Experiment 1

Computer Communications

Name: DEBASYA SAHOO

Faculty: Prof. Berlin Hency

Slot:L49+L50

Registration Number:

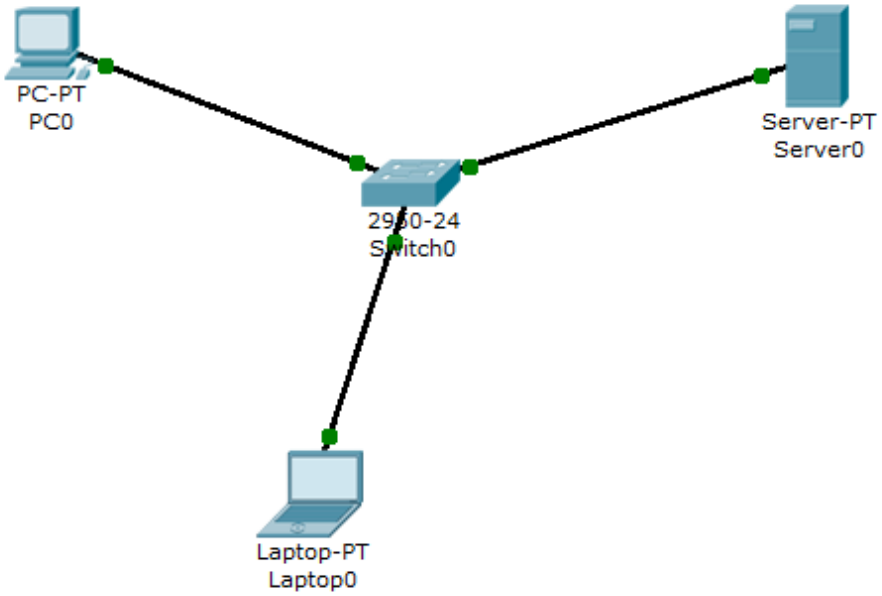
15BEC1111

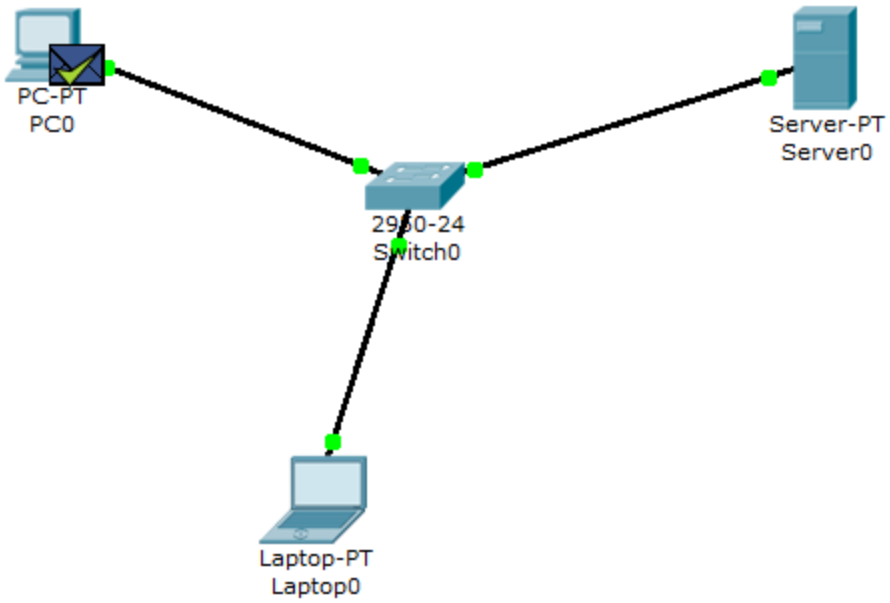
Aim:

1) Simulation of client server network topology using cisco packet tracer

Software used: Cisco Packet Tracer

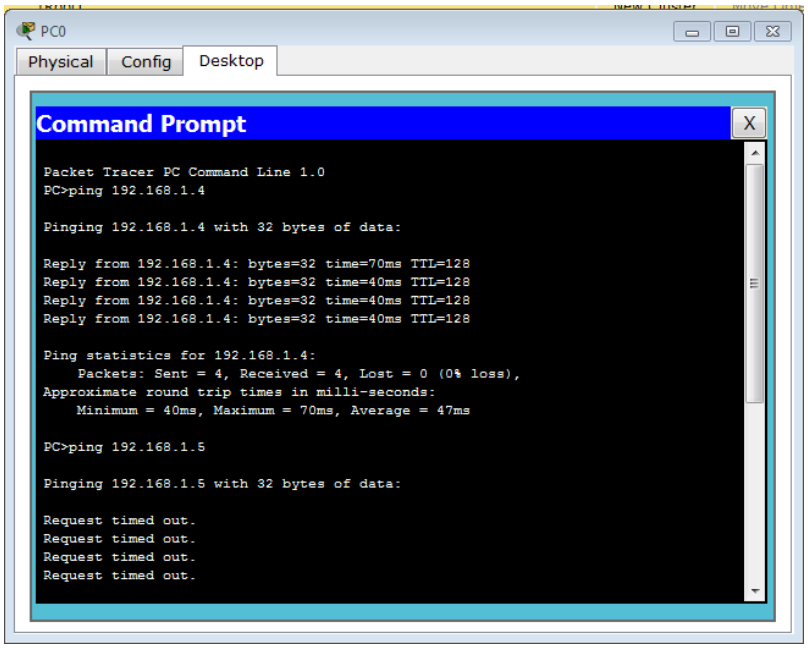
Simulation and outputs:



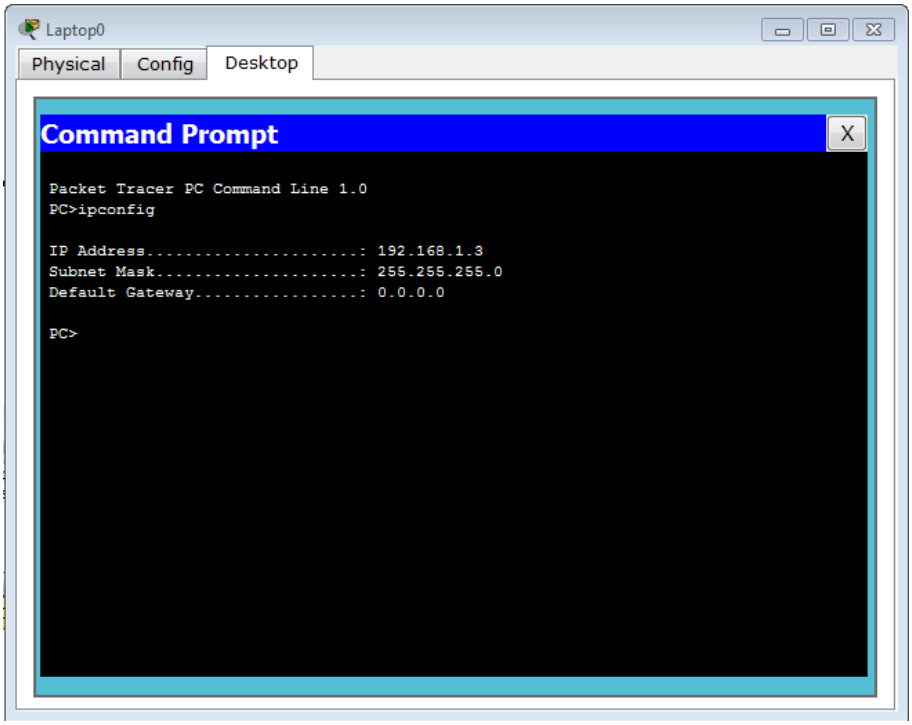


Each pc, laptop, switch and the server are given different IP addresses and are connected using copper wire.

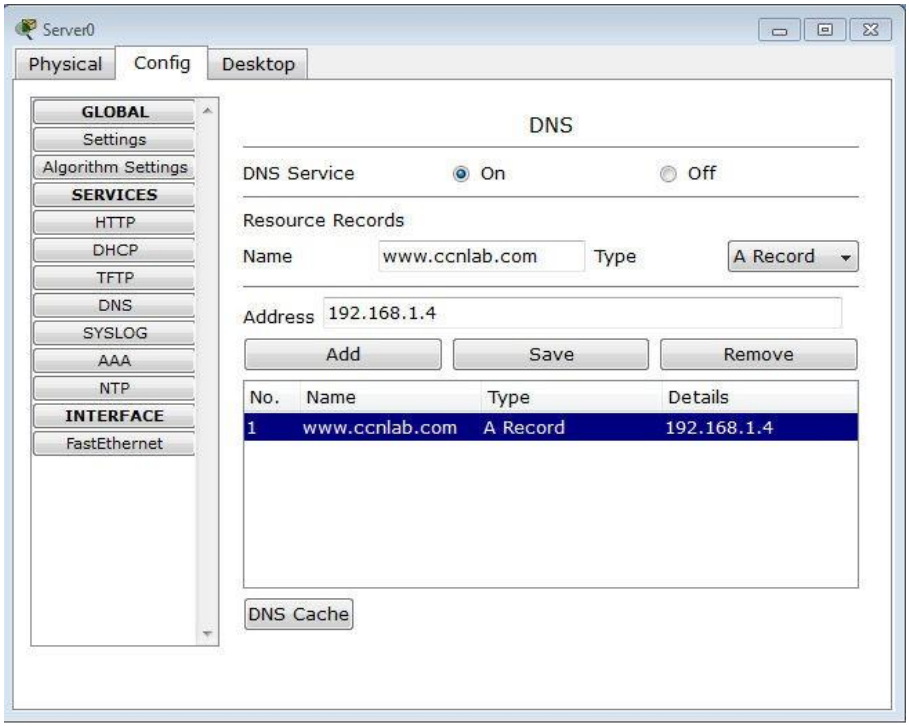
Pinging to check connectivity:

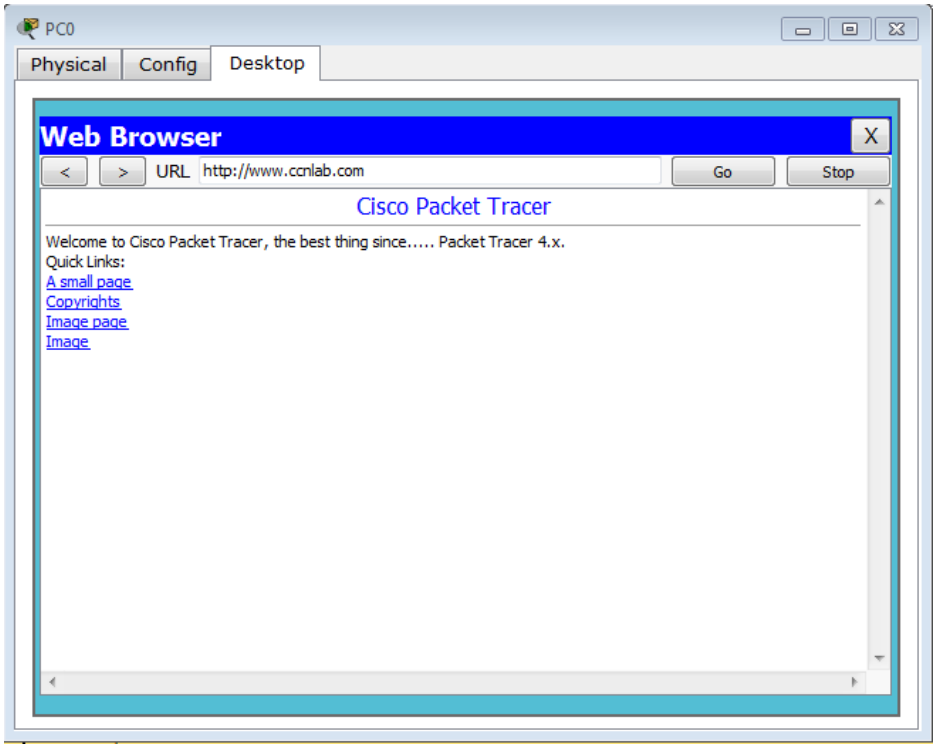


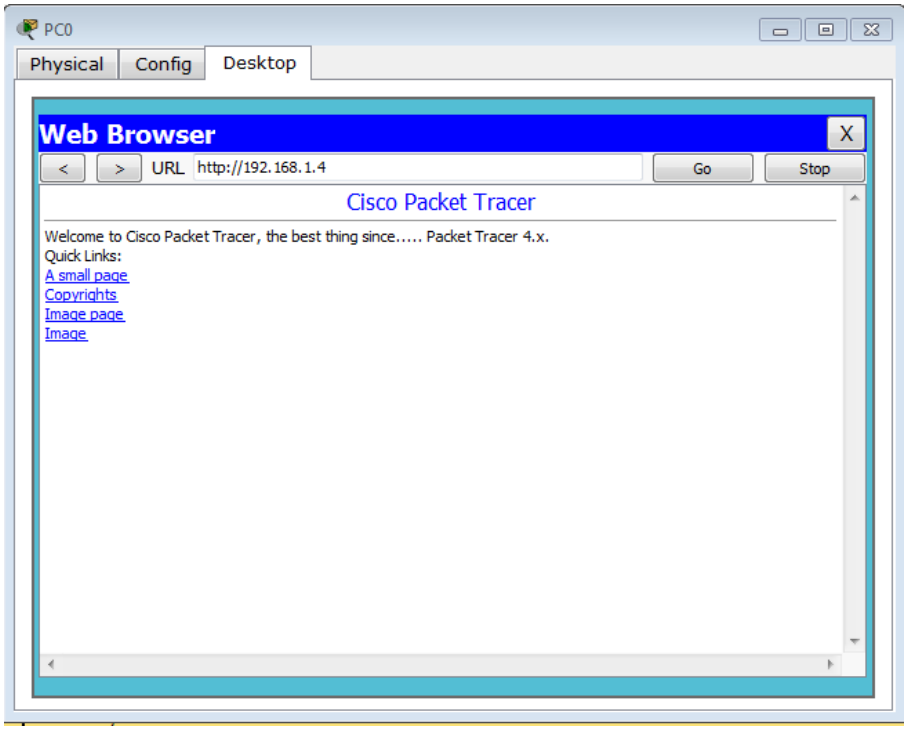
ipconfig:



Enabling the DNS Service:

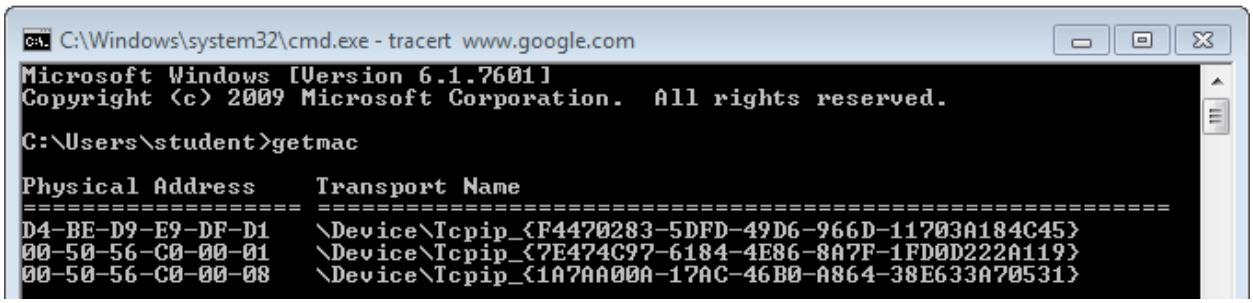




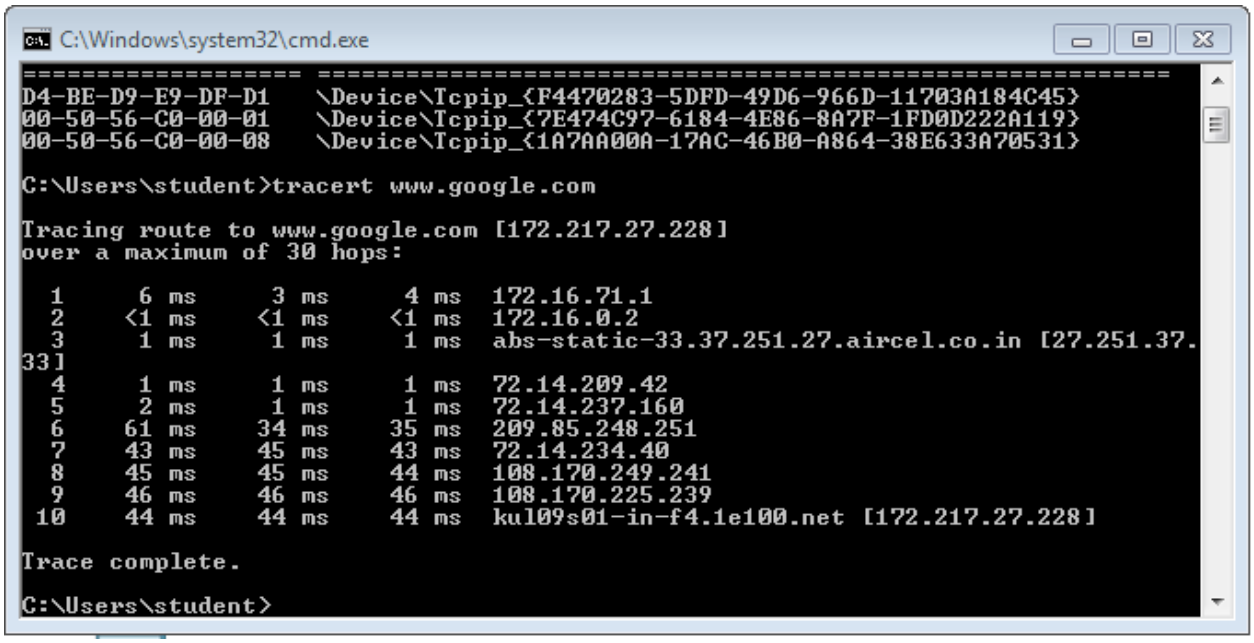


After enabling the DNS service we gave a name to the server's IP address, '*www.ccnlab.com'* after which we could access it by using the domain name in the web browser.

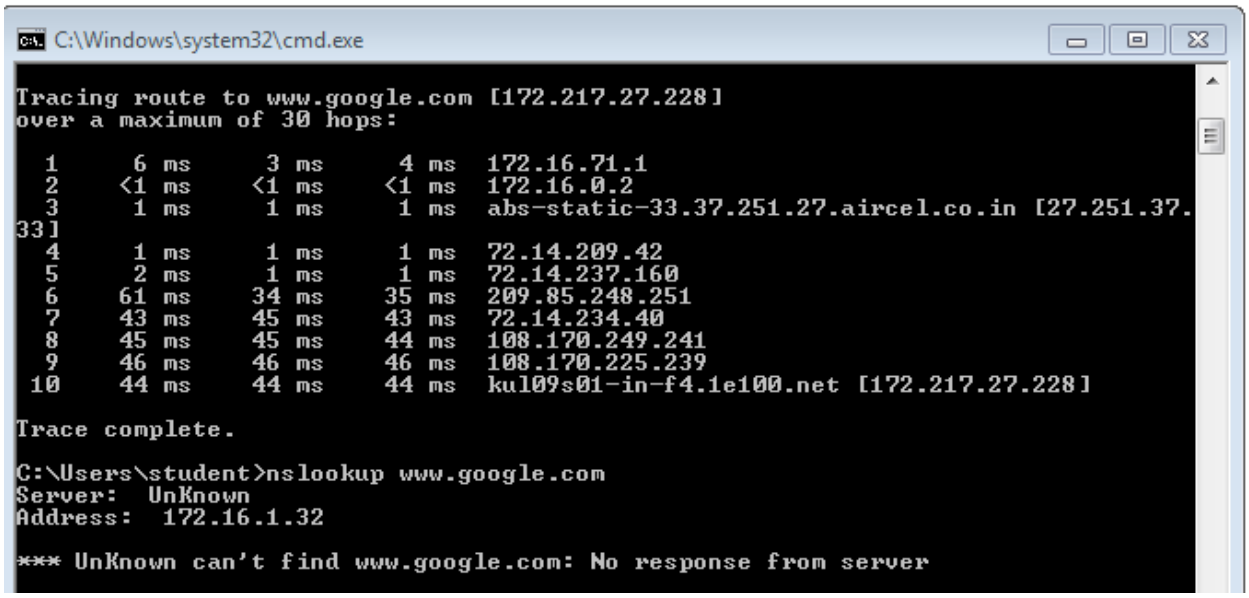
getmaccommand:



Trace route command:



Name Service lookup:



ns look up *domain_name* shows us the server name and displays one of the addresses.

Inferences:

- IP address is 32 bits in size.
- **'ping *destination_address*'**is used to verify IP-level connectivity. It sends few packets of data and then receives the packets back(ACK messages) then compares the number of packets sent and number of packets received to determine the lost packets (% loss).
- **Round Trip Time (RTT)** is the amount of time it takes to send a signal and receive acknowledgement of that signal.
- **'ipconfig'** displays IP address, Subnet Mask and Default Gateway for that device.
- **ICMP** - **Internet Control Message Protocol**. This protocol is used by *ping*.
- **DNS - Domain Name Server**. They maintain a directory of domain names and translate them to IP addresses.
- **HTTP(S) – Hypertext Transfer Protocol (Secure).** HTTP the foundation of data communication for the world wide web.
- **'nslookup' – Name Service look up**is used for querying the DNS and the IP address.

- **‘getmac’** – **get Medium Access Control**, gives the MAC address. It is a physical address of the device which cannot be changed. It is 48 bits address represented in hexadecimal format.
- **‘tracert*domain_name(destination)*’** – Trace route, is used to show several details about the path that a packet takes from the computer or device you're on to whatever destination you specify.This tracert option specifies the maximum number of hops in the search for the *target*. If you do not specify *MaxHops*, and a *target* has not been found by 30 hops, tracert will stop looking.

Experiment 2

Name: Debasya Sahoo

Registration number: 15BEC1111

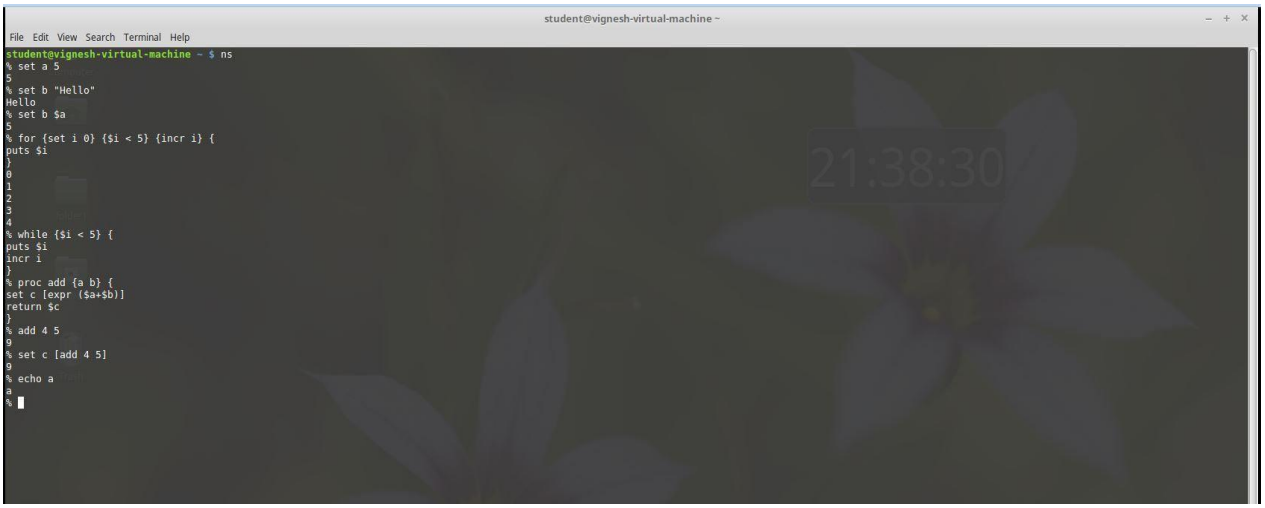
Aim:

- **Study of TCL (Tool Command Language).**
- **Simulation and analysis of point to point wired network using NS2**

Software: linux cmd prompt

List of cmds:

- **pwd: present working directory**
- **ls: listing**
- **ls -l : long listing format**
  presence of d->directory, - -> files
- **ls -ltr : long listing sorting in time reverse order**
- **clear: clear screen**
- **mkdir: make new directory**
- **cd: change directory**
- **cd .. : take to previous directory**
- **cd ~ : go to home directory**
- **gedit: graphic editor**
- **gedit test.txt & : separate process**
- **grep “hello” test1.txt : to find number of occurrence (case sensitive)**
- **grep -i “hello” test.txt: to find number of occurrence (case insensitive)**
- **grep -i “hello” test.txt | wc -l : word cound**
- **cp test.txt test1.txt : copy from one file to another**
- **cat test.txt : contents displayed in terminal**
- **ls D* : list all files starting with D**



Aim: Simulation and analysis of point to point wired network using NS2

Software used: NS2

Description:

- First create a new directory.
- Go to that directory and create a file sample2.tcl
- Open the file and write the tcl script.
- Save and compile and run the tcl script.

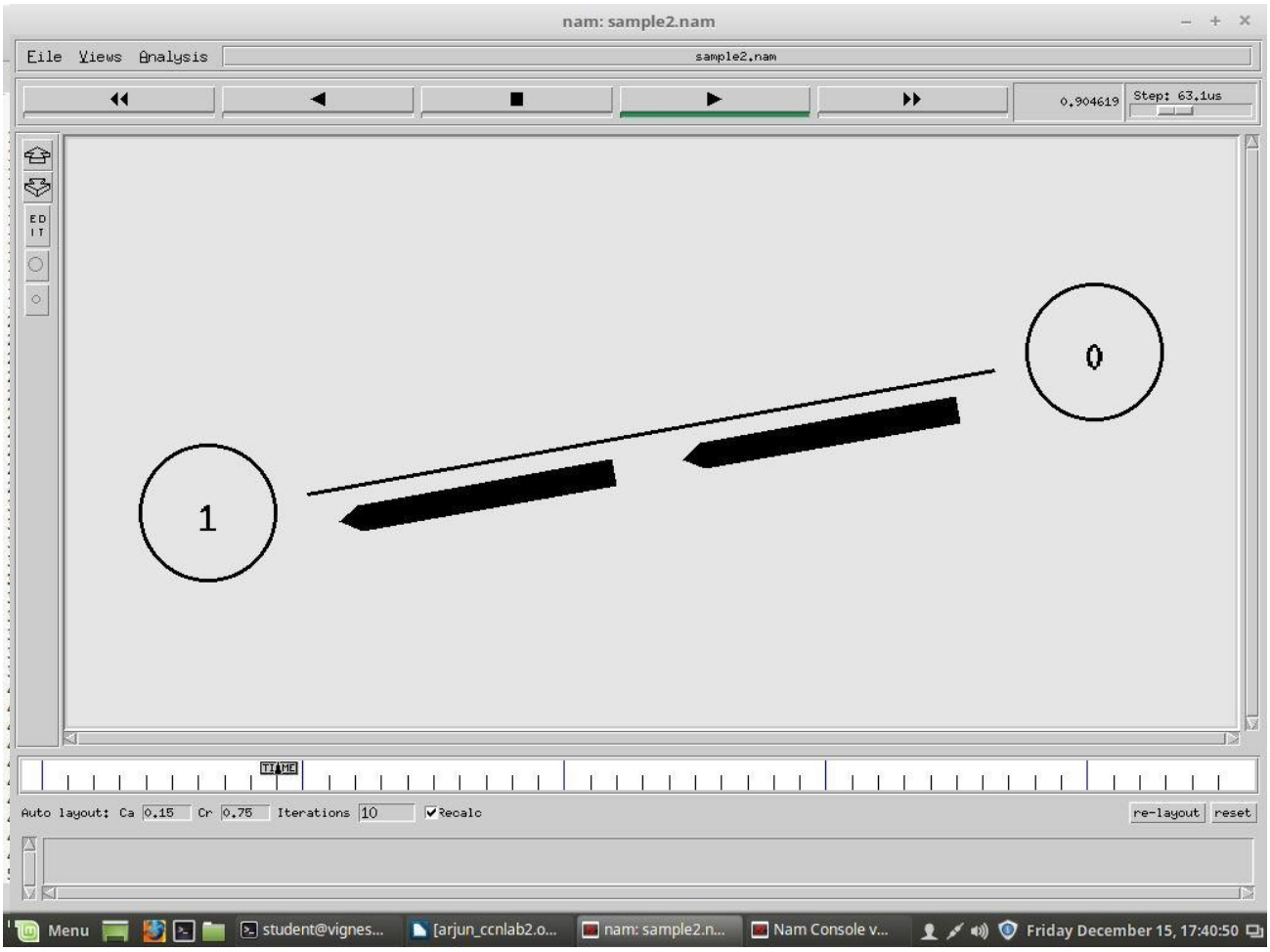
Code:

```
#Create an intance of Simulator class

set ns [new Simulator]

#Create a Trace file
set tf [open sample.tr w]
$ns trace-all $tf


#Create a NAM file
set nf [open sample2.nam w]
$ns namtrace-all $nf

#Create nodes and duplex links
set n0 [$ns
node] set n1
[$ns node]

# Create duplex link between source and target of 1Mb BW, 10ms propagation delay and
DropTail queueing mechanism
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

# Create Transport Agents

#Create UDP object
set udp0 [new Agent/UDP]

#Attach the UDP object to $n0
$ns attach-agent $n0 $udp0

#Create a NULL object
set null0 [new Agent/Null]

#Attach the NULL object to $n0
$ns attach-agent $n1 $null0
```

