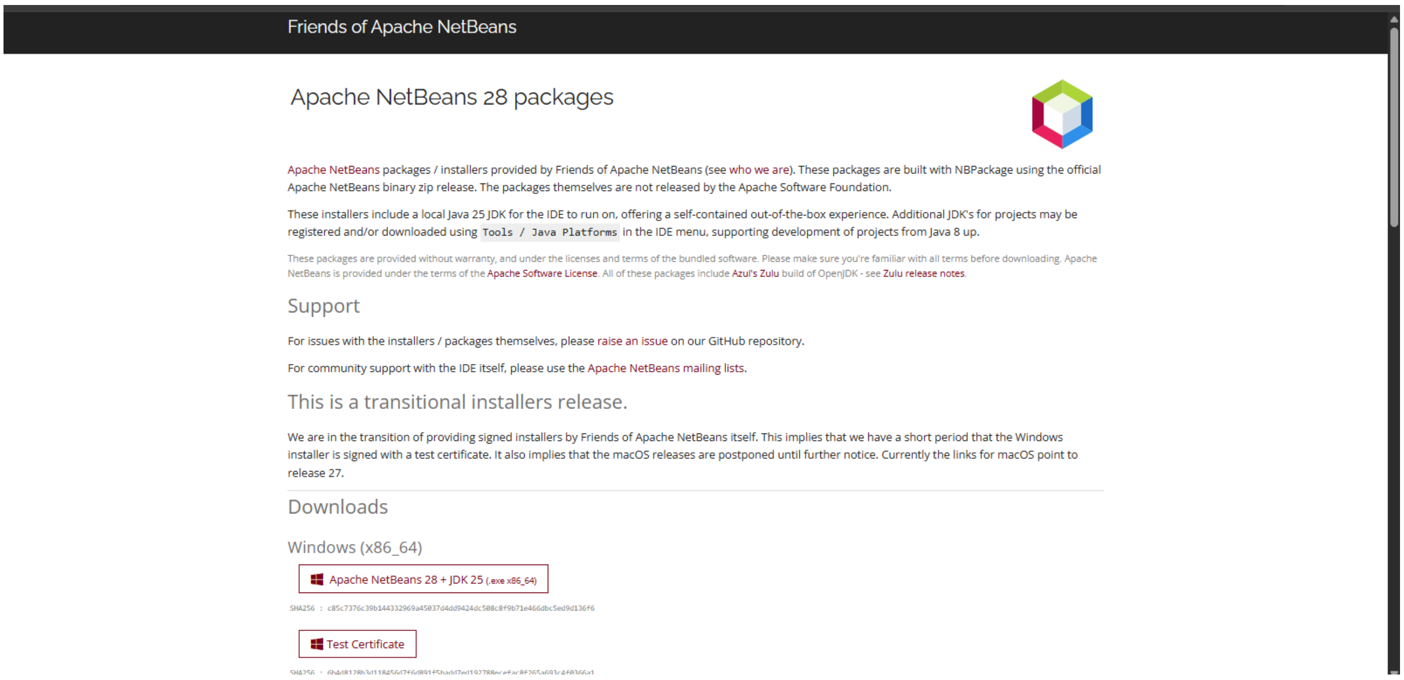
Not as many plugins/extensions as Eclipse Marketplace or IntelliJ plugins.

5. Less Popular for Large-Scale Industry Projects

Though good for learning and mid-size applications, IntelliJ IDEA dominates enterprise-level Java development.

How to Download and Install NetBeans IDE on your Computer:

To install NetBeans IDE on your system, follow the steps below:
Step 1: Open the NetBeans Download Page
Go to the official download link: https://netbeans.org/downloads



Friends of Apache NetBeans

Apache NetBeans 28 packages

Apache NetBeans packages / installers provided by Friends of Apache NetBeans (see who we are). These packages are built with NBPackage using the official Apache NetBeans binary zip release. The packages themselves are not released by the Apache Software Foundation.

These installers include a local Java 25 JDK for the IDE to run on, offering a self-contained out-of-the-box experience. Additional JDK's for projects may be registered and/or downloaded using `Tools / Java Platforms` in the IDE menu, supporting development of projects from Java 8 up.

These packages are provided without warranty, and under the licenses and terms of the bundled software. Please make sure you're familiar with all terms before downloading. Apache NetBeans is provided under the terms of the Apache Software License. All of these packages include Azul's Zulu build of OpenJDK - see Zulu release notes.

Support

For issues with the installers / packages themselves, please raise an issue on our GitHub repository.

For community support with the IDE itself, please use the Apache NetBeans mailing lists.

This is a transitional installers release.

We are in the transition of providing signed installers by Friends of Apache NetBeans itself. This implies that we have a short period that the Windows installer is signed with a test certificate. It also implies that the macOS releases are postponed until further notice. Currently the links for macOS point to release 27.

Downloads

Windows (x86_64)

SHA256 : c85c7376c39b144332969a45037d4dd9424dc508c8f9b71e466dbc5ed9d136f6

SHA256 : 6b4d8128b3d118456d7f6d891f5badd7ed19278Becefac8f265a693c4f0366a1

Step 2: View the Available NetBeans Packages
On the page, you will see a heading similar to:

“Apache NetBeans 28 packages”
This page is provided by the Friends of Apache NetBeans who create installer packages for easier installation.
The page explains:
- These installers bundle the NetBeans IDE along with required JDK or dependencies.
- They are transitional releases (not yet the official ASF releases).
- They currently provide installers for Windows (x86_64).

Step 3: Scroll to the Downloads Section
Under the Downloads section, you will see available installers for your operating system.

For Windows (x86_64), you will find a button such as:
Apache NetBeans 28 + JDK 22 (SFX exe 453 MB)
This is the installer file you need for Windows.

