

Name: Mareena Fernandes

Roll no.: 8669

Class: SE IT

Batch: B

Exp 1

Computer Network SE (IT)

MAREENA FERNANDES
8669 SEIT(B)

Academic Term: Jan-May 2020


FR. CONCEICAO RODRIGUES COLLEGE OF ENGG. , BANDRA

S.E (Information Technology)

Subject: Networking Lab

Experiment No:	1
Title:	To study different hardware components of communication network
Date of performance:	20 th JANUARY 2020

Sr.No.	Rubrics	Grade
1	Performance (4)	04
2	Understanding(2)	0
3	Post Lab/Quiz (2)	02
4	On time submission (2)	02

Signature of the Teacher: 

Date: 27/1

FR. CONCEICAO RODRIGUES COLLEGE OF ENGG. , BANDRA

Exp 2

#ifconfig

ifconfig

```
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 8.8.5.19 netmask 255.255.255.224 broadcast 8.8.5.31
    inet6 fe80::42a8:f0ff:fe43:96e2 prefixlen 64 scopeid 0x20<link>
    ether 40:a8:f0:43:96:e2 txqueuelen 1000 (Ethernet)
    RX packets 15211 bytes 10945138 (10.9 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 11243 bytes 1306285 (1.3 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

```
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 1047 bytes 75687 (75.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1047 bytes 75687 (75.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

#ping

ping 8.8.5.1

PING 8.8.5.1 (8.8.5.1) 56(84) bytes of data.

64 bytes from 8.8.5.1: icmp_seq=1 ttl=64 time=0.135 ms

64 bytes from 8.8.5.1: icmp_seq=2 ttl=64 time=0.164 ms

#Traceroute

traceroute 8.8.5.1

traceroute to 8.8.5.1 (8.8.5.1), 30 hops max, 60 byte packets

1 _gateway (8.8.5.1) 0.438 ms 0.451 ms 0.434 ms

#route

route

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
default	_gateway	0.0.0.0	UG	100	0	0	enp3s0
8.8.5.0	0.0.0.0	255.255.255.224	U	100	0	0	enp3s0

#arp

arp -a

_gateway (8.8.5.1) at 70:62:b8:94:72:bd [ether] on enp3s0

#host

host 8.8.5.1

1.5.8.8.in-addr.arpa domain name pointer _gateway.

```
#hostname  
hostname  
hp19
```

```
#iwconfig  
iwconfig  
lo      no wireless extensions.  
enp3s0  no wireless extensions.
```

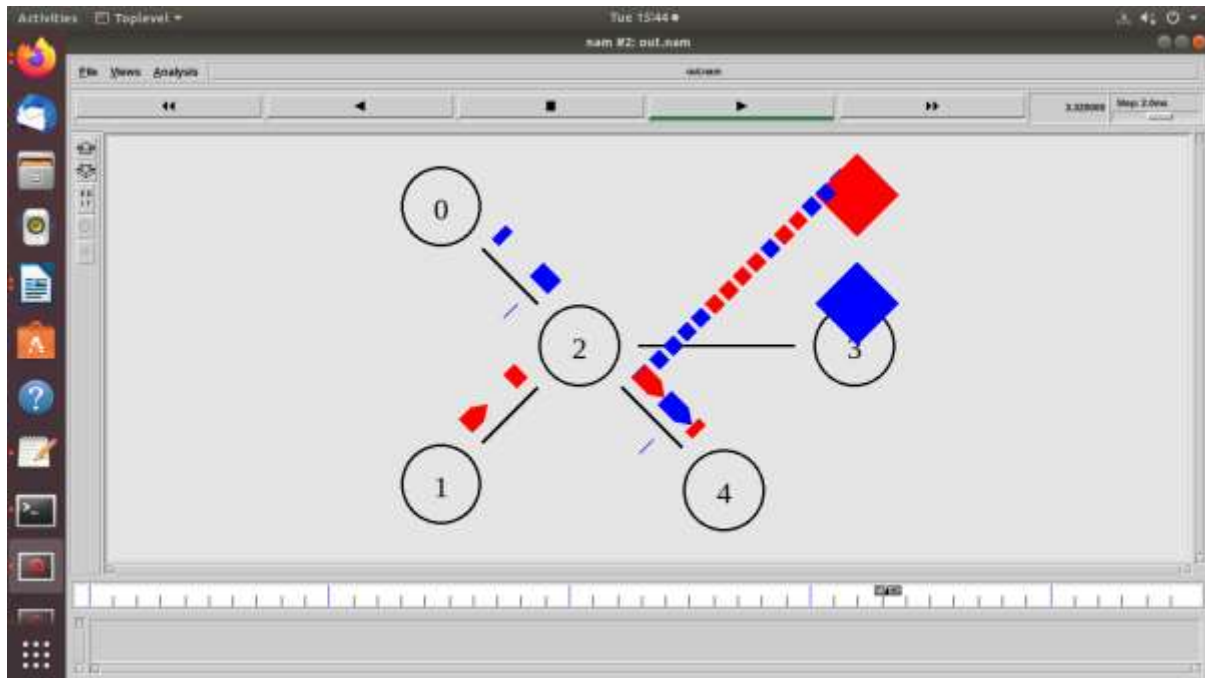
Exp 3

```
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
    global ns nf
    $ns flush-trace
    #Close the NAM trace file
    close $nf
    #Execute NAM on the trace file
    exec nam out.nam &
    exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$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
$ns duplex-link $n2 $n4 1.7Mb 11ms DropTail
$ns queue-limit $n2 $n3 10
$ns queue-limit $n2 $n4 10
$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 $n4 orient right-down
$ns duplex-link-op $n2 $n3 queuePos 0.5
$ns duplex-link-op $n2 $n4 queuePos 0.5
set tcp [new Agent/TCP]
set tcp1 [new Agent/TCP]
$tcp set class_2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_1
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp set type_ FTP
$ftp1 set type_ FTP
```

```

set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns attach-agent $n4 $null
$ns connect $udp $null
$udp set fid_ 2
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
$ns at 5.0 "$cbr stop"
$ns at 5.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
$ns at 5.5 "finish"
#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"
$ns run

```



Postlabs:

Q. Explain Trace File Format.

A: The file written by an application (or by Coverage Server) to store coverage information or overall network information and in NS2, it is called as Trace File. It is a file that contains events logs during a simulation process. For instance, when the network simulation is run, some events happens such as packet drops and packet reception by different nodes in the network. These events are then logged in the text file and this is normally called a trace file. Events in Queues can be recorded in a trace file. Statistical information (such as throughput on a link/queue) can be computed using the trace data. In order to generate a trace file, we have to create a file in Otcl script.