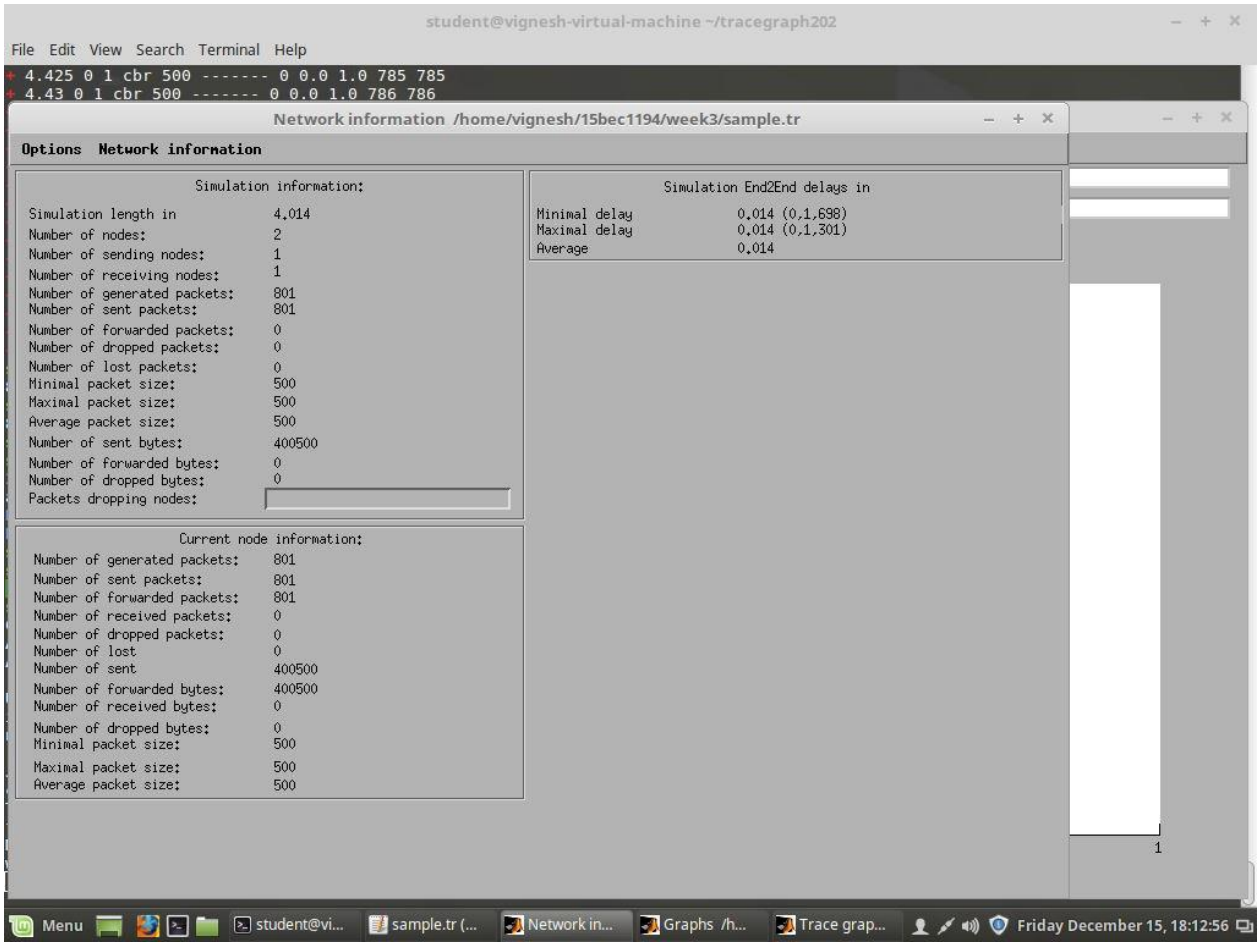
```
# Generate Application Traffic
# Constant Bit Rate Application (CBR)
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Virtually connect these 2 agents
$ns connect $udp0 $null0

#Finish Procedure
proc finish {} {
global ns nf tf
# flush-trace: Dumps all content to trace and nam file
$ns flush-
trace close $tf
close $nf
exec nam sample2.nam
& exit 0
}


$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

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

Output:

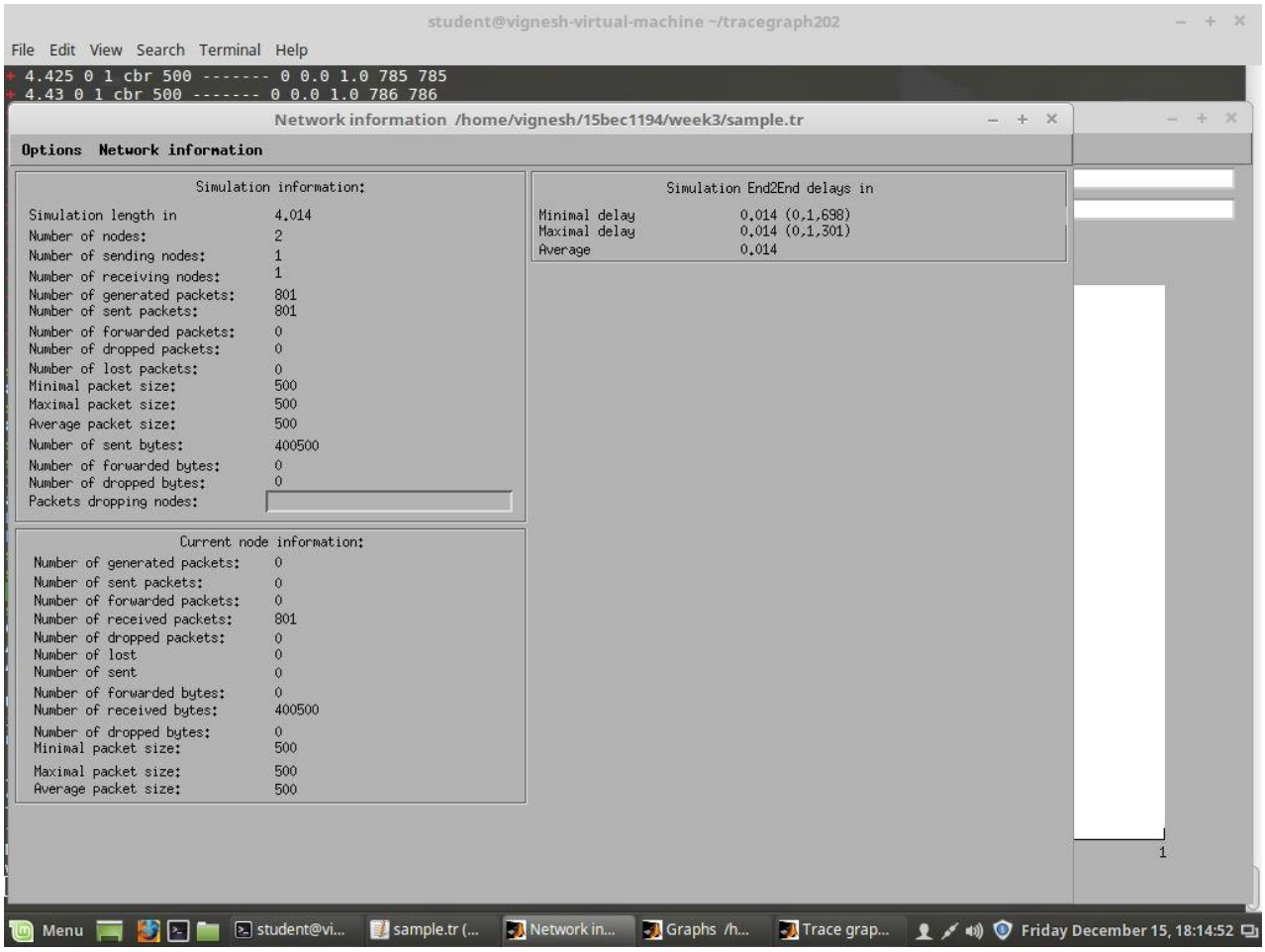
NS2 simulation of point to point wired network



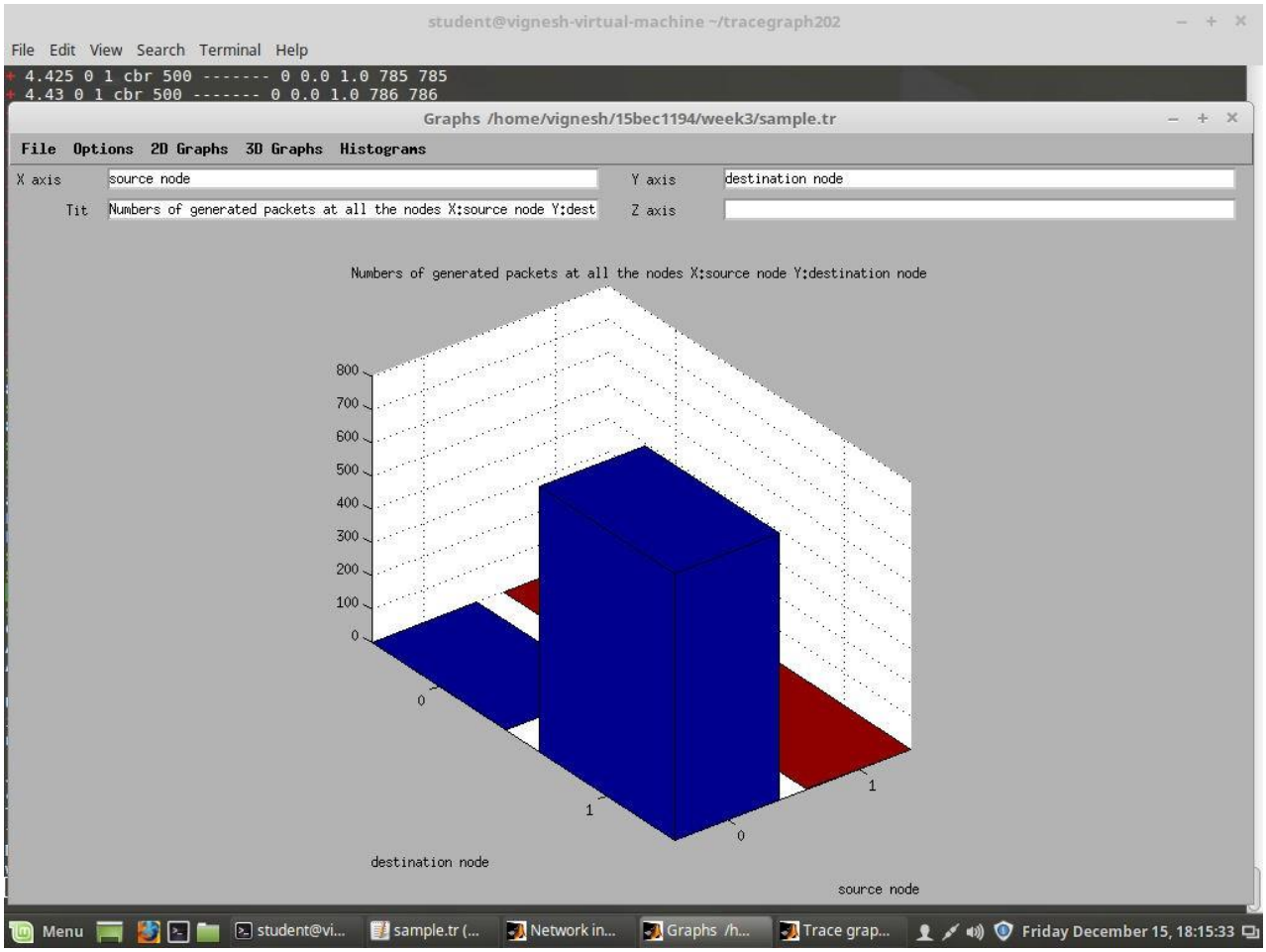
In trace graph we can observe avg end to end delay, PDR and another node information.



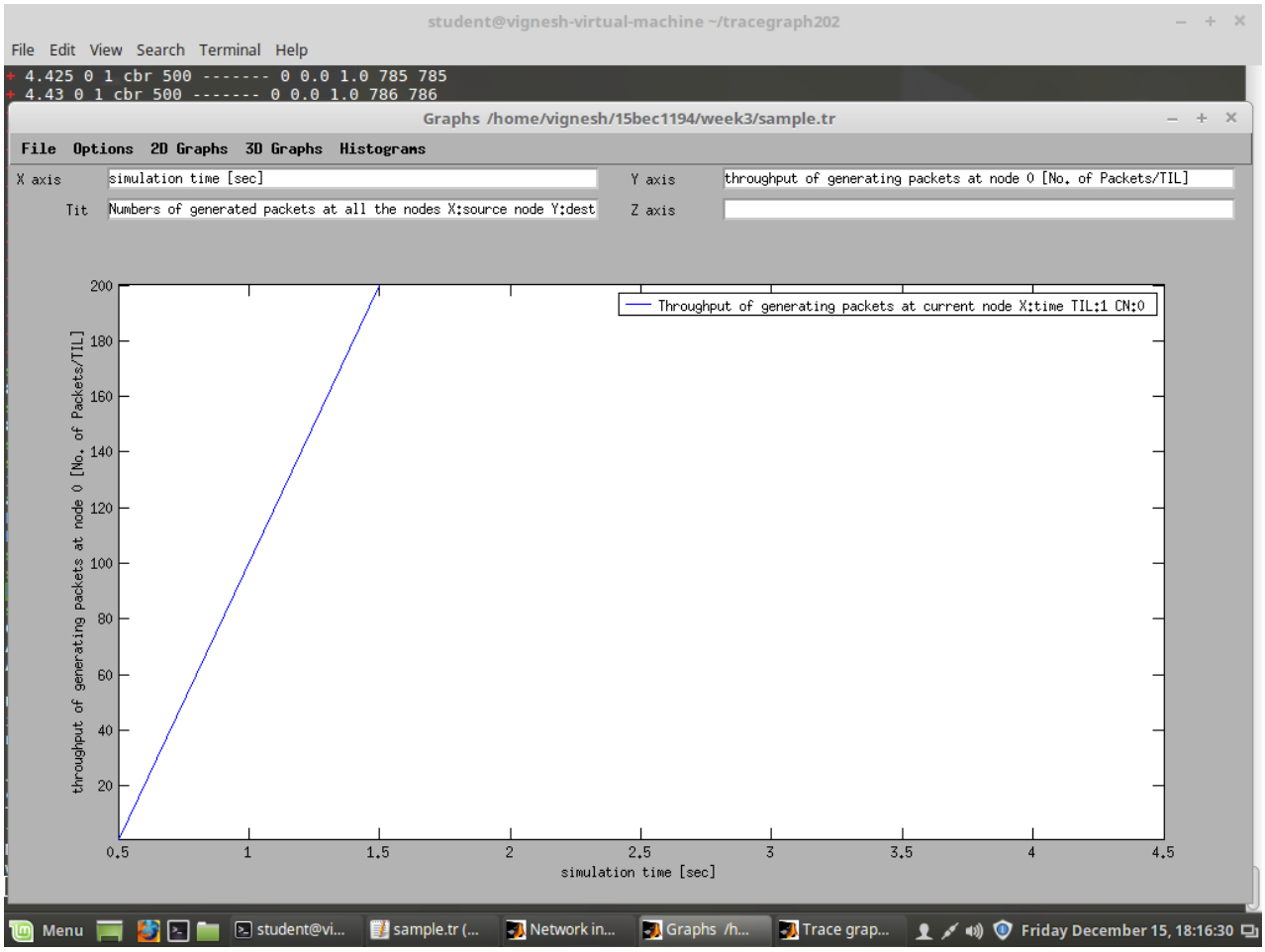
Network info of source node. Current node information of source node. We observe no packets are lost as link capacity is capable of servicing all the packets.



Current node info of destination



3-D graph of number of packets generated at all nodes.



Throughput of source node.

We changed interval to 2ms, we get:
Code:

#Create an intance of Simulator class
set ns [new Simulator]

#Create a Trace file
set tf [open sample.tr w]
$ns trace-all $tf

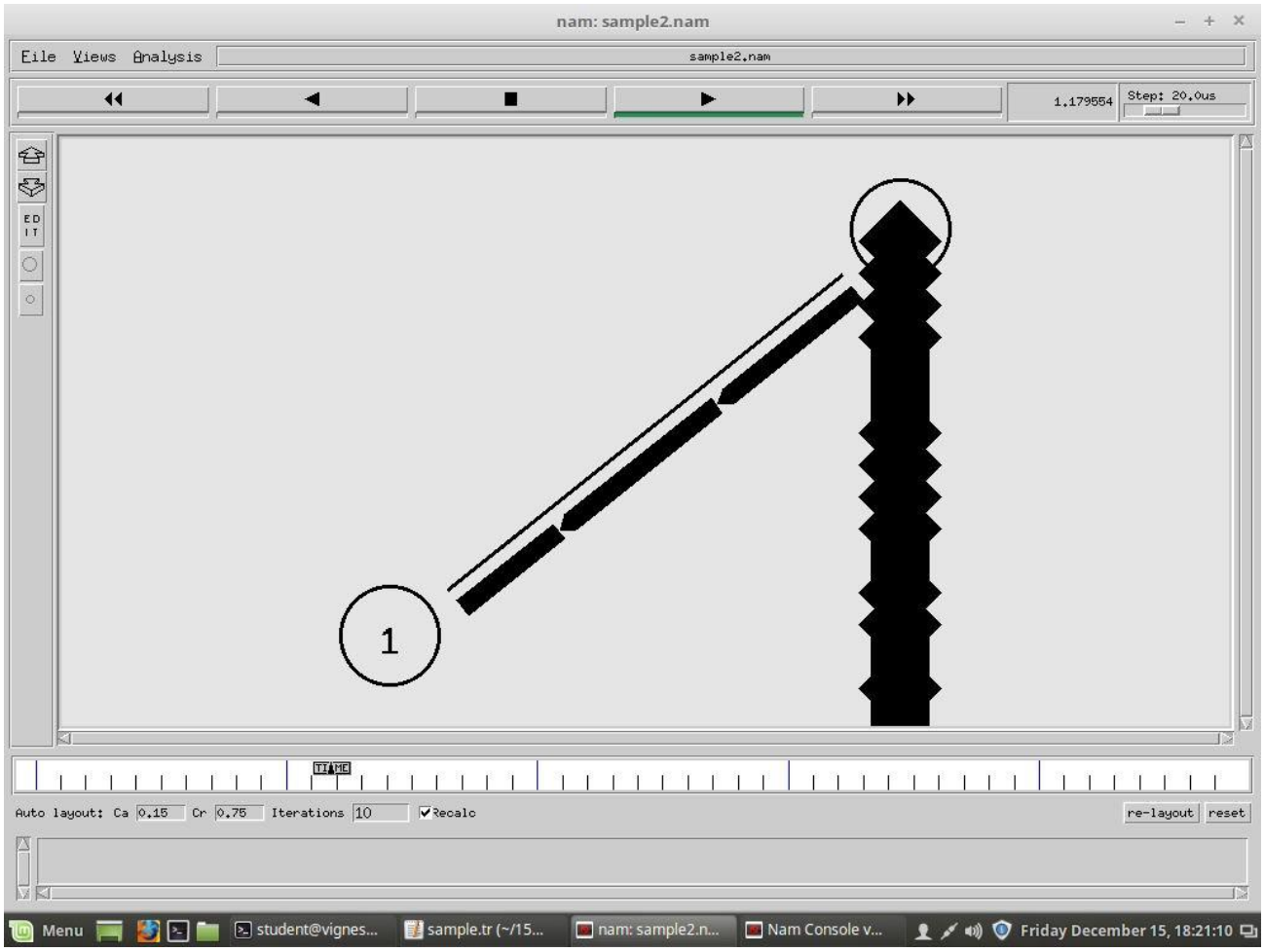
#Create a NAM file
set nf [open sample2.nam w]
$ns namtrace-all $nf

#Create nodes and duplex links
set n0 [$ns
node] set n1
[$ns node]

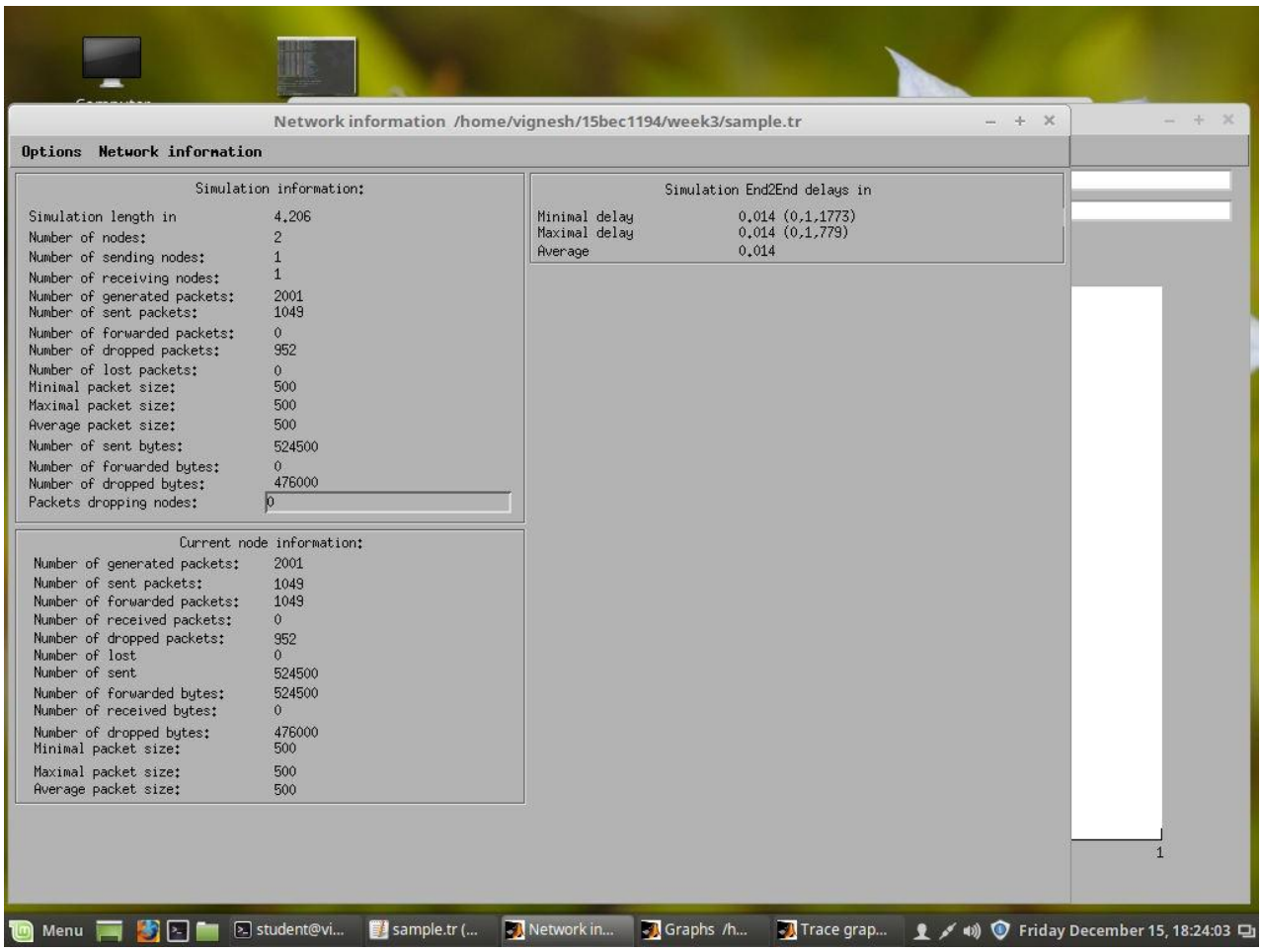
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

 # Create Transport Agents
set udp0 [new Agent/UDP]
$ns attach-agent $n0
$udp0 set null0 [new
Agent/Null]

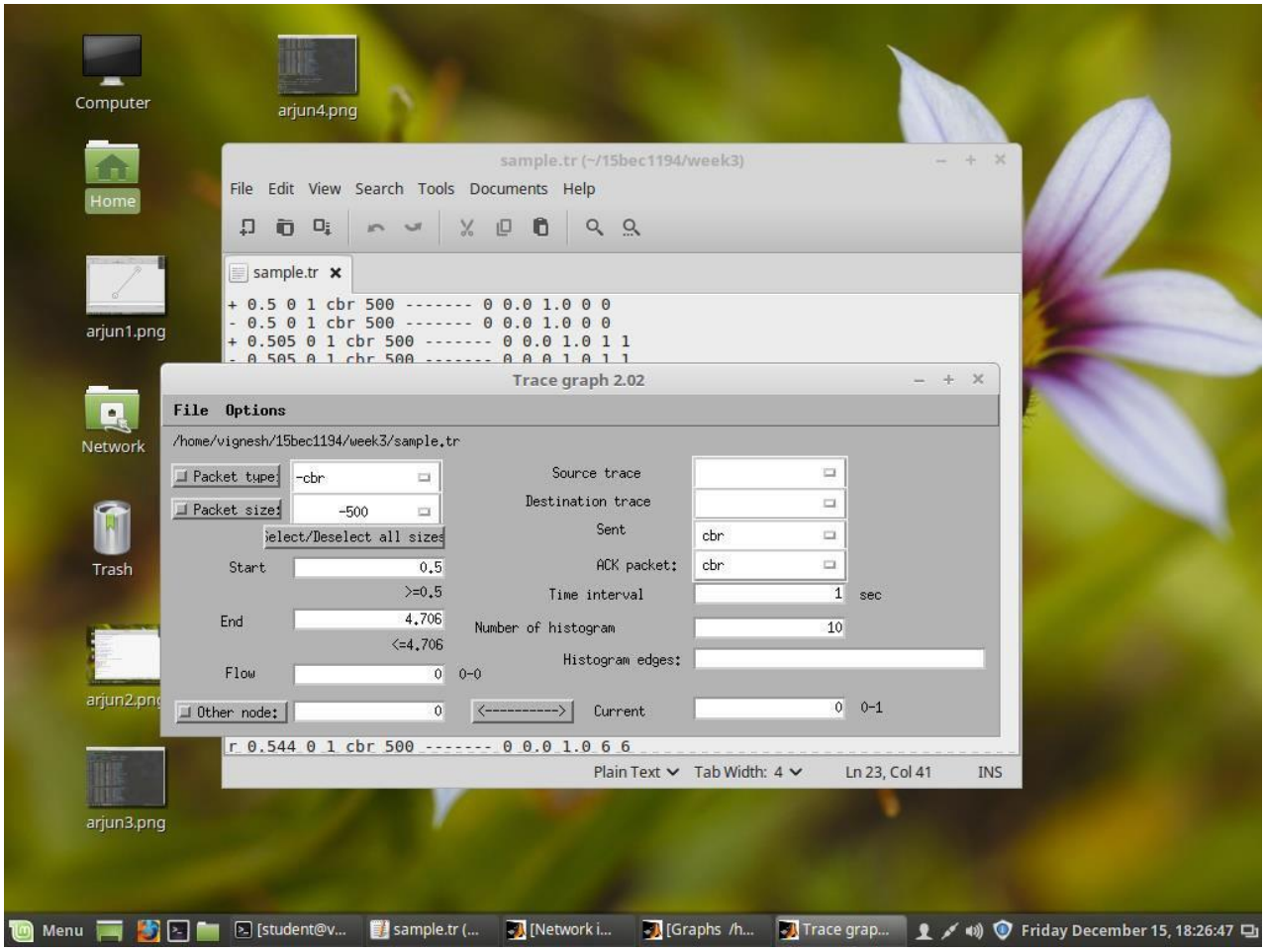
```
$ns attach-agent $n1 $null0

# Generate Application Traffic
# Constant Bit Rate Application (CBR)
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.002
$cbr0 attach-agent $udp0
#Virtually connect these 2 agents
$ns connect $udp0 $null0


#Finish Procedure
proc finish {} {
global ns nf tf
$ns flush-
trace close $tf
close $nf
exec nam sample2.nam
& exit 0
}


$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

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



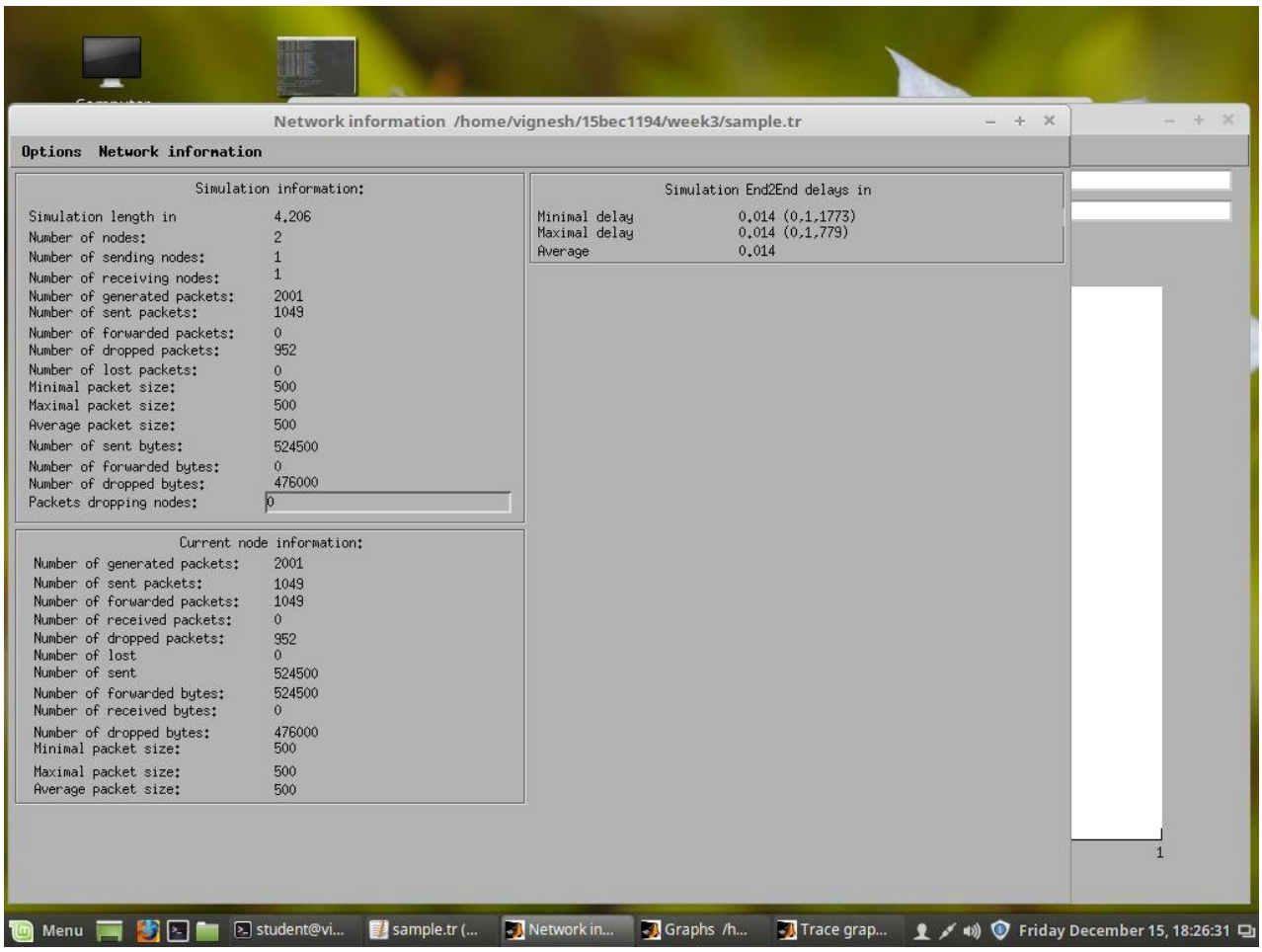
We observe that there are dropped packets involved. This is because our link capacity is 1Mbps, but we have set out time interval of packets as 2ms and packet size is 500 bytes. Thus 500*1/0.002*8=2Mbps. As we can see that our link capacity is less than that of packet delivery rate, some of the packets get dropped.



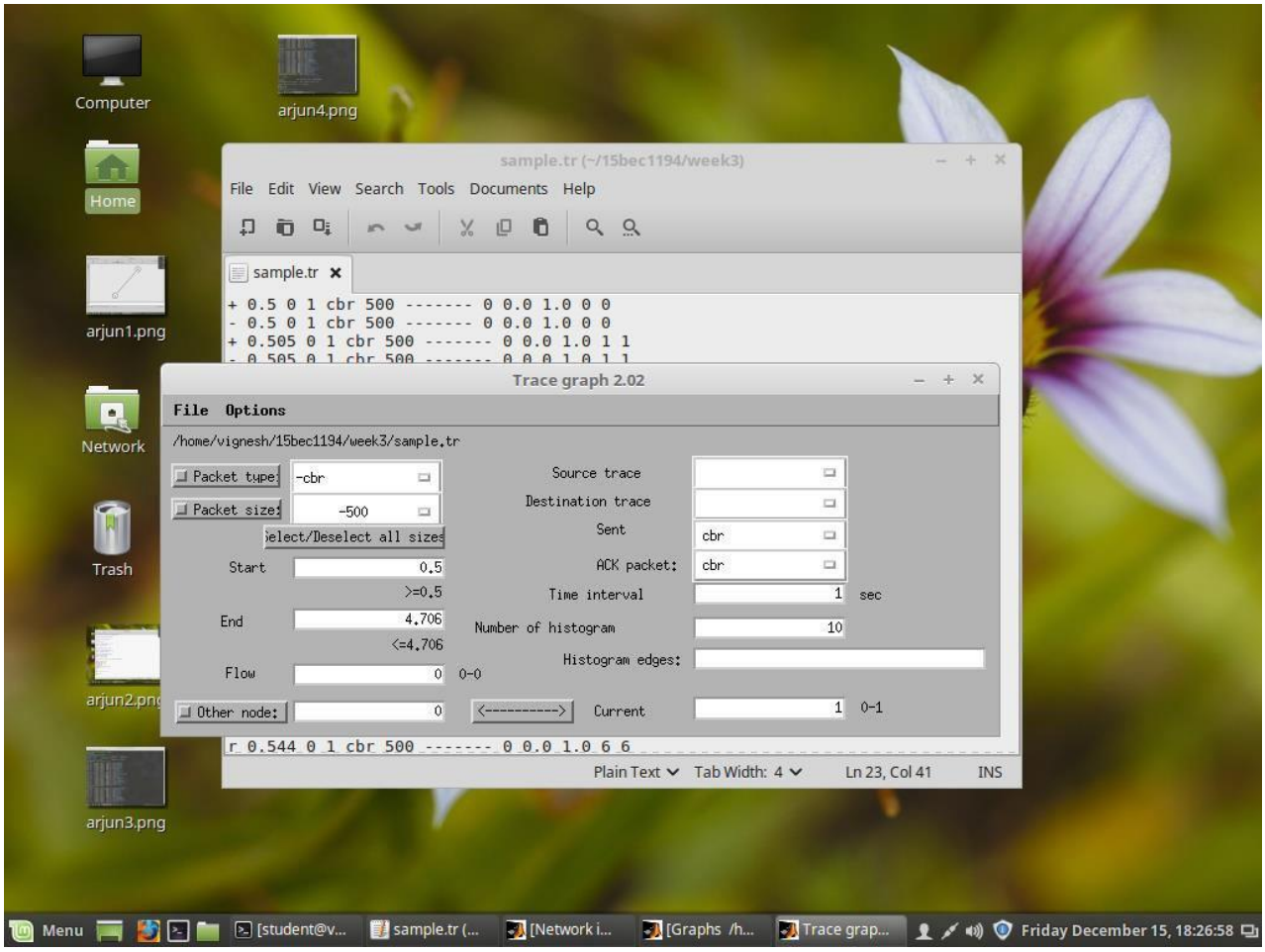
Network info of source node.



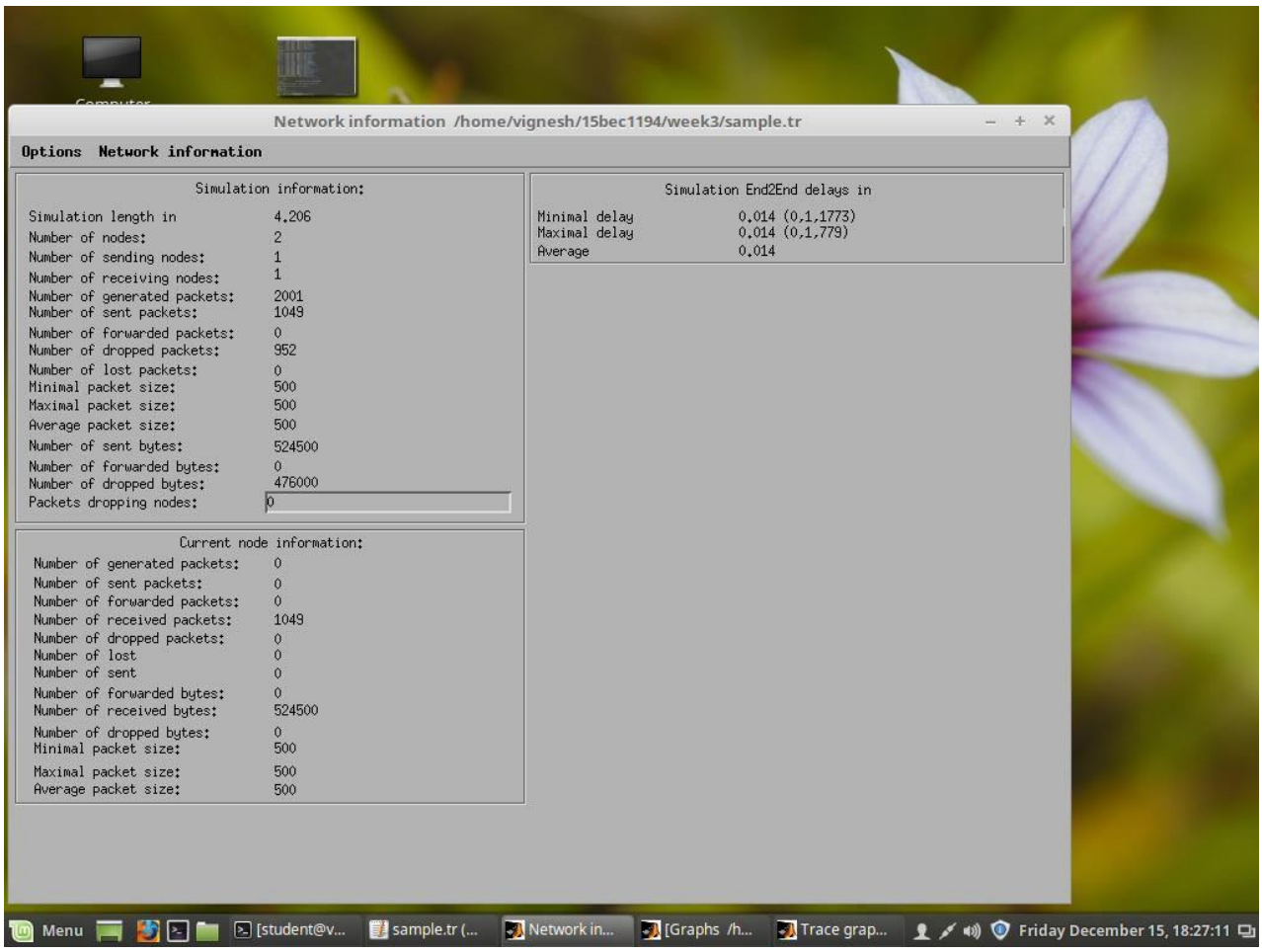
Selecting the source node by setting the current as 0.



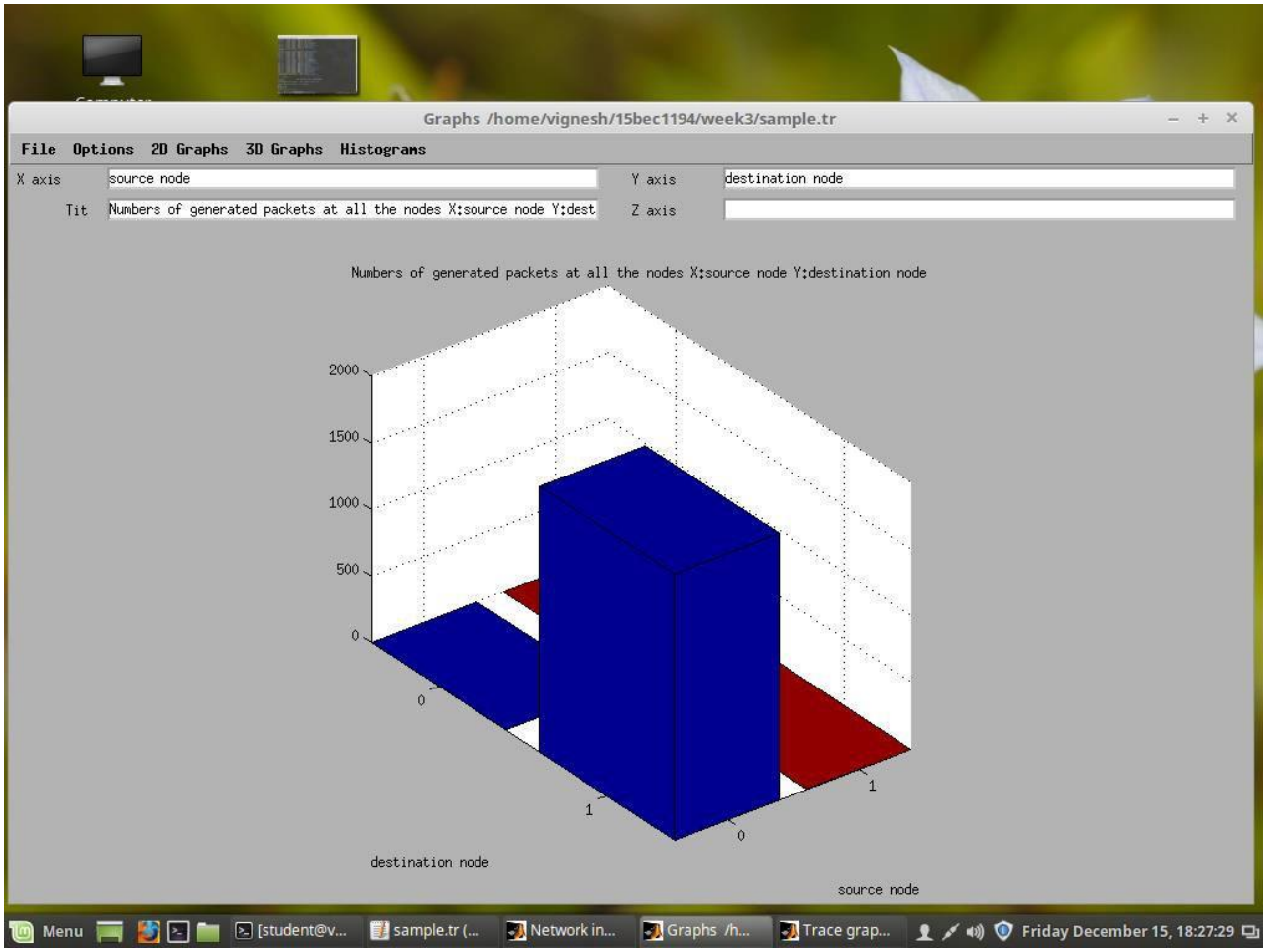
Information of the source node using trace graph



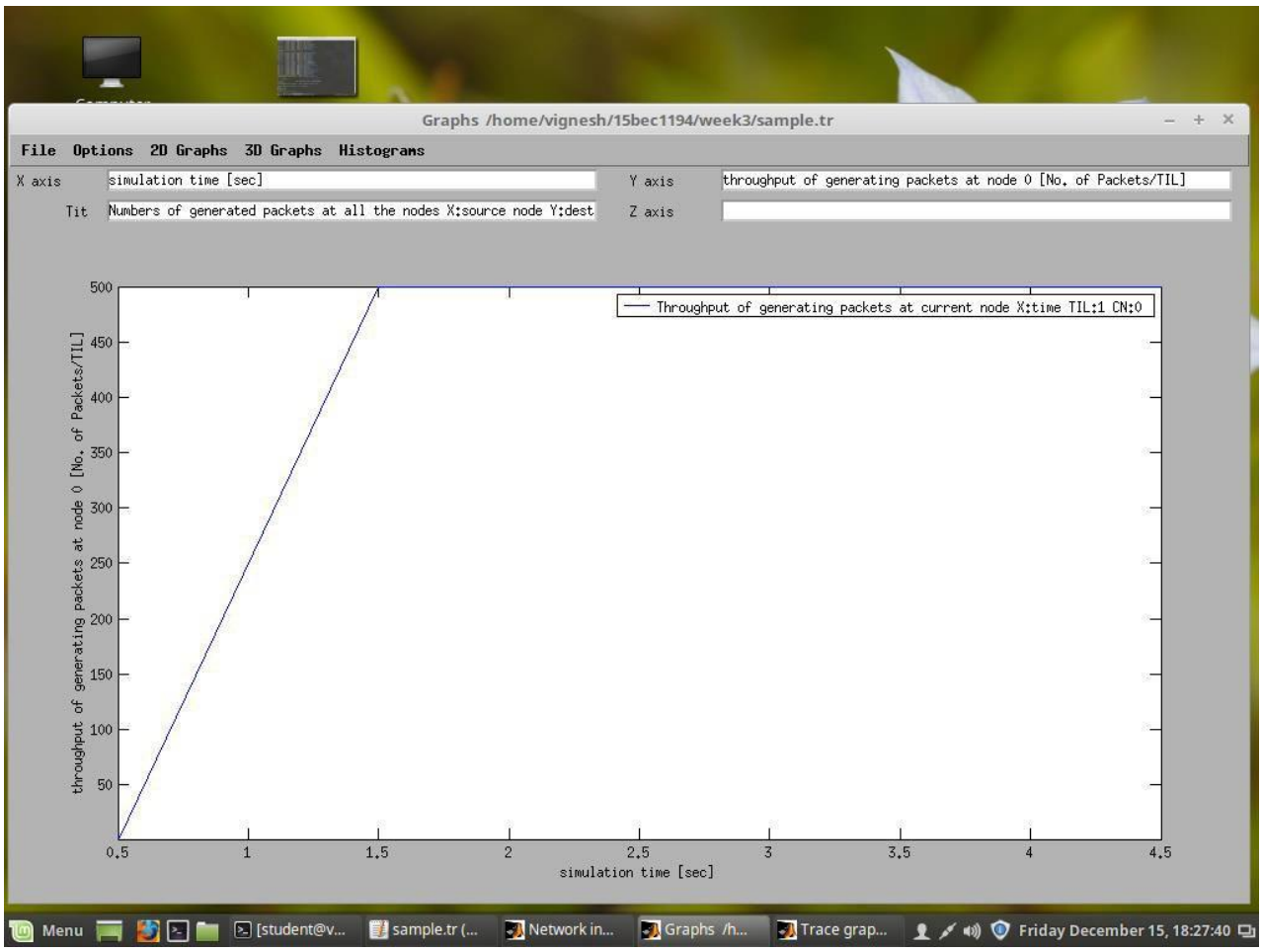
Selecting the destination node by making the current value equal to 1



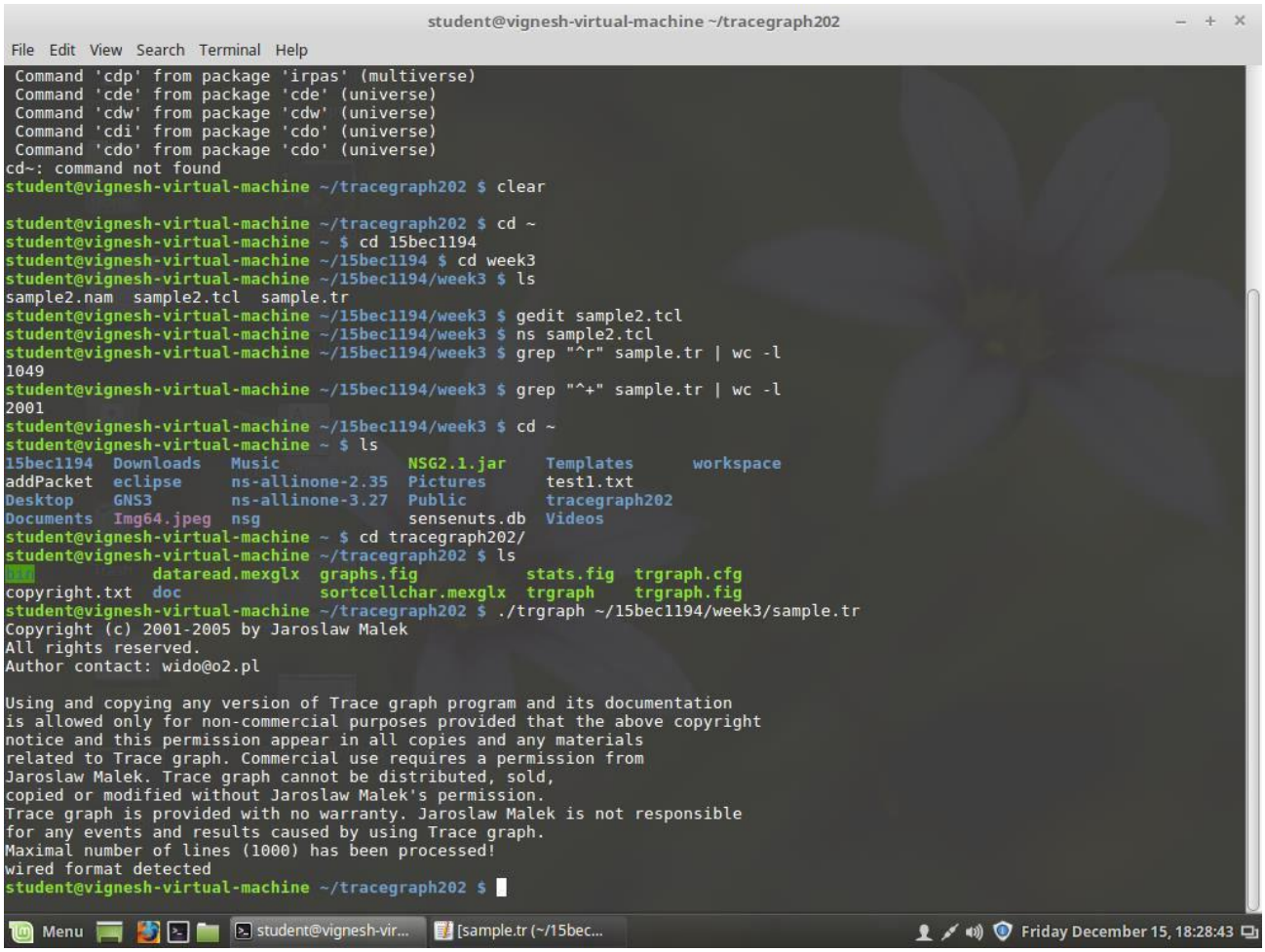
Information at the destination node using trace graph



3-D graph of number of packets generated at all nodes.



Throughput graph of the source node.



Inference: If the data generation rate is greater than the link rate, not all the packets get serviced, thus we observe that some of the packets getting dropped.

Result: The end to end delay is 0.014 seconds.

PDR in case 1 is 100
PDR in case 2 is 52.52

Thus study of linux commands and TCL, and simulation and analysis of point to point wired network using NS2 were successfully executed.

Experiment 3

COMPUTER COMMUNICATION LAB ECE4004

Name: Debasya Sahoo

Registration number: 15BEC1111

Aim:

- Simulation and analysis of queuing mechanism in star topology using NS2.

Software: NS2 and Trace Graph

Description:

- First create a new directory.
- Go to that directory and create a file sample3.tcl
- Open the file and write the tcl script.
- Save and compile and run the tcl script
- Use grep commands for pdr calculation.
- Using trace graph for visual simulation.