Step 4: Click the Installer Download Button
Click on the button to start downloading the NetBeans installer (.exe file).
You may also see a button labeled:

Test Certificate
This is optional and is used to verify the authenticity of the installer package.

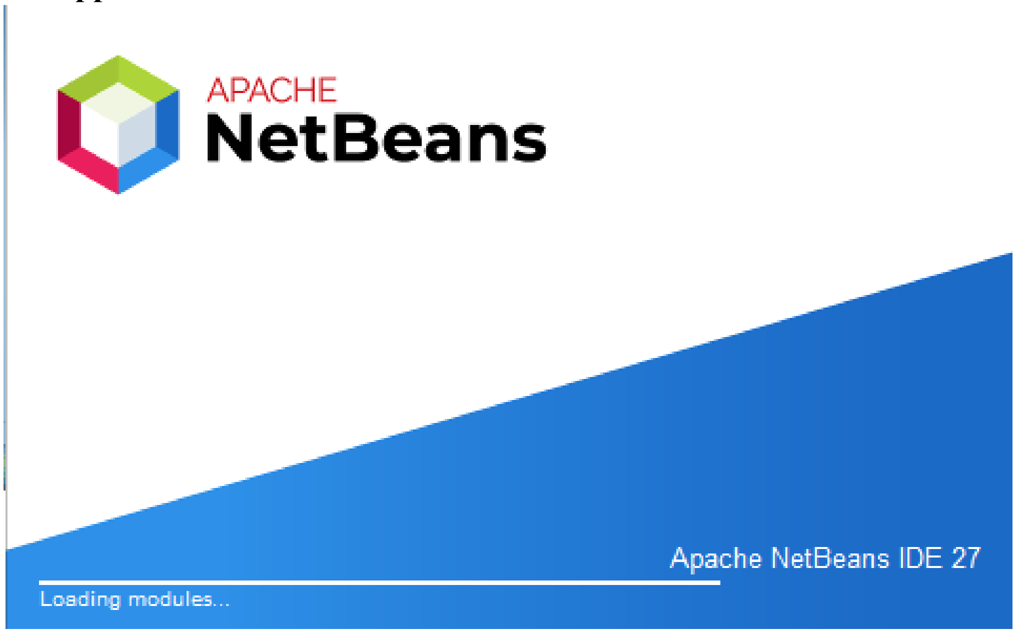
Step 5: Wait for the Download to Complete
The installer file is usually large (around 400–500 MB).
Once downloaded, you will get a .exe file on Windows.

Step 6: Run the Installer
Double-click the downloaded installer file (e.g., netbeans-28-jdk22.exe).
The setup wizard will open.

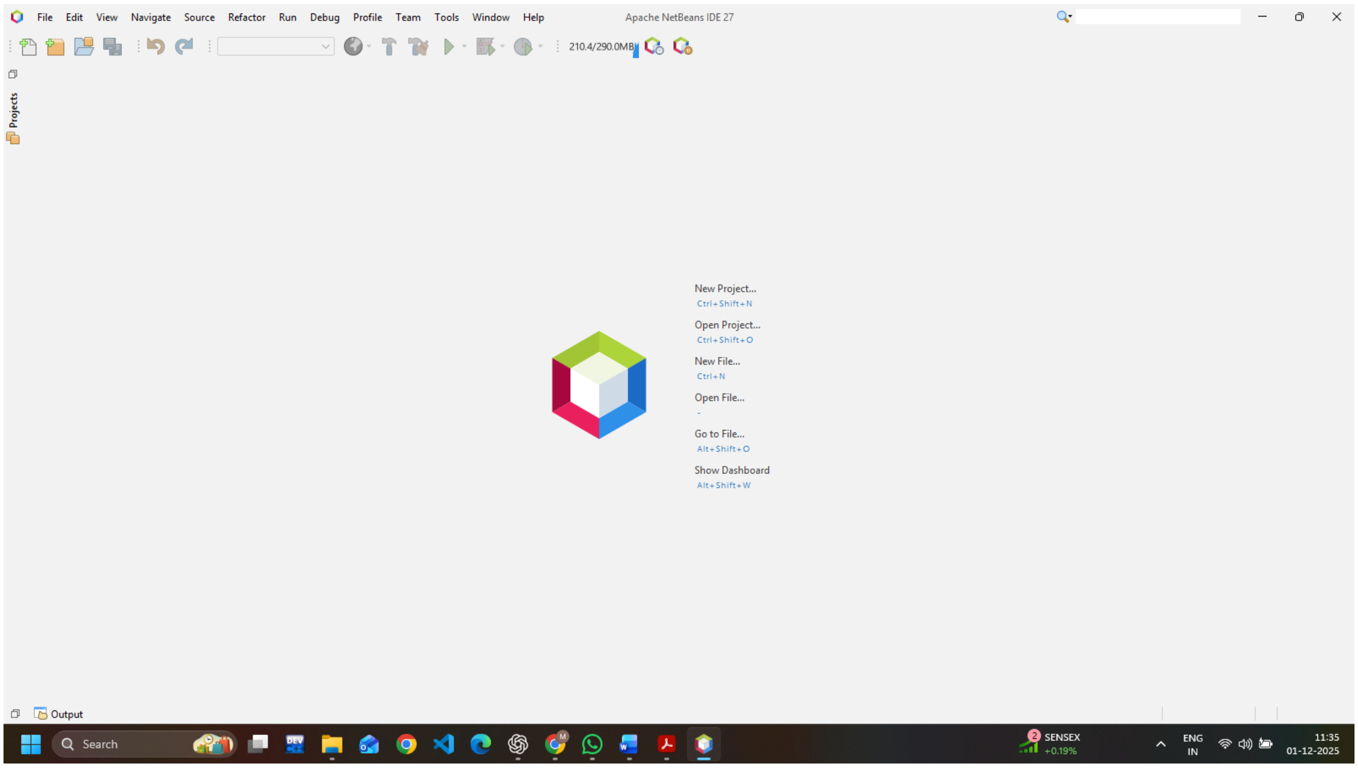
Step 7: Follow the Installation Wizard
Proceed with the default installation options:
- Accept license agreement
- Choose installation path
- Install the required JDK (if included)
- Complete the installation
After installation, NetBeans IDE will be ready to use.