Exp 4

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

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

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

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

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

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n4 1.7Mb 20ms DropTail
$ns duplex-link $n4 $n3 1.7Mb 30ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n4 20

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n4 orient right
```

```
$ns duplex-link-op $n4 $n3 orient right
```

```
#Monitor the queue for link (n2-n3). (for NAM)
```

```
$ns duplex-link-op $n2 $n4 queuePos 0.5
```

```
#Setup a TCP connection
```

```
set tcp [new Agent/TCP]
```

```
$tcp set class_ 2
```

```
$ns attach-agent $n0 $tcp
```

```
set sink [new Agent/TCPSink]
```

```
$ns attach-agent $n3 $sink
```

```
$ns connect $tcp $sink
```

```
$tcp set fid_ 1
```

```
#Setup a FTP over TCP connection
```

```
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
```

```
$ftp set type_ FTP
```

```
#Setup a UDP connection
```

```
set udp [new Agent/UDP]
```

```
$ns attach-agent $n1 $udp
```

```
set null [new Agent/Null]
```

```
$ns attach-agent $n3 $null
```

```
$ns connect $udp $null
```

```
$udp set fid_ 2
```

```
#Setup a CBR over UDP connection
```

```
set cbr [new Application/Traffic/CBR]
```

```
$cbr attach-agent $udp
```

```
$cbr set type_ CBR
```

```
$cbr set packet_size_ 1000
```

```
$cbr set rate_ 1mb
```

```
$cbr set random_ false
```

```
#Schedule events for the CBR and FTP agents
```

```
$ns at 0.1 "$cbr start"
```

```
$ns at 1.0 "$ftp start"
```

```
$ns at 4.0 "$ftp stop"
```

```
$ns at 4.5 "$cbr stop"
```



```
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
```

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

\$ns at 5.0 "finish"

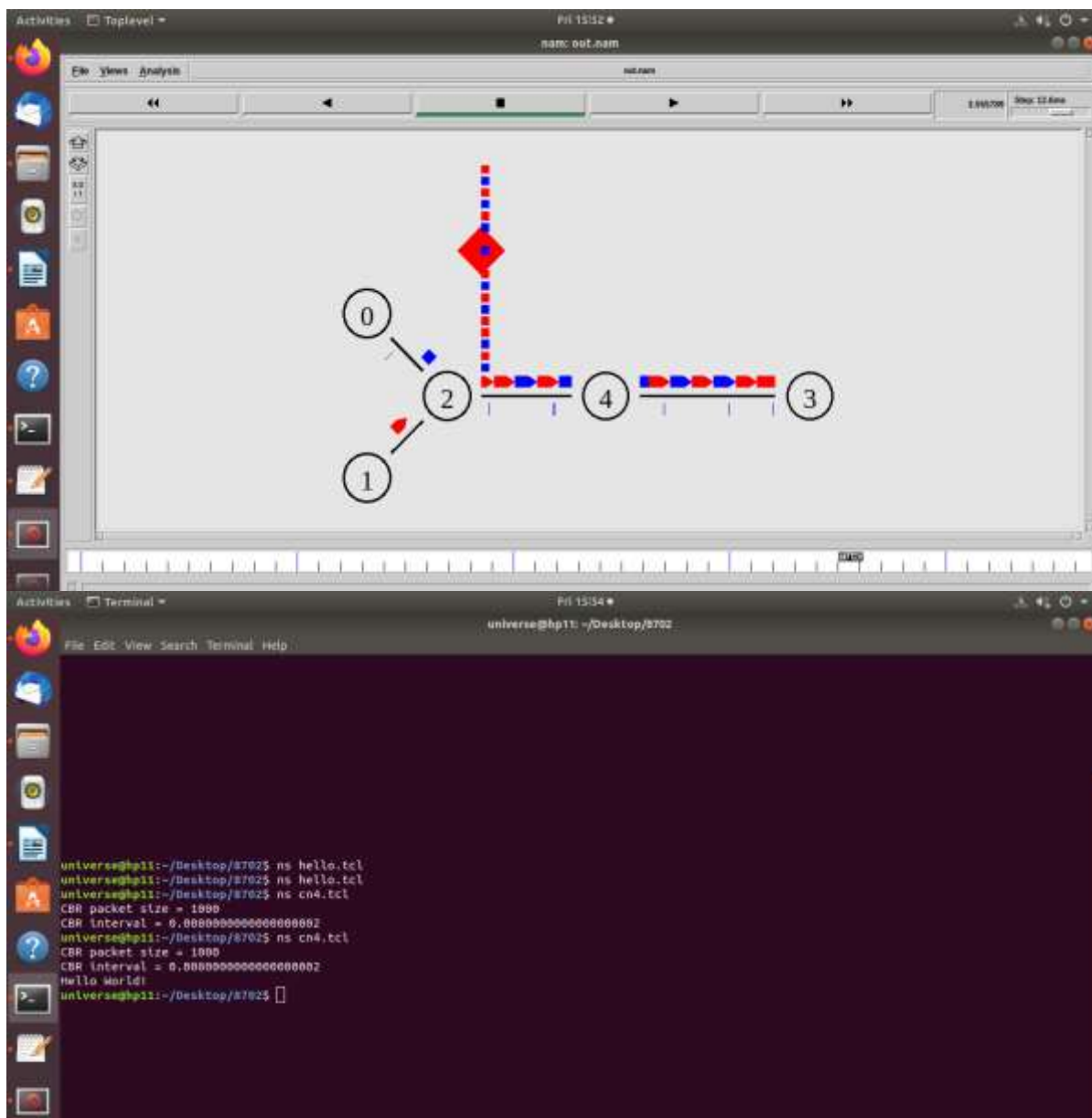
```
#Print CBR packet size and interval
```