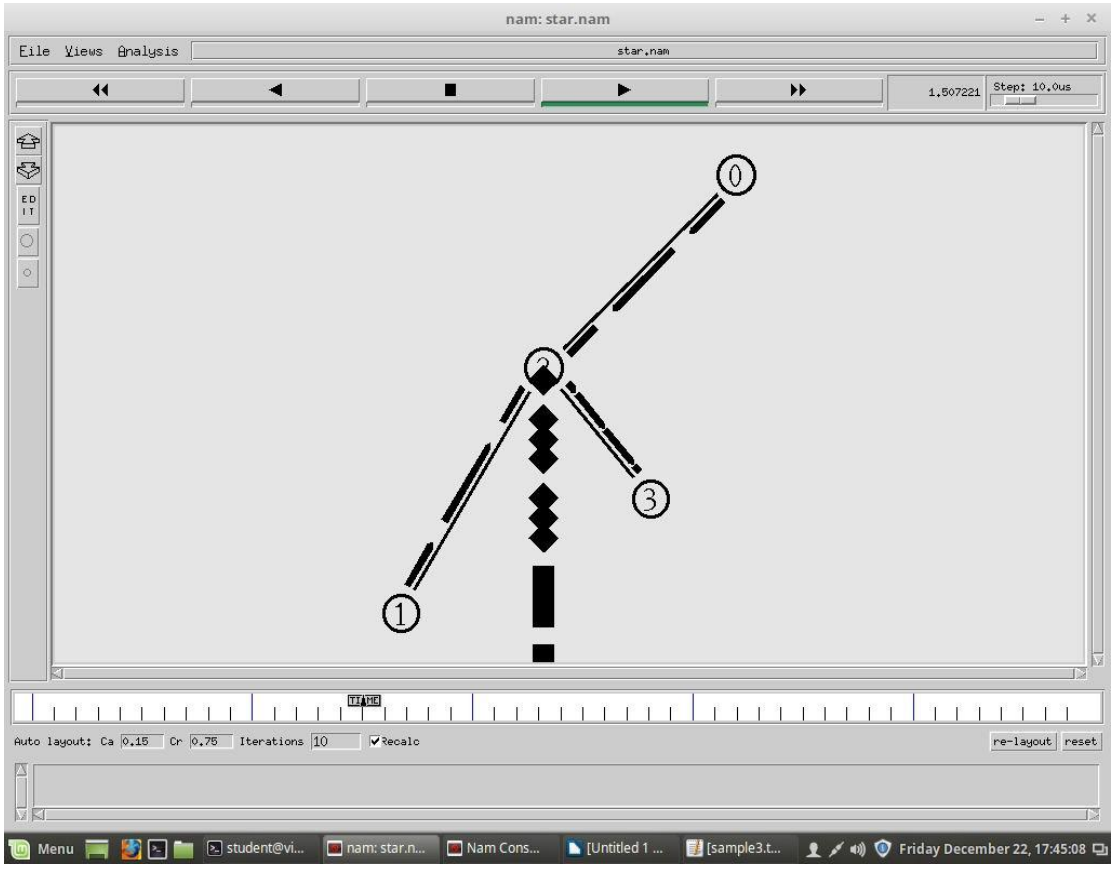
Code:

```
#Create an instance of simulator class
set ns [new Simulator]

#Create a Trace file
set tf [open star.tr w]
$ns trace-all $tf

#Create a NAM file
set nf [open star.nam w]
$ns namtrace-all $nf

#Create nodes and duplex links

set num 4

for {set i 0} {$i< $num} {incri} {
set n($i) [$ns node]
}

# Create duplex link between source and target of 1Mb BW, 10ms propagation delay and DropTail queueing mechanism
$ns duplex-link $n(0) $n(2) 1Mb 10ms DropTail
```

```
$ns duplex-link $n(1) $n(2) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 1Mb 10ms DropTail

# Create Transport Agents

#Create UDP object
set udp0 [new Agent/UDP]

#Attach the UDP object to $n0
$ns attach-agent $n(0) $udp0

# Generate Application Traffic
# Constant Bit Rate Application (CBR)
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a NULL object and attach it to node 3
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

#Virtually connect these 2 agents
$ns connect $udp0 $null0


# Doing the same for node 1
set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

$ns connect $udp1 $null0

#Finish Procedure
proc finish {} {
global ns nftf
# flush-trace: Dumps all content to trace and nam file
$ns flush-trace
close $nf close
$tf
exec namstar.nam&
exit 0
}

#Specifying the start and the stop time
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
```

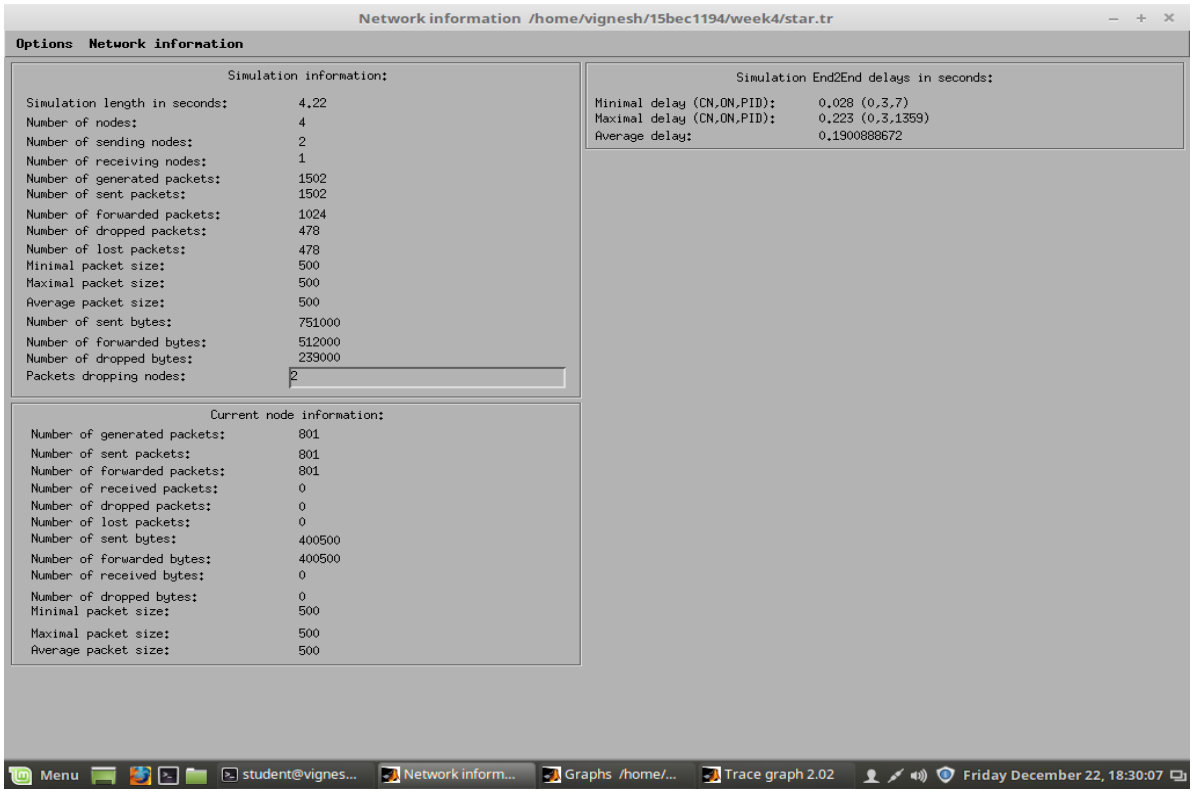
```
$ns at 4.5 "$cbr0 stop"
$ns at 4.5 "$cbr1 stop"
$ns at 5.0 "finish"
$ns run
```

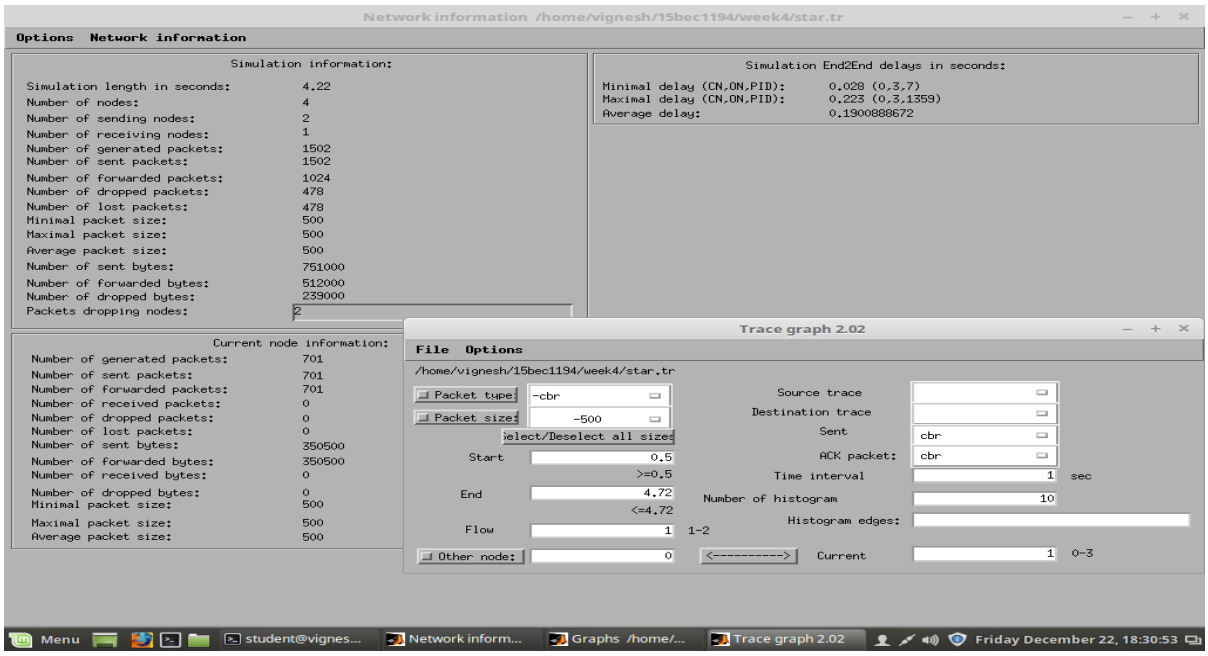
Output:

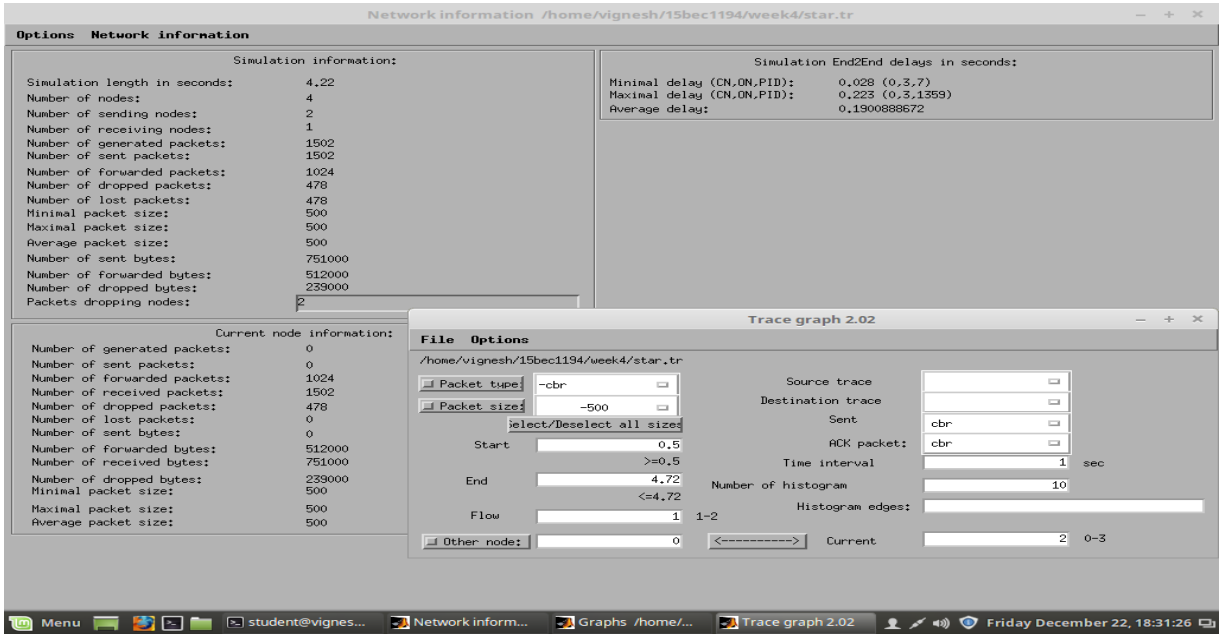


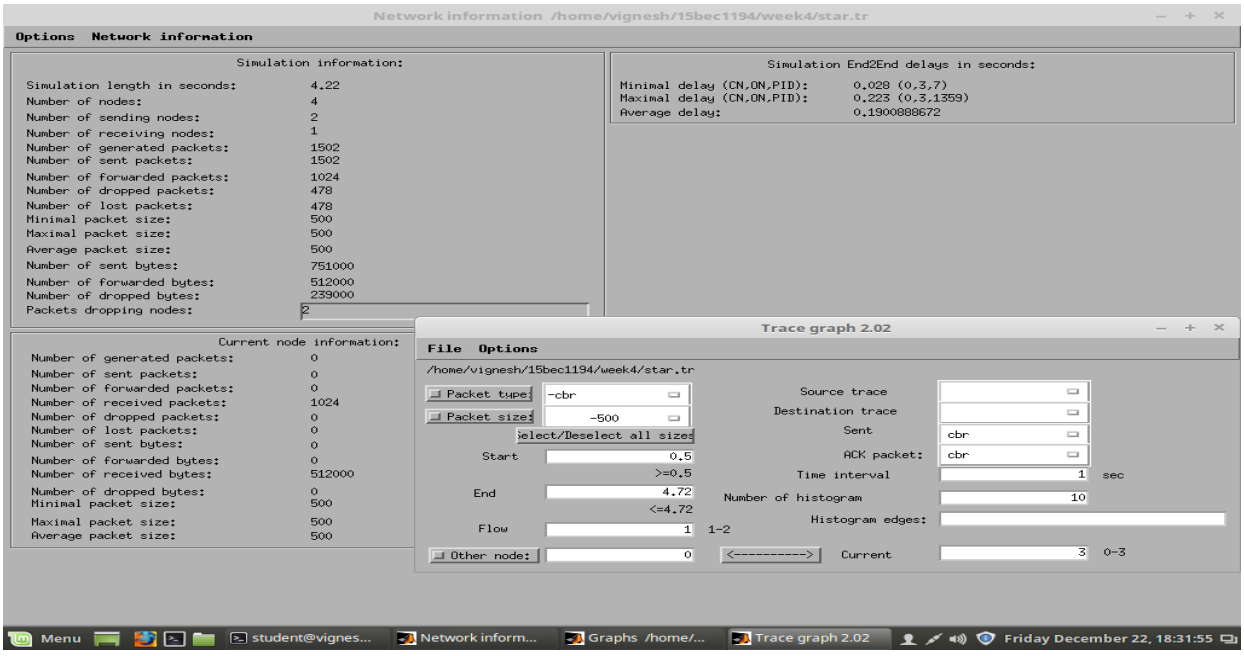
Calculating pdr using grep commands: Trace

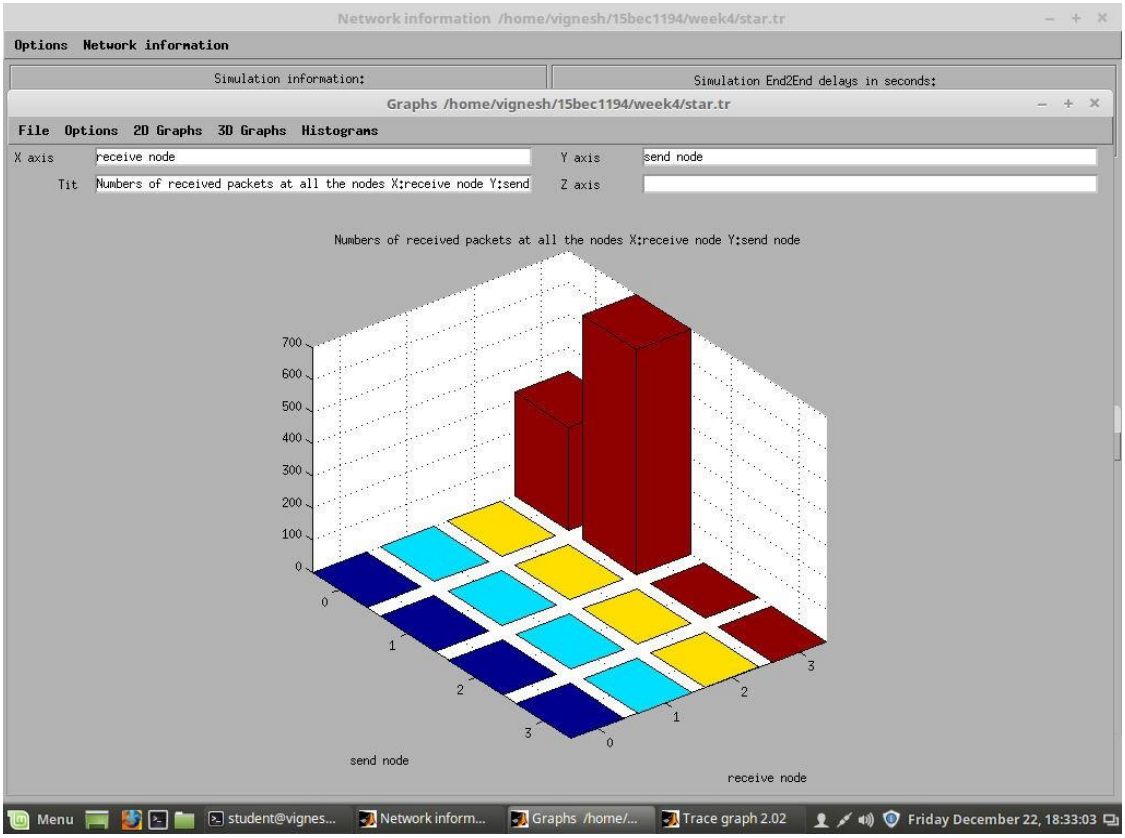
graph simulation output for all nodes:

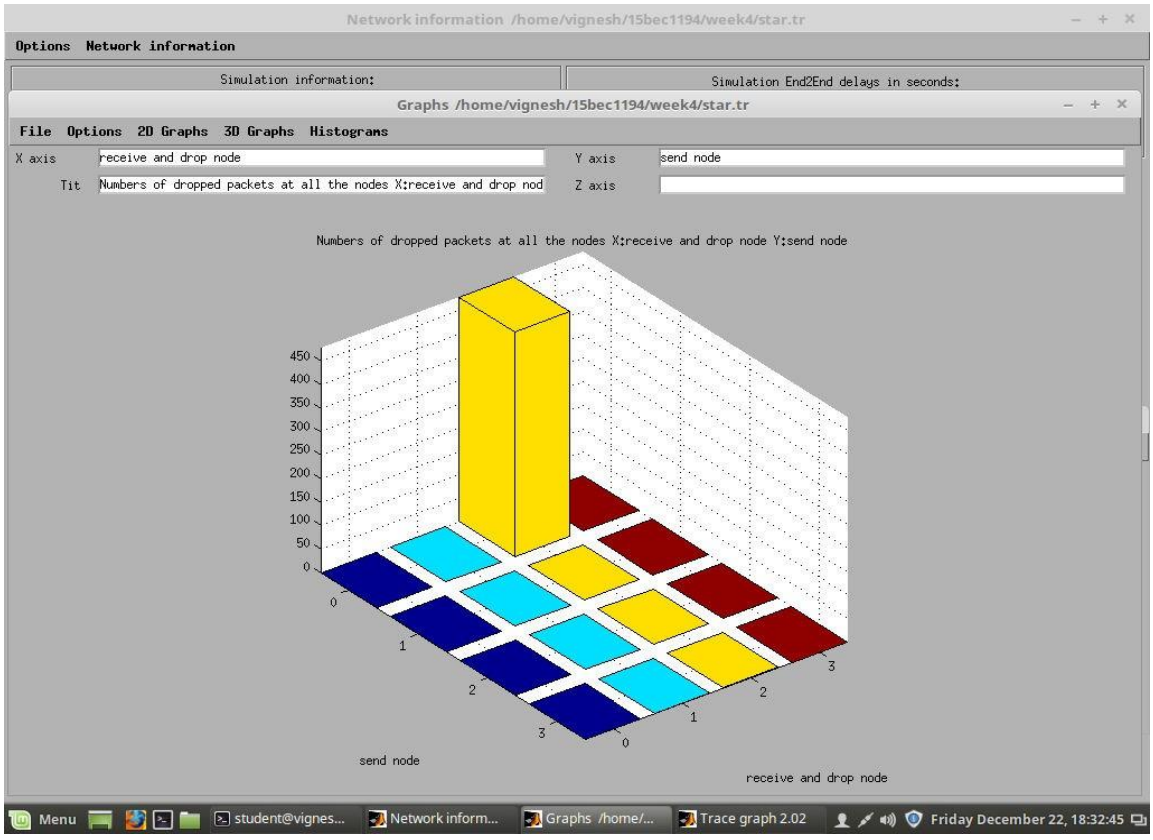



Network information /home/vignesh/15bec1194/week4/star.tr
Options Network information
Simulation information:
Simulation length in seconds: 4.22
Number of nodes: 4
Number of sending nodes: 2
Number of receiving nodes: 1
Number of generated packets: 1502
Number of sent packets: 1502
Number of forwarded packets: 1024
Number of dropped packets: 478
Number of lost packets: 478
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Number of sent bytes: 751000
Number of forwarded bytes: 512000
Number of dropped bytes: 239000
Packets dropping nodes: 2
Simulation End2End delays in seconds:
Minimal delay (CN,ON,PID): 0.028 (0,3,7)
Maximal delay (CN,ON,PID): 0.223 (0,3,1359)
Average delay: 0.1900888672
Current node information:
Number of generated packets: 701
Number of sent packets: 701
Number of forwarded packets: 701
Number of received packets: 0
Number of dropped packets: 0
Number of lost packets: 0
Number of sent bytes: 350500
Number of forwarded bytes: 350500
Number of received bytes: 0
Number of dropped bytes: 0
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Trace graph 2.02
File Options
/home/vignesh/15bec1194/week4/star.tr
Packet type: -cbr
Packet size: -500
Select/Deselect all sizes
Start 0.5 >=0.5
End 4.72 <=4.72
Flow 1 1-2
Other node: 0
Source trace
Destination trace
Sent cbr
ACK packet: cbr
Time interval 1 sec
Number of histogram 10
Histogram edges:
<---------->
Current 1 0-3
Menu
student@vignes...
Network inform...
Graphs /home/...
Trace graph 2.02
Friday December 22, 18:30:53


Network information /home/vignesh/15bec1194/week4/star.tr
Options Network information
Simulation information:
Simulation length in seconds: 4.22
Number of nodes: 4
Number of sending nodes: 2
Number of receiving nodes: 1
Number of generated packets: 1502
Number of sent packets: 1502
Number of forwarded packets: 1024
Number of dropped packets: 478
Number of lost packets: 478
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Number of sent bytes: 751000
Number of forwarded bytes: 512000
Number of dropped bytes: 239000
Packets dropping nodes: 2
Simulation End2End delays in seconds:
Minimal delay (CN,ON,PID): 0.028 (0,3,7)
Maximal delay (CN,ON,PID): 0.223 (0,3,1359)
Average delay: 0.1900888672
Current node information:
Number of generated packets: 0
Number of sent packets: 0
Number of forwarded packets: 1024
Number of received packets: 1502
Number of dropped packets: 478
Number of lost packets: 0
Number of sent bytes: 0
Number of forwarded bytes: 512000
Number of received bytes: 751000
Number of dropped bytes: 239000
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Trace graph 2.02
File Options
/home/vignesh/15bec1194/week4/star.tr
Packet type: -cbr
Packet size: -500
Select/Deselect all sizes
Start 0.5 >=0.5
End 4.72 <=4.72
Flow 1 1-2
Other node: 0
Source trace
Destination trace
Sent cbr
ACK packet: cbr
Time interval 1 sec
Number of histogram 10
Histogram edges:
<---------->
Current 2 0-3
Menu
student@vignes...
Network inform...
Graphs /home/...
Trace graph 2.02
Friday December 22, 18:31:26


Network information /home/vignesh/15bec1194/week4/star.tr
Options Network information
Simulation information:
Simulation length in seconds: 4.22
Number of nodes: 4
Number of sending nodes: 2
Number of receiving nodes: 1
Number of generated packets: 1502
Number of sent packets: 1502
Number of forwarded packets: 1024
Number of dropped packets: 478
Number of lost packets: 478
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Number of sent bytes: 751000
Number of forwarded bytes: 512000
Number of dropped bytes: 239000
Packets dropping nodes: 2
Simulation End2End delays in seconds:
Minimal delay (CN,ON,PID): 0.028 (0,3,7)
Maximal delay (CN,ON,PID): 0.223 (0,3,1359)
Average delay: 0.1900888672
Current node information:
Number of generated packets: 0
Number of sent packets: 0
Number of forwarded packets: 0
Number of received packets: 1024
Number of dropped packets: 0
Number of lost packets: 0
Number of sent bytes: 0
Number of forwarded bytes: 0
Number of received bytes: 512000
Number of dropped bytes: 0
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Trace graph 2.02
File Options
/home/vignesh/15bec1194/week4/star.tr
Packet type: -cbr
Packet size: -500
Select/Deselect all sizes
Start 0.5 >=0.5
End 4.72 <=4.72
Flow 1 1-2
Other node: 0
Source trace
Destination trace
Sent cbr
ACK packet: cbr
Time interval 1 sec
Number of histogram 10
Histogram edges:
<---------->
Current 3 0-3
Menu
student@vignes...
Network inform...
Graphs /home/...
Trace graph 2.02
Friday December 22, 18:31:55





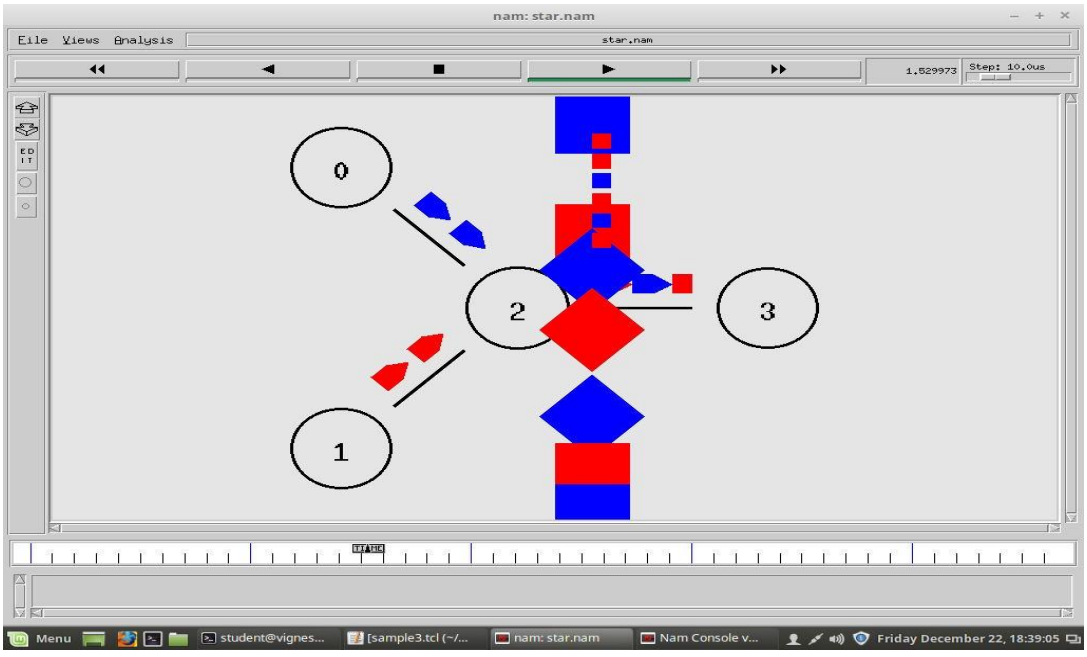
Biased dropping of packets of only node 0

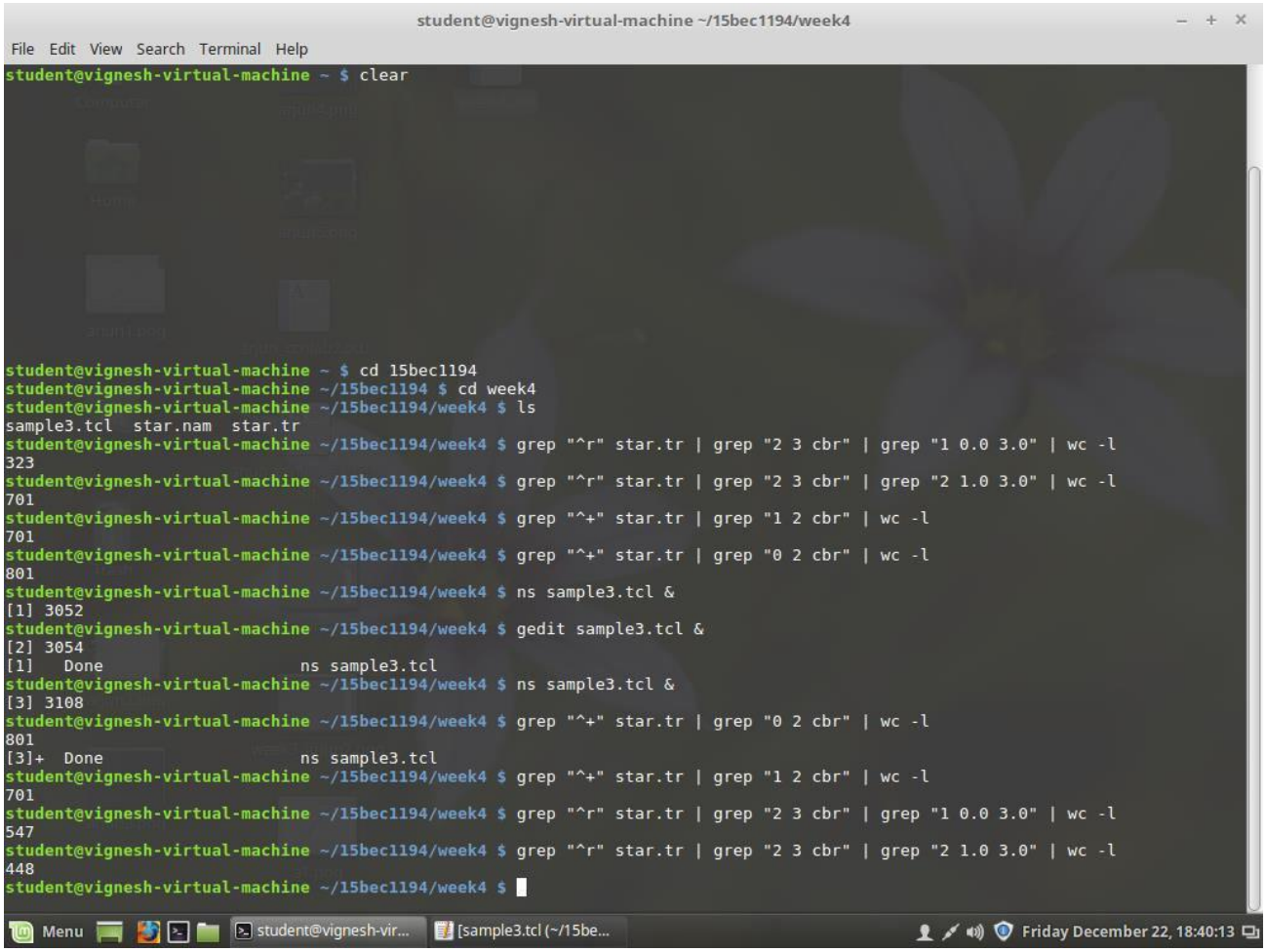
Code 2 using SFQ mechanism.

```
#Create an instance of simulator class
set ns [new Simulator]

set tf [open star.tr w]
$ns trace-all $tf
set nf [open star.nam w]
$ns namtrace-all

$nf set num 4

$ns color 1 Blue
$ns color 2 Red

for {set i 0} {$i< $num} {incri} {
set n($i) [$ns node]
}


$ns duplex-link $n(0) $n(2) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1Mb 10ms DropTail
# SFQ - Stochastic Fair Queing
$ns duplex-link $n(2) $n(3) 1Mb 10ms SFQ



# Duplex link orientation
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(1) $n(2) orient right-up
$ns duplex-link-op $n(2) $n(3) orient right

# queue position see how the buffer gets filled
$ns duplex-link-op $n(2) $n(3) queuePos 0.5

set udp0 [new Agent/UDP]
$udp0 set class_ 1
$ns attach-agent $n(0) $udp0

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

set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

$ns connect $udp0 $null0
```

```
set udp1 [new Agent/UDP]
$udp1 set class_ 2

$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

$ns connect $udp1

$null0 proc finish {} {
global ns nftf
$ns flush-trace
close $nf close
$tf
exec namstar.nam&
exit 0
}

$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.5 "$cbr0 stop"
$ns at 4.5 "$cbr1 stop"
$ns at 5.0 "finish"

$ns run
```

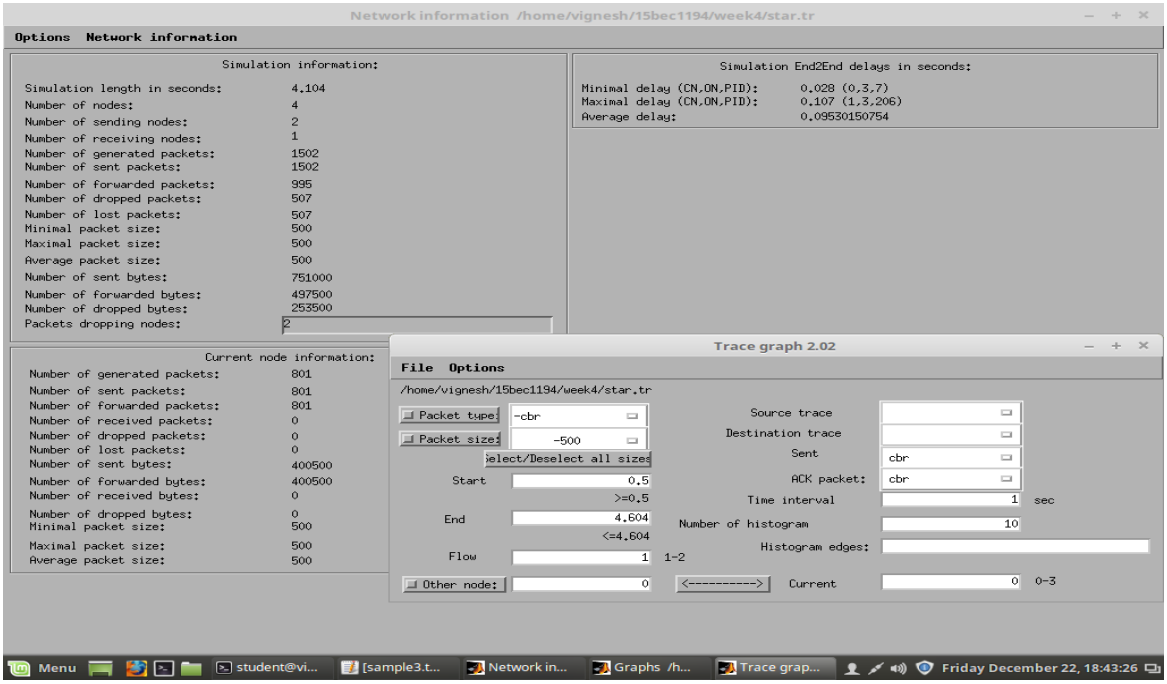
Output:

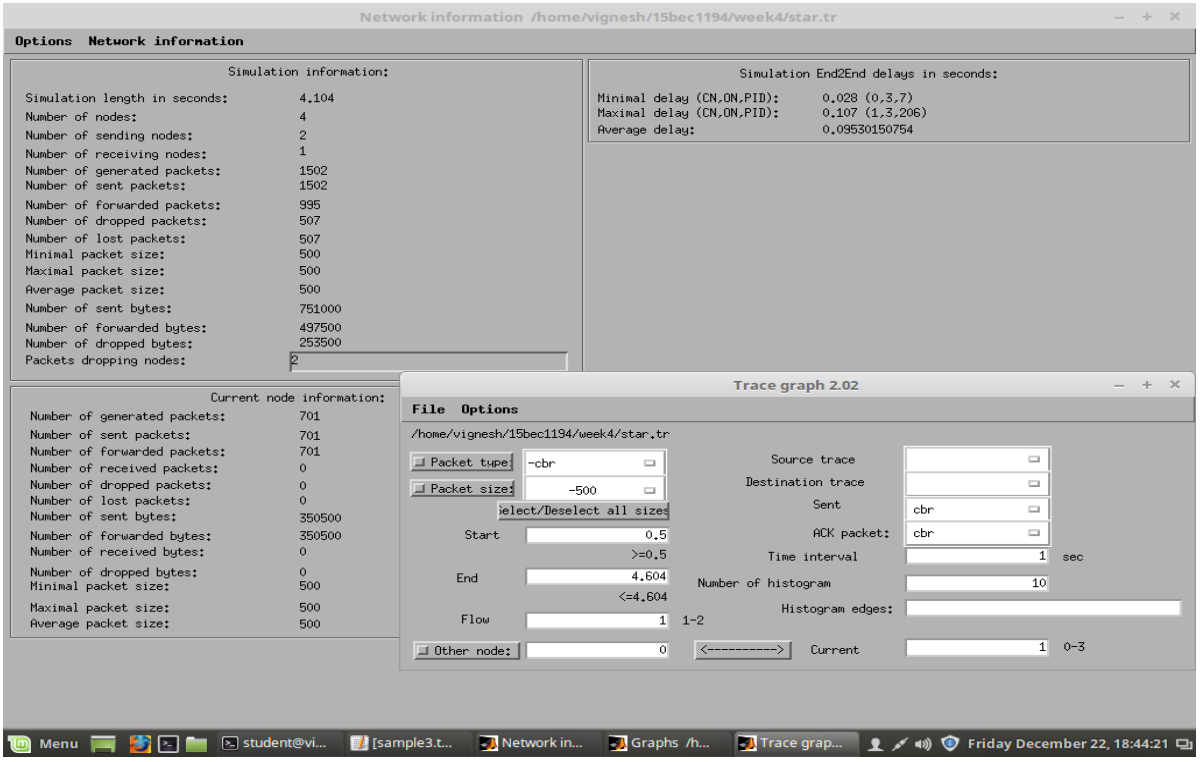


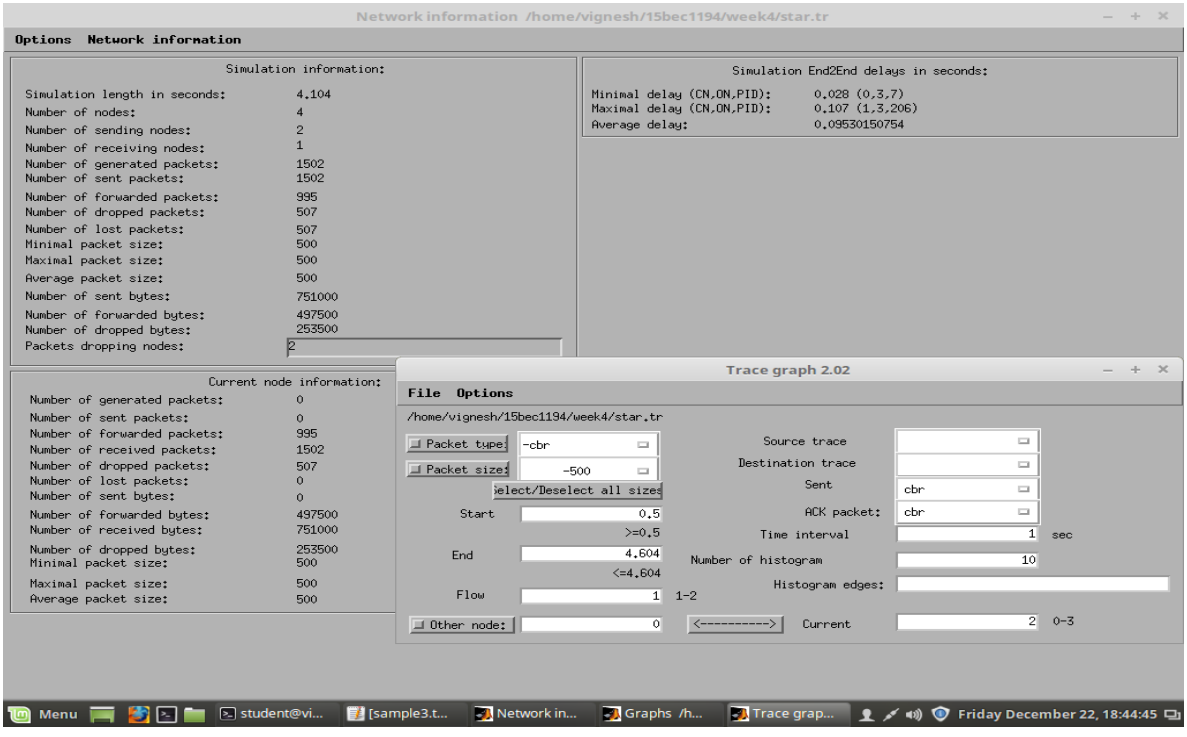


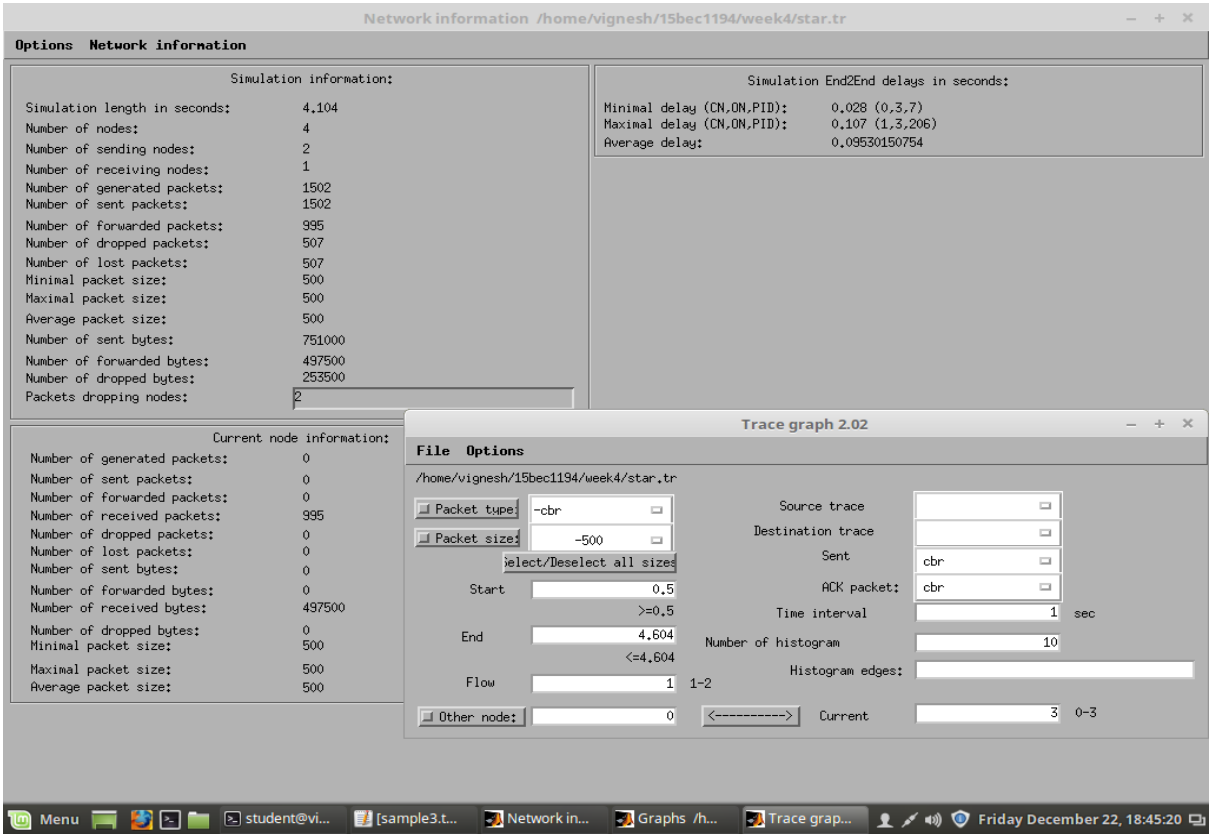
Calculating pdr using grep commands

Trace graph simulation outputs for different nodes.