Step 8: Launch NetBeans IDE
Go to:
- Start Menu → NetBeans IDE
or
- Double-click the NetBeans desktop icon

NetBeans will start with a default workspace.
The splash screen appears:

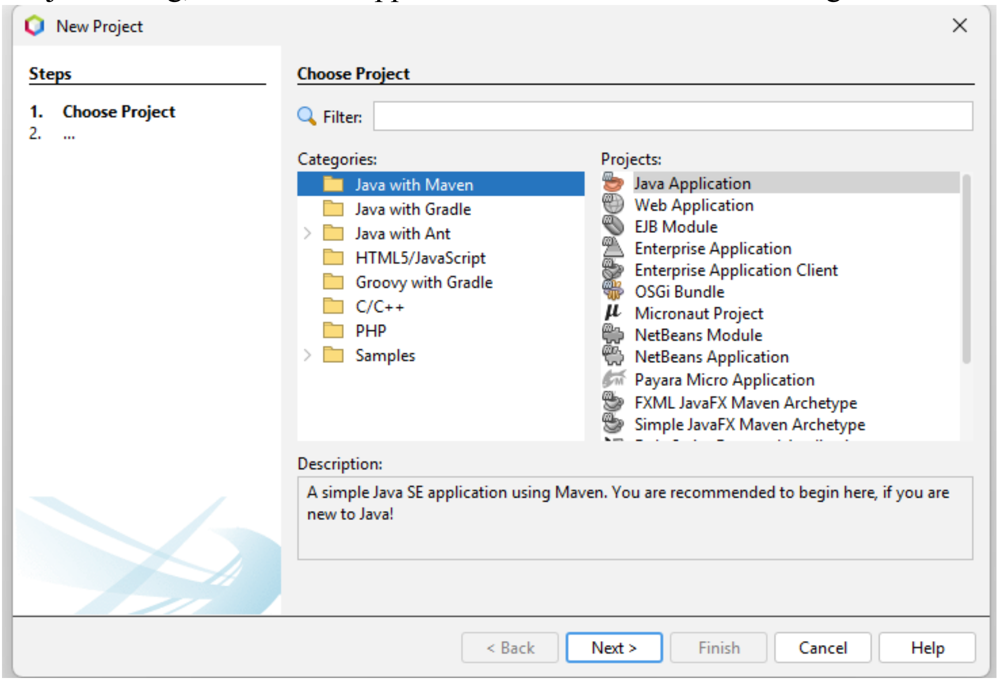


And you should see the home screen of NetBeans:


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Apache NetBeans IDE 27
Projects
New Project... Ctrl+Shift+N
Open Project... Ctrl+Shift+O
New File... Ctrl+N
Open File...
Go to File... Alt+Shift+O
Show Dashboard Alt+Shift+W
Output

How to Create Your First Java Project:-

Now, let's create a Java project using NetBeans IDE. Go to menu **File > New Project…**
Under the New Project dialog, choose Java application as shown in the following screenshot:



New Project

Steps
1. Choose Project
2. ...

Choose Project

Filter:

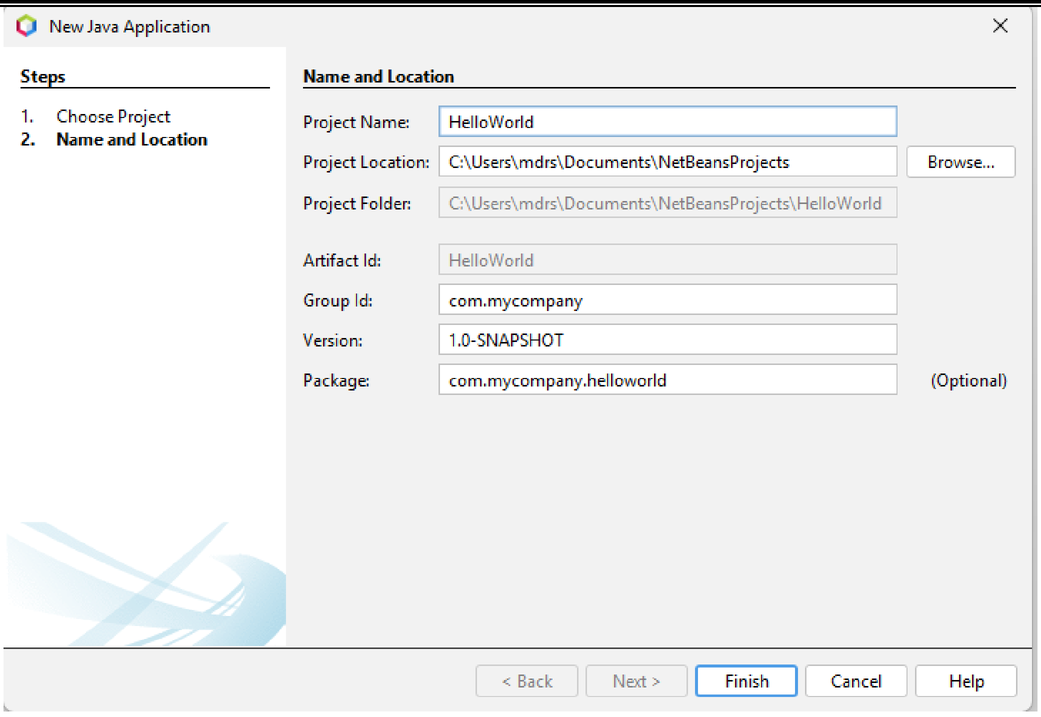
Categories:
- Java with Maven
- Java with Gradle
- Java with Ant
- HTML5/JavaScript
- Groovy with Gradle
- C/C++
- PHP
- Samples

Projects:
- Java Application
- Web Application
- EJB Module
- Enterprise Application
- Enterprise Application Client
- OSGi Bundle
- Micronaut Project
- NetBeans Module
- NetBeans Application
- Payara Micro Application
- FXML JavaFX Maven Archetype
- Simple JavaFX Maven Archetype

Description:
A simple Java SE application using Maven. You are recommended to begin here, if you are new to Java!