```
puts "CBR packet size = [$cbr set packet_size_]"
```

```
puts "CBR interval = [$cbr set interval_]"
```

```
#Run the simulation
```

\$ns run



Exp 5

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
```

```
set tr [open out.tr w]
$ns trace-all $tr
```

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

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

```
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail
```

```
$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up
```

```
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
```

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

```
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
```

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

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
```

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

```
$ns connect $tcp $sink  
$ns connect $udp $null
```

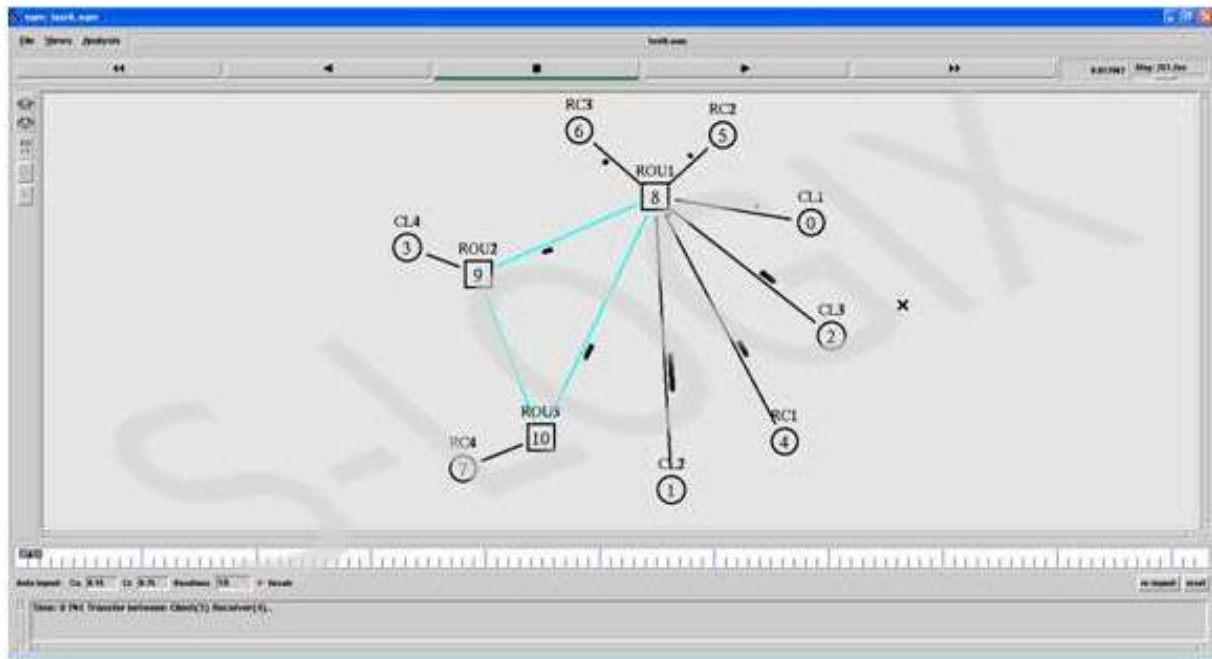
```
$ns rtmodel-at 1.0 down $n1 $n3  
$ns rtmodel-at 2.0 up $n1 $n3
```

```
$ns rtproto LS
```

```
$ns at 0.0 "$ftp start"  
$ns at 0.0 "$cbr start"
```

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

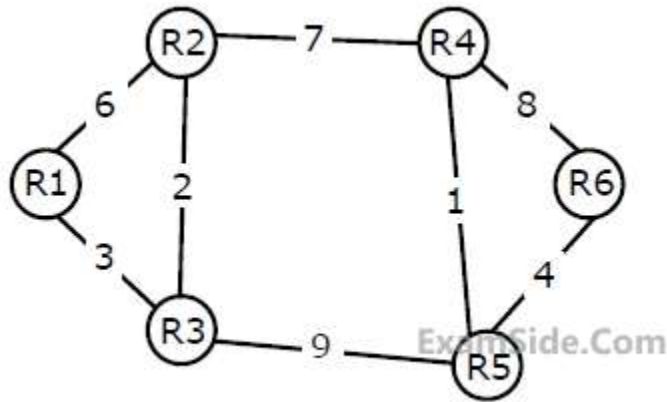
```
$ns run
```



Postlabs:

GATE -CS-2010

Consider a network with 6 routers R1 to R6 connected with links having weights as shown in the following diagram



All the routers use the distance vector based routing algorithm to update their routing tables. Each router starts with its routing table initialized to contain an entry for each neighbour with the weight of the respective connecting link. After all the routing tables stabilize, how many links in the network will never be used for carrying any data?

Solution:

We can check one by one all shortest distances. When we check for all shortest distances for R_i we don't need to check its distances to R_0 to R_{i-1} because the network graph is undirected.

Following will be distance vectors of all nodes.

Shortest Distances from R1 to R2, R3, R4, R5 and R6

R1 (5, 3, 12, 12, 16)

Links used: R1-R3, R3-R2, R2-R4, R3-R5, R5-R6

Shortest Distances from R2 to R3, R4, R5 and R6

R2 (2, 7, 8, 12)

Links used: R2-R3, R2-R4, R4-R5, R5-R6

Shortest Distances from R3 to R4, R5 and R6

R3 (9, 9, 13)

Links used: R3-R2, R2-R4, R3-R5, R5-R6

Shortest Distances from R4 to R5 and R6

R4 (1, 5)

Links used: R4-R5, R5-R6

Shortest Distance from R5 to R6

R5 (4)

Links Used: R5-R6

If we mark, all the used links one by one, we can see that following links are never used.

R1-R2

R4-R6

Exp 7