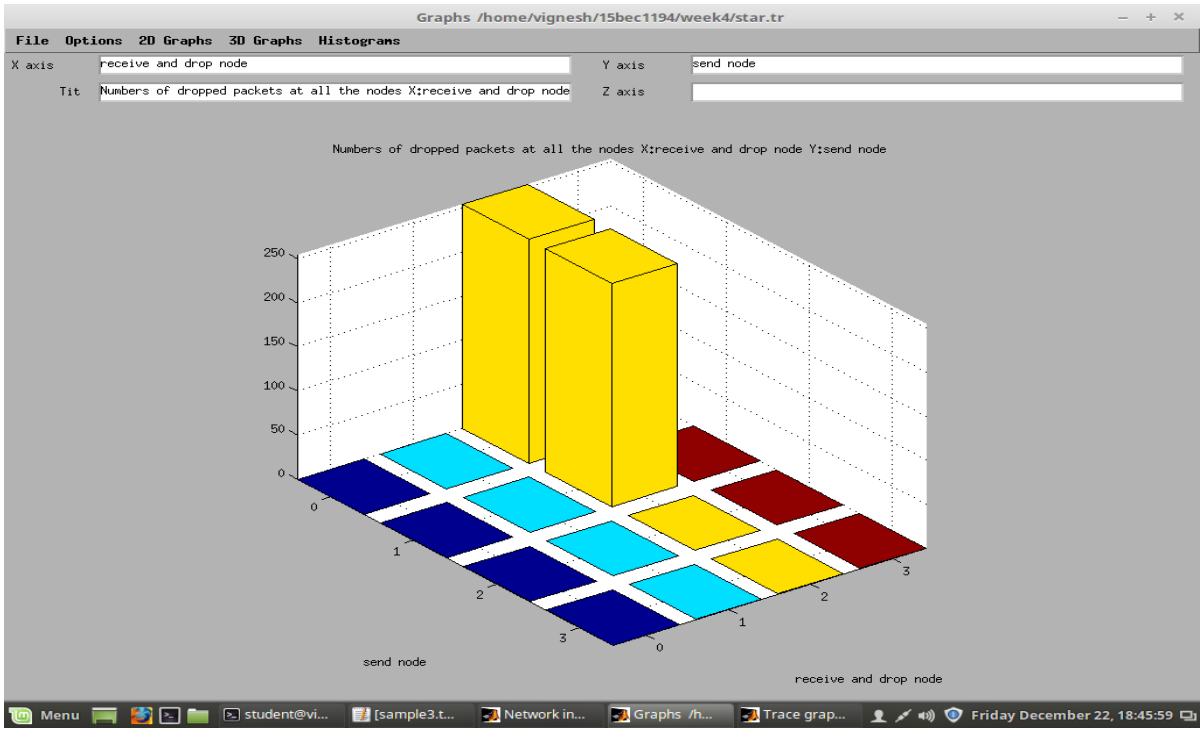


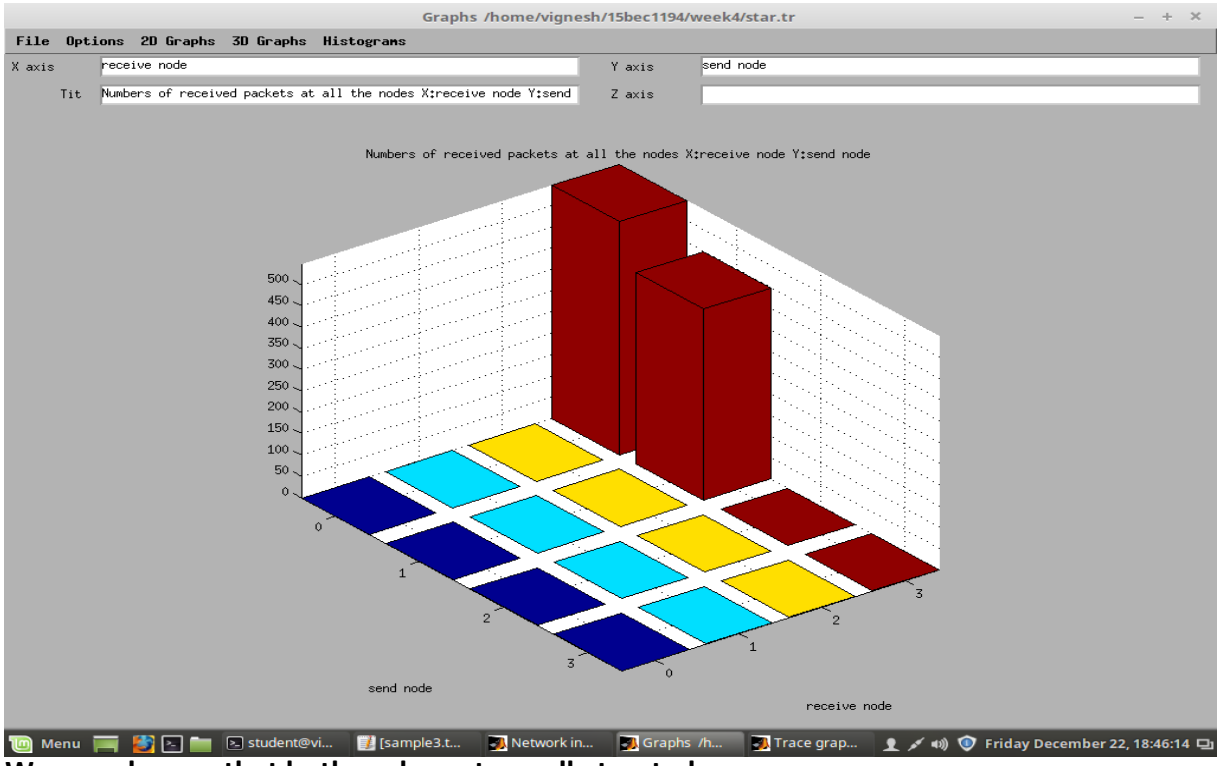

Network information /home/vignesh/15bec1194/week4/star.tr
Options Network information
Simulation information:
Simulation length in seconds: 4.104
Number of nodes: 4
Number of sending nodes: 2
Number of receiving nodes: 1
Number of generated packets: 1502
Number of sent packets: 1502
Number of forwarded packets: 995
Number of dropped packets: 507
Number of lost packets: 507
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Number of sent bytes: 751000
Number of forwarded bytes: 497500
Number of dropped bytes: 253500
Packets dropping nodes: 2
Simulation End2End delays in seconds:
Minimal delay (CN,ON,PID): 0.028 (0,3,7)
Maximal delay (CN,ON,PID): 0.107 (1,3,206)
Average delay: 0.09530150754
Current node information:
Number of generated packets: 701
Number of sent packets: 701
Number of forwarded packets: 701
Number of received packets: 0
Number of dropped packets: 0
Number of lost packets: 0
Number of sent bytes: 350500
Number of forwarded bytes: 350500
Number of received bytes: 0
Number of dropped bytes: 0
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Trace graph 2.02
File Options
/home/vignesh/15bec1194/week4/star.tr
Packet type: -cbr
Packet size: -500
Select/Deselect all sizes
Start 0.5
>=0.5
End 4.604
<=4.604
Flow 1 1-2
Other node: 0
<----------> Current 1 0-3
Source trace
Destination trace
Sent cbr
ACK packet: cbr
Time interval 1 sec
Number of histogram 10
Histogram edges:
Menu
student@vi...
[sample3.t...
Network in...
Graphs /h...
Trace grap...
Friday December 22, 18:44:21


Network information /home/vignesh/15bec1194/week4/star.tr
Options Network information
Simulation information:
Simulation length in seconds: 4.104
Number of nodes: 4
Number of sending nodes: 2
Number of receiving nodes: 1
Number of generated packets: 1502
Number of sent packets: 1502
Number of forwarded packets: 995
Number of dropped packets: 507
Number of lost packets: 507
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Number of sent bytes: 751000
Number of forwarded bytes: 497500
Number of dropped bytes: 253500
Packets dropping nodes: 2
Simulation End2End delays in seconds:
Minimal delay (CN,ON,PID): 0.028 (0,3,7)
Maximal delay (CN,ON,PID): 0.107 (1,3,206)
Average delay: 0.09530150754
Current node information:
Number of generated packets: 0
Number of sent packets: 0
Number of forwarded packets: 995
Number of received packets: 1502
Number of dropped packets: 507
Number of lost packets: 0
Number of sent bytes: 0
Number of forwarded bytes: 497500
Number of received bytes: 751000
Number of dropped bytes: 253500
Minimal packet size: 500
Maximal packet size: 500
Average packet size: 500
Trace graph 2.02
File Options
/home/vignesh/15bec1194/week4/star.tr
Packet type: -cbr
Packet size: -500
Select/Deselect all sizes
Start 0.5
>=0.5
End 4.604
<=4.604
Flow 1 1-2
Other node: 0
<----------> Current 2 0-3
Source trace
Destination trace
Sent cbr
ACK packet: cbr
Time interval 1 sec
Number of histogram 10
Histogram edges:
Menu
student@vi...
[sample3.t...
Network in...
Graphs /h...
Trace grap...
Friday December 22, 18:44:45



Packets of both get dropped.





We can observe that both nodes get equally treated.

Inferences:

- **From the above mechanism we inferred that the DropTail queuing mechanism is unfair, since there is biasing involved, the node 1 gets more service compared to node 0.**
- **On the other hand, the Stochastic Fair Queuing (SFQ) is much better since it services both the nodes fairly and there's no biasing involved.**
- **We can also see that the PDR for SFQ is fairer than PDR for DropTail.**

Results:

- Packet Delivery Ratio (PDR)

Using DropTail mechanism PDR

(n0-n3) = 323/801 = 40.37% PDR

(n1-n3) = 701/701 = 100%

Using SFQ mechanism

PDR (n0-n3) = 547/801 = 68.28%

PDR (n1-n3) = 448/701 = 63.90%

- **Average end to end delay** of 0.0953s (95.3 ms) is observed.

Experiment 4

COMPUTER

COMMUNICATION LAB

ECE4008

Name: *Debasya Sahoo*

Registration number: *15BEC1111*

Aim: Simulation of ring topology and analysis of static/dynamic routing in case of link/node failures.

Software: NS2 and Trace Graph

Description:

- Open the terminal and go to your own directory.
- Create a new directory week4 and open a tcl file.
- Write the tcl script, save and compile.
- Run the tcl script.
- Make use of grep commands for calculation of pdr.
- Using trace graph calculate the average end to end delay.

Code:

```
#Create an instance of simulator class
set ns [new Simulator]

# Enable Dyanamic Routing in Ns2

#DV - Distance Vector Routing Protocol

$ns rtproto DV

#Create a Trace file
set tf [open ring.tr w]

$ns trace-all $tf

#Create a nam file
set nf [open ring.nam w]

$ns namtrace-all $nf

# Creating nodes

set num 7

for {set i 0} {$i< $num} {incri} {

set n($i) [$ns node]

}
```

```
#Creating duplex-link

for {set i 0} {$i< $num} {incri} {

$ns duplex-link $n($i) $n([expr ($i+1)%$num]) 1Mb 10ms DropTail

}

#Create Transport Agent

#Create UDP object
set udp0 [new Agent/UDP]

#Attach the UDP object to $n0
$ns attach-agent $n(0) $udp0

# Generate Application Traffic
# Constant Bit Rate Application (CBR)
set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 500

$cbr0 set interval_ 0.005

$cbr0 attach-agent $udp0

#Create a NULL object and attach it to node 3
set null0 [new Agent/Null]

$ns attach-agent $n(3) $null0

#Virtually connect these 2 agents
$ns connect $udp0 $null0

#Finish Procedure
proc finish {} {

global ns nftf

# flush-trace: Dumps all content to trace and nam file
$ns flush-

trace close $tf

close $nf

exec namring.nam&

exit 0

}

# Schedule Traffic

$ns at 0.5 "$cbr0 start"

# Use rtmodel-at to set a link/node down/up

$ns rtmodel-at 1.0 down $n(1) $n(2)
```

$ns rtmodel-at 2.0 up $n(1) $n(2)

$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"

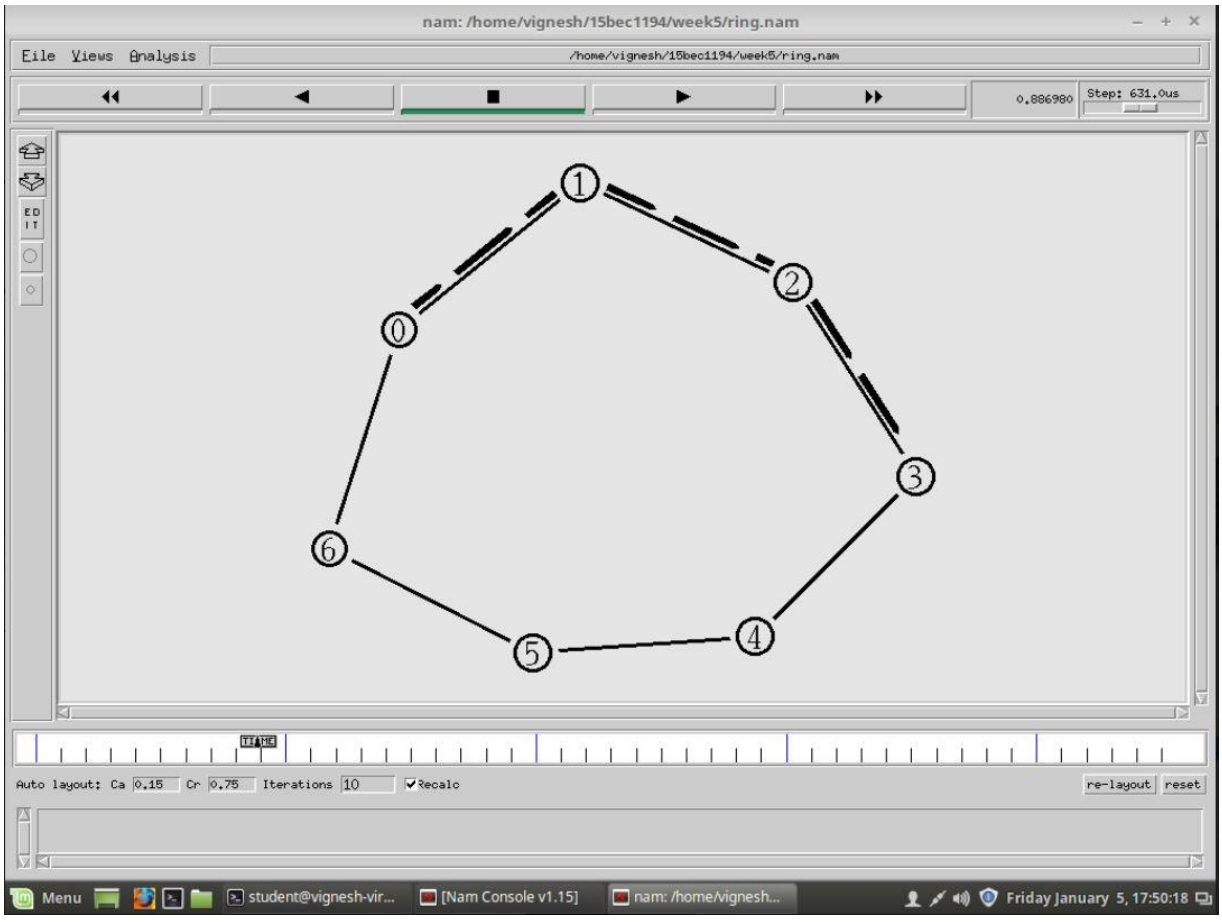
 # Assign different weights to edges

$ns cost $n(2) $n(3) 5

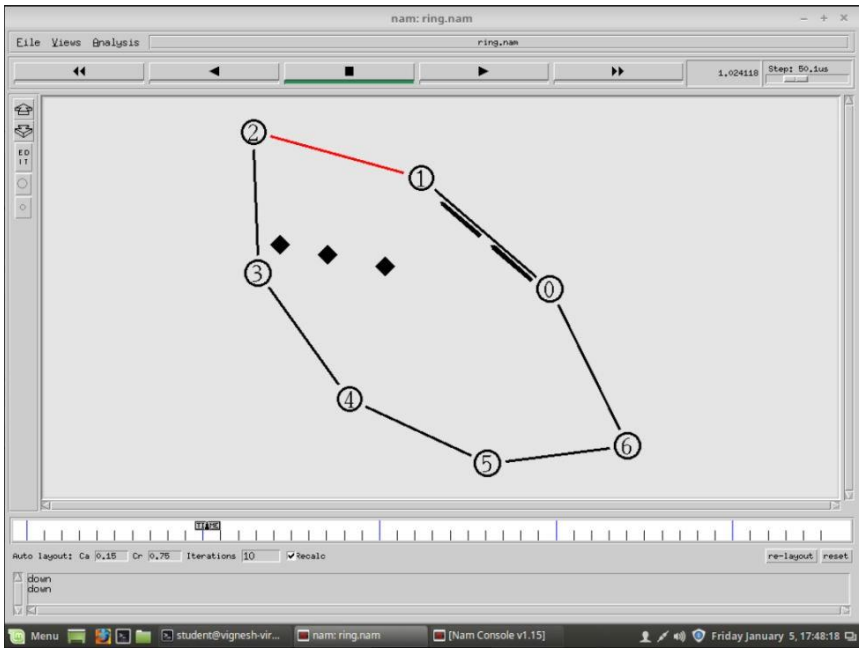
 # Run the Script

$ns run

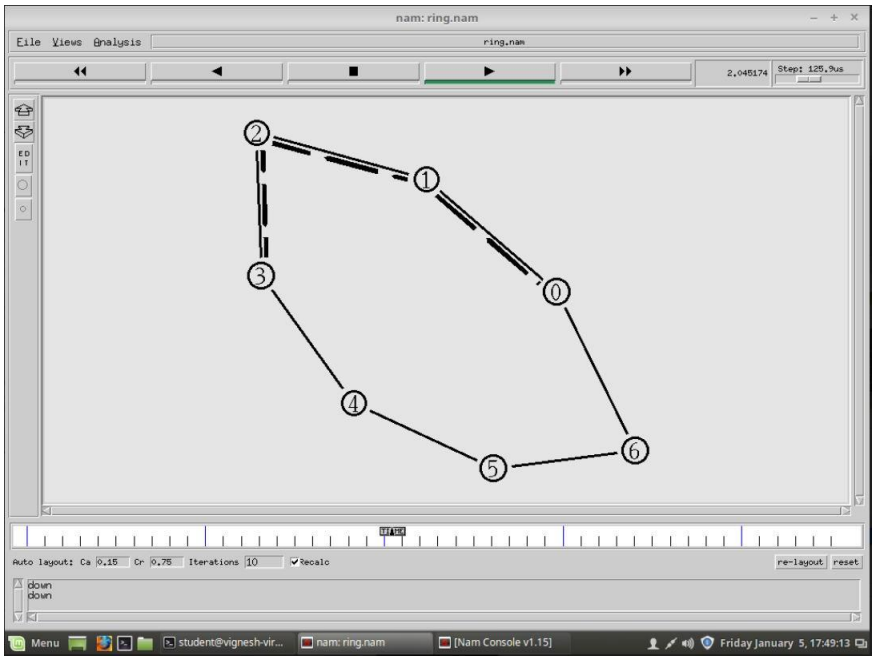
Output:



The shortest path is taken by default.

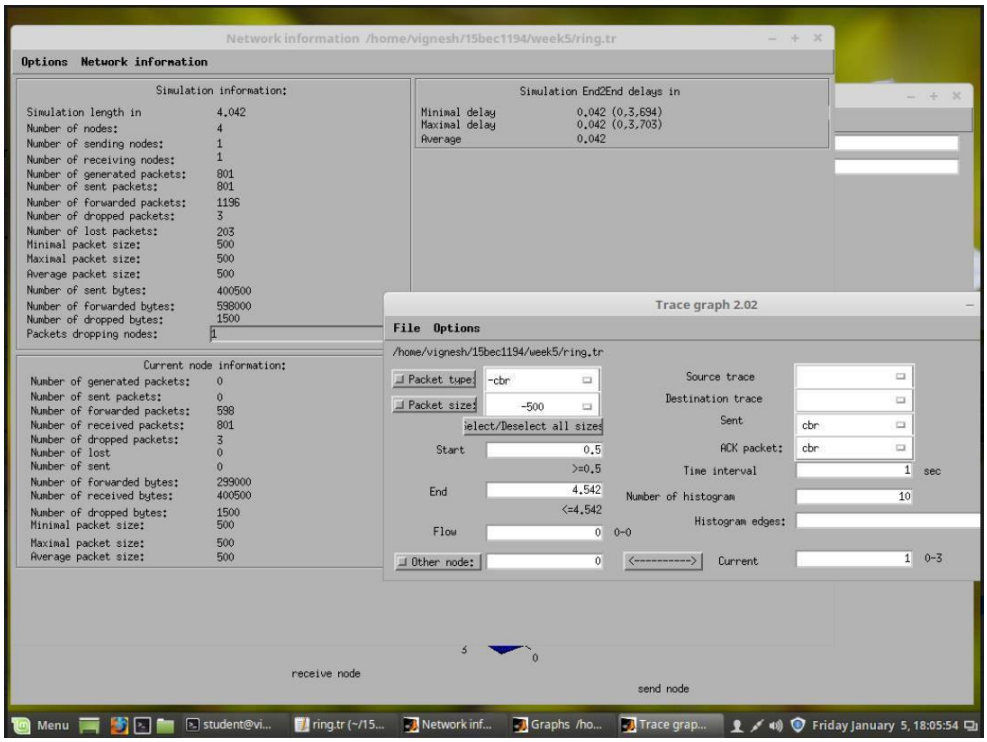


During 1s to 2s the link between node 1 and 2 is down thus no packets received at node 3



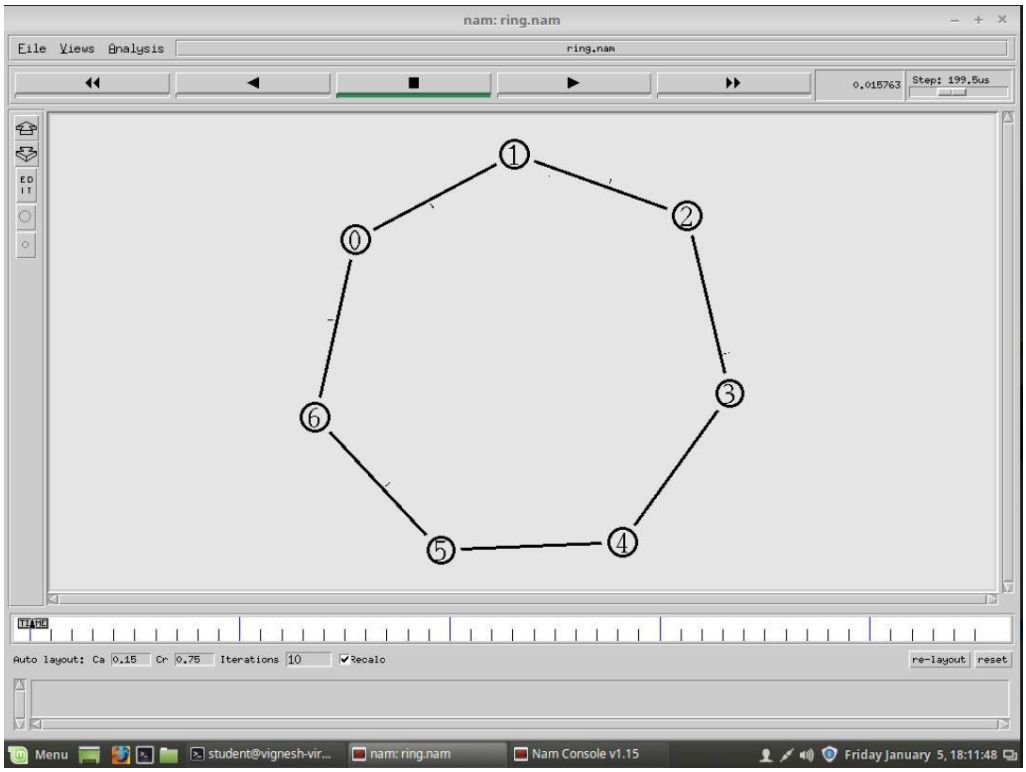
After 2s the link is up again thus packets get transferred

Calculation of pdr using grep commands

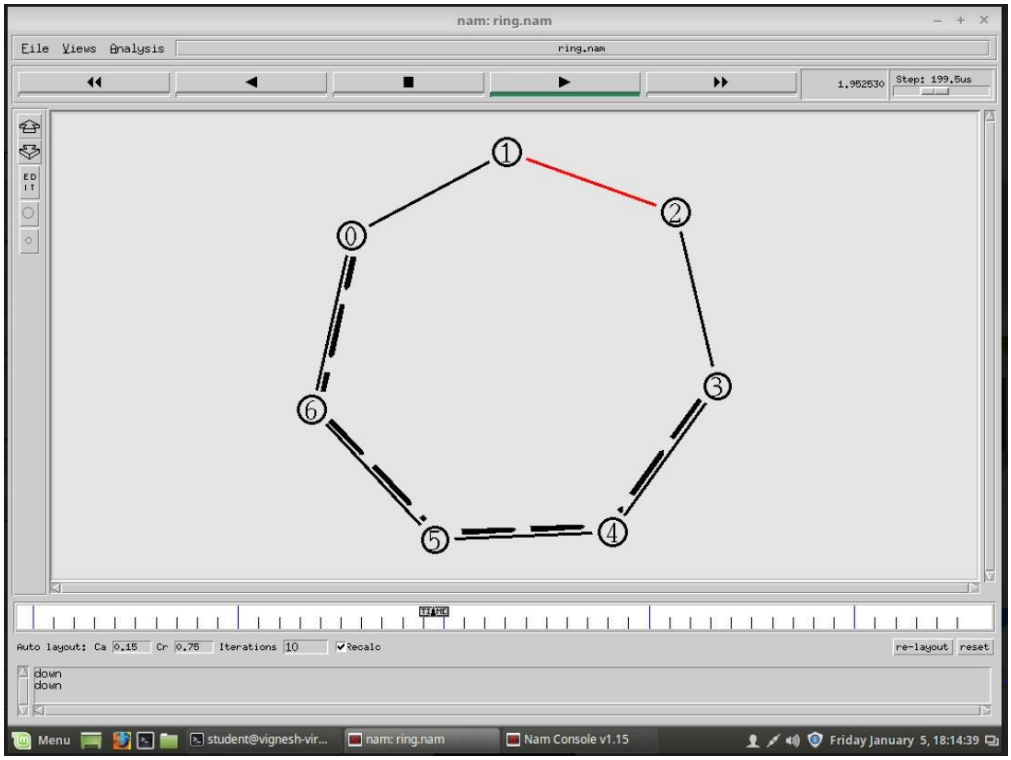


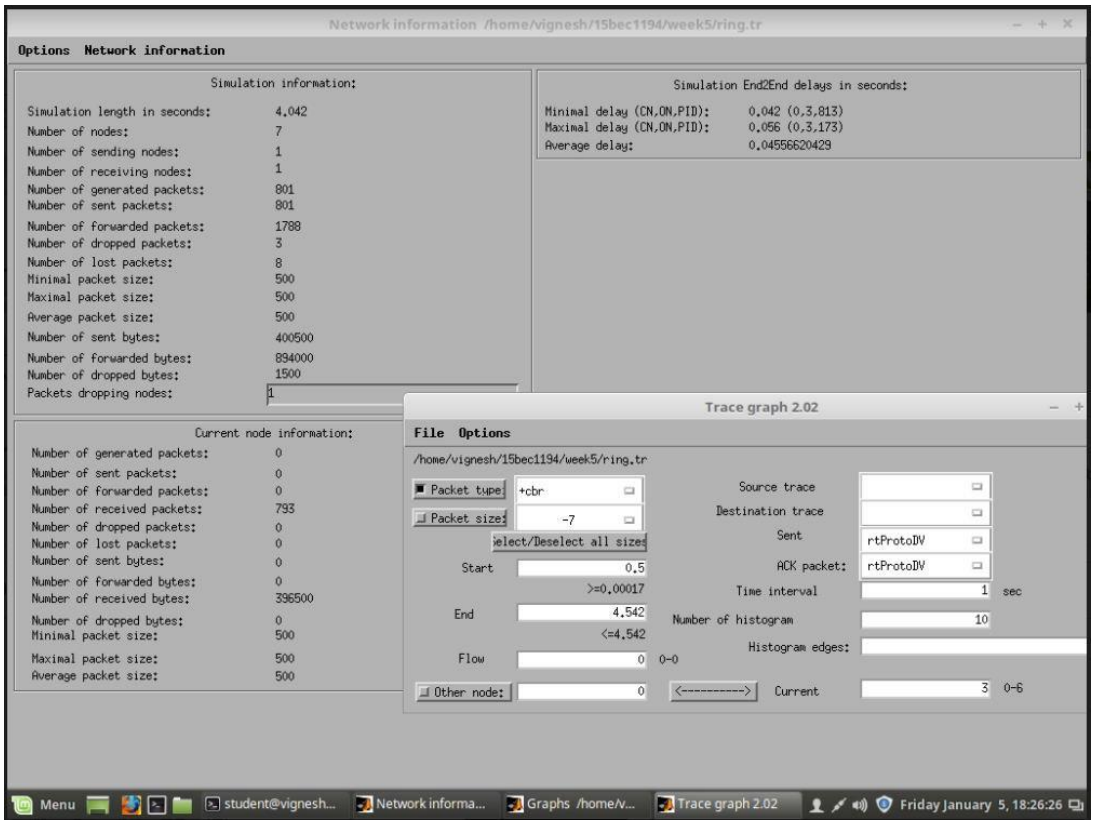
Using trace graph to determine average end to end delay.

Making use of dynamic routing protocol so that the packets are sent via different path when link between node 1 and node 2 fails.



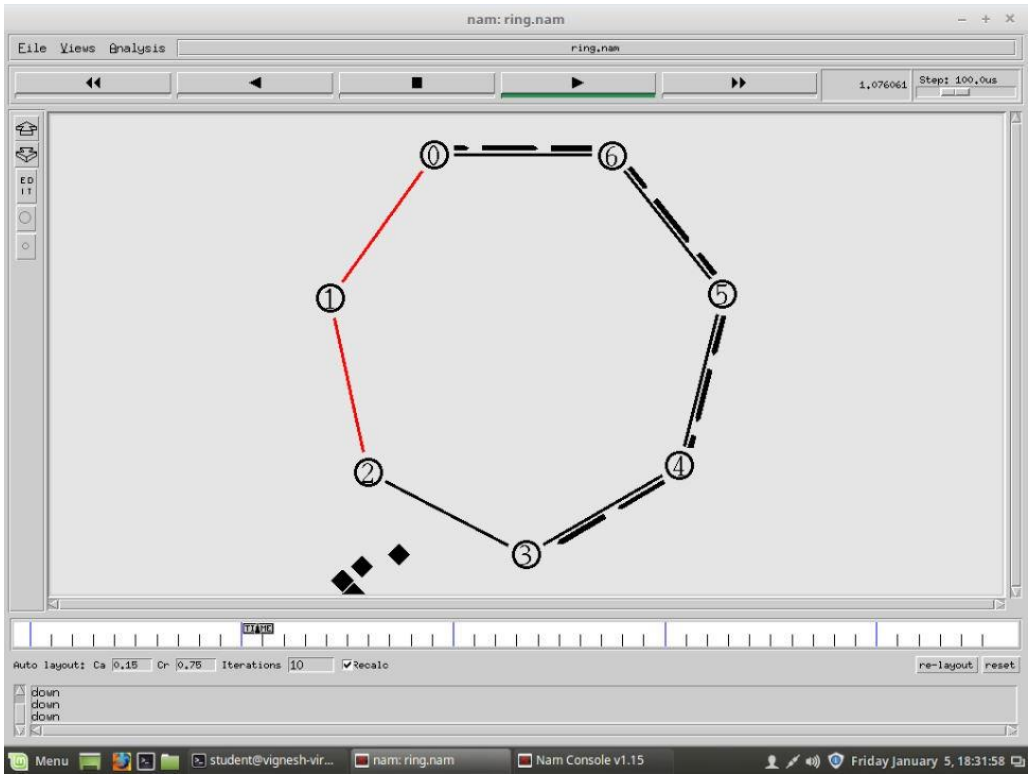
When the link fails the longer path is chosen for delivering the packets.



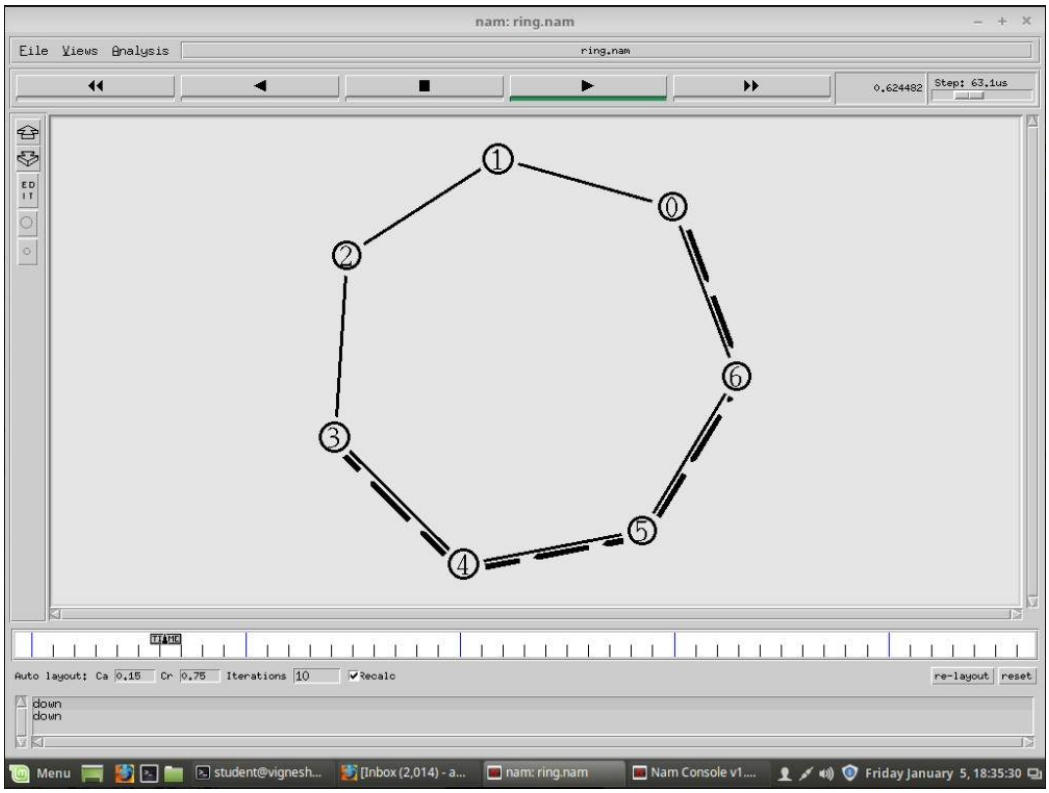


Using trace graph to determine average end to end delay.

In case of node failure

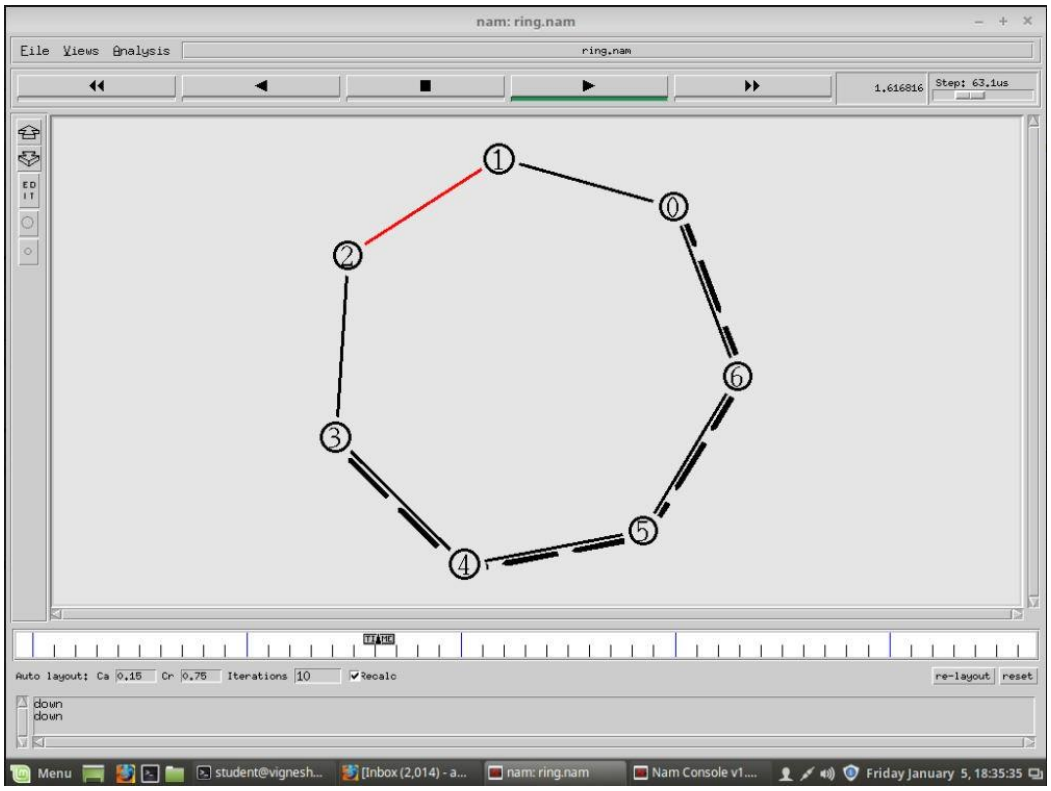


In case the weights of the links are changed



The weights from 0-1-2-3 are more as compared to 0-6-5-4-3.

Thus, the shortest path is now the longest path earlier. Even when the link fails this path is not affected, thus the pdr remains 100%



Inferences:

- When we create a ring topology and set the start and destination node, the packets are sent via the shortest path by default.
- In case the link is down the packets start getting lost unless we start using dynamic routing protocol, after which in case the link fails, the packets are transferred via another path.
- This protocol sends few bits after the start time which analyses all the path from starting node to destination node. These values are stored routing table which is referred in case of link/node failure.
- In case we assign weights to the links the shortest path get changed accordingly.
- We observe that the average end to end delay increases after we use the dynamic routing protocol this is because when the link fails the protocol switches to another route, thus switching and the time to travel via another route increase the delay.
- Even though the delay is increased we are able to achieve higher PDR after the use to dynamic routing protocol, which is much intended.