< Back | Next > | Finish | Cancel | Help

Click **Next** to advance to the next step. In the New Java Application screen, type Project Name, specify Project Location and the main class:

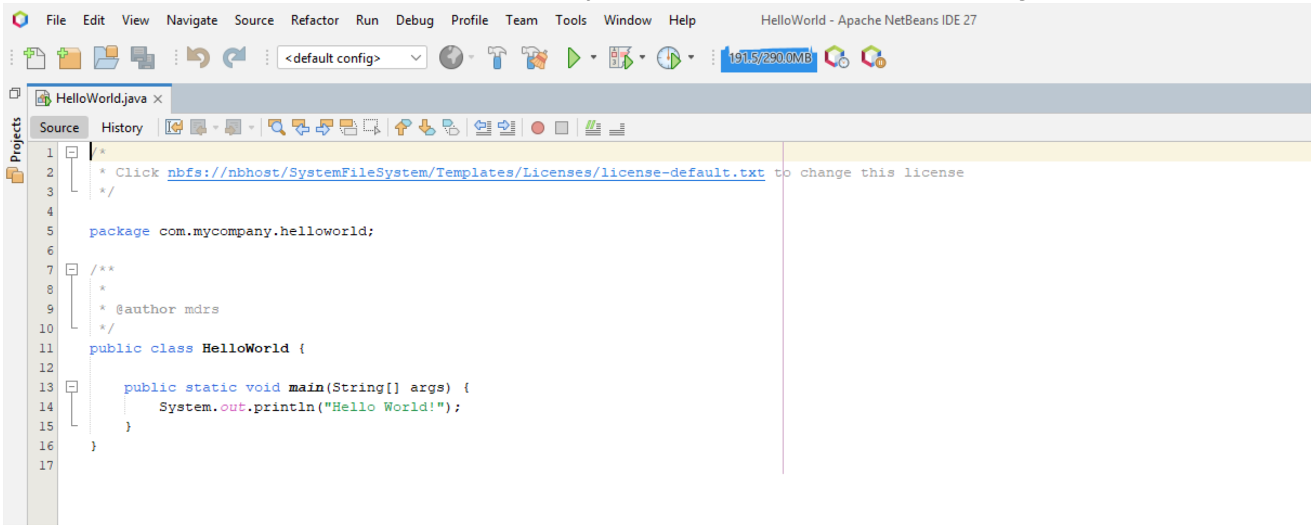


Name and Location

Project Name: HelloWorld
Project Location: C:\Users\mdrs\Documents\NetBeansProjects
Project Folder: C:\Users\mdrs\Documents\NetBeansProjects\HelloWorld
Artifact Id: HelloWorld
Group Id: com.mycompany
Version: 1.0-SNAPSHOT
Package: com.mycompany.helloworld (Optional)

Note that we check the option Create Main Class to generate the main class for the application. Here we specify the package name net.codejava before the class name HelloWorld.
Click Finish. NetBeans create the project with a main class very quickly:

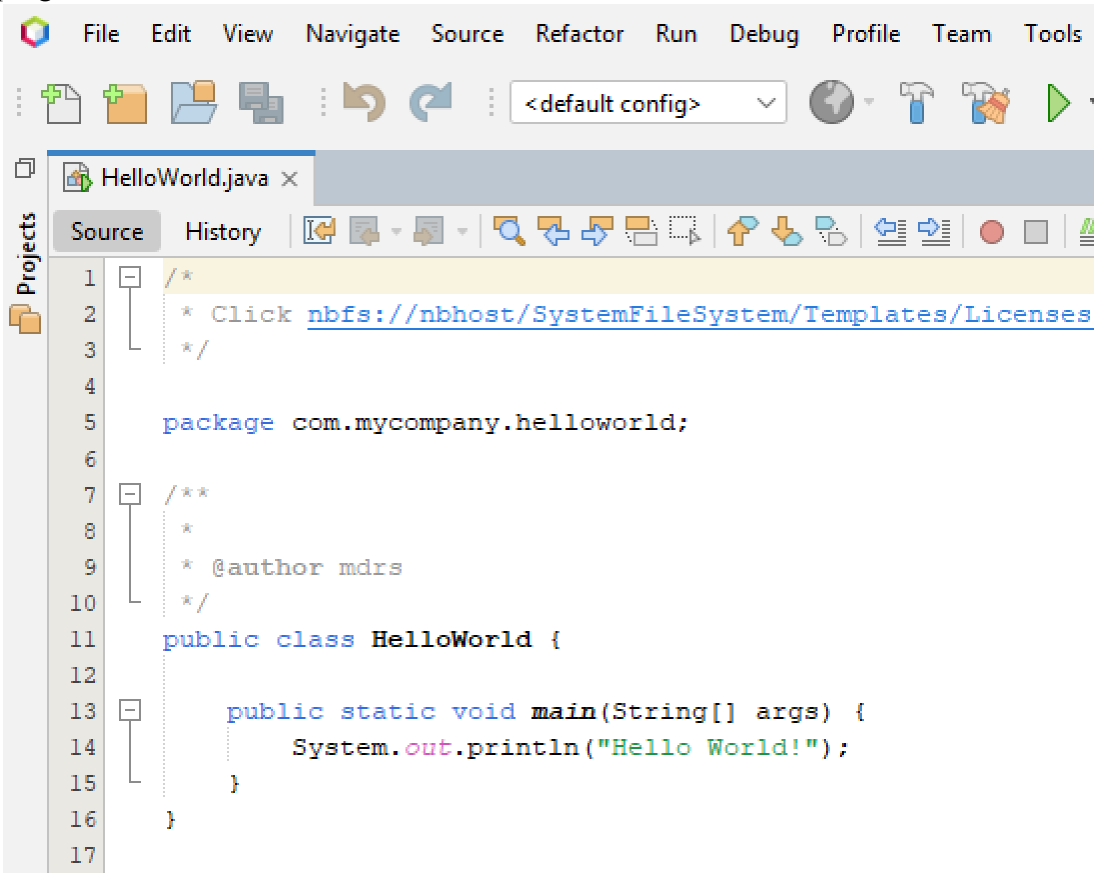
Write Your First Java Code:-
You can see a code editor for the HelloWorld.java file as shown in the following screenshot:


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
HelloWorld - Apache NetBeans IDE 27
<default config>
191.5/290.0MB
HelloWorld.java
Source History
Projects

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 */

package com.mycompany.helloworld;

/**
 *
 * @author mdrs
 */
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

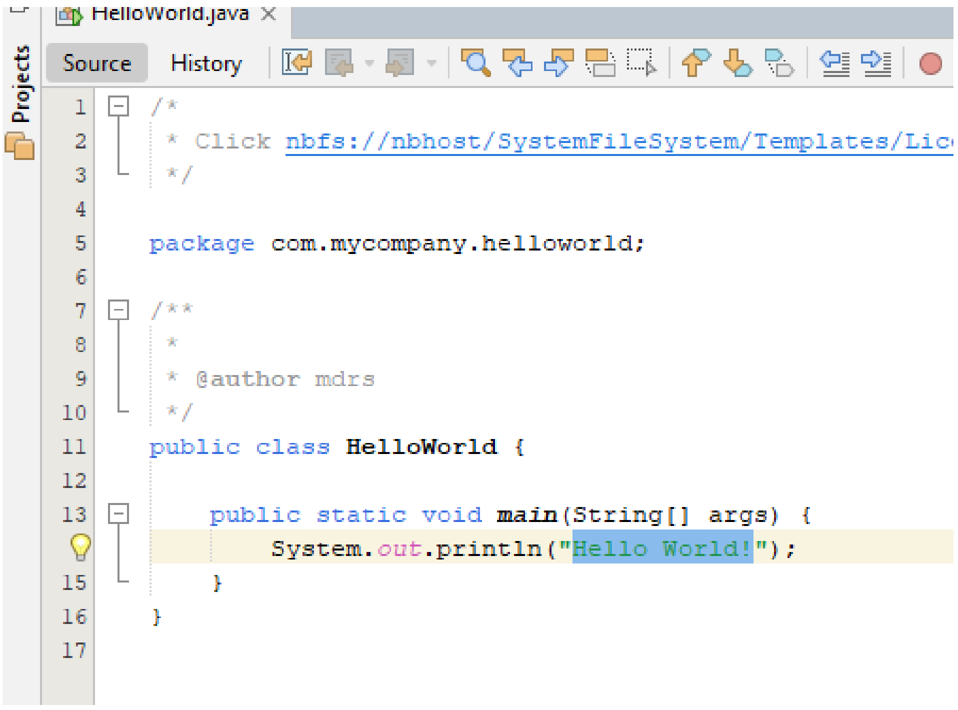
The method main() is the main entry to a Java application. All Java programs start from the main() method. Now, let’s type some code in this method to print “Hello World Java!” on the screen:

The whole program should look like this:



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses
 */

package com.mycompany.helloworld;

/**
 *
 * @author mdrs
 */
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

NetBeans is very smart, as it compiles the code instantly while you are typing the code. So if there's any error, the IDE will inform you by underlining the errors with green color, as shown in the following screenshot:



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Lic
 */

package com.mycompany.helloworld;

/**
 *
 * @author mdrs
 */
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

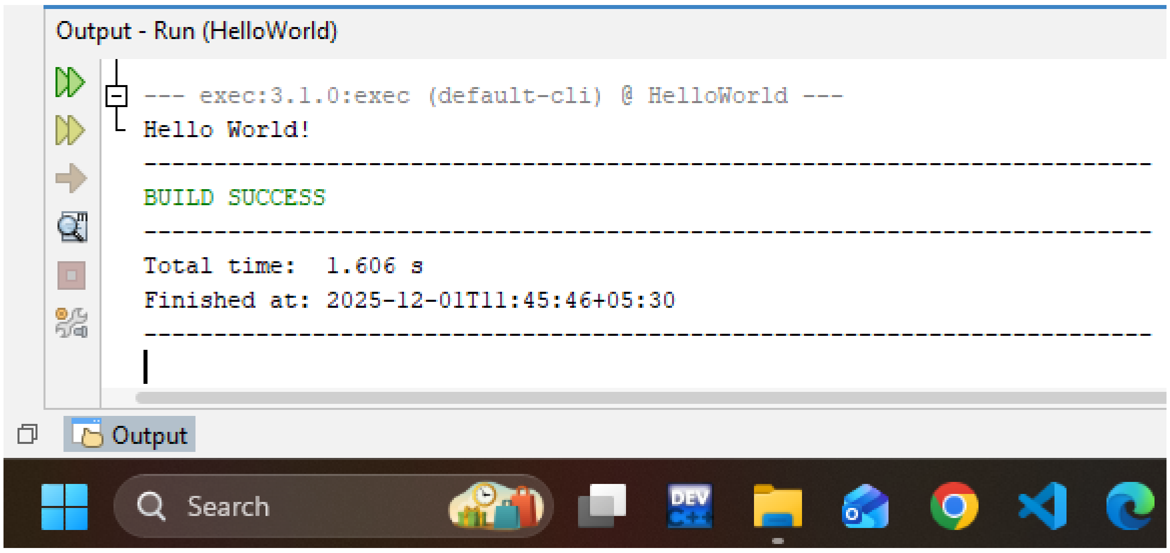
If there's no red marks like this, the code is fine and we're ready to run the program.