```
set opt(chan)    Channel/WirelessChannel
set opt(prop)    Propagation/TwoRayGround
set opt(netif)   Phy/WirelessPhy
set opt(mac)     Mac/802_11
set opt(ifq)     Queue/DropTail/PriQueue
set opt(ll)      LL ;                               #Link Layer
set opt(ant)     Antenna/OmniAntenna
set opt(x)       500 ;                             # X dimension of the topography
set opt(y)       500 ;                             # Y dimension of the topography
set opt(ifqlen)  50 ;                               # max packet in ifq
set opt(nn)      100
set opt(connections) 50

set opt(stop)    50
set opt(dataRate) [expr 1.0*256*8] ;               #packet size=256 bytes
set opt(adhocRouting) AODV

set ns_          [new Simulator]
set topo         [new Topography]
set opt(fn) "wireless"
set tracefd      [open $opt(fn).tr w]
set namtrace     [open $opt(fn).nam w]

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# declare finish program
proc finish {} {
    global ns_ tracefd namtrace
    $ns_ flush-trace
    close $tracefd
    close $namtrace
    #exec nam $namtrace
    exit 0
}

# define topology
$topo load_flatgrid $opt(x) $opt(y)

# Create God(Generate Operations Director): stores table of shortest no of hops from 1 node to another
set god_ [create-god $opt(nn)]

# define how node should be created
#global node setting
$ns_ node-config -adhocRouting $opt(adhocRouting) \
    -llType $opt(ll) \
```

```

-macType $opt(mac) \
-ifqType $opt(ifq) \
-ifqLen $opt(ifqlen) \
-antType $opt(ant) \
-propType $opt(prop) \
-phyType $opt(netif) \
-channelType $opt(chan) \
                    -topoInstance $topo \
                    -agentTrace ON \
-movementTrace ON \
-routerTrace ON \
-macTrace ON

```

Create the specified number of nodes [\$opt(nn)] and "attach" them to the channel.

```

for {set i 0} {$i < $opt(nn)} {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 1;
}

```

disable random motion

```

for {set i 0} {$i < $opt(nn)} {incr i} {
    $node_($i) set X_ [expr rand()*500]
    $node_($i) set Y_ [expr rand()*500]
    $node_($i) set Z_ 0
}

```

```

for {set i 0} {$i < $opt(nn)} {incr i} {
    $ns_ initial_node_pos $node_($i) 20
}

```

```

for {set i 0} {$i < $opt(connections)} {incr i} {

```

```

    #Setup a UDP connection
    set udp_($i) [new Agent/UDP]
    $ns_ attach-agent $node_($i) $udp_($i)
    set null_($i) [new Agent/Null]
    $ns_ attach-agent $node_([expr $i+2]) $null_($i)
    $ns_ connect $udp_($i) $null_($i)

```

```

    #Setup a CBR over UDP connection
    set cbr_($i) [new Application/Traffic/CBR]
    $cbr_($i) attach-agent $udp_($i)
    $cbr_($i) set type_ CBR
    $cbr_($i) set packet_size_ 256
    $cbr_($i) set rate_ $opt(dataRate)
    $cbr_($i) set random_ false
    $ns_ at 0.0 "$cbr_($i) start"
    $ns_ at $opt(stop) "$cbr_($i) stop"
}

```

```

# random motion
for {set j 0} {$j < 10} {incr j} {
    for {set i 0} {$i < $opt(nn)} {incr i} {
        set xx_ [expr rand()*$opt(x)]
        set yy_ [expr rand()*$opt(y)]
        set rng_time [expr rand()*$opt(stop)]
        $ns_ at $rng_time "$node_($i) setdest $xx_ $yy_ 15.0" ;
    }
}

# Tell nodes when the simulation ends
for {set i 0} {$i < $opt(nn)} {incr i} {
    $ns_ at $opt(stop) "$node_($i) reset";
}