Results:

- PDR before dynamic routing protocol:

  PDR = 598/801 * 100 = 74.65%
  Average end to end delay = 0.042 s

- PDR after dynamic routing protocol

  PDR = (591+202)/(599+202) * 100 = 99%

Average end to end delay = 0.0455 s

Experiment 5

COMPUTER COMMUNICATION LAB

Study and Analysis of Network Performance using NETSIM

Name: Debasya Sahoo

Registration number: 15BEC1111

Aim: To study the performance of a Computer Network, by changing various factors of the network, using NetSim.

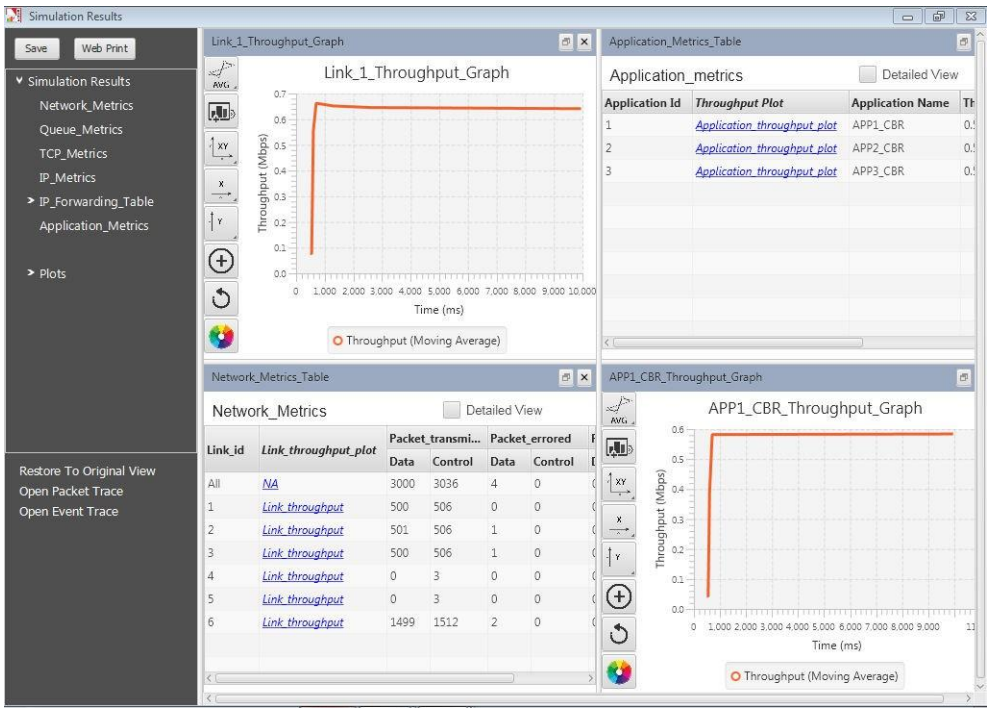
Different Factors are

- Increasing the traffic (Increasing the Packet Generation Rate for fixed number of Applications)
- Increasing the number of users (Increasing the number of nodes / Applications with fixed packet generation rate)
- Increasing the bit error rate across links

Network Simulation Specifications:

1. No of Nodes = 6 ; No of Destination Node = 1 (Node 6) ; No of Applications (CBR) = 3 (3 nodes Application Traffic) ; Vary the Packet Inter-arrival Time : 20ms, 15ms, 10ms, 5ms
   Link Speed = Switch to Destination = 5Mbps (Duplex);
   Only UDP sources ; Dynamic ARP enabled ; Simulation time : 10seconds
   - ✓ **Inference (Plots):** Calculate the Average Application Throughput & Average Delay vs Packet Inter-arrival Time

20ms:



Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	0.582832	18317.454247
2	*Application_throughput_plot*	APP2_CBR	0.582832	35778.330249
3	*Application_throughput_plot*	APP3_CBR	0.582832	22713.576373

Average application throughput: 0.58 Mbps

Average delay: 25603 us

15ms:

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	0.777888	36211.821393
2	*Application_throughput_plot*	APP2_CBR	0.777888	39264.033867
3	*Application_throughput_plot*	APP3_CBR	0.777888	7883.122054

Average application throughput: 0.78Mbps

Average delay: 27786 us

10ms:

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	1.163328	30972.161357
2	*Application_throughput_plot*	APP2_CBR	1.136464	75178.847574
3	*Application_throughput_plot*	APP3_CBR	1.163328	93912.396885

Average application throughput: 1.15Mbps

Average delay: 66687 us

5ms:

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	1.319840	2248728.935641
2	*Application_throughput_plot*	APP2_CBR	1.593152	1328288.722103
3	*Application_throughput_plot*	APP3_CBR	1.742656	1791018.949393

Average application throughput: 1.55Mbps

Average delay: 1789344 us

Thus, we have,

Packet Inter-arrival time (ms)	Average application throughput (Mbps)	Average delay (us)
20	0.58	25603
15	0.78	27786
10	1.15	66687
5	1.55	1789344

2. No of Nodes = 6 ; Packet Inter-arrival Time = 5ms ; Increase the number of Application Traffics (nodes in this case) = 1, 2, 3, 4, 5 ; No of Destination Node = 1 Link Speed = Switch to Destination = 5Mbps (Duplex)
   - ✓ **Inference (Plots):** Calculate the Average Application Throughput & Average Delay vs No of Application Traffic/Nodes

Number of nodes: 1

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	2.334832	53030.944743

Number of nodes: 2

Application_Metrics_Table

Application_metrics Detailed View

Application Id	Throughput Plot	Application Name	Throughput (Mbps)	Delay(microsec)
1	Application_throughput_plot	APP1_CBR	2.325488	148807.860281
2	Application_throughput_plot	APP2_CBR	2.324320	156054.894975

Number of nodes: 3

Application_Metrics_Table

Application_metrics Detailed View

Application Id	Throughput Plot	Application Name	Throughput (Mbps)	Delay(microsec)
1	Application_throughput_plot	APP1_CBR	1.319840	2248728.935641
2	Application_throughput_plot	APP2_CBR	1.593152	1328288.722103
3	Application_throughput_plot	APP3_CBR	1.742656	1791018.949393

Number of nodes: 4

Application_Metrics_Table

Application_metrics Detailed View

Application Id	Throughput Plot	Application Name	Throughput (Mbps)	Delay(microsec)
1	Application_throughput_plot	APP1_CBR	1.212384	3137211.032517
2	Application_throughput_plot	APP2_CBR	1.757840	895670.798244
3	Application_throughput_plot	APP3_CBR	1.120112	1895396.928575
4	Application_throughput_plot	APP4_CBR	0.628384	4598300.719238

Number of nodes: 5

Application_Metrics_Table

Application_metrics Detailed View

Application Id	Throughput Plot	Application Name	Throughput (Mbps)	Delay(microsec)
1	Application_throughput_plot	APP1_CBR	0.971776	3620304.005334
2	Application_throughput_plot	APP2_CBR	0.421648	5351094.985485
3	Application_throughput_plot	APP3_CBR	1.186688	1498288.209255
4	Application_throughput_plot	APP4_CBR	1.502048	972417.557506
5	Application_throughput_plot	APP5_CBR	0.648240	5178123.573960

Number of application traffic/node	Average application throughput (Mbps)	Average delay (us)
1	2.33	53030
2	2.32	152430
3	1.55	1789344
4	1.18	2631644
5	0.95	3324045

3. No of Nodes = 6 ; No of Destination Node = 1 ; No of Applications (CBR) = 3 (3 nodes Application Traffic) ; Packet Inter-arrival Time : 10ms ; Vary the BER : 10^-8, 10^-6, 10^-5 across active links
   Link Speed = Switch to Destination = 5Mbps (Duplex)
   - ✓ **Inference (Plots):** Calculate the Average Application Throughput & Average Delay vs BER

10^-8:

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	1.166832	27571.032932
2	*Application_throughput_plot*	APP2_CBR	1.166832	50932.997057
3	*Application_throughput_plot*	APP3_CBR	1.166832	32170.513253

10^-6:

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	1.104928	123577.926784
2	*Application_throughput_plot*	APP2_CBR	1.124784	149381.347219
3	*Application_throughput_plot*	APP3_CBR	1.166832	243651.302099

10^-5:

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	0.307184	4158198.656806
2	*Application_throughput_plot*	APP2_CBR	0.061904	532544.039276
3	*Application_throughput_plot*	APP3_CBR	0.059568	1258302.947880

BER	Average application throughput (Mbps)	Average delay (us)
10^{-8}	1.17	36891
10^{-6}	1.13	172203
10^{-5}	0.14	1983014

4. No of Nodes = 6; No of Destination Nodes = 5 ; No of Applications (CBR) = 5 ; Source-Destination Pairs : 1-2, 2-3, 3-4, 4-5, 5-6 ; Vary the Packet Inter-arrival Time : 5ms, 10ms, 15ms and 20ms
   Link Speed = Switch to Destination = 5Mbps (Duplex)
   - ✓ **Inference (Plots):** Calculate the Average Application Throughput & Average Delay vs Packet Inter-arrival Time
   - ✓ Compare this scenario with Scenario 1 and infer if any difference in metrics results.

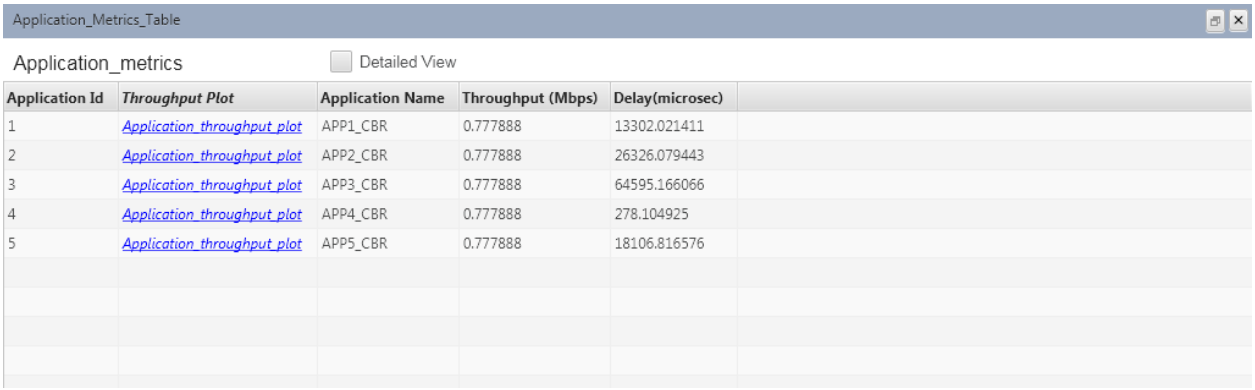
20ms:

Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	0.573488	278.109898
2	*Application_throughput_plot*	APP2_CBR	0.582832	26551.323032
3	*Application_throughput_plot*	APP3_CBR	0.582832	64878.966854
4	*Application_throughput_plot*	APP4_CBR	0.582832	278.213667
5	*Application_throughput_plot*	APP5_CBR	0.582832	32500.162998

15ms:

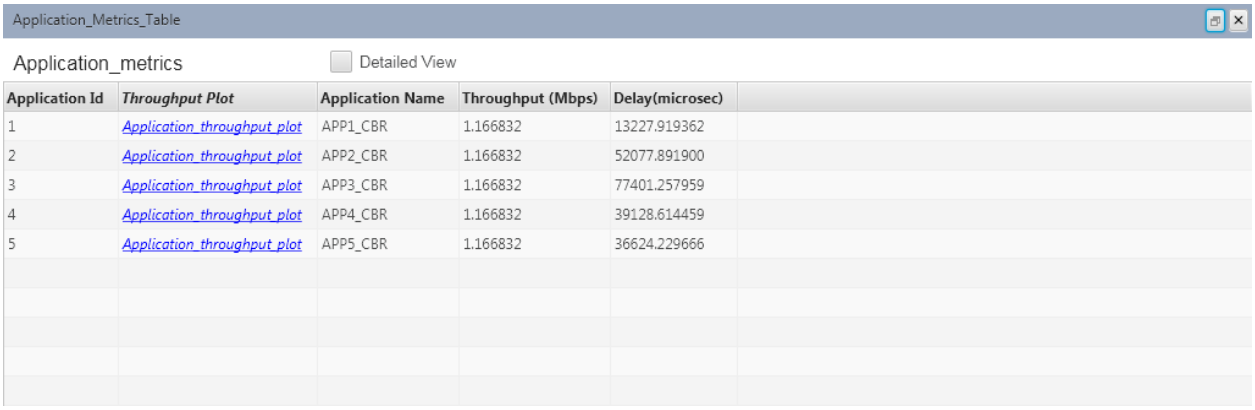


Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	0.777888	13302.021411
2	*Application_throughput_plot*	APP2_CBR	0.777888	26326.079443
3	*Application_throughput_plot*	APP3_CBR	0.777888	64595.166066
4	*Application_throughput_plot*	APP4_CBR	0.777888	278.104925
5	*Application_throughput_plot*	APP5_CBR	0.777888	18106.816576

10ms:

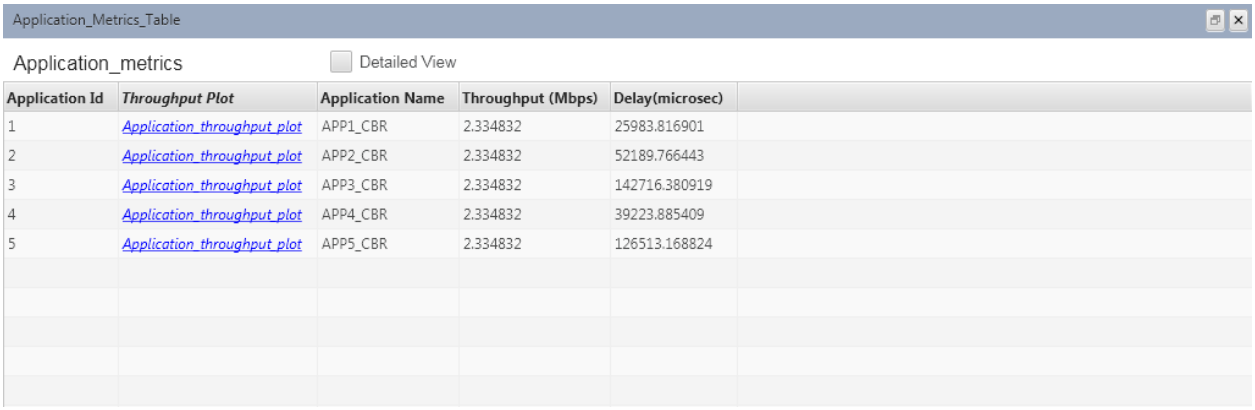


Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	1.166832	13227.919362
2	*Application_throughput_plot*	APP2_CBR	1.166832	52077.891900
3	*Application_throughput_plot*	APP3_CBR	1.166832	77401.257959
4	*Application_throughput_plot*	APP4_CBR	1.166832	39128.614459
5	*Application_throughput_plot*	APP5_CBR	1.166832	36624.229666

5ms:



Application_Metrics_Table

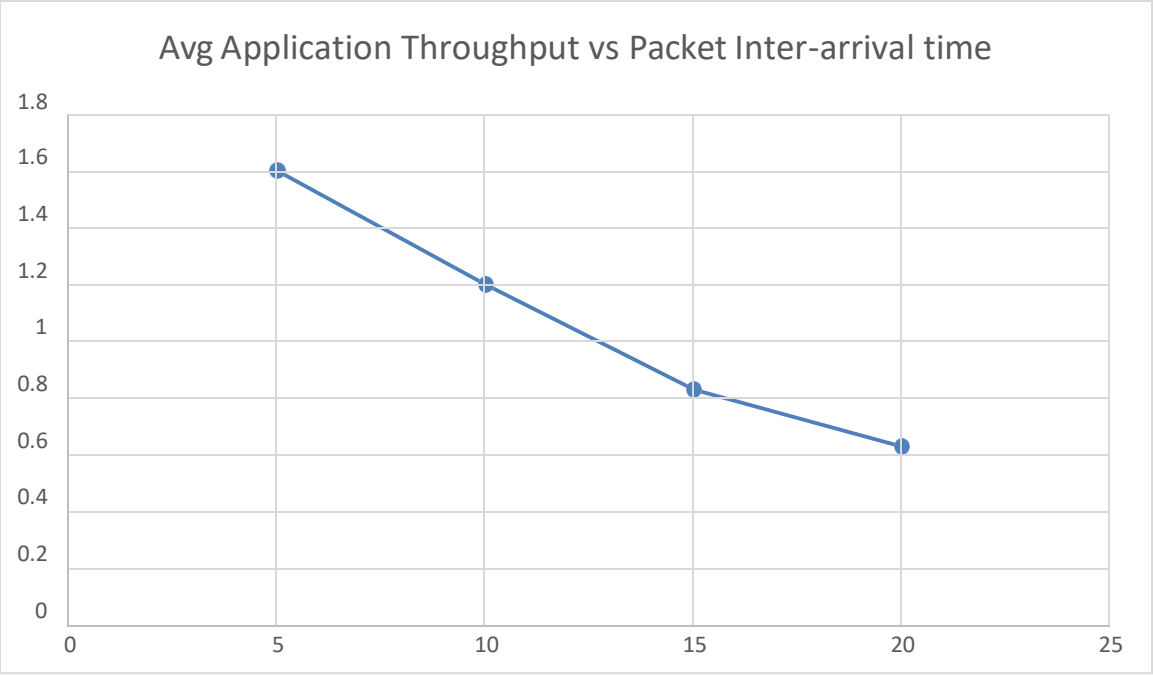
Application_metrics Detailed View

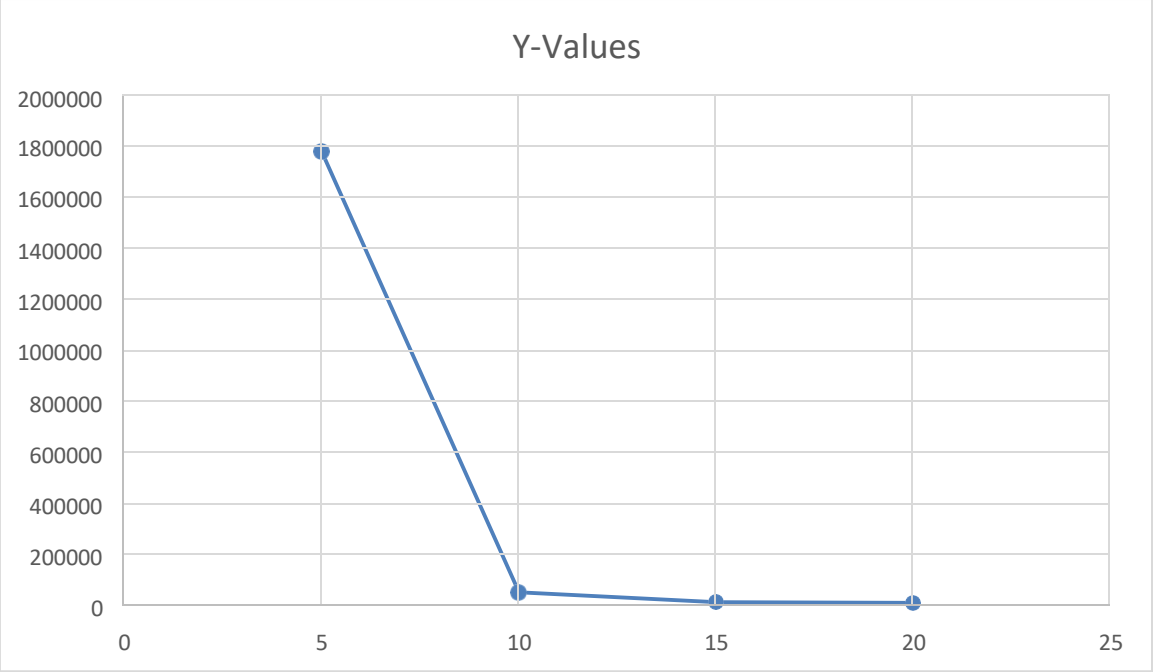
Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	2.334832	25983.816901
2	*Application_throughput_plot*	APP2_CBR	2.334832	52189.766443
3	*Application_throughput_plot*	APP3_CBR	2.334832	142716.380919
4	*Application_throughput_plot*	APP4_CBR	2.334832	39223.885409
5	*Application_throughput_plot*	APP5_CBR	2.334832	126513.168824

Packet Inter-arrival time (ms)	Average application throughput (Mbps)	Average delay (us)
20	0.58	24897
15	0.78	24521
10	1.17	43691
5	2.33	77324

Observations, Graphs and Inferences:

Packet Inter-arrival time (ms)	Average application throughput (Mbps)	Average delay (us)
20	0.58	25603
15	0.78	27786
10	1.15	66687
5	1.55	1789344

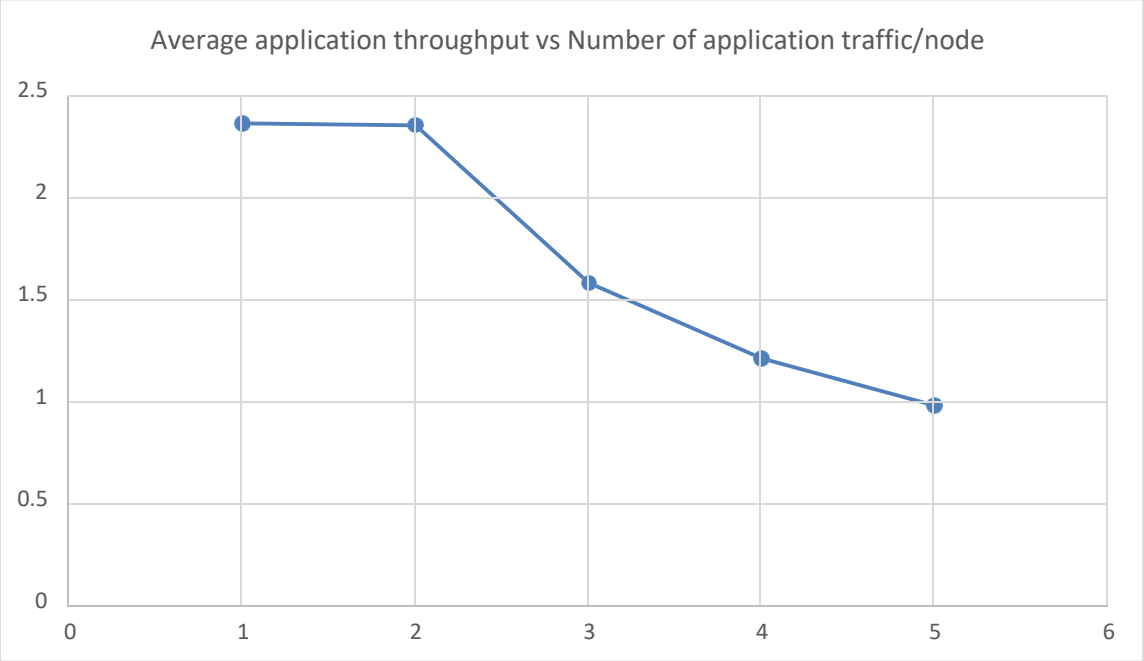


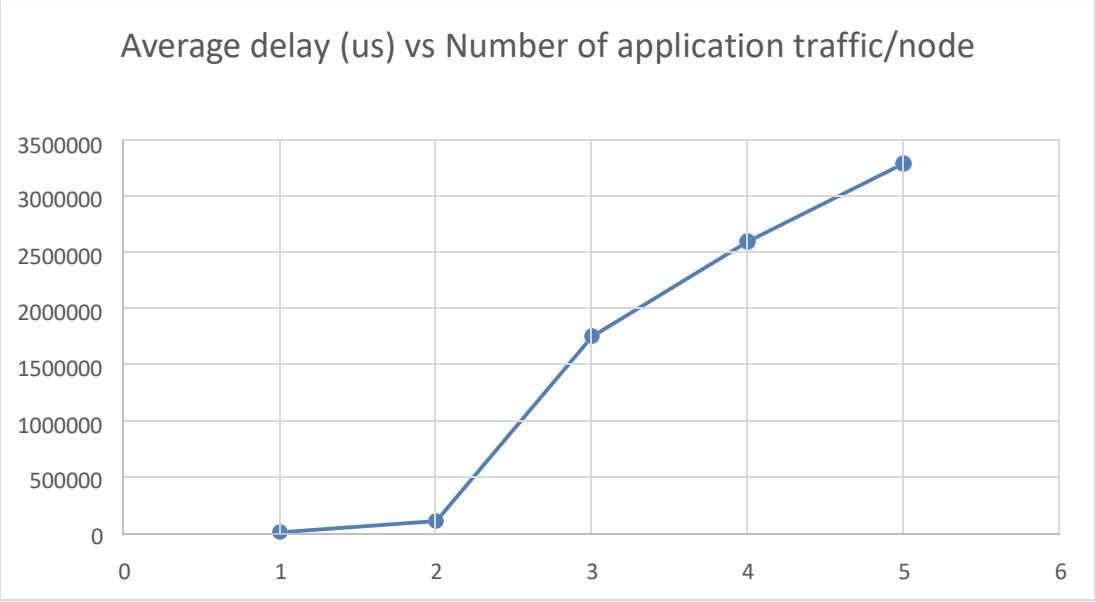


Inferences:

- As the packet inter-arrival time increases the average application throughput decreases.
- As the packet inter-arrival time increases the average delay decreases.

Number of application traffic/node	Average application throughput (Mbps)	Average delay (us)
1	2.33	53030
2	2.32	152430
3	1.55	1789344
4	1.18	2631644
5	0.95	3324045

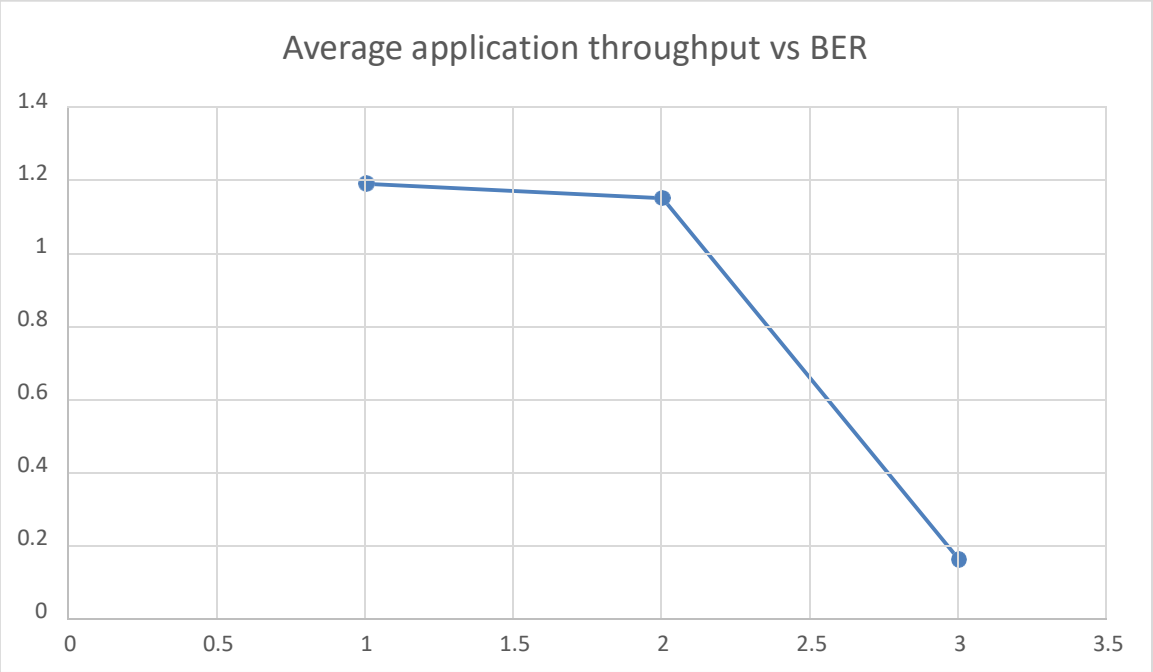


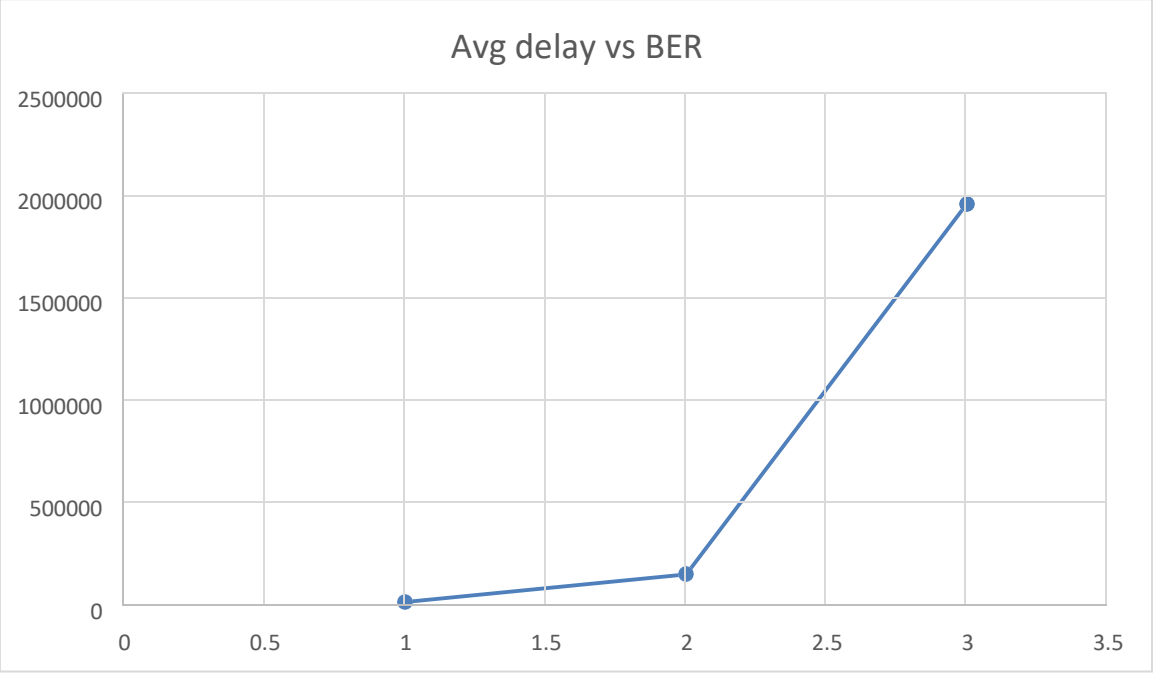


Inferences:

- As the number of application traffic/node increases the average application throughput decreases.
- As the number of application traffic/node increases the average delay increases

BER	Average application throughput (Mbps)	Average delay (us)
10^{-8}	1.17	36891
10^{-6}	1.13	172203
10^{-5}	0.14	1983014

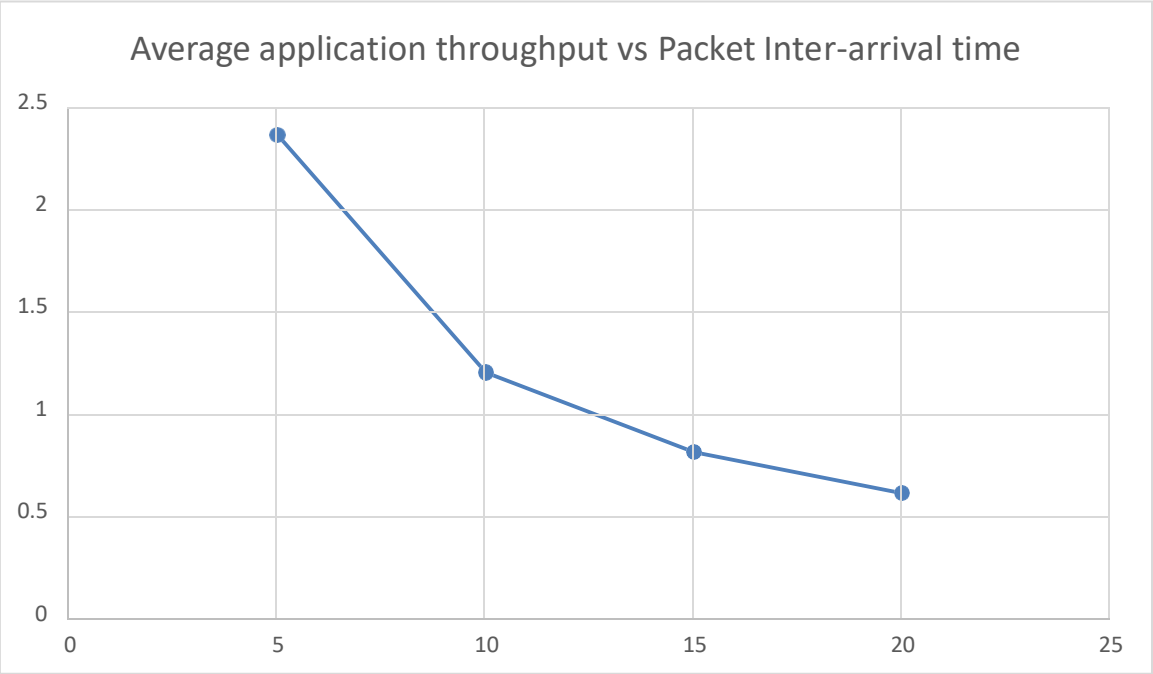


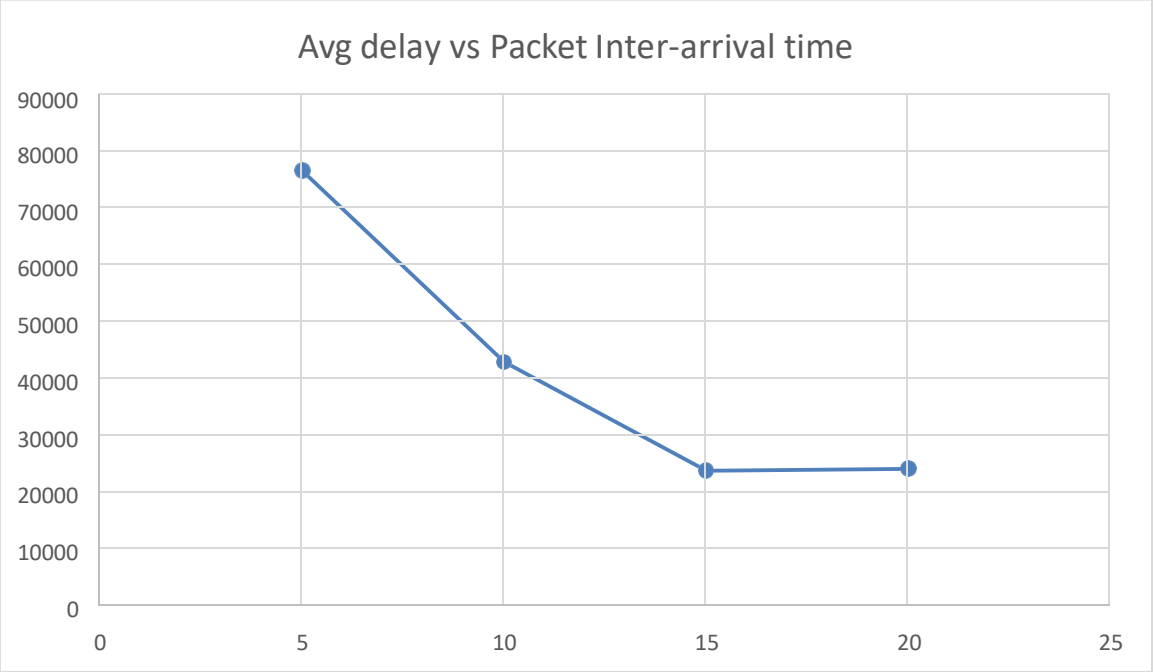


Inferences:

- As the BER increases the average application throughput decreases.
- As the BER increases the average delay increases

Packet Inter-arrival time (ms)	Average application throughput (Mbps)	Average delay (us)
20	0.58	24897
15	0.78	24521
10	1.17	43691
5	2.33	77324

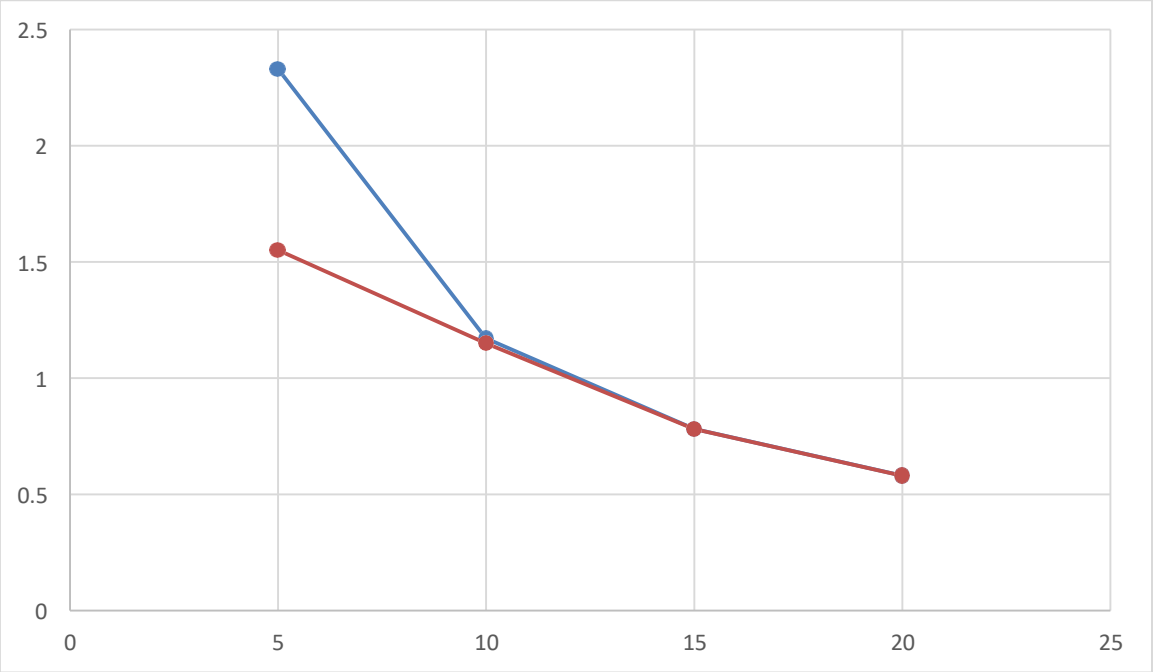


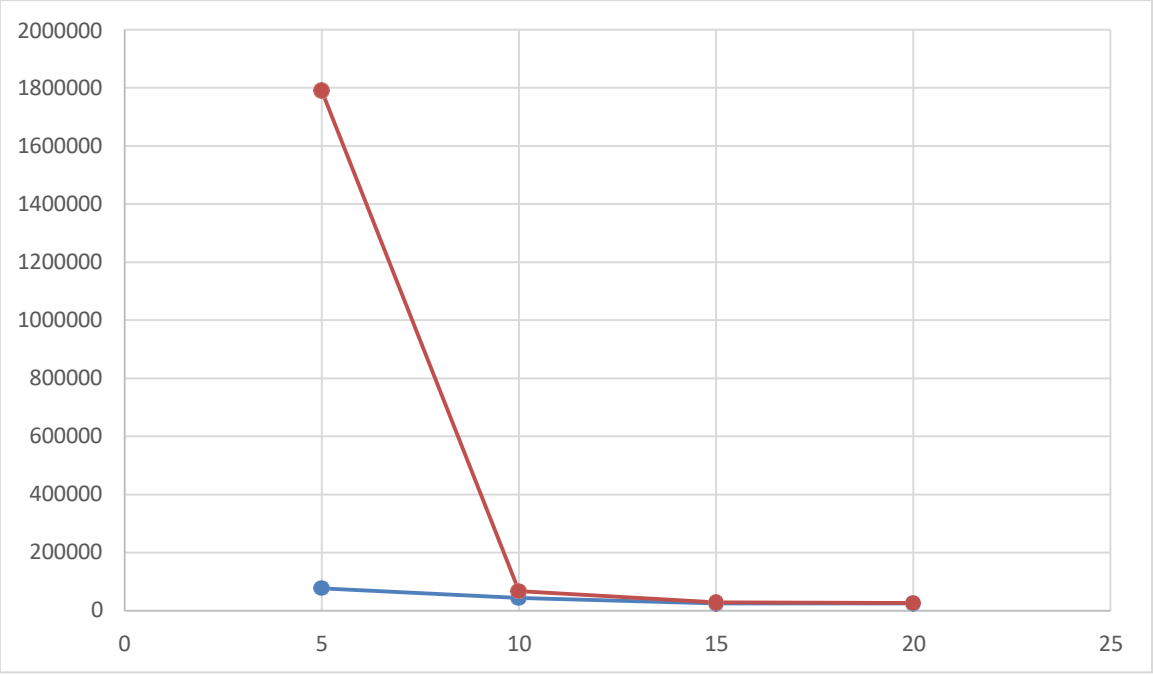


Inferences:

- As the packet inter-arrival time increases the average application throughput decreases.
- As the packet inter-arrival time increases the average delay decreases

Comparing scenarios (i) and (iv)





Inferences:

- The throughput decreases in both the cases as we increase the packet inter-arrival time.
- When we compare the two cases (i) and (iv) we observe that the average delay has decreased in the latter case as the packet inter-arrival time has increased.