Run Your First Java Program

To run the HelloWorld program above, there are several ways:

- Go to menu **Run > Run Project <ProjectName>**
- Click **Run Project** icon in the toolbar.
- Press **F6** key.
- Right click in the code editor, and select **Run File** (or press **Shift + F6**).

You should see the output of this program like this:



```
--- exec:3.1.0:exec (default-cli) @ HelloWorld ---
Hello World!
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  1.606 s
Finished at: 2025-12-01T11:45:46+05:30
------------------------------------------------------------------------
```

That's it! The HelloWorld program has run and printed the output "Hello World".
You have successfully created and run your first Java program with NetBeans IDE.

Program 4:-
Write a program for error detecting code using CRC-CCITT (16-bits).

Program Objective:-
Understand the operation of CRC-CCITT.

```
import java.util.*;
import java.util.Scanner;

public class Crc {
   // Function to perform CRC division (XOR operation with generator polynomial)
   void div(int a[], int k) {
      // Generator polynomial (CRC-16 type)
      int gp[] = {1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};
      int count = 0;

      // Perform modulo-2 division
      for (int i = 0; i < k; i++) {
         // If the current bit is 1, perform XOR with the generator polynomial
         if (a[i] == gp[0]) {
            for (int j = i; j < 17 + i; j++) {
               a[j] = a[j] ^ gp[count++]; // XOR operation
            }
            count = 0; // Reset counter after one division cycle
         }
      }
   }

   public static void main(String args[]) {
      int a[] = new int[100]; // Array to hold data bits (including CRC)
      int b[] = new int[100]; // Copy of original data
      int len, k; // len = total bits after appending zeros, k = original message length
      Crc ob = new Crc(); // Create object of Crc class

      System.out.println("Enter the length of Data Frame : ");
      Scanner sc = new Scanner(System.in);
      len = sc.nextInt(); // Read length of message bits

      int flag = 0; // To detect error in received data

      System.out.println("\nEnter the Messege : ");
      // Input message bits
      for (int i = 0; i < len; i++) {
         a[i] = sc.nextInt();
      }

      // Append 16 zeros at the end (since generator is of degree 16)
      for (int i = 0; i < 16; i++) {
         a[len++] = 0;
```

```
        }

        // k = length of message bits (excluding appended zeros)
        k = len - 16;

        // Copy message (with appended zeros) into another array b
        for (int i = 0; i < len; i++) {
            b[i] = a[i];
        }

        // Perform division to get CRC remainder
        ob.div(a, k);

        // XOR remainder with original message bits to get transmitted data (message + CRC)
        for (int i = 0; i < len; i++) {
            a[i] = a[i] ^ b[i];
        }

        System.out.println("Data to be transmitted : ");
        // Display transmitted data bits
        for (int i = 0; i < len; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println();

        System.out.println("Enter the Reveived Data : ");
        // Input received data bits (after transmission)
        for (int i = 0; i < len; i++) {
            a[i] = sc.nextInt();
        }

        // Perform division again to check for remainder (error detection)
        ob.div(a, k);

        // If any bit of remainder is non-zero, error detected
        for (int i = 0; i < len; i++) {
            if (a[i] != 0) {
                flag = 1;
                break;
            }
        }

        // Display result
        if (flag == 1)
            System.out.println("Error in data");
        else
            System.out.println("No Error");
    }
}
```

Output :-

1.
```
Enter the length of Data Frame :
4
Enter the Messege :
1 0 1 1
Data to be transmitted :
1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Reveived Data :
1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0
Error in data
```

2.
```
Enter the length of Data Frame :
4
Enter the Messege :
1 0 1 1
Data to be transmitted :
1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Reveived Data :
1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
No Error
```

Program 5:-
Develop a program to implement a sliding window protocol in the data link layer.

```
import java.util.*;
public class SlidingWindowGBN {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter total number of frames: ");
        int totalFrames = sc.nextInt();

        System.out.print("Enter window size: ");
        int windowSize = sc.nextInt();

        int currentFrame = 0;

        Random rand = new Random();

        while (currentFrame < totalFrames) {

            int lastFrame = Math.min(currentFrame + windowSize, totalFrames);

            System.out.println("\nSending frames:");
            for (int i = currentFrame; i < lastFrame; i++) {
                System.out.print(i + " ");
            }

            System.out.println("\n\nReceiving ACKs:");

            boolean timeout = false;

            for (int i = currentFrame; i < lastFrame; i++) {

                // Randomly drop an ACK
                if (rand.nextInt(10) < 2) {   // 20% loss
                    System.out.println("ACK for frame " + i + " LOST!");
                    timeout = true;
                    break;
                } else {
                    System.out.println("ACK received for frame " + i);
                }
            }

            if (timeout) {
                System.out.println("\nTimeout occurred! Resending window from frame " + currentFrame);
            } else {
                currentFrame = lastFrame; // Move window
            }
        }

        System.out.println("\nAll frames transmitted successfully!");
    }
}
```

Output :-

```
Enter total number of frames :
 10
Enter the window size : 4

Sending frames:
0 1 2 3

Receiving ACKs:
ACK received for frame 0
ACK received for frame 1
ACK received for frame 2
ACK received for frame 3

Sending frames:
4 5 6 7

Receiving ACKs:
ACK received for frame 4
ACK received for frame 5
ACK received for frame 6
ACK received for frame 7

Sending frames:
8 9

Receiving ACKs:
ACK received for frame 8
ACK received for frame 9

All frames transmitted successfully!
```

Program 6:-

Write a program to find the shortest path between vertices using bellman-ford algorithm and path vector routing algorithm.