$ns_ at $opt(stop) "finish"
$ns_ run

```

1. Running exp7.tcl file:

Mareena@DESKTOP-I8DD1RQ:~/CodeWork\$ ns exp7.tcl

When configured, ns found the right version of tclsh in /usr/bin/tclsh8.6

but it doesn't seem to be there anymore, so ns will fall back on running the first tclsh in your path. The wrong version of tclsh may break the test suites. Reconfigure and rebuild ns if this is a problem.

num_nodes is set 100

warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl

INITIALIZE THE LIST xListHead

channel.cc:sendUp - Calc highestAntennaZ_ and distCST_

highestAntennaZ_ = 1.5, distCST_ = 550.0

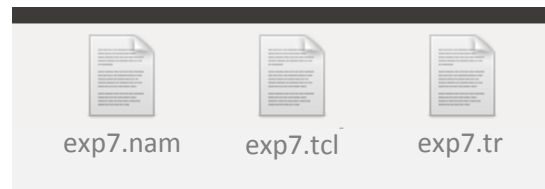
SORTING LISTS ...DONE!

2. Outputs after running above tcl file:

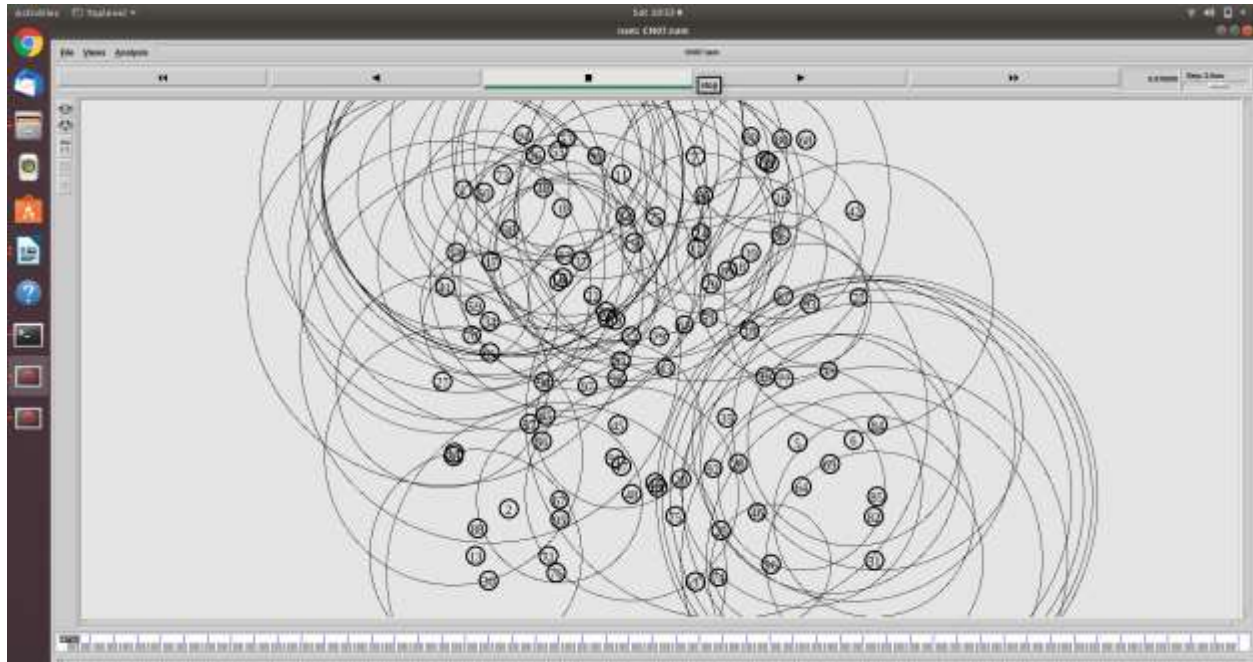
2 files are generated as output:

exp7.tf file i.e., Trace File

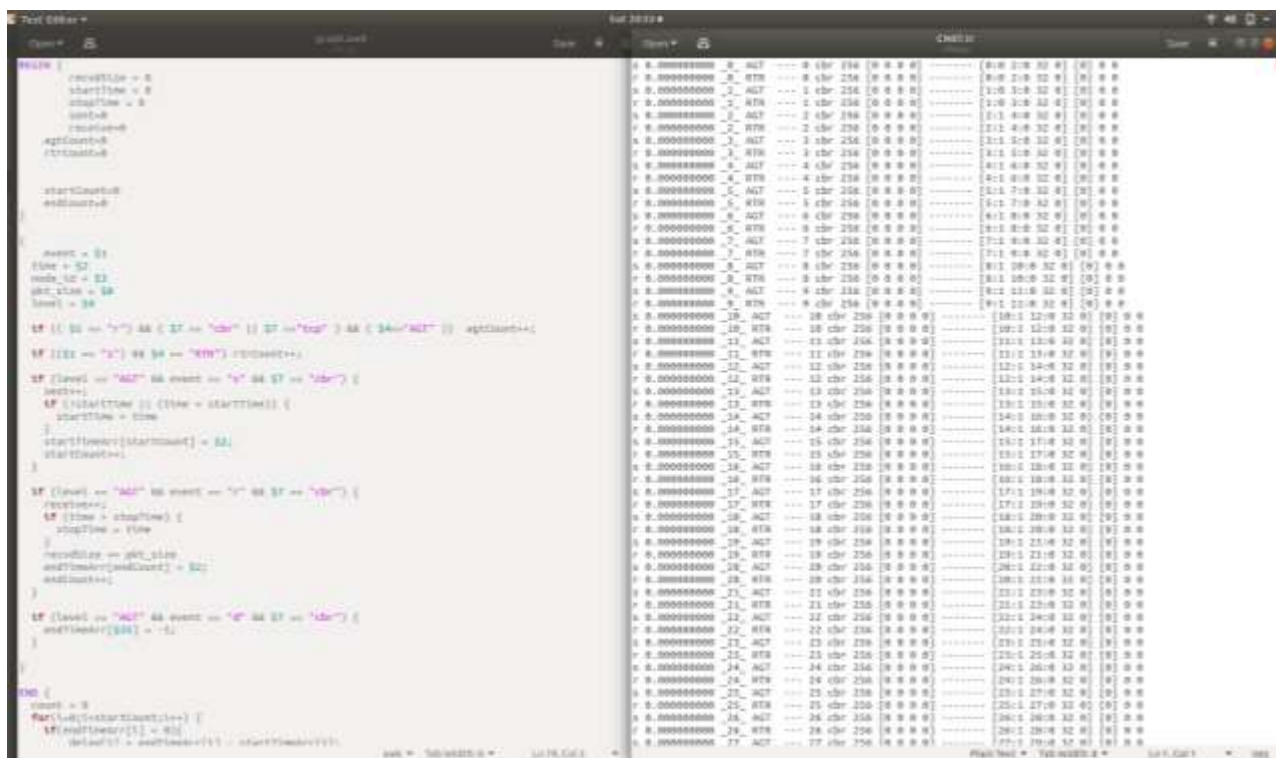
exp7.nam i.e., NS simulation file



3. NS simulation (100 nodes):



4.Trace File and awk file Content :



5.AWK command:

AWK Scripts are very good in processing the data from the log (trace files) which we get from NS2 also if you want to process the trace file manually, they are useful.