Result: The above experiment was successfully implemented and output verified.

Experiment No: 6

COMPUTER COMMUNICATION LAB

Implementation of CRC (Cyclic Redundancy Check) using C-Program to perform Error Detection mechanism

Name : Debasya Sahoo
Reg. No : 15BEC1111

AIM:

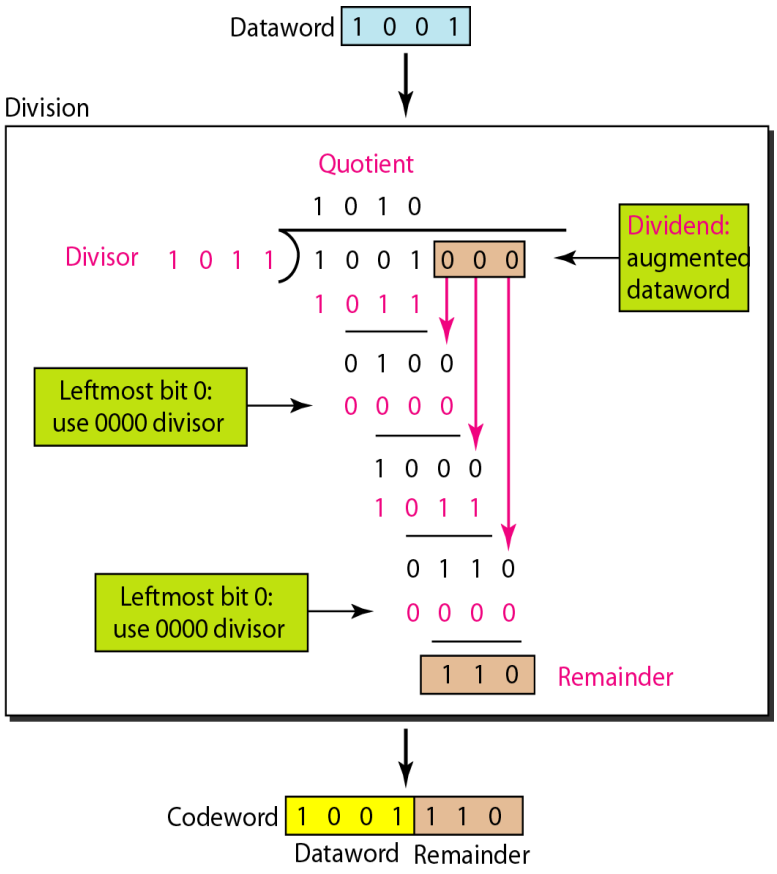
a. To Implement CRC (Cyclic Redundancy check) using C-Program to perform Error Detection mechanism

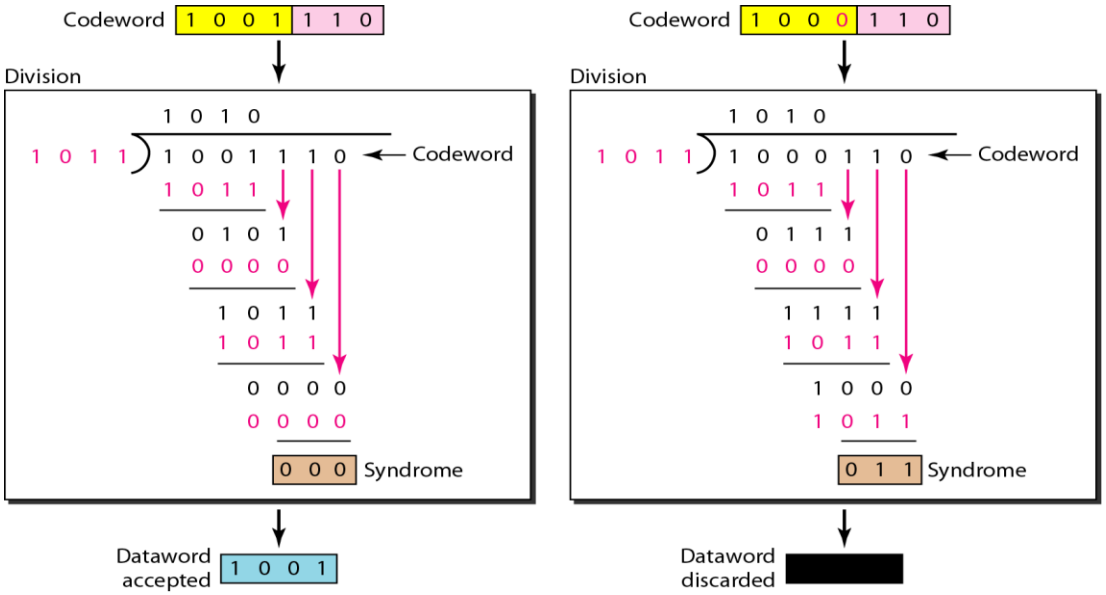
SOFTWARE USED:

C Programming Language, Turbo C/C++ compiler, gcc compiler (Linux)

PROCEDURE:

Division in CRC encoder





Division in the CRC decoder for two cases

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define N strlen(g)

char t[10],cs[10],g[]="1011";
int a,e,c;

void xor(){
   for(c = 1;c < N; c++)
   cs[c] = (( cs[c] == g[c])?'0':'1');
}

void crc(){
   for(e=0;e<N;e++)
      cs[e]=t[e];
   do{
      if(cs[0]=='1')
         xor();
      for(c=0;c<N-1;c++)
         cs[c]=cs[c+1];
      cs[c]=t[e++];
    }while(e<=a+N-1);
}

int main()
{
   printf("\nEnter data :
   "); scanf("%s",t);
   printf("\n---------------------------------------");
```

```
        printf("\nGeneratng polynomial : %s",g);
        a=strlen(t);
        for(e=a;e<a+N-1;e++) //append zeros to the dataword
            t[e]='0';
        printf("\n----------------------------------------");
        printf("\nModified data is : %s",t);
        printf("\n----------------------------------------");
        crc(); //perform crc
        printf("\nChecksum is : %s",cs);
        for(e=a;e<a+N-1;e++)
            t[e]=cs[e-a];
        printf("\n----------------------------------------");
        printf("\nFinal codeword is : %s",t);
        printf("\n----------------------------------------");
        printf("\nTest error detection 0(yes) 1(no)? :
        "); scanf("%d",&e);
        if(e==0)
        {
            do{
                printf("\nEnter the position where error is to be inserted :
                "); scanf("%d",&e);
            }while(e==0 || e>a+N-1);
            t[e-1]=(t[e-
            1]=='0')?'1':'0';
            printf("\n----------------------------------------");
            printf("\nErroneous data : %s\n",t);
        }
        crc();
        for(e=0;(e<N-1) && (cs[e]!='1');e++);
            if(e<N-1)
                printf("\nError
            detected\n\n"); else
                printf("\nNo error detected\n\n");
                printf("\n----------------------------------------\n");
            return 0;
}
```

A CRC is constructed to generate a 3 bit FCS for a 4-bit message. The generator pattern is 1011.

a. Show the generation of the codeword at the sender site
b. Show the checking of the codeword at the receiver site (assume no error).
c. Show that the detection algorithm detects error if received codeword is corrupted.

RESULT & INFERENCES:

Dataword: k

Codeword: n

Divisor: n-k+1

Augment n-k zeros to dataword

```
Enter data : 1001

---------------------------------------
Generatng polynomial : 1011
---------------------------------------
Modified data is : 1001000
---------------------------------------
Checksum is : 110
---------------------------------------
Final codeword is : 1001110
---------------------------------------
Test error detection 0(yes) 1(no)? : 0

Enter the position where error is to be inserted : 1

---------------------------------------
Erroneous data : 0001110

Error detected
```

```
Enter data : 1001

---------------------------------------
Generatng polynomial : 1011
---------------------------------------
Modified data is : 1001000
---------------------------------------
Checksum is : 110
---------------------------------------
Final codeword is : 1001110
---------------------------------------
Test error detection 0(yes) 1(no)? : 1

No error detected
```

- A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data.
- Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents.
- On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error correction
- CRCs are so called because the *check* (data verification) value is a *redundancy* (it expands the message without adding information) and the algorithm is based on *cyclic* codes.

Result:

The above experiment was successfully executed and output verified.

Experiment No: 7

COMPUTER COMMUNICATION LAB

Implementation of Bit Stuffing/De-stuffing and Byte Stuffing/De-stuffing to delimit MAC frames using 'C' Program

Name : Debasya Sahoo
Reg. No : 15BEC1111

AIM:

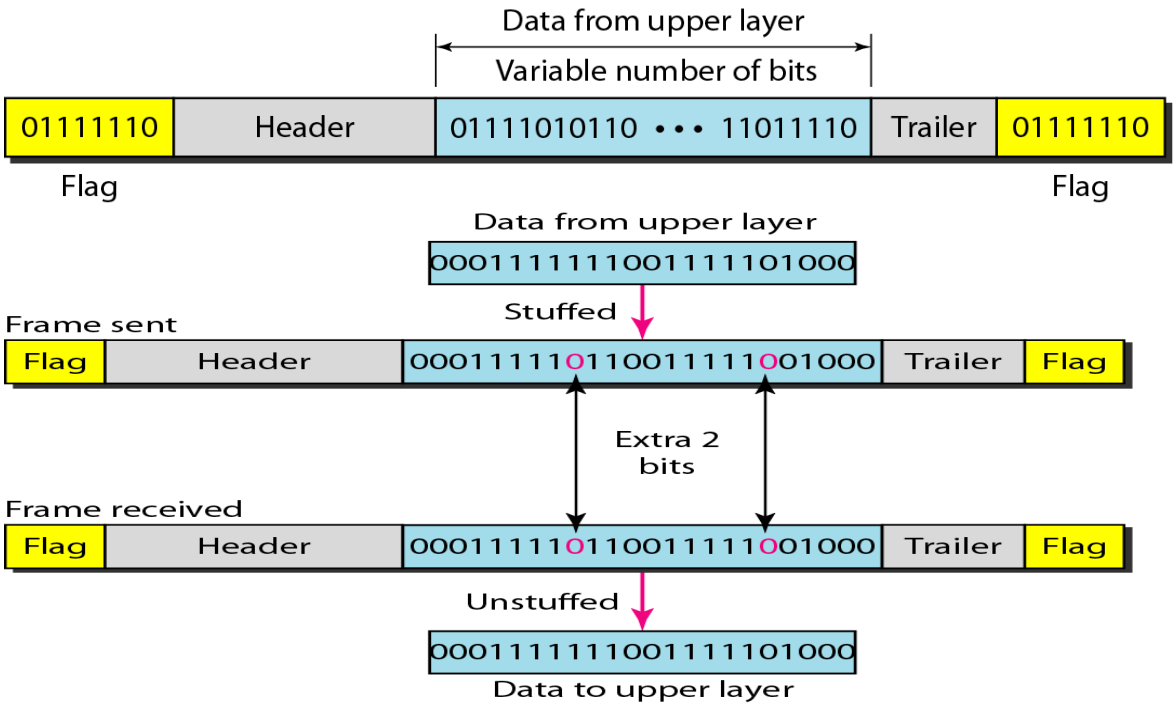
a. **To Implement Bit Stuffing/De-stuffing using C programming language**
b. **To Implement Byte Stuffing/De-stuffing using C programming language and understand the importance of Character oriented and Bit Oriented Protocols and need for stuffing MAC frames.**

SOFTWARE USED:

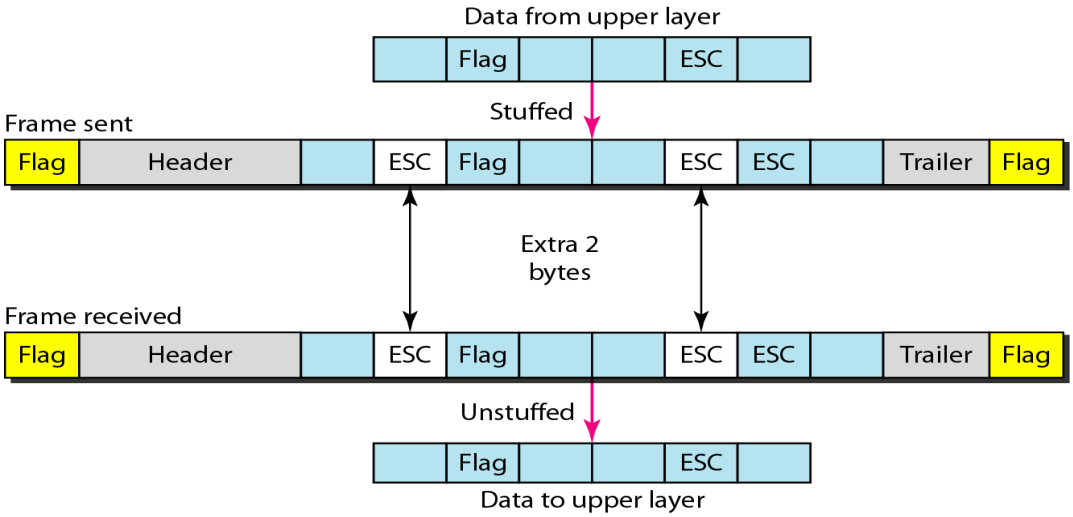
C Programming Language, Turbo C/C++ compiler or gcc compiler (Linux)

PROCEDURE:

Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.



Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.



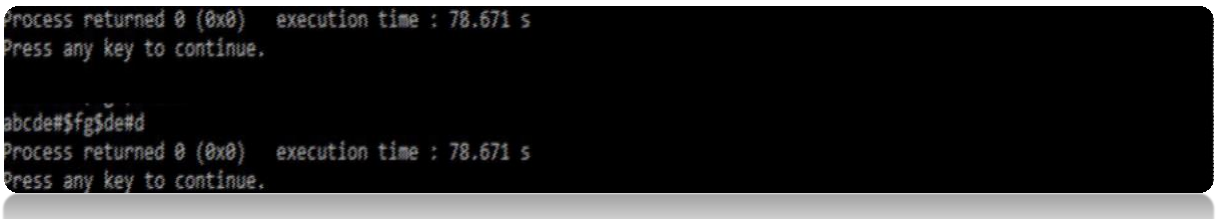
Byte Stuffing

```
#include <stdio.h>
#include <string.h>
intmain()
{
   char s1[50], stuf[50], dstuf[50];
inti, j=0;

printf("Enter Data Frame: ");
scanf("%s", s1);

   for(i=0;i<=strlen(s1);i++)
   {
      if(s1[i]=='#' || s1[i]=='$')
      {
stuf[j]='#';
j++;
```

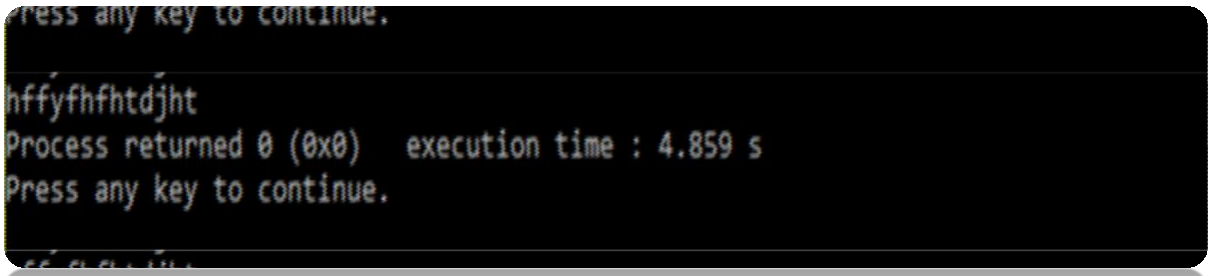
```
        }

stuf[j]=s1[i];
j++;
    }

printf("%s\n",stuf);
j=0;
for(i=0;i<=strlen(stuf);i++)
{
    if(stuf[j]=='#')
    {
dstuf[i]=stuf[j+1];
        j=j+2;
    }
    else
    {
dstuf[i]=stuf[j];
j++;
    }
}

printf("%s",dstuf);

    return 0;
}
```

Output:

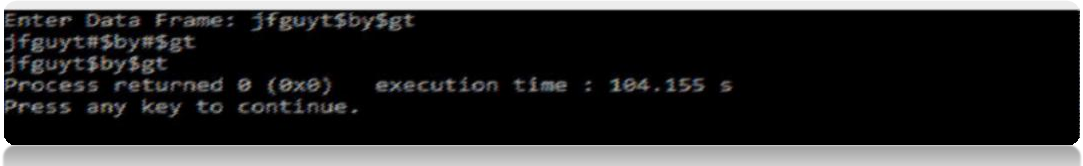
a) If both Flag and ESC is part of data



b) Either no flag or esc character in frame- no stuffing



c) If flag is part of frame, stuff '#' before '$'



d) If ESC is part of frame, stuff '#' before '#'

```
Enter Data Frame: hfhg#jfj#fg
hfhg##jfj##fg
hfhg#jfj#fg
Process returned 0 (0x0)   execution time : 10.007 s
Press any key to continue.
```

Bit Stuffing

```
#include<stdio.h>

#include<stdlib.h>


intmain()

{

 char *p,*q;

 char t;

 char in[50],stuf[50],dstuf[50];


int c=0;


printf("input data Frame data: ");

scanf("%s",in);


 p=in;

 q=stuf;
```

```
 while(*p!='\0')
 {
  if(*p=='0')
  {
   *q=*p;
   q++;
   p++;
  }
  else
  {
while(*p=='1' && c!=5)
   {
c++;
     *q=*p;
     q++;
     p++;
   }

   if(c==5)
   {
      if(*(q-6)=='0'){
     *q='0';
     q++;
      }
   }
   c=0;
  }
 }
 *q='\0';
printf("\nStuffed data frame: ");
printf("%s",stuf);
```

```
 p=stuf;
 q=dstuf;
 while(*p!='\0')
 {
  if(*p=='0')
  {
   *q=*p;
   q++;
   p++;
  }
  else
  {
while(*p=='1' && c!=5)
   {
c++;
     *q=*p;
     q++;
     p++;
   }
   if(c==5)
   {
     p++;
   }
   c=0;
  }
 }
 *q='\0';
printf("\ndestuffed data frame: ");
printf("%s\n",dstuf);
 return 0;
```

}

Output:

a) 00000000000000000000000000000000

```
input data Frame data: 00000000000000000000000000000000

Stuffed data frame: 000000000000000000000000000000000
destuffed data frame: 00000000000000000000000000000000

Process returned 0 (0x0)   execution time : 8.927 s
Press any key to continue.
```

b) 11111111111111111111111111

```
input data Frame data: 11111111111111111111111111

Stuffed data frame: 111111111111111111111111111
destuffed data frame: 11111111111111111111

Process returned 0 (0x0)   execution time : 5.651 s
Press any key to continue.
```

c) 011111000011111111111111011111

```
Input data Frame data: 011111000011111111111111011111

Stuffed data frame: 0111110000011111011111011101111110
destuffed data frame: 011111000011111111111111011111

Process returned 0 (0x0)   execution time : 20.393 s
Press any key to continue.
```

d) 011111111111000000000111110

```
input data Frame data: 011111111111000000000111110

Stuffed data frame: 0111110111111010000000001111100
destuffed data frame: 011111111111000000000111110