Program Objective:

Understand the Implementation of the shortest path for bellman-ford algorithm.

```
import java.util.Scanner;

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

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

   public void BellmanFordEvaluation(int source, int A[][])
   {
      // Initialize distances
      for (int node = 1; node <= NoV; node++)
      {
         D[node] = MAX_VALUE;
      }
      D[source] = 0;

      // Relax edges (NoV - 1) times
      for (int node = 1; node <= NoV - 1; node++)
      {
         for (int i = 1; i <= NoV; i++)
         {
            for (int j = 1; j <= NoV; j++)
            {
               if (A[i][j] != MAX_VALUE)
               {
                  if (D[i] != MAX_VALUE && D[j] > D[i] + A[i][j])
                  {
                     D[j] = D[i] + A[i][j];
                  }
               }
            }
         }
      }

      // Check for negative cycles
      for (int i = 1; i <= NoV; i++)
      {
         for (int j = 1; j <= NoV; j++)
```

```
            {
                if (A[i][j] != MAX_VALUE)
                {
                    if (D[i] != MAX_VALUE && D[j] > D[i] + A[i][j])
                    {
                        System.out.println("The Graph contains a negative edge cycle");
                        return;
                    }
                }
            }
        }

        // Print final distances
        for (int vertex = 1; vertex <= NoV; vertex++)
        {
            System.out.println("Distance from source " + source + " to vertex "
                              + vertex + " is " + D[vertex]);
        }
    }

    public static void main(String... arg)
    {
        int NoV = 0, source;
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of vertices: ");
        NoV = scanner.nextInt();

        int A[][] = new int[NoV + 1][NoV + 1];

        System.out.println("Enter the adjacency matrix (use 999 for no edge):");
        for (int i = 1; i <= NoV; i++)
        {
            for (int j = 1; j <= NoV; j++)
            {
                A[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter the source vertex: ");
        source = scanner.nextInt();

        BellmanFord bellmanford = new BellmanFord(NoV);
        bellmanford.BellmanFordEvaluation(source, A);

        scanner.close();
    }
}
```

Output:-

```
Enter the number of vertices
5
Enter the adjacency matrix
0   6   999 7   999
999 0   5   999 999
999 999 0   999 2
999 999 999 0   3
999 999 999 999 0
Enter the source vertex
1

Shortest distances from source 1:
Distance to vertex 1 = 0
Distance to vertex 2 = 6
Distance to vertex 3 = 11
Distance to vertex 4 = 7
Distance to vertex 5 = 10
```

Program No. 7 :-
Using TCP/IP sockets, write a client–server program where the client sends a filename and the server returns the file contents if present.

Server Program — Server.java

```
import java.net.*;
import java.io.*;

public class TCPS
{
   public static void main(String[] args) throws Exception
   {
      // Create server socket
      ServerSocket sersock = new ServerSocket(4000);
      System.out.println("Server ready for connection...");

      // Wait for client
      Socket sock = sersock.accept();
      System.out.println("Client connected. Waiting for filename...");

      // Read filename sent by client
      BufferedReader fileRead = new BufferedReader(
            new InputStreamReader(sock.getInputStream()));
      String fname = fileRead.readLine();

      // Open requested file
      BufferedReader contentRead = null;
      OutputStream ostream = sock.getOutputStream();
      PrintWriter pwrite = new PrintWriter(ostream, true);

      try {
         contentRead = new BufferedReader(new FileReader(fname));
         String str;

         // Send file contents line by line
         while ((str = contentRead.readLine()) != null) {
            pwrite.println(str);
         }
      }
      catch (FileNotFoundException e) {
         pwrite.println("Error: File Not Found!");
      }

      // Close connections
      pwrite.close();
      fileRead.close();
      if (contentRead != null) contentRead.close();
      sock.close();
```

```
        sersock.close();
    }
}
```

Client Program — Client.java

```
import java.net.*;
import java.io.*;

public class TCPC
{
    public static void main(String[] args) throws Exception
    {
        // Create client socket
        Socket sock = new Socket("127.0.0.1", 4000);

        // Read filename from user
        System.out.println("Enter the filename: ");
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        String fname = keyRead.readLine();

        // Send filename to server
        PrintWriter pwrite = new PrintWriter(sock.getOutputStream(), true);
        pwrite.println(fname);

        // Receive file contents from server
        BufferedReader socketRead = new BufferedReader(
                new InputStreamReader(sock.getInputStream()));
        String str;

        while ((str = socketRead.readLine()) != null) {
            System.out.println(str);
        }

        // Close connections
        pwrite.close();
        socketRead.close();
        keyRead.close();
        sock.close();
    }
}
```

Server Output:-

```
Server ready for connection...
Connection established. Waiting for filename...
Client requested file: sample.txt
File transfer completed.
```

Client Output:-

```
Enter the filename
sample.txt

--- File Contents Received From Server ---

This is a sample file.
It contains multiple lines.
TCP client-server file transfer successful.
```

Program No. 8
Develop a program on a datagram socket for client/server to display the messages on client side typed at the server side.