```
Mareena@DESKTOP-I8DD1RQ:~/CodeWork$ awk -f graph.awk exp7.tr
```

Sent 1900

Received 1594

Dropped 306

PDR 83.89

Average Throughput[kbps] = 97.59 StartTime=1.00 StopTime = 37.06

Normalized Load 0.102

Average End-to-End Delay = 5236.52 ms

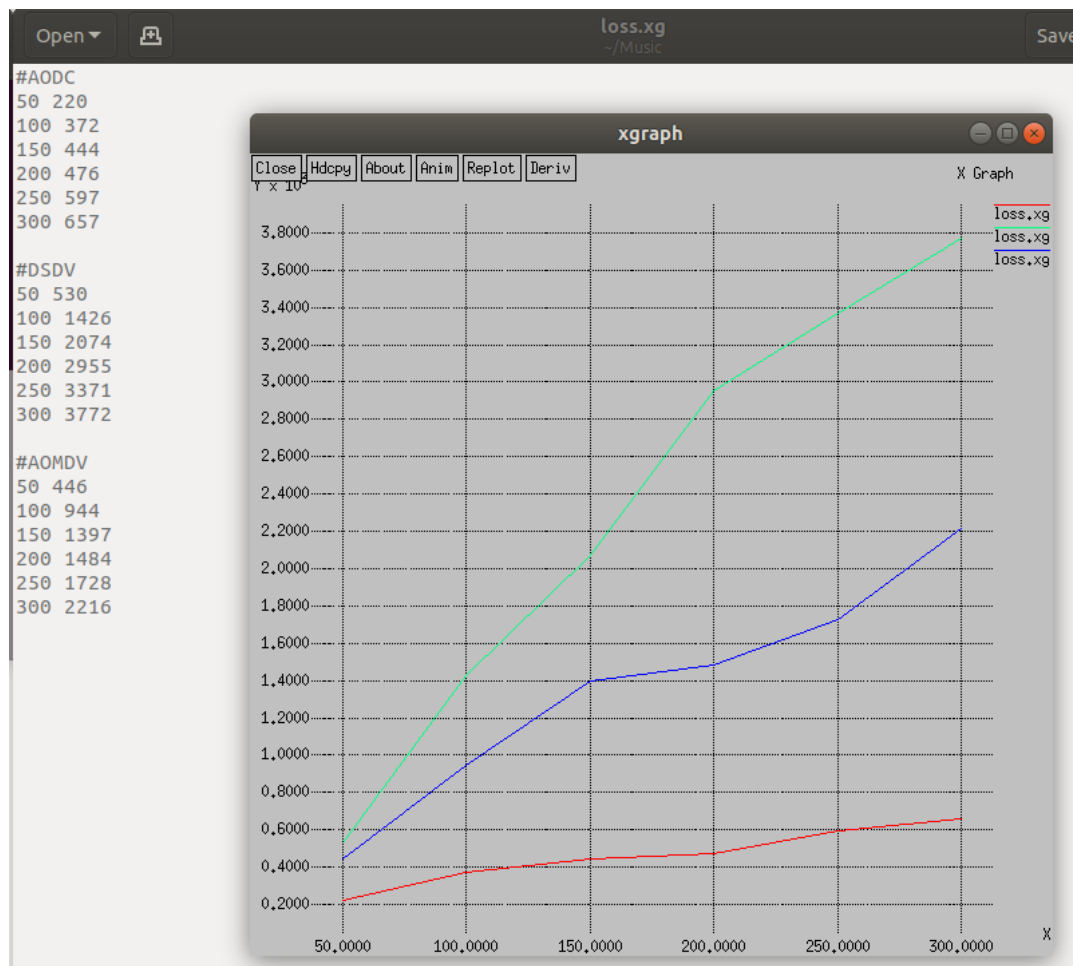
6.Packet Loss Graph:

Packet loss reflects the number of packets lost per 100 packets sent by a host