Process returned 0 (0x0)   execution time : 7.516 s
Press any key to continue.
```

Inference:

1. In case of when text contains one or more escape characters followed by a flag. To solve this problem, the escape characters that are part of the text are marked by another escape character i.e., if the escape character is part of the text, an extra one is added to show that the second one is part of the text.
2. Bit stuffing cannot be done if we only get 5 consecutive 1 it applicable only if whenever we encounter 11111(5 ones) after a 0 we would insert 0 in the data after 011111 0

Result: Hence, we successfully implemented of Bit Stuffing/De-stuffing and Byte Stuffing/De-stuffing to delimit MAC frames using ‘C’ Program

EXPERIMENT 8

COMPUTER COMMUNICATION LAB

NAME: Debasya Sahoo

REG NO-15BEC1111

Study and Analysis the Performance of CSMA/CA and CSMA/CD using NETSIM

Aim: To study the performance of a CSMA/CA, by changing the number of the network nodes , using NetSim and plotting the graph using excel.

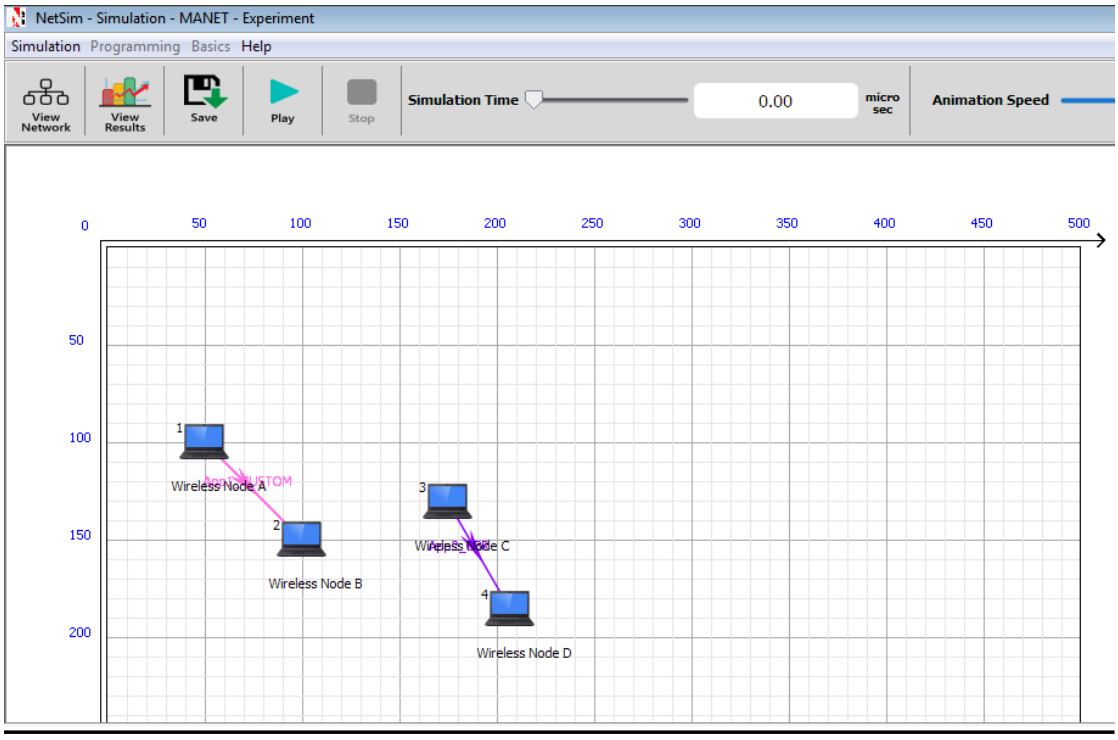
ONLY 2 NODES:

DELAY AND THROUGHPUT:

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CUSTOM	0.194589	1691.533973

4 NODES:



DELAY AND THROUGHPUT

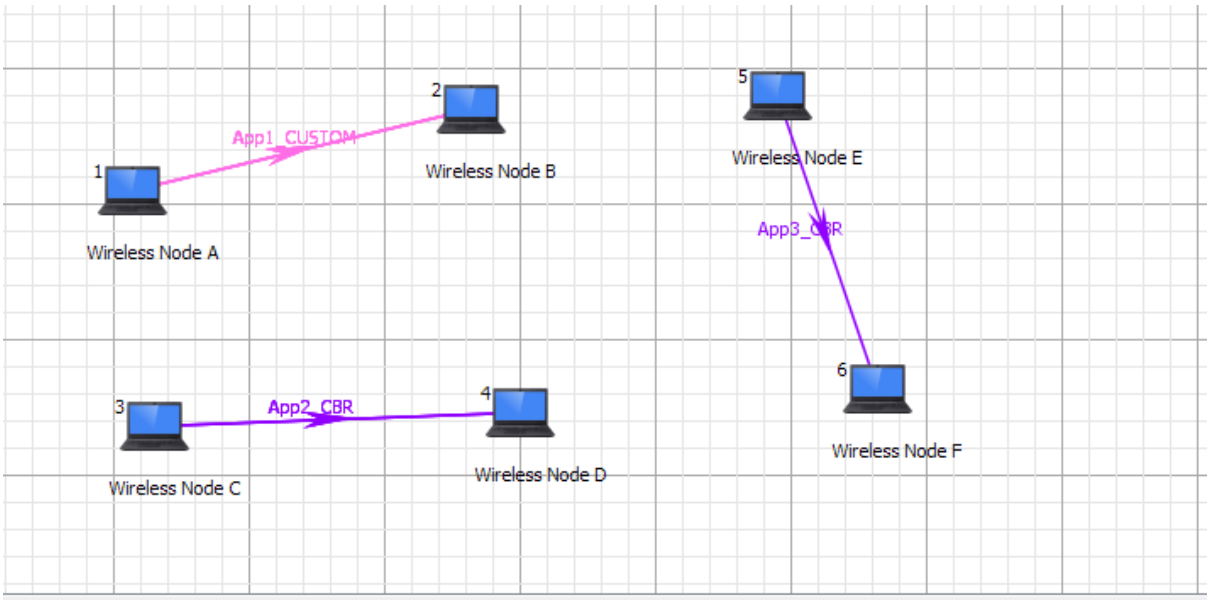
Application_Metrics_Table

Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CUSTOM	0.194589	2523.771778
2	*Application_throughput_plot*	APP2_CBR	0.194589	2537.330407

AVG THROUGHPUT: 0.1945Mbps
AVG DELAY: 2530.55 ms

6 NODES:



DELAY AND THROUGHPUT

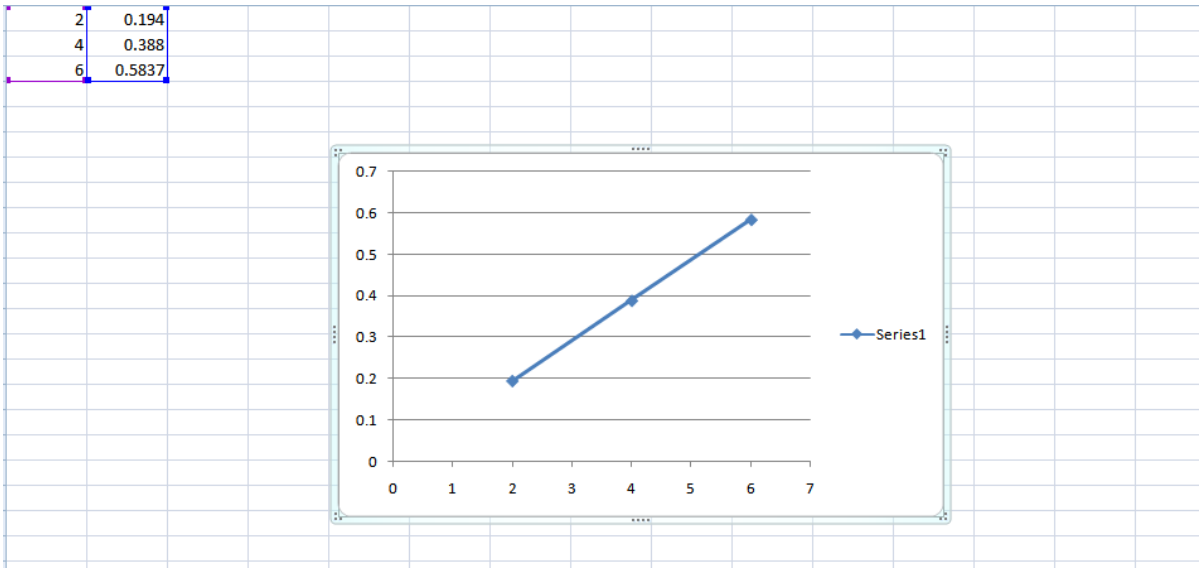
Application_metrics Detailed View

Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CUSTOM	0.194706	3277.918772
2	*Application_throughput_plot*	APP2_CBR	0.194589	3317.547331
3	*Application_throughput_plot*	APP3_CBR	0.194589	3586.903153

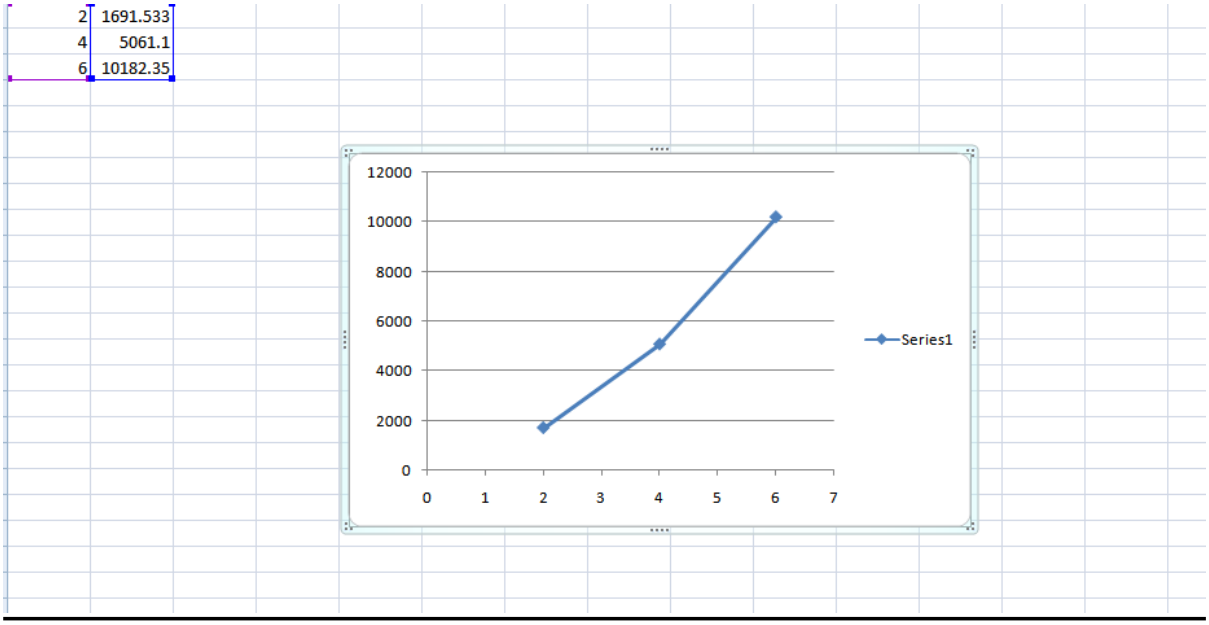
AVG THROUGHPUT:0.1945Mbps

AVG DELAY:3394.16ms

PLOT OF THORUGHPUT VS NODES:



PLOT OF DELAY VS NODES:



Study and Analysis the Performance of CSMA/CD using NETSIM

Aim: To study the performance of a CSMA/CD, by changing various factors of the network, using NetSim.

Different Factors are

- Increasing the persistance value with fixed number of nodes and applications.
- Increasing the number of application traffics (light load to heavy load)

Network Simulation Specifications:

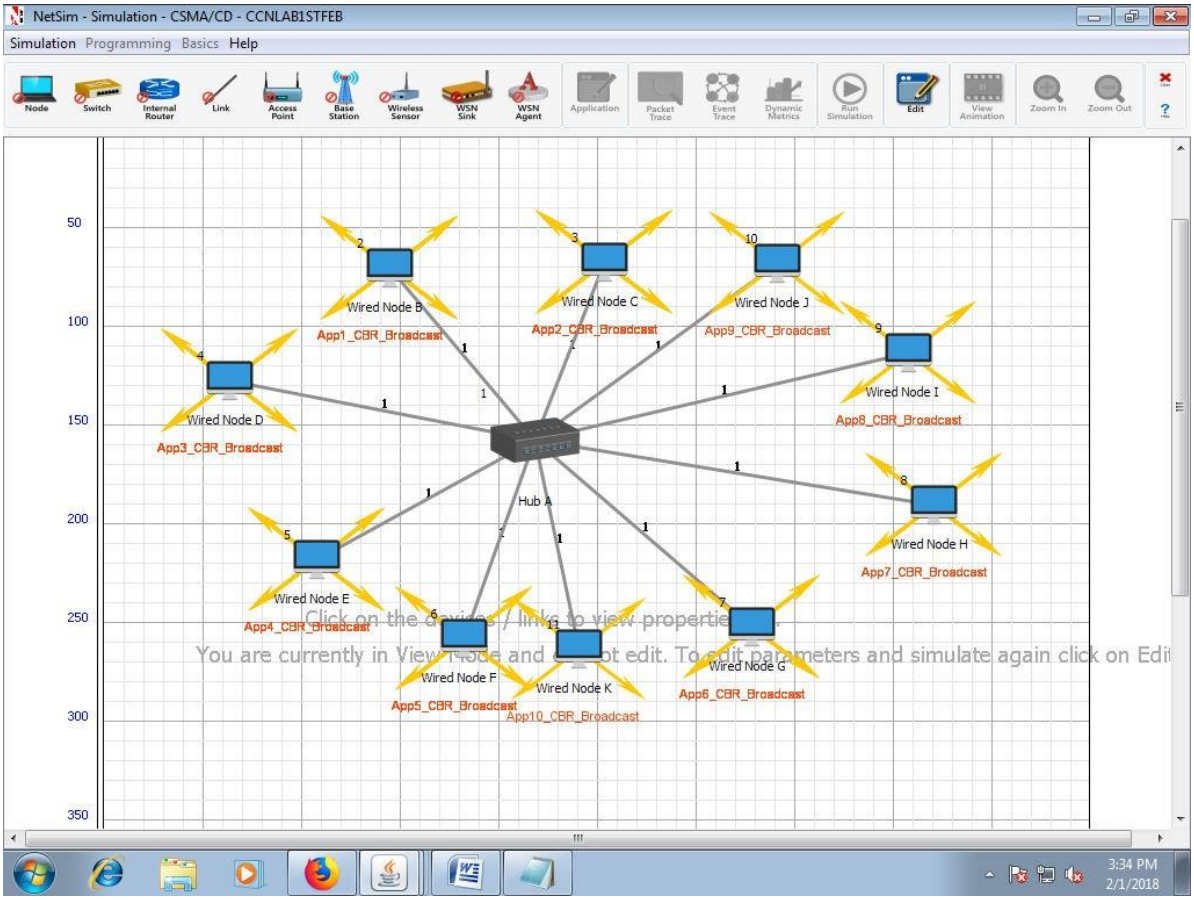
1. No of Nodes = 10 ; Set Destination Node = 0 (for broadcast) ; No of Applications (CBR) = 10; Inter-arrival Time : 1000ms; uplink and downlink bit error rate = 0; persistance value= 1 (vary the values:1/2, 1/3, 1/4, 1/5,1/6,1/7,1/8,1/9 1/10, 1/11)
   Only UDP sources ; Dynamic ARP enabled ; Simulation time : 10seconds

   ✓ **Inference (Plots):** Calculate the Average Application Throughput vs Persistance value.

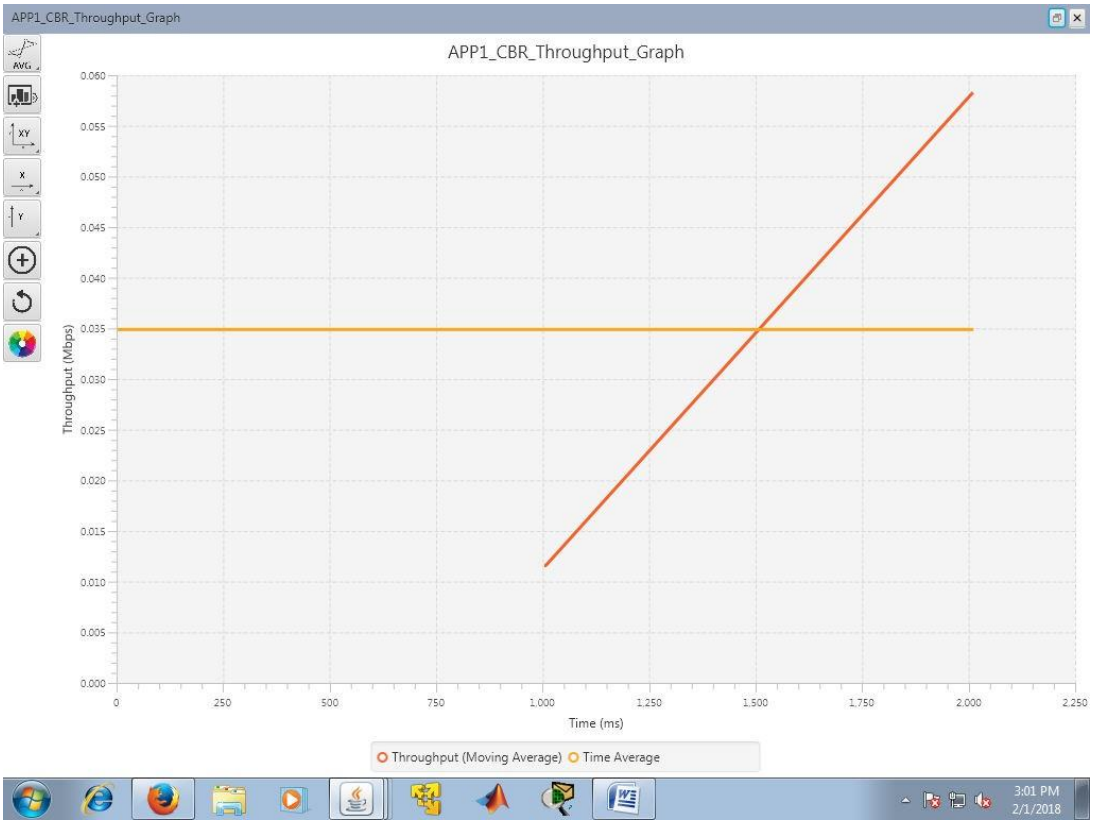
2. No of Nodes = 10; Packet Inter-arrival Time = 2500ms ; Increase the number of Application Traffics (nodes in this case) = 1, 2, 3, 5, 7 ; No of Destination Node = 1 (node 9) ; Link Speed = Switch to Destination = 10Mbps (Duplex)

   ✓ **Inference (Plots):**
   ✓ Calculate the Total Packets Errored vs No of Application Traffics
   ✓ Calculate the Average Application Throughput vs No of Application Traffic/Nodes and draw the comparison graph for different number of transmission.

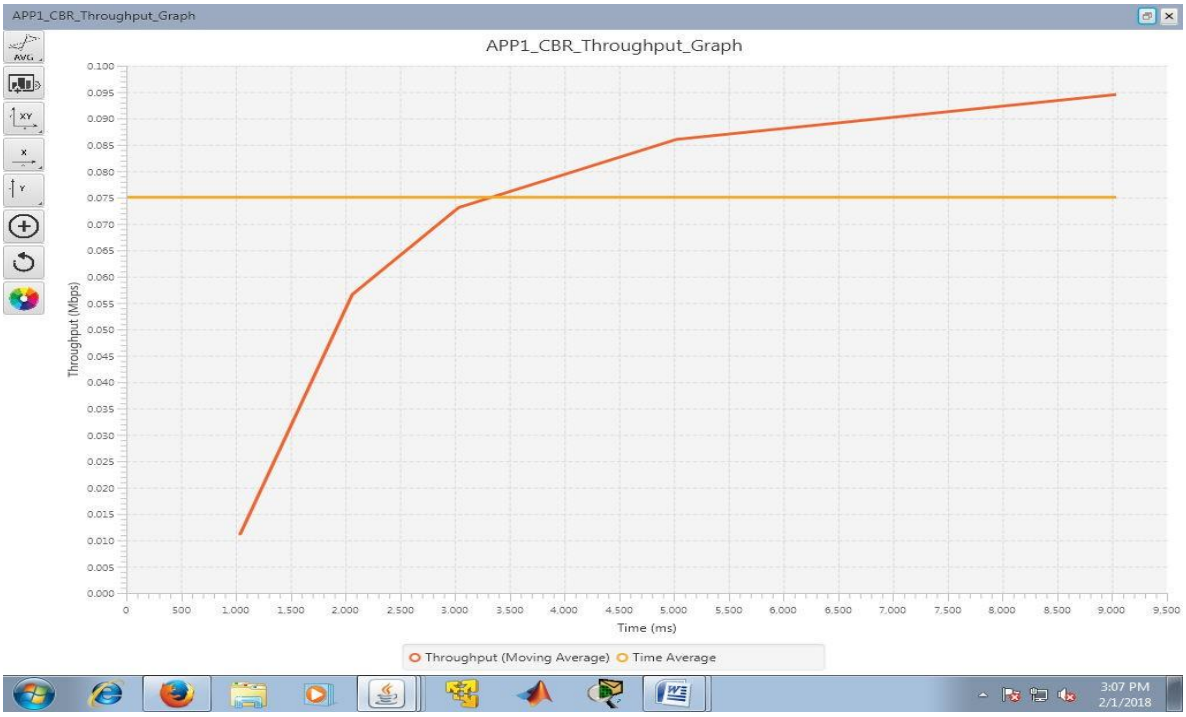
SOLUTION 1:


NetSim - Simulation - CSMA/CD - CCNLAB1STFEB
Simulation Programming Basics Help
Node
Switch
Internal Router
Link
Access Point
Base Station
Wireless Sensor
WSN Sink
WSN Agent
Application
Packet Trace
Event Trace
Dynamic Metrics
Run Simulation
Edit
View Animation
Zoom In
Zoom Out
Wired Node B
App1_CBR_Broadcast
Wired Node C
App2_CBR_Broadcast
Wired Node J
App9_CBR_Broadcast
Wired Node D
App3_CBR_Broadcast
Wired Node I
App8_CBR_Broadcast
Hub A
Wired Node H
App7_CBR_Broadcast
Wired Node E
App4_CBR_Broadcast
Wired Node F
App5_CBR_Broadcast
Wired Node K
App10_CBR_Broadcast
Wired Node G
App6_CBR_Broadcast
Click on the devices / links to view properties.
You are currently in View Mode and cannot edit. To edit parameters and simulate again click on Edit
3:34 PM
2/1/2018

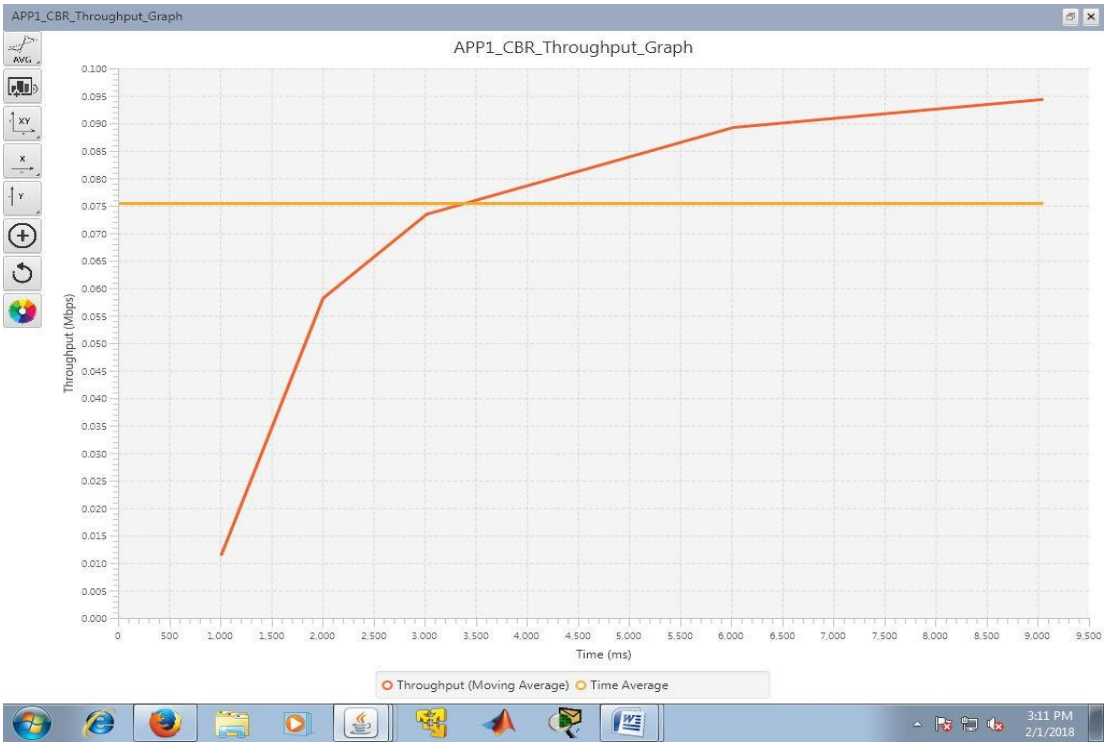
PERSISTENCE=1,THROUGHPUT=0.035



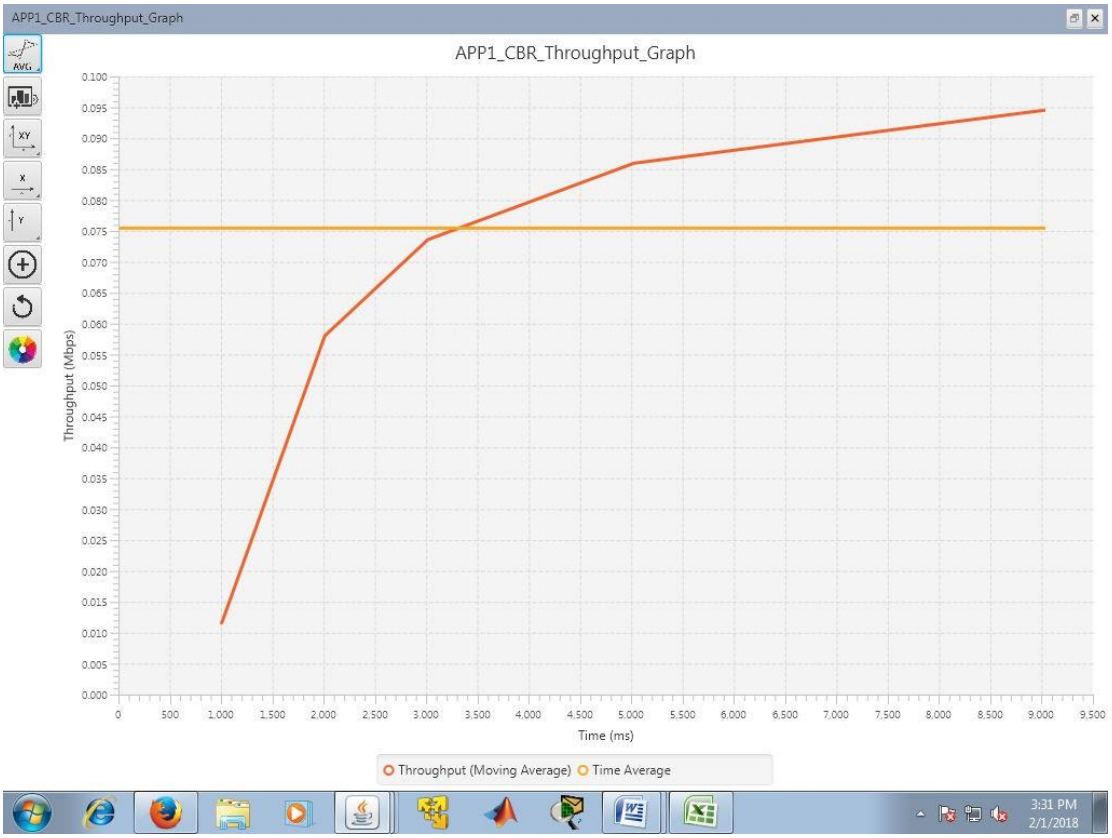
PERSISTENCE=1/2,THROUGHPUT=0.075



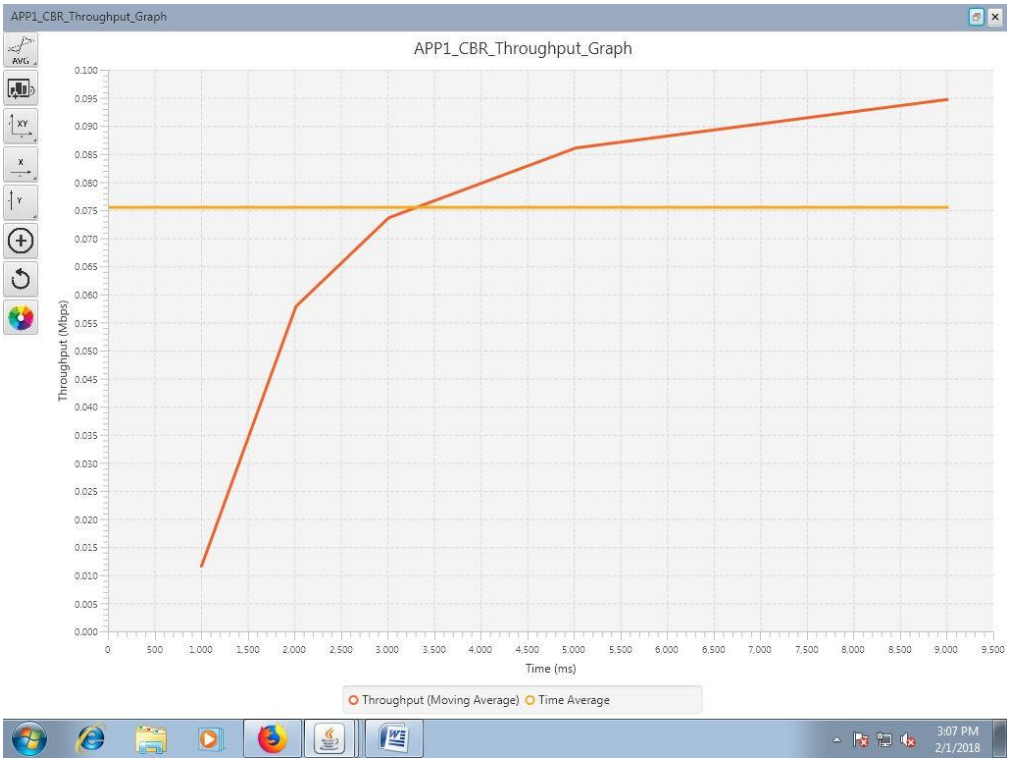
PERSISTENCE=1/3,THROUGHPUT=0.076



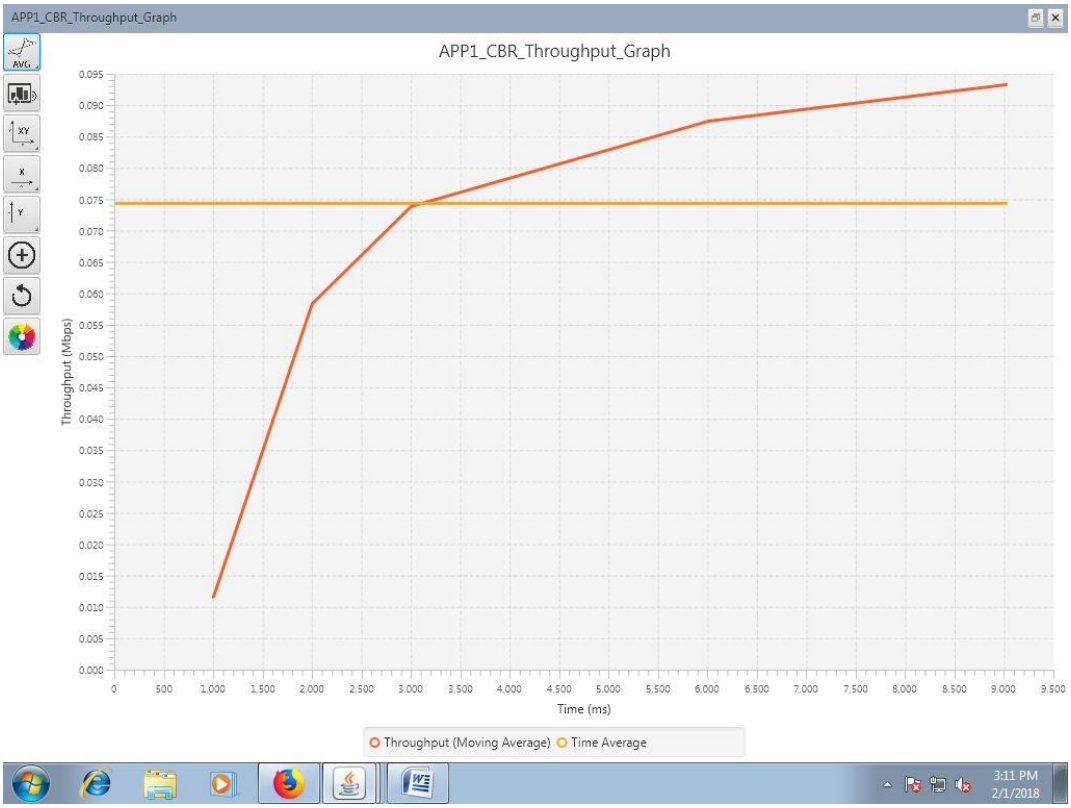
PERSISTENCE=1/4,THROUGHPUT=0.0754



PERSISTENCE=1/5,THROUGHPUT=0.076

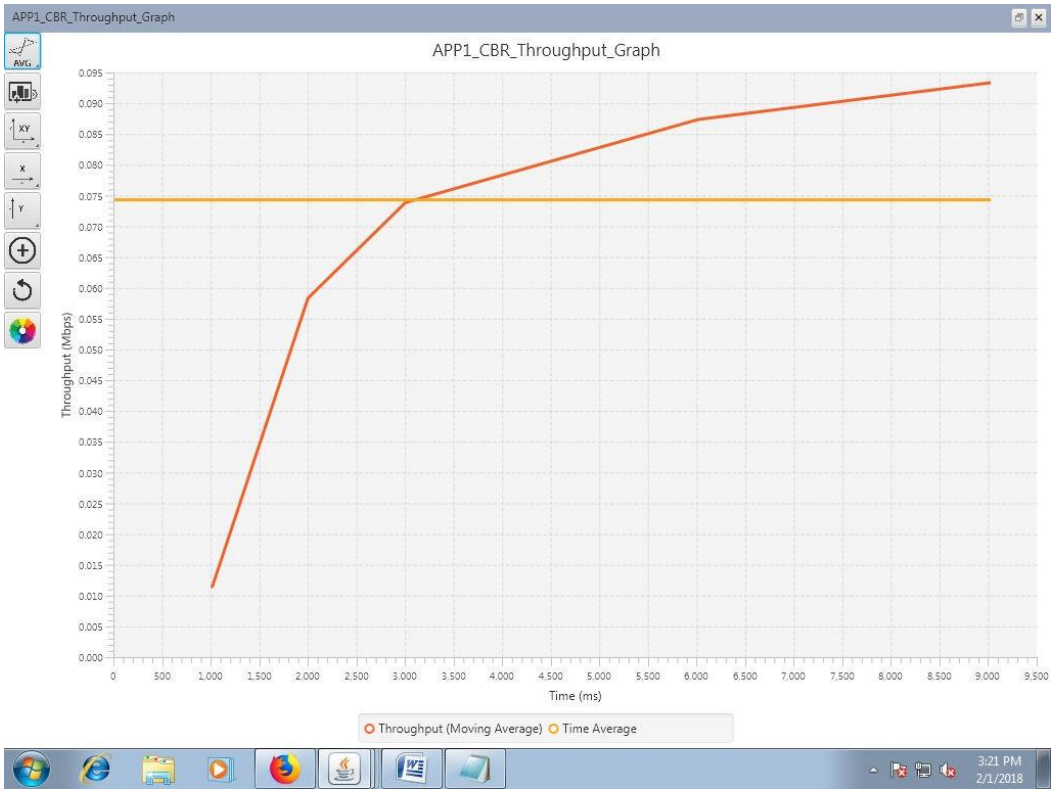


PERSISTENCE=1/6,THROUGHPUT=0.0743

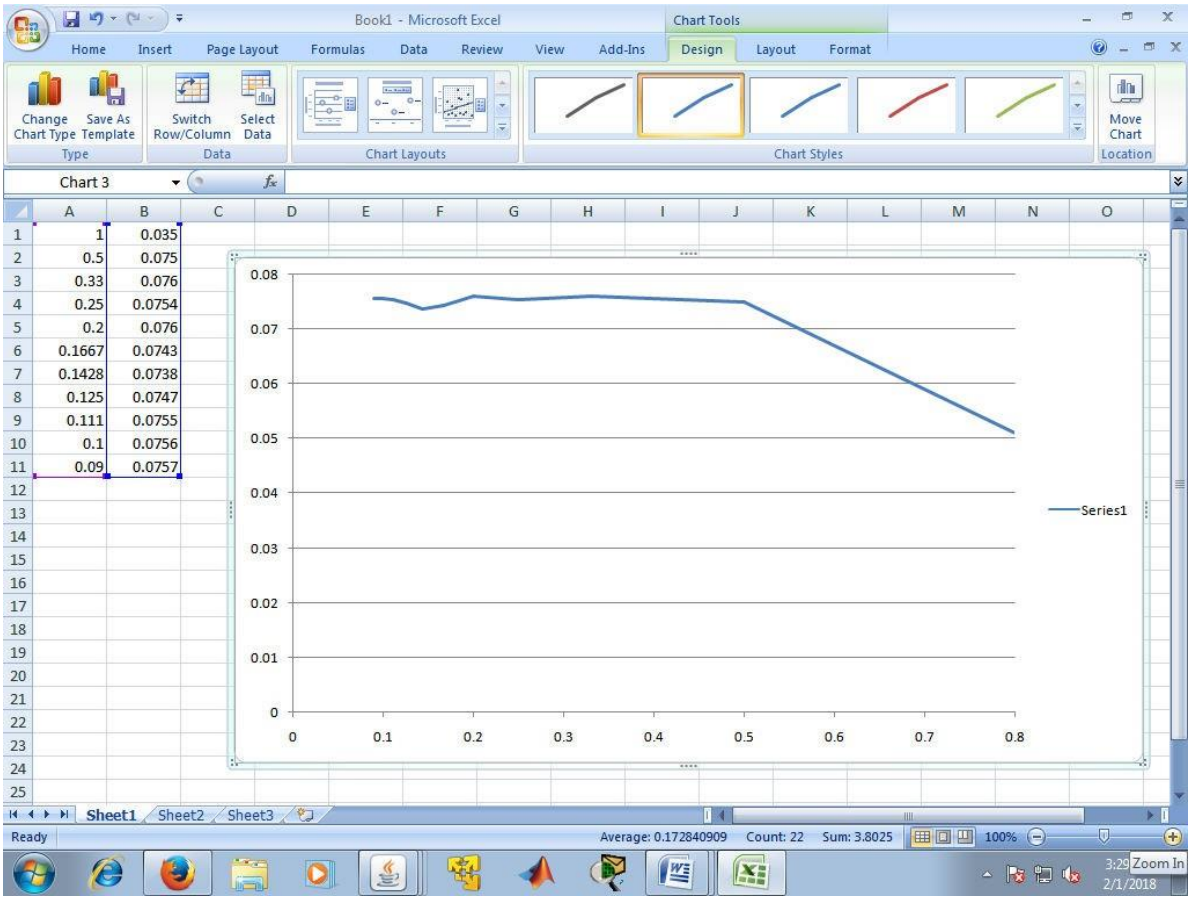


PERSISTENCE=1/7,THROUGHPUT=0.0738

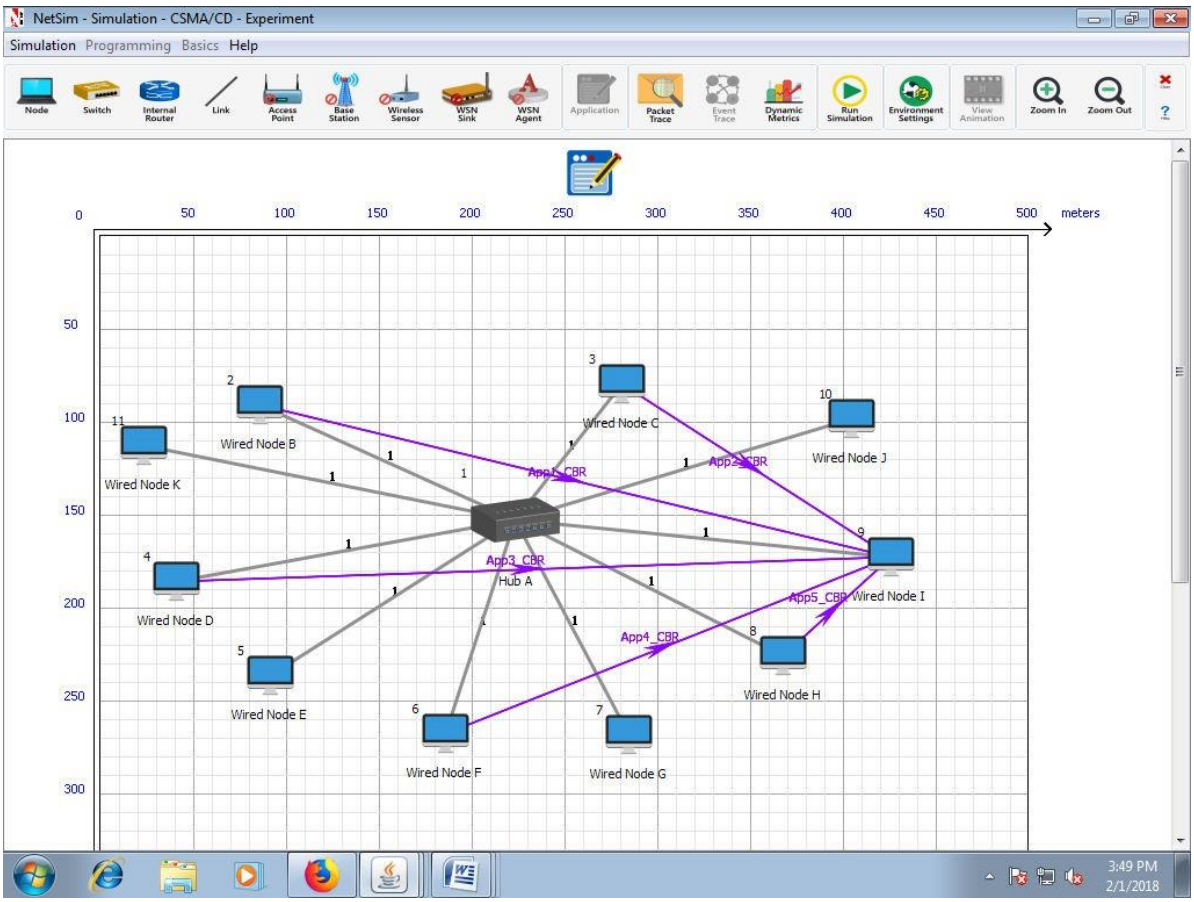
PERSISTENCE=1/8,THROUGHPUT=0.0747



AVERAGE THROUGHPUT VS PERSISTENCE GRAPH:


Book1 - Microsoft Excel
Chart Tools
Home
Insert
Page Layout
Formulas
Data
Review
View
Add-Ins
Design
Layout
Format
Change Chart Type
Save As Template
Type
Switch Row/Column
Select Data
Data
Chart Layouts
Chart Styles
Move Chart
Location
Chart 3
A B C D E F G H I J K L M N O
1 1 0.035
2 0.5 0.075
3 0.33 0.076
4 0.25 0.0754
5 0.2 0.076
6 0.1667 0.0743
7 0.1428 0.0738
8 0.125 0.0747
9 0.111 0.0755
10 0.1 0.0756
11 0.09 0.0757
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
Series1
Sheet1
Sheet2
Sheet3
Ready
Average: 0.172840909
Count: 22
Sum: 3.8025
100%
Zoom In
3:29
2/1/2018

solution 2:


NetSim - Simulation - CSMA/CD - Experiment
Simulation
Programming
Basics
Help
Node
Switch
Internal Router
Link
Access Point
Base Station
Wireless Sensor
WSN Sink
WSN Agent
Application
Packet Trace
Event Trace
Dynamic Metrics
Run Simulation
Environment Settings
View Animation
Zoom In
Zoom Out
0 50 100 150 200 250 300 350 400 450 500 meters
50
100
150
200
250
300
Wired Node B
Wired Node C
Wired Node J
Wired Node K
Wired Node D
Wired Node E
Wired Node F
Wired Node G
Wired Node H
Wired Node I
Hub A
App1_CBR
App2_CBR
App3_CBR
App4_CBR
App5_CBR
3:49 PM
2/1/2018

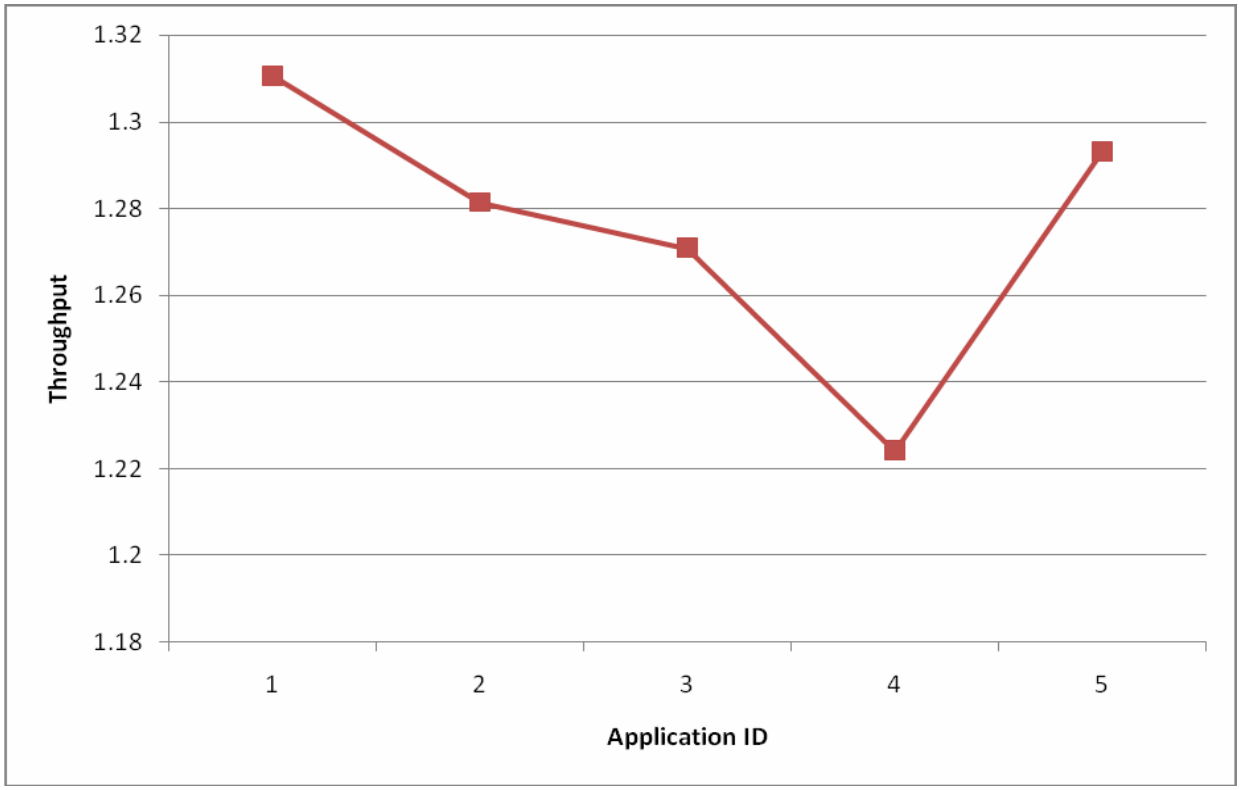
Network Matrix

Network_Metrics_Table

Network_Metrics Detailed View

Link_id	Link_throughput_plot	Packet_transmi...		Packet_errored		Packet_collided	
		Data	Control	Data	Control	Data	Control
All	NA	9542	0	752	0	3328	0
1	Link_throughput	9542	0	752	0	3328	0

THROUGHPUT VS APPLICATION ID



RESULTS : Experiment is successfully verified using NetSim

Experiment 9

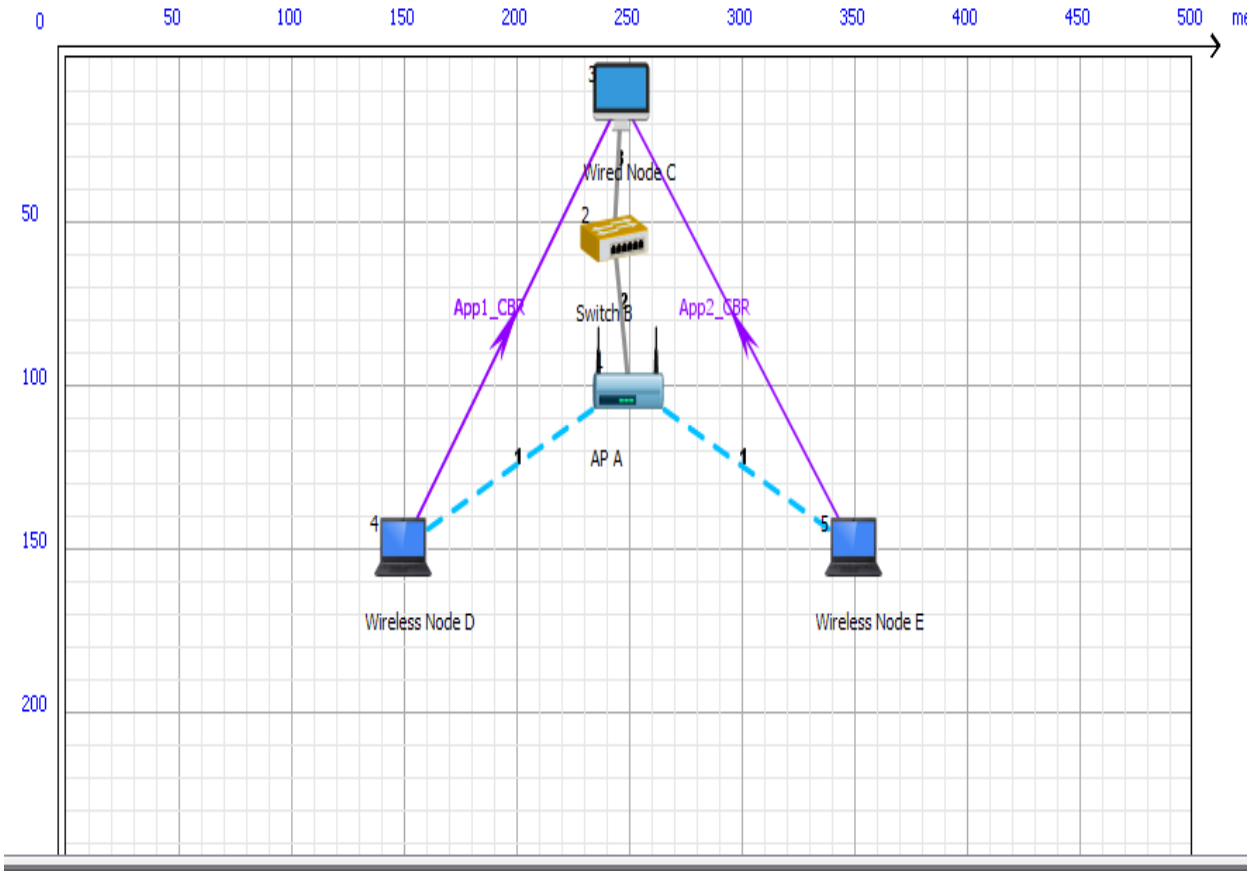
Computer Communication Lab

Name : Debasya Sahoo

Reg. :15BEC1111

Aim: To study Hidden Node Problem with and without RTS/CTS in WLAN.

Device connection:



Simulation Results:

Sample 1:

Application Metrics - Average throughput

Application_Metrics_Table

Application_metrics Detailed View

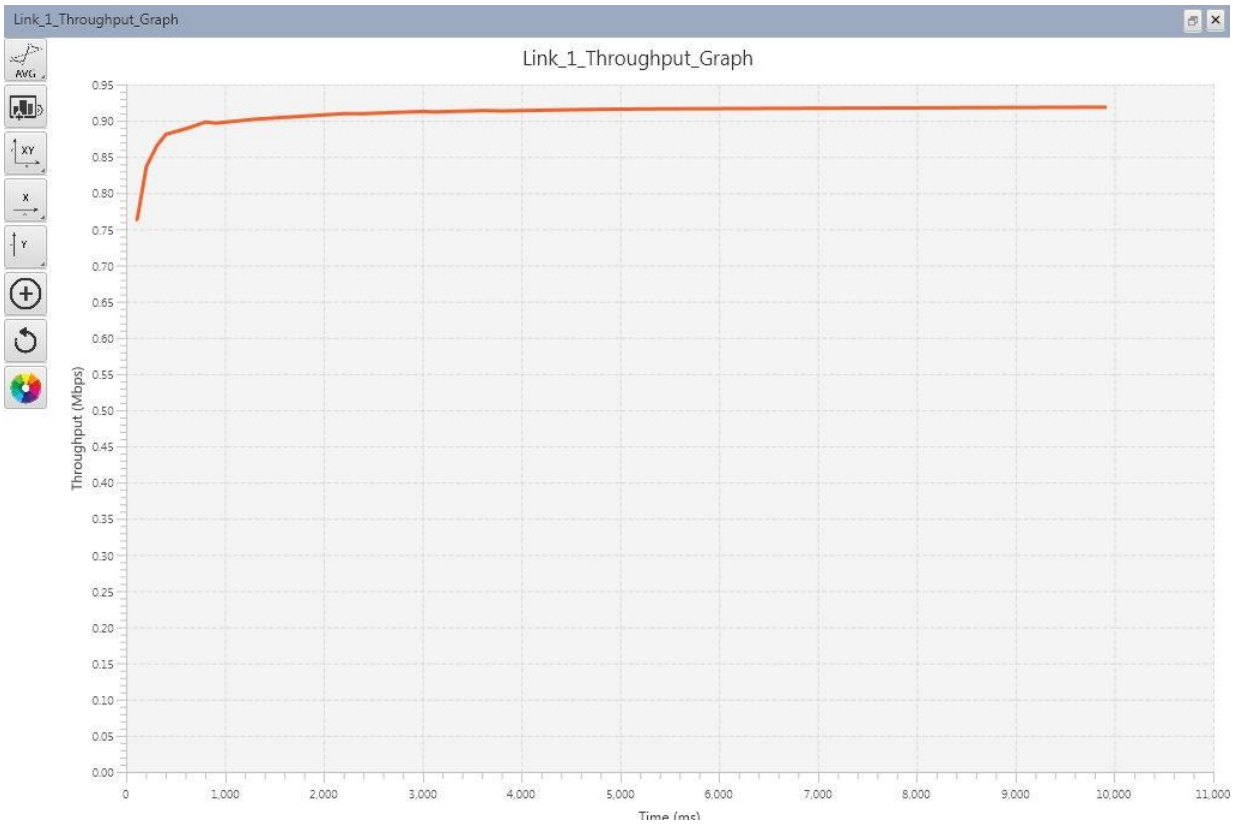
Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	0.192720	3458856.232936
2	*Application_throughput_plot*	APP2_CBR	0.230096	3022196.016555

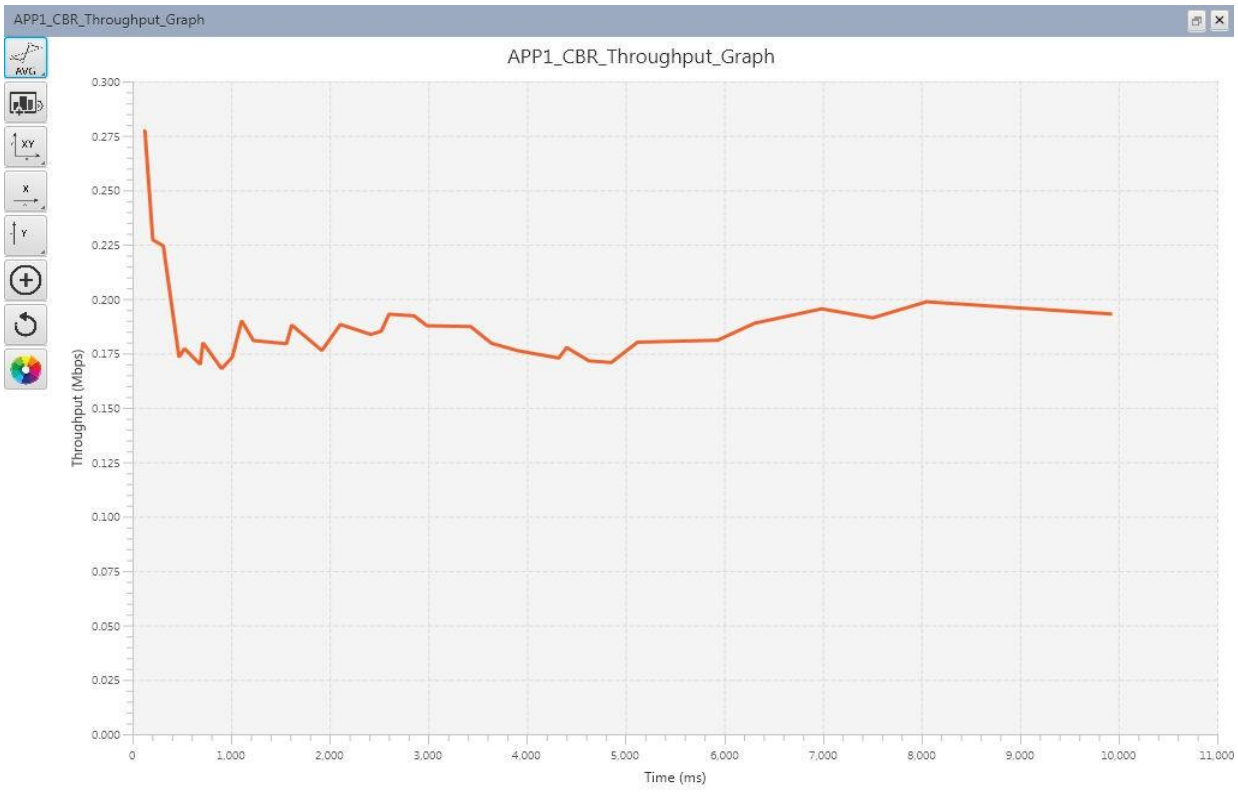
Network Metrics- Packets collided:

Network_Metrics_Table

Network_Metrics Detailed View

Link_id	*Link_throughput_plot*	Packet_transmi...		Packet_errored		Packet_collided	
		Data	Control	Data	Control	Data	Control
All	*NA*	1473	363	3	0	383	0
1	*Link_throughput*	748	363	2	0	383	0
2	*Link_throughput*	363	0	1	0	0	0
3	*Link_throughput*	362	0	0	0	0	0


Link_1_Throughput_Graph
Link_1_Throughput_Graph
Throughput (Mbps)
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
10,000
11,000
Time (ms)


APP1_CBR_Throughput_Graph
APP1_CBR_Throughput_Graph
Throughput (Mbps)
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
10,000
11,000
Time (ms)

Sample 2:

Application metrics- Average throughput:

Application_Metrics_Table

Application_metrics Detailed View

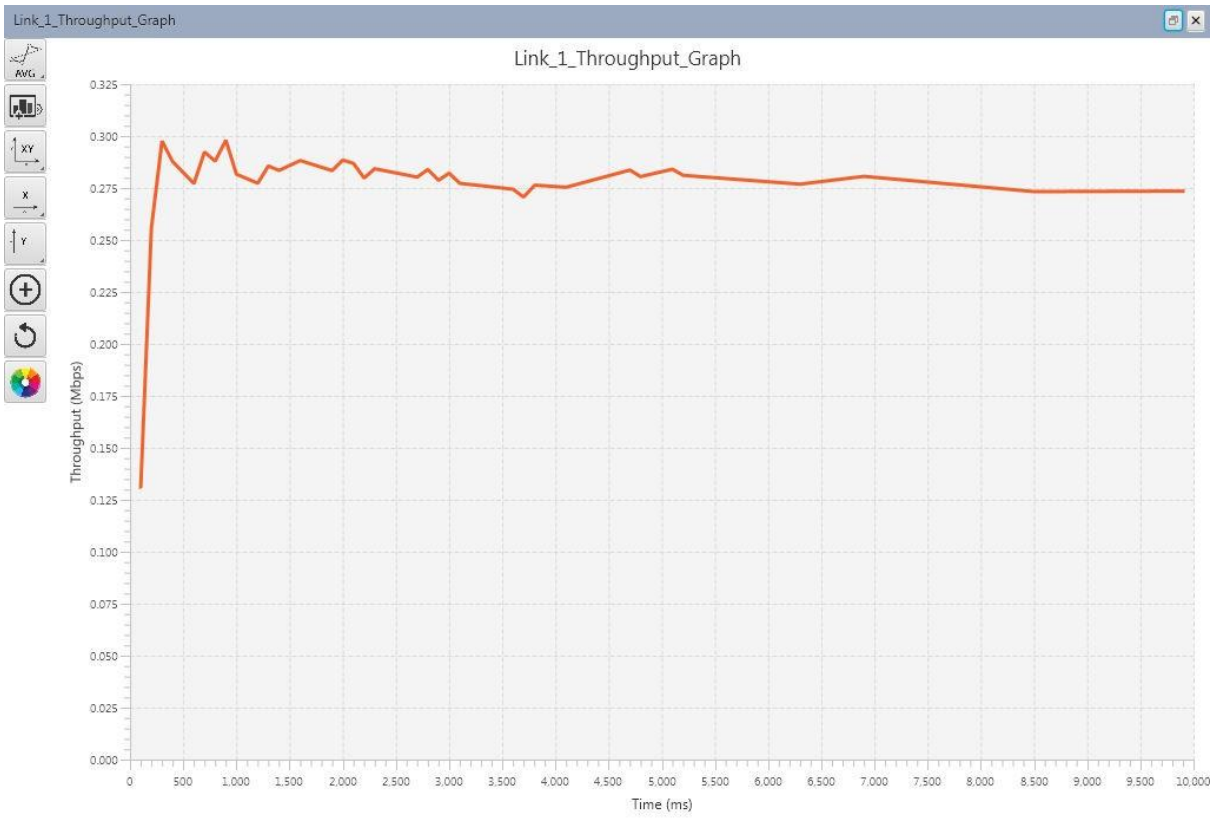
Application Id	*Throughput Plot*	Application Name	Throughput (Mbps)	Delay(microsec)
1	*Application_throughput_plot*	APP1_CBR	0.128480	3340336.770797
2	*Application_throughput_plot*	APP2_CBR	0.119136	4263686.205463

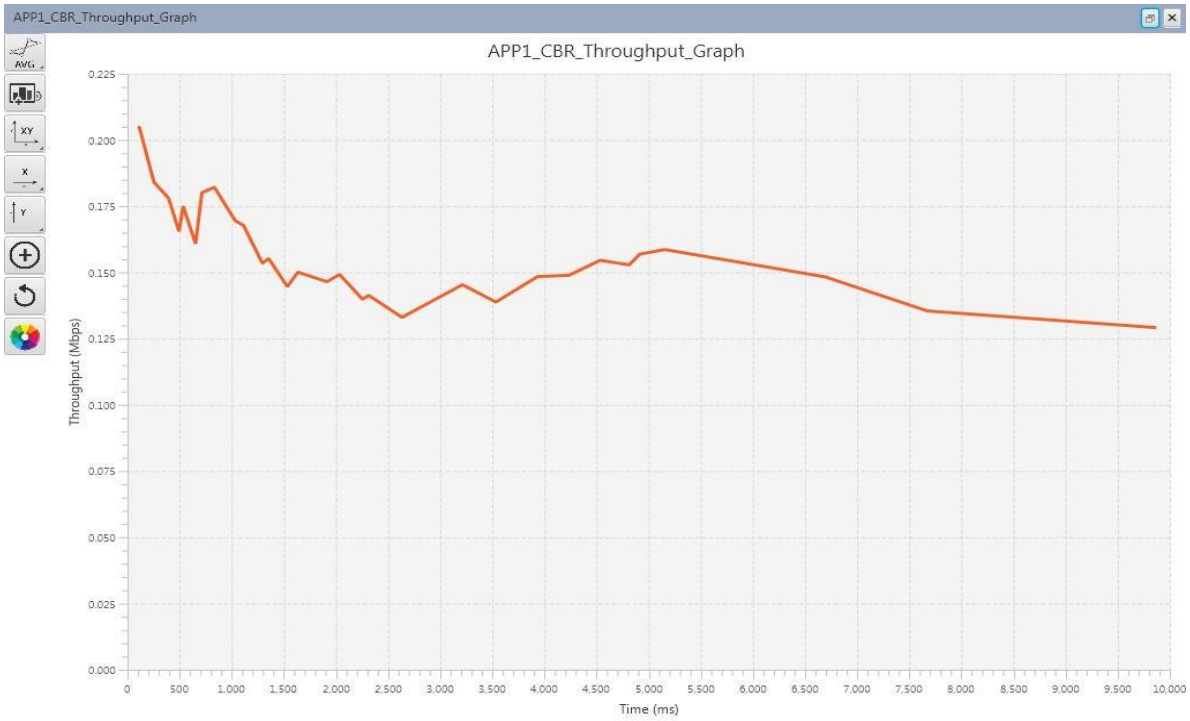
Network metrics- Packets collided:

Network_Metrics_Table

Network_Metrics Detailed View

Link_id	*Link_throughput_plot*	Packet_transmi...		Packet_errored		Packet_collided	
		Data	Control	Data	Control	Data	Control
All	*NA*	638	943	1	0	0	295
1	*Link_throughput*	213	943	0	0	0	295
2	*Link_throughput*	213	0	1	0	0	0
3	*Link_throughput*	212	0	0	0	0	0


Link_1_Throughput_Graph
Link_1_Throughput_Graph
Throughput (Mbps)
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
5,000
5,500
6,000
6,500
7,000
7,500
8,000
8,500
9,000
9,500
10,000
Time (ms)


APP1_CBR_Throughput_Graph
APP1_CBR_Throughput_Graph
Throughput (Mbps)
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
5,000
5,500
6,000
6,500
7,000
7,500
8,000
8,500
9,000
9,500
10,000
Time (ms)

Observation:

Packets Collision:

Collided packets	Sample 1(Without RTS/CTS)	Sample 2(With RTS/CTS)
Data packets	383	0
Control Packets	0	295

Average throughput:

Metrics(Throughput in Mbps)	Sample 1(Without RTS/CTS)	Sample 2(With RTS/CTS)
Application 1	0.1927	0.1284
Application 2	0.2300	0.1191

Inference:

The comparison with hidden stations shows that RTS/CTS mechanism reduces the data packet collision.

In sample 1 due to hidden node problem, packets collide continuously. In sample 2 on enabling RTS/CTS, the source node will refrain from sending a data frame until it completes a RTS/CTS handshake with another station, such as an access point. A station initiates the process by sending a RTS frame. The access point receives the RTS and responds with a CTS frame. The station must receive a CTS frame before sending the data frame. The CTS also contains a time value that alerts other stations to hold off from accessing the medium while the station initiating the RTS transmits its data. Hence in RTS/CTS mechanism throughput of

the applications are less than the other sample. Because data packets are transmitted after the successful RTS/CTS handshake happened.

Experiment No: 10

Implementation of Dijkstra Algorithm using C

Name : Debasya Sahoo
Reg. No: 15BEC1111

AIM: Implementation of Dijkstra Algorithm – Shortest Path Tree Computation using C

SOFTWARE USED:
C Programming Language (DEV C)

ALGORITHM:

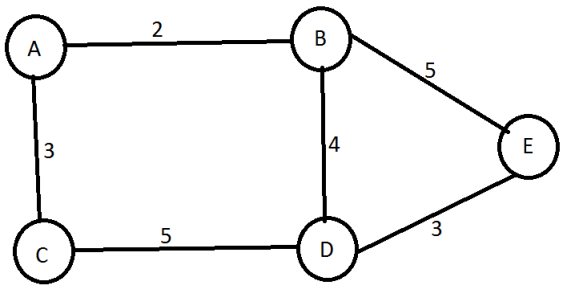
INPUT:

1) **Input the Number of Nodes in the Graph (N)**
2) **Input the 2-D Matrix Link State Database (lsdb[N][N]) or Adjacency Matrix (in Graph Theory Terms) of Size N*N.**
    a. **Weight of the edges – represent the link cost**
    b. **If node is not adjacent neighbor, mark the link cost to infinity (say 999 means unreachable directly) in the Adjacency Matrix**
    c. **Link cost of Node to itself is 0.**
3) **Input the Root Node out of N nodes which computes the Shortest path Tree (running Dijkstra's)**

OUTPUT:

1) **Output a 1-D Array say D[] of size N whose values represents the least cumulative cost for root node to reach to every other node in the network.**

Input Graph (V,E)



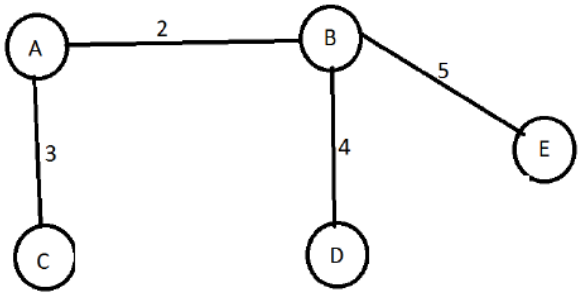
	A	B	C	D	E
A	0	2	3	∞	∞
B	2	0	∞	4	5
C	3	∞	0	5	∞
D	∞	4	5	0	3
E	∞	5	∞	3	0

Output :

Say Root Node is 'A' (say A is node index 0, B is node index 1 etc..)

Output D[] array values :

D[0] = 0 ; D[1] = 2 ; D[2] = 3 ; D[3] = 6 ; D[4]= 7



ALGORITHM

Initialization

1) Initialize D[] values with D[i] = lsdb[root][i] (Pick the row of the root node from the adjacency matrix) (i ∀N)
2) Initialize flag[i] array to 0 for all N nodes
    a. To track whether node is processed or unprocessed.
    b. [Processed means – node is visited and the least cumulative cost is calculated]
    c. If processed, value will be changed to 1

3) **Track two lists Permanent List and Tentative List via flag[i] array.**

   - ✓ **Permanent list are the list of nodes processed whose shortest (least) cumulative cost is calculated and nodes are marked as flagged (flag[i]=1)**
   - ✓ **Tentative – unprocessed – nodes unmarked**

4) **Start with the root node (which was in tentative list) and move it to permanent list & set flag[root]=1**

5) **Among the unprocessed neighbors of root node and find the node 'w' with D[w] minimum among all nodes and put that node in permanent list.**

6) **For every node x which is a neighbor of w and not in permanent list ,**
   find the cumulative cost to reach x via node w
   D[x] = D[w]+ lsdb[w][x]
   and if D[x] value is smaller than it's existing value then update the D[x] value.
   i.e D[x] = min{D[x],D[w]+lsdb[w][x]}.
   Find the node x with min D[x] and put that in permanent list

7) **Repeat step 6 until there is no node left in tentative list, i.e. all f[i] ==1.**

8) **Print the cumulative cost of each node w.r.t. to root node i.e print the values of D[] array**

Code:

```
#include<stdio.h>
#define infinity 999
/*void dijkstra(int num, int root, int lsdb[10][10], int D[10]){


   }*/

int min(int a, int b){
   int min=0;
   if(a<b){
        return a;
   }else{
```

```
        return b;
    }
}

int main(){
    int num,root, lsdb[10][10], D[10], flag[10];
    int x,i,j,k,w;
    printf("enter the number of nodes");
    scanf("%d",&num);
    printf("enter the root node");
    scanf("%d", &root);
    printf("Enter the lsdb matrix");
    for(i=0;i<num;i++){
        for(j=0; j<num;j++){
            scanf("%d", &lsdb[i][j]);
            }
        }
    for(i=0;i<num;i++){
        D[i]= lsdb[root][i];
        }
    printf("Initial root row:");
    for(i=0;i<num;i++){
            printf("%d\t", D[i]);
    }

    for(k =0;k<num;k++){
        flag[i]=0;
        }

    for(x=0;x<num;x++){
        for(w=0;w<num;w++){
        D[x]=min(D[x],D[w]+lsdb[w][x]);
        }
    }
    printf("\nFinal row:\n");
    for(i=0;i<num;i++){
            printf("%d\t", D[i]);
            }
    //dijkstra(num,root,lsdb,D);
    getchar();
    return 0;
    }
```

Out put

```
enter the number of nodes5
enter the root node0
Enter the lsdb matrix0 2 3 999 999
2 0 999 4 5
3 999 0 5 999
999 4 5 0 3
999 5 999 3 0
Initial root row:0      2       3       999     999
Final row:
0       2       3       6       7
```

Experiment 11

Simulation and Analysis of TCP Congestion Control Mechanism in Ns2

Name: Debasya Sahoo

Registration Number: 15BEC1111

AIM:

To simulate the TCP congestion control mechanism in NS2 and analyze the different phases of TCP on adaptive data rate control to control network congestion.

SOFTWARE USED:

Network Simulator (NS2), 2.35 version.

THEORY:

Network Simulator

A package of tools that simulates behaviour of networks:

Create network topologies.
Log events that happen under any load.
Analyze events to understand the network behaviour.

Network Animator (NAM)

A visual aid showing how packets flow along the network.

STEPS FOR EXECUTION:

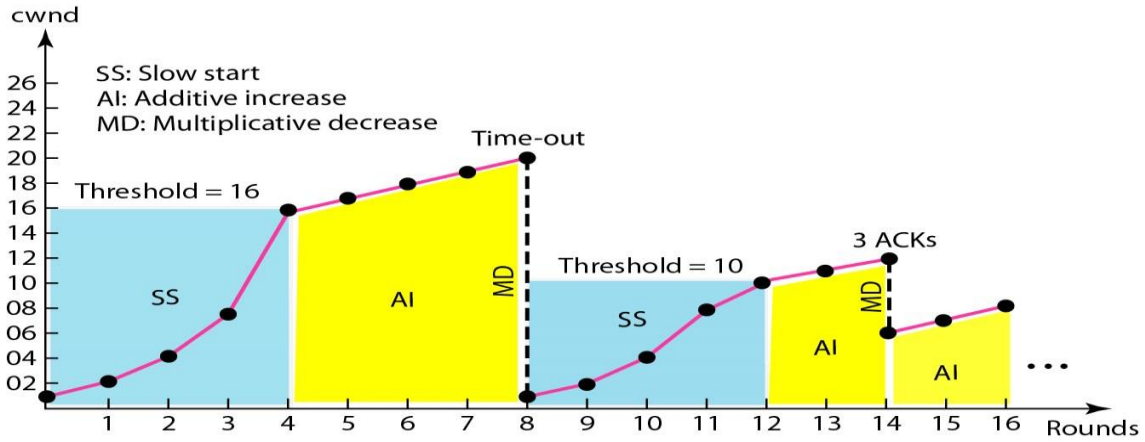
- Initialise a simulator class.
- Define the nodes.
- Queuing of the models.
- Topology is defined.
- Define the agents.

- Stack agents and applications.
- Execute the visualization to see how definition is going by NAM.
- Performance analysis by trace.
- Post analysis using tracegraph

PROCEDURE:

3 phases

- Slow-Start Phase
    - Sender starts with very slow rate of transmission
    - Increases rapidly till threshold
- Congestion Avoidance
    - When Threshold reached, Reduce Data rate to avoid congestion
- Congestion Detection
    - Sender goes back to Slow Start / Congestion Avoidance phase depending on how congestion is detected

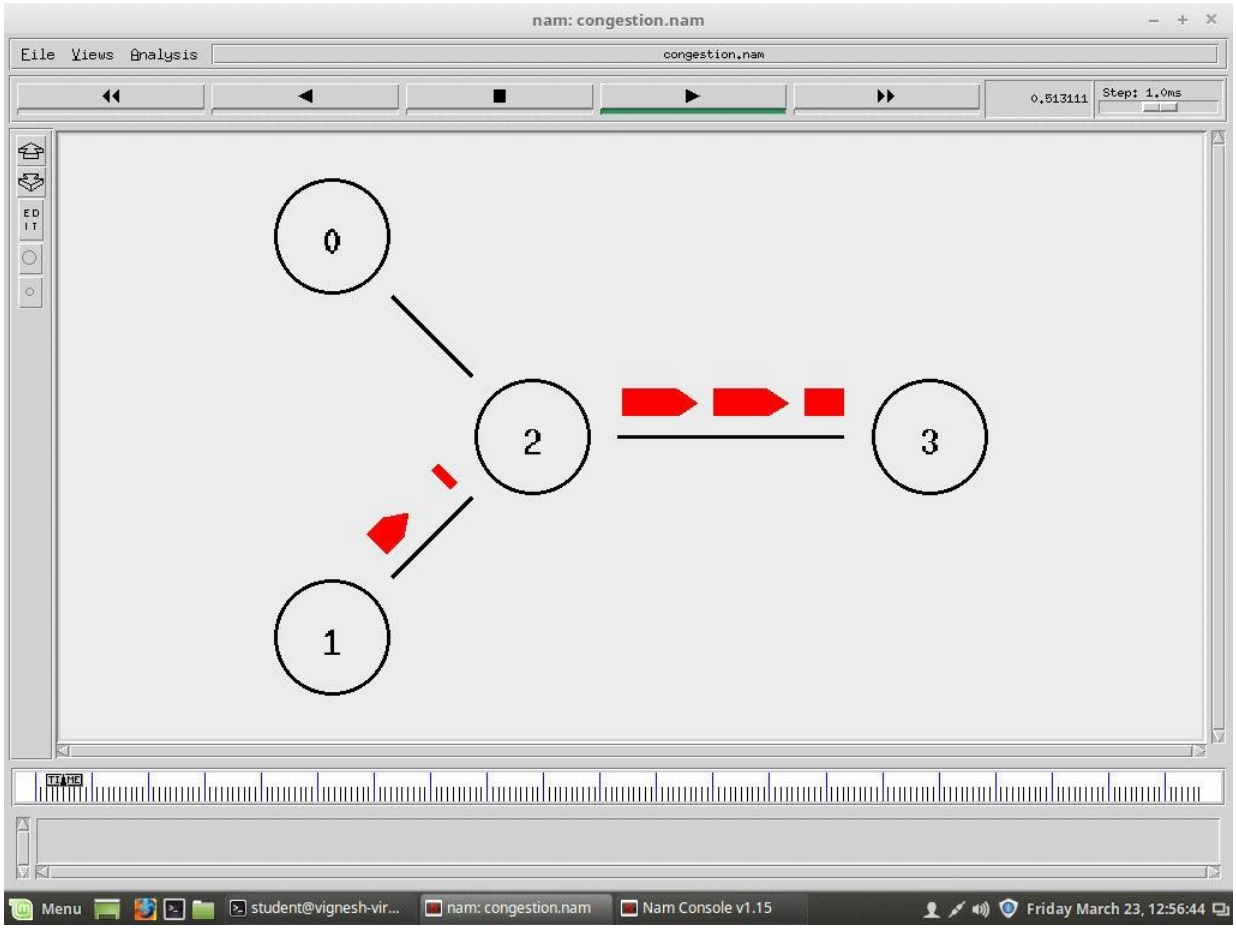


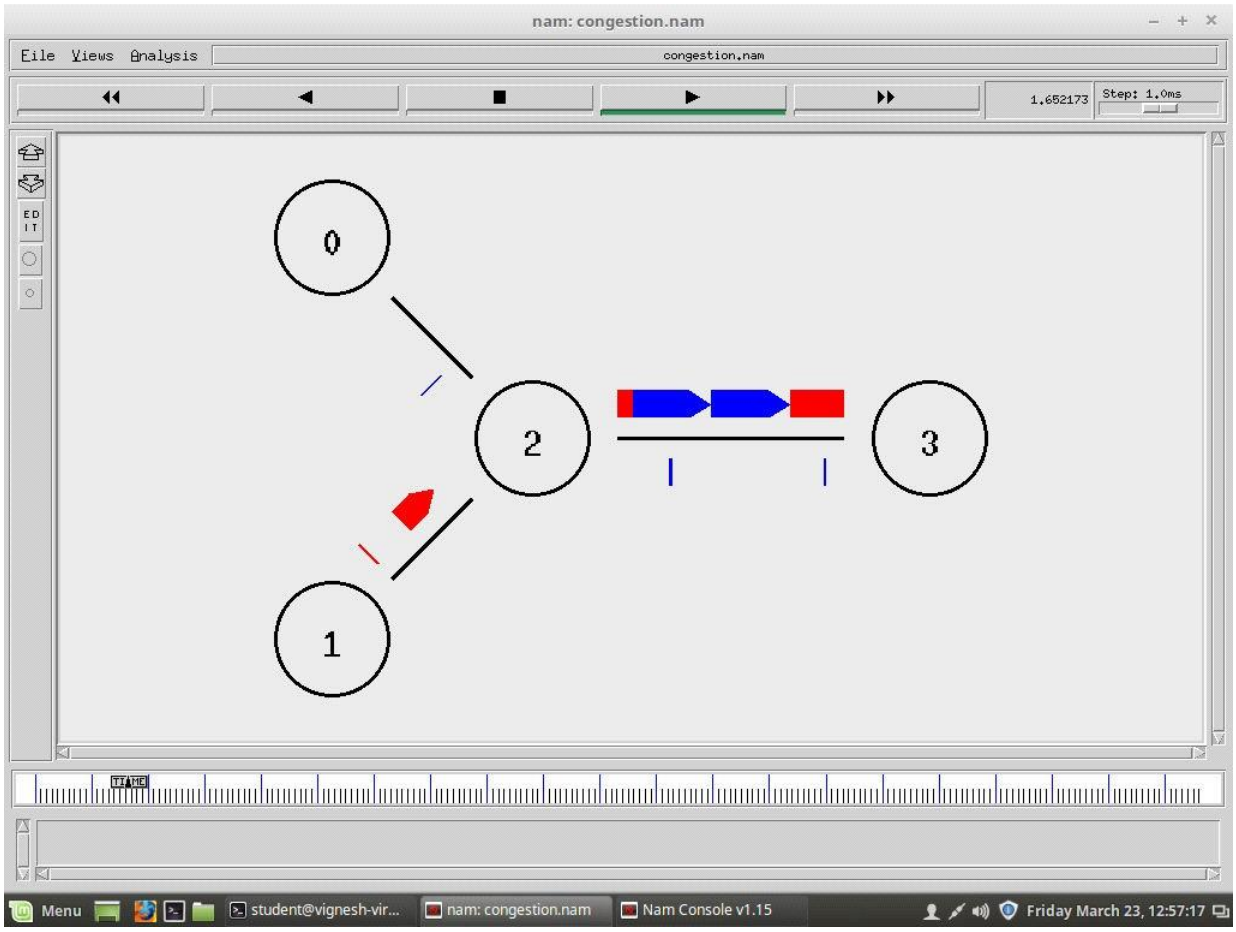
CODING

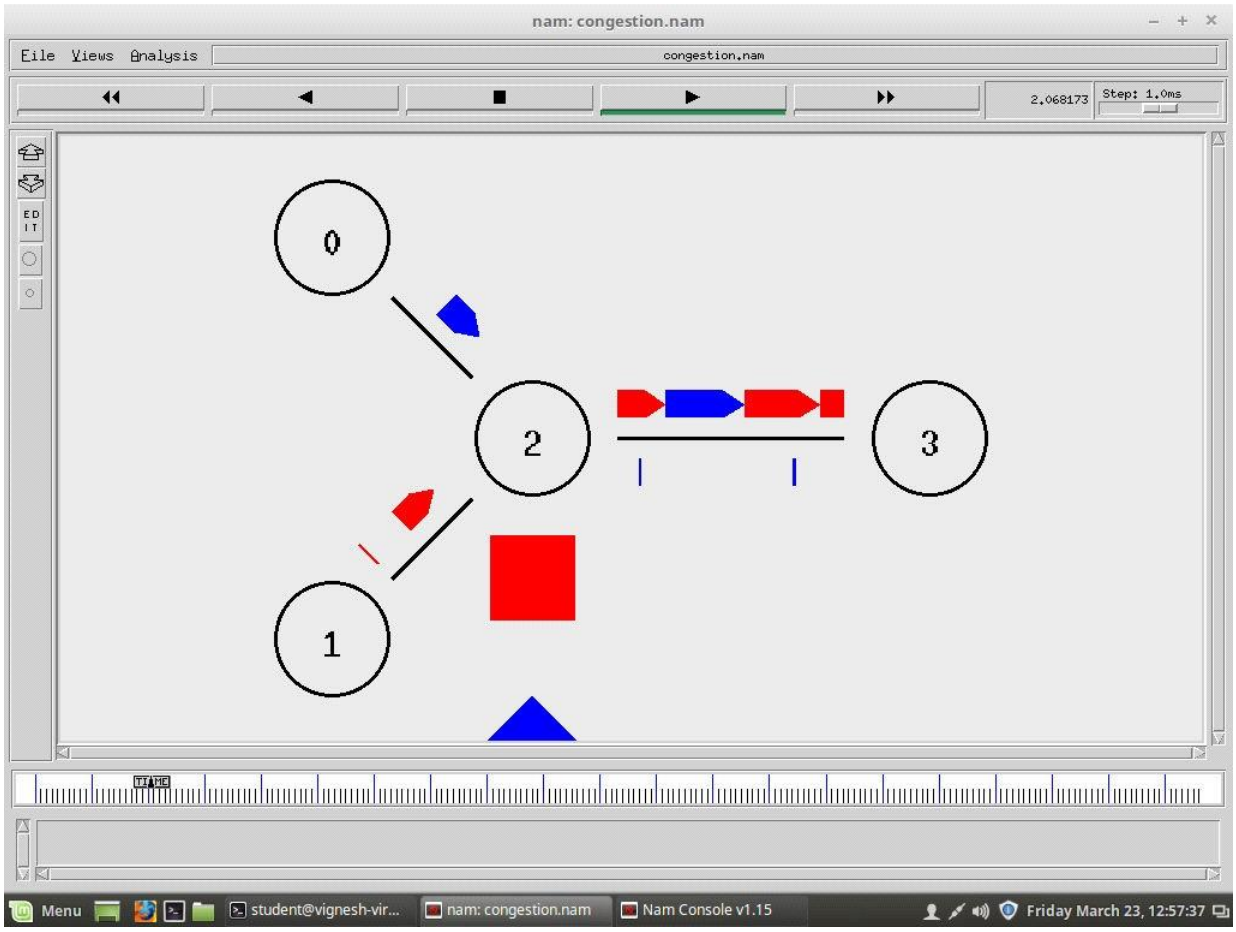
```
# Create an object of Simulator Class set ns [new
Simulator]

#Create NAM and trace file  set nf [open
congestion.nam w]
$ns namtrace-all $nf

set tf [open congestion.tr w]
$ns trace-all $tf
$ns color 1 Blue
$ns color 2 Red
#Create Nodes
set n0 [$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 1.2Mb 20ms 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

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

#Create transport agents set tcp
[new Agent/TCP]
$ns attach-agent $n0 $tcp
$tcp set class_ 1
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink set udp
[new Agent/UDP]
$udp set class_ 2
$ns attach-agent $n1 $udp set null
[new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
#Create Application Traffic
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1000
$cbr set rate_ 1Mb
$cbr attach-agent $udp

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

#Default setting in ns2 - ssthresd_ 20
```

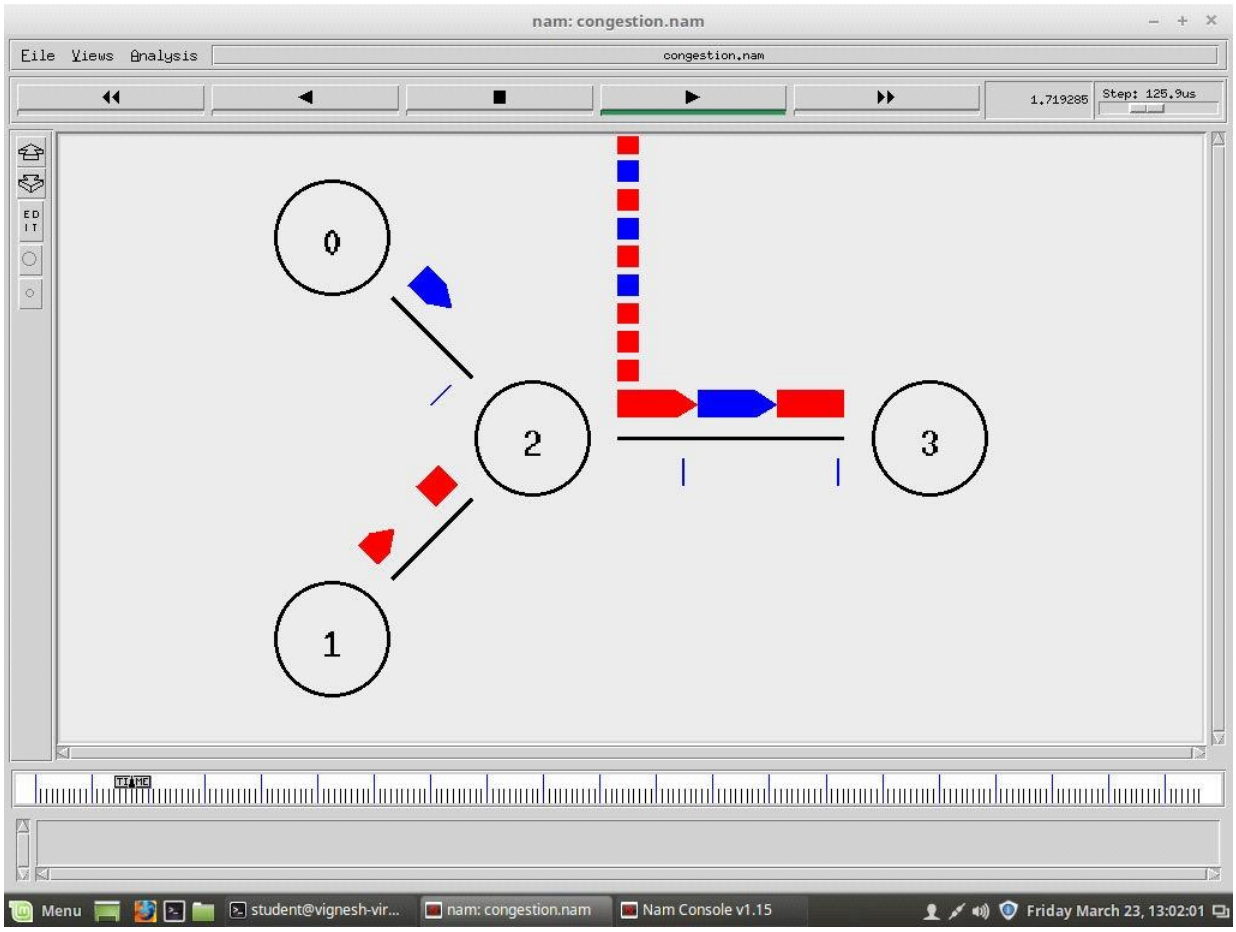
```
set outfile [open congestion.txt w]

proc plotWindow {tcpSource outfile} { global ns
set now [$ns now]
set cwnd [$tcpSource set cwnd_] puts
$outfile "$now $cwnd"
$ns at [expr $now+0.1] "plotWindow $tcpSource $outfile"
}

$ns at 1.0 "plotWindow $tcp $outfile"
# Schedule traffic
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 20.0 "$ftp stop"
$ns at 20.5 "$cbr stop"
$ns at 20.6 "finish"

proc finish {} { global ns
nf tf
$ns flush-trace close
$nf
close $tf
exec nam congestion.nam & exit 0
}
#Finish and run
$ns run
```

Sample OUTPUT




nam: congestion.nam
File Views Analysis
congestion.nam
1.652173
Step: 1.0ms
0
1
2
3
Menu
student@vignesh-vir...
nam: congestion.nam
Nam Console v1.15
Friday March 23, 12:57:17


nam: congestion.nam
File Views Analysis
congestion.nam
2.068173
Step: 1.0ms
0
1
2
3
Menu
student@vignesh-vir...
nam: congestion.nam
Nam Console v1.15
Friday March 23, 12:57:37


nam: congestion.nam
File Views Analysis
congestion.nam
1.719285
Step: 125.9us
0
1
2
3
Menu
student@vignesh-vir...
nam: congestion.nam
Nam Console v1.15
Friday March 23, 13:02:01

RECORD THE INFERENCES & RESULTS

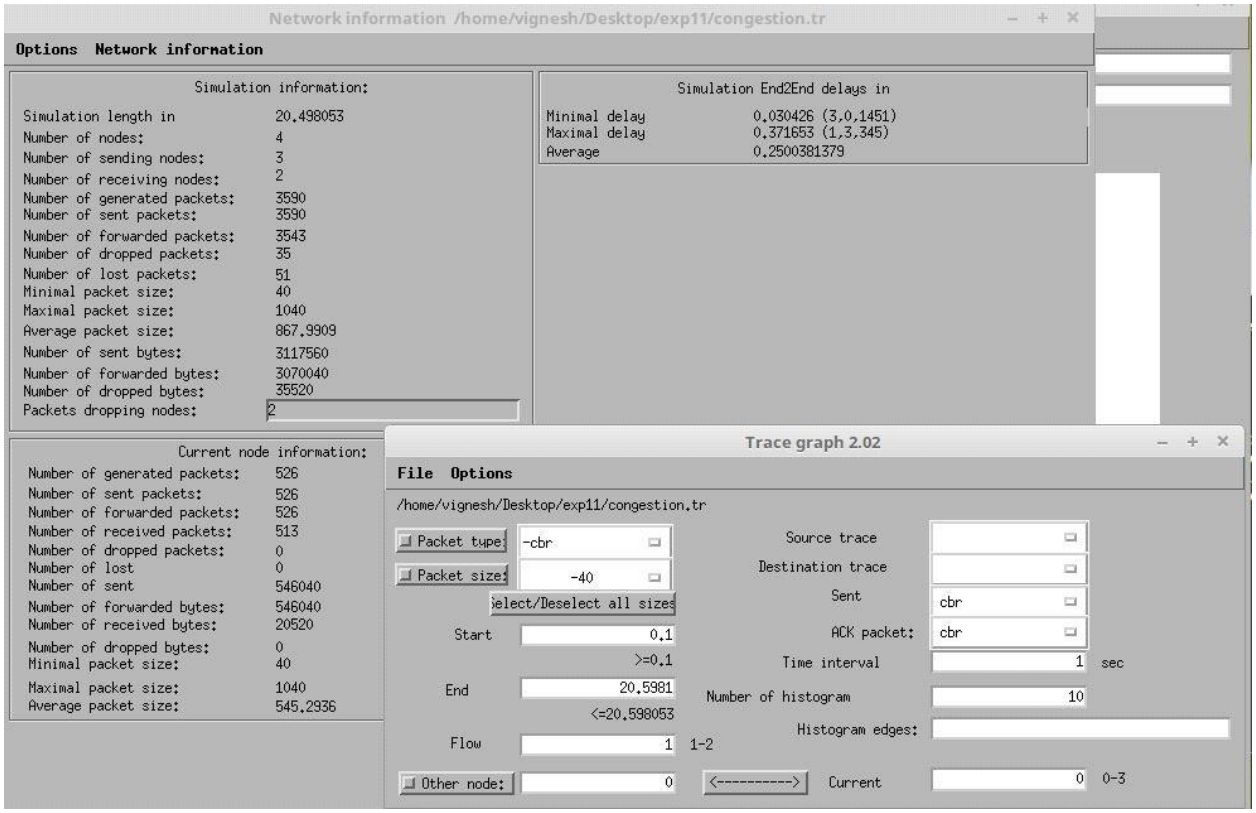
- TCP uses a congestion window and a congestion policy that avoid congestion.

- If the network cannot deliver the data as fast as it is created by the sender, it must tell the sender to slow down, i.e. in addition to the receiver, the network is a second entity that determines the size of the sender's window.

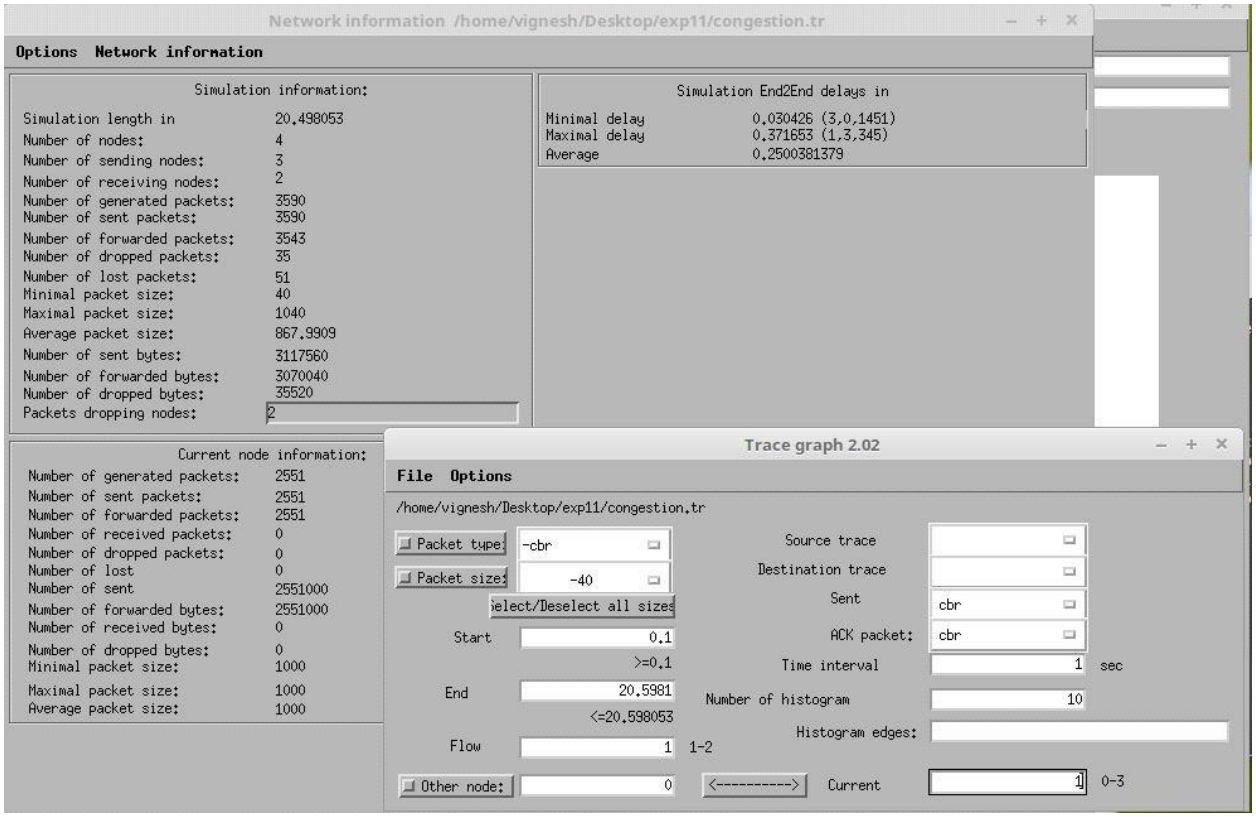
- Congestion policy in TCP –

- **Slow Start Phase:** starts slowly increment is exponential to threshold
- **Congestion Avoidance Phase:** After reaching the threshold increment is by one. That is there is linear increment.
- **Congestion Detection Phase:** Sender goes back to Slow start phase or Congestion avoidance phase.

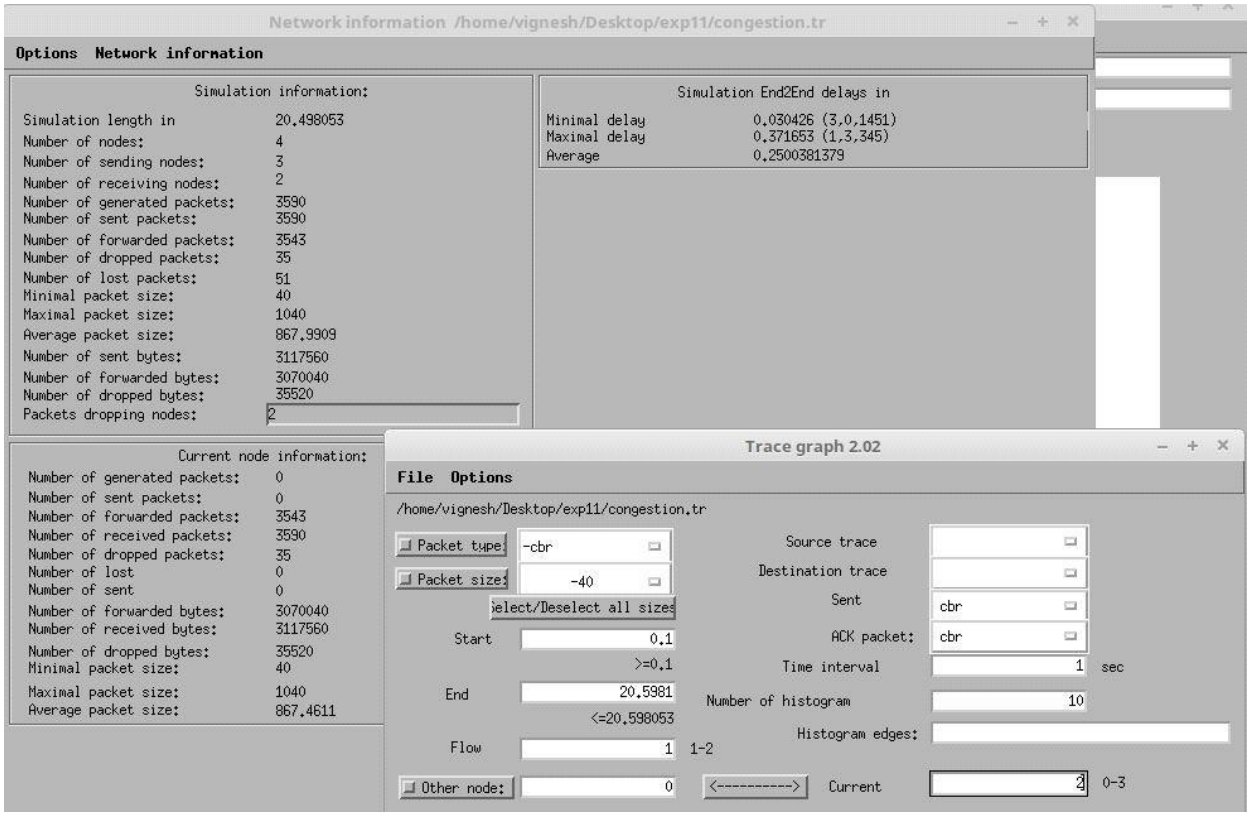
Node analysis and PDR calculations using tracegraph and grep commands:



PDR (0-3) TCP packets = 513/526 = 97.52%



PDR (1-3) UDP packets = 2551/2551 = 100%



Node 2 to Node 3:

The number of received packets are 3590 which include reception from node 0 and node 1 and also receiving ACK from node 3

The number of forwarded packets is 3543 which include forwarding the received packets from node 0 and node 1 and forwarding the ACK to node 0.

Results: The above experiment was successfully executed, and output verified.