UDP Server:- udps.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class udps
{
   public static void main(String[] args)
   {
      DatagramSocket skt = null;
      Scanner sc = new Scanner(System.in);

      try
      {
         skt = new DatagramSocket(2400);
         byte[] buffer = new byte[1000];

         while (true)
         {
            // Receive packet from client
            DatagramPacket request =
                  new DatagramPacket(buffer, buffer.length);
            skt.receive(request);

            // Read message from server user
            String message = sc.nextLine();
            byte[] sendMsg = message.getBytes();

            // Send reply back to client
            DatagramPacket reply =
                  new DatagramPacket(sendMsg, sendMsg.length,
                        request.getAddress(), request.getPort());
            skt.send(reply);
         }
      }
      catch (Exception ex)
      {
         System.out.println("Error: " + ex);
      }
   }
}
```

UDP Client (udpc.java)

```
import java.io.*;
import java.net.*;

public class udpc
{
   public static void main(String[] args)
   {
      DatagramSocket skt;

      try
      {
         skt = new DatagramSocket();

         String msg = "text message";
         byte[] b = msg.getBytes();

         InetAddress host = InetAddress.getByName("127.0.0.1");
         int serverSocket = 2400;

         // Send request to server
         DatagramPacket request =
               new DatagramPacket(b, b.length, host, serverSocket);
         skt.send(request);

         // Receive reply
         byte[] buffer = new byte[1000];
         DatagramPacket reply =
               new DatagramPacket(buffer, buffer.length);
         skt.receive(reply);

         System.out.println("Client received: " +
               new String(reply.getData()).trim());

         skt.close();
      }
      catch (Exception ex)
      {
         System.out.println("Error: " + ex);
      }
   }
}
```

Sample Output:-

Server Side

```
Hello Client!
How are you?
```

Client Side

```
Client received: Hello Client!
```

Program 9:-

Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

```
import java.math.BigInteger;
import java.util.Random;

public class RSAExample {

    private BigInteger p, q, n, phi, e, d;
    private int bitLength = 32 ;
    private Random r;

    public RSAExample() {
        r = new Random();

        // Step 1: Choose two large prime numbers
        p = BigInteger.probablePrime(bitLength, r);
        q = BigInteger.probablePrime(bitLength, r);

        // Step 2: Compute n = p*q
        n = p.multiply(q);

        // Step 3: Compute phi = (p-1)*(q-1)
        phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

        // Step 4: Choose e (public key)
        e = BigInteger.probablePrime(bitLength / 2, r);

        // Step 5: Compute d (private key)
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0) {
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }

    // Encryption: C = M^e mod n
    public BigInteger encrypt(BigInteger message) {
        return message.modPow(e, n);
    }

    // Decryption: M = C^d mod n
    public BigInteger decrypt(BigInteger encrypted) {
        return encrypted.modPow(d, n);
    }

    public static void main(String[] args) {
        RSAExample rsa = new RSAExample();
        String plainText = "hello";
        System.out.println("\n--- RSA Algorithm Output ---");

        System.out.println("Prime p = " + rsa.p);
        System.out.println("Prime q = " + rsa.q);
        System.out.println("n = " + rsa.n);
        System.out.println("phi = " + rsa.phi);
        System.out.println("Public key e = " + rsa.e);
        System.out.println("Private key d = " + rsa.d);

        // Convert message to BigInteger
        BigInteger message = new BigInteger(plainText.getBytes());

        // Encrypt
        BigInteger encrypted = rsa.encrypt(message);
        System.out.println("\nEncrypted message: " + encrypted);

        // Decrypt
        BigInteger decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypted message: " + new String(decrypted.toByteArray()));

        System.out.println("\n--- End ---");
    }
}
```

Output:

```
--- RSA Algorithm Output ---
Prime p = 4289309881
Prime q = 2758150099
n = 11830560472921828219
phi = 11830560465874368240
Public key e = 48341
Private key d = 3296287187167453421

Encrypted message: 7170850868341140022
Decrypted message: hello
```

Program 10:-

Write a program for congestion control using leaky bucket algorithm.

```
import java.util.Scanner;
import java.lang.*;
public class Leakybucket
{
   public static void main(String [] args)
   {
      int i;
      int a[]=new int[20]; // array to store incoming packets
      int buck_rem=0, buck_cap=4, rate=3, sent=0,recv=0; // initialize bucket parameters

      Scanner in=new Scanner(System.in);
      System.out.println("Enter the number of packets");
      int n=in.nextInt(); // read number of packets

      System.out.println("Enter the packets: ");

      // read packet sizes
      for(i=1; i<=n; i++)
         a[i]=in.nextInt();

      System.out.println("Clock \t packets size\t accept \t sent \t remaining");

      // simulate leaky bucket algorithm for each packet
      for(i=1; i<=n; i++)
      {
         // if there is an incoming packet
         if(a[i]!=0)
         {
            // check if bucket will overflow
            if(buck_rem+a[i]>buck_cap)
               recv=-1; // packet dropped
            else
            {
               recv=a[i]; // packet accepted
               buck_rem+=a[i]; // add to bucket
            }
         }
         else
         {
            // no incoming packet
            recv=0;

            // check if bucket has any packets to send
            if(buck_rem!=0)
            {
               // send packets based on available amount and rate
```

```
                if(buck_rem<rate)
                {
                   sent=buck_rem;
                   buck_rem=0; // bucket empty
                }
                else
                {
                   sent=rate;
                   buck_rem=buck_rem-rate; // reduce bucket by sending rate amount
                }
             }
             else
                sent=0; // nothing to send
          }

          // display the current clock cycle details
          if(recv==-1)
             System.out.println(+i+ "\t\t" +a[i]+ "\t dropped \t" + sent + "\t" +buck_rem);
          else
             System.out.println(+i+ "\t\t" +a[i]+ "\t\t" +recv+ "\t" +sent +"\t" + buck_rem);
       }

    }
}
```

Output:-

```
n = 6
packets = 2 1 5 0 3 2

Clock   packet  accept      sent    remaining
1          2          2         0             2
2          1          1         0             3
3          5      dropped     0             3
4          0          0         3             0
5          3          3         0             3
6          2      dropped     0             3
```