```
Mareena@DESKTOP-I8DD1RQ:~/CodeWork$ xgraph loss.xg
```

Parameter LabelFont: can't translate 'helvetica-10' into a font (defaulting to 'fixed')

Parameter TitleFont: can't translate 'helvetica-18' into a font (defaulting to 'fixed')



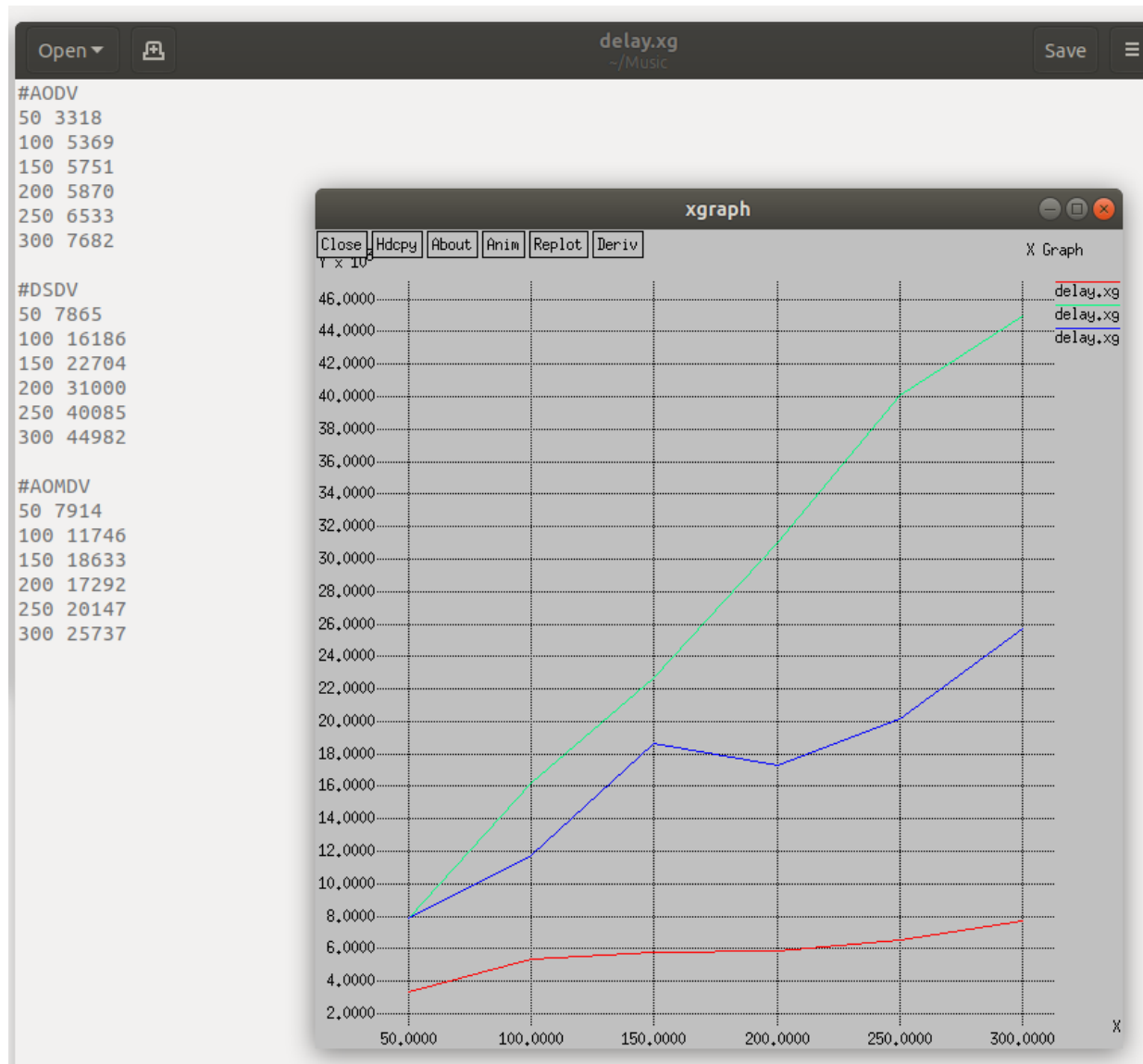
7.Delay Graph:

The **delay** of a **network** specifies how long it takes for a bit of data to travel across the **network** from one communication endpoint to another

Mareena@DESKTOP-I8DD1RQ:~/CodeWork\$ xgraph delay.xg

Parameter LabelFont: can't translate `helvetica-10' into a font (defaulting to `fixed')

Parameter TitleFont: can't translate `helvetica-18' into a font (defaulting to `fixed')



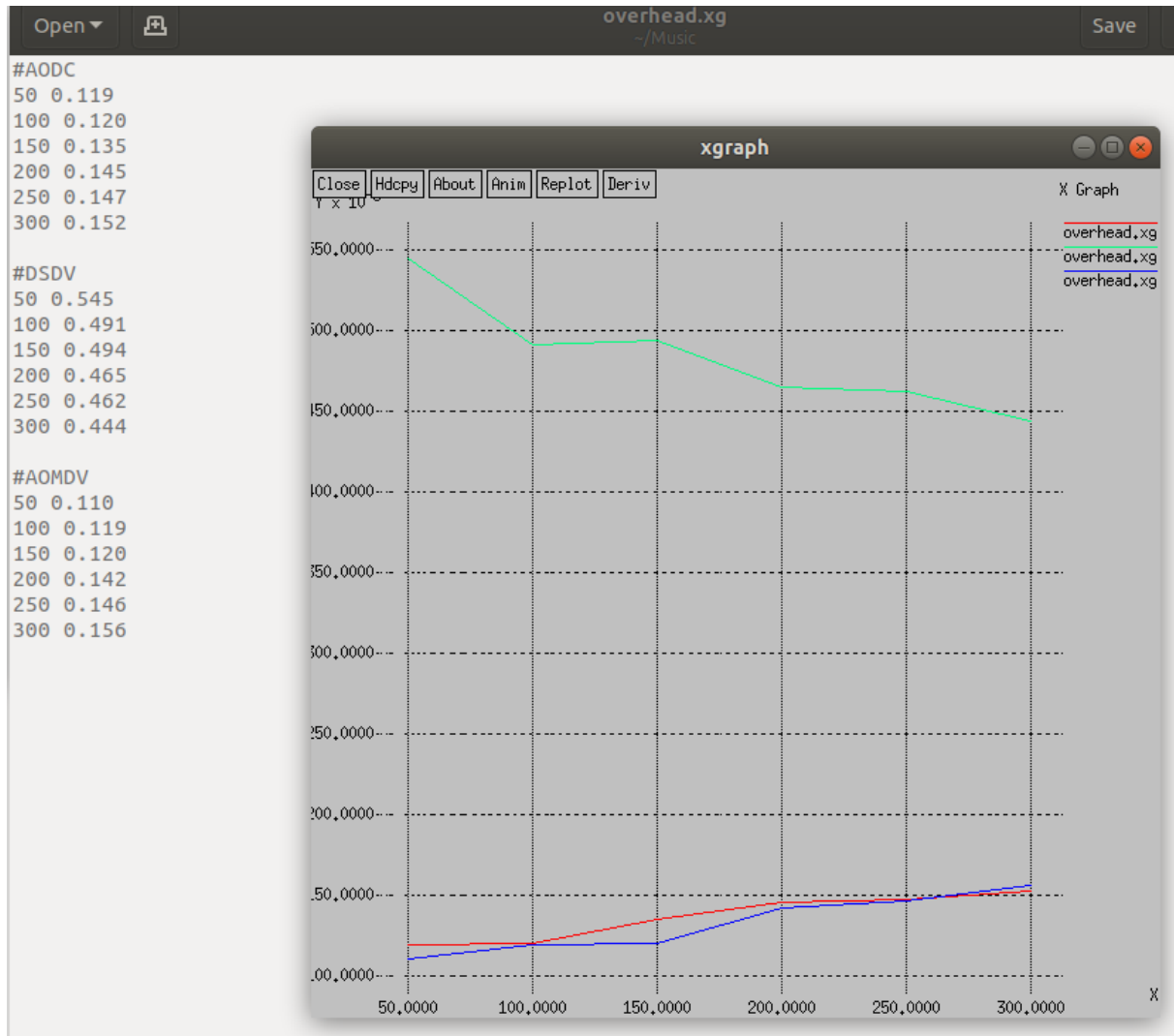
8.Overhead Graph:

The time it takes to transmit data on a **packet**-switched network

Mareena@DESKTOP-I8DD1RQ:~/CodeWork\$ xgraph overhead.xg

Parameter LabelFont: can't translate `helvetica-10' into a font (defaulting to `fixed')

Parameter TitleFont: can't translate `helvetica-18' into a font (defaulting to `fixed')



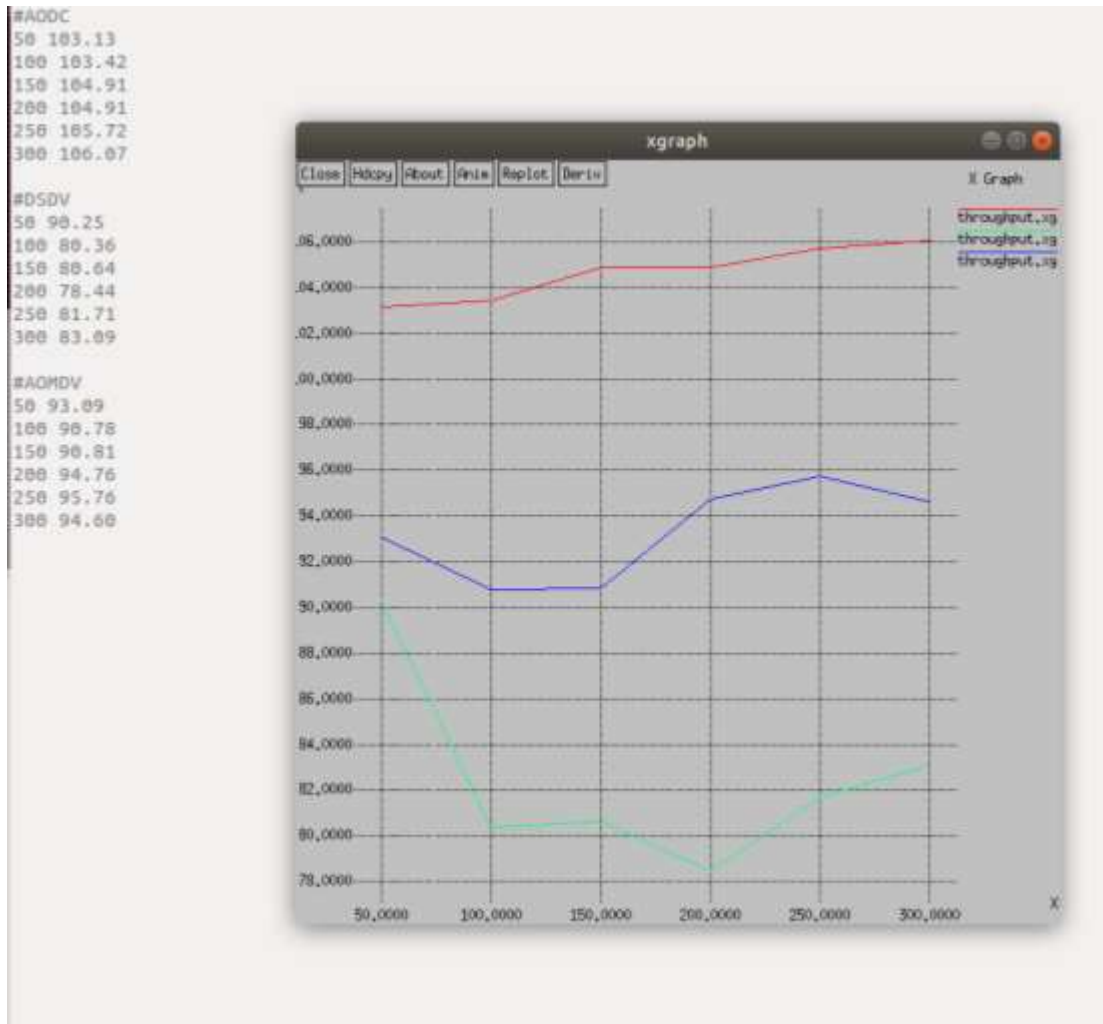
9.Throughput Graph:

Throughput is defined as the quantity of data being sent/received by unit of time

Mareena@DESKTOP-I8DD1RQ:~/CodeWork\$ xgraph throughput.xg

Parameter LabelFont: can't translate `helvetica-10' into a font (defaulting to `fixed')

Parameter TitleFont: can't translate `helvetica-18' into a font (defaulting to `fixed')



(All the data passed to draw graphs is sorted out with the help of awk script from Trace File which was generated after running ns filename command in step 2)

Conclusion:

We learnt to analyze Different Routing protocols on **xgraph** as above on the basis of loss packets, Throughput, Delay and Overhead.

Postlabs:

Q. Compare different types of routing protocols

A:

1. **Routing Information Protocol (RIP):**
Routing Information Protocol or RIP is one of the first routing protocols to be created. RIP is used in both Local Area Networks (LANs) and Wide Area Networks (WANs), and also runs on the Application layer of the OSI model. There are multiple versions of RIP including RIPv1 and RIPv2. The original version or RIPv1 determines network paths based on the IP destination and the hop count of the journey.
2. **Interior Gateway Protocol (IGRP):**
Interior Gateway Protocol or IGRP is a distance vector protocol produced by Cisco. IGRP was designed to build on the foundations laid down on RIP to function more effectively within larger networks and removed the 15-hop cap that was placed on RIP. IGRP uses metrics such as bandwidth, delay, reliability, and load to compare the viability of routes within the network. However, only bandwidth and delay are used under IGRP's default settings.
3. **Open Shortest Path First (OSPF):**
Open Shortest Path First or OSPF protocol is a link-state IGP that was tailor-made for IP networks using the Shortest Path First (SPF) algorithm. The SPF algorithm is used to calculate the shortest path spanning-tree to ensure efficient transmission of packets. OSPF routers maintain databases detailing information about the surrounding topology of the network. This database is filled with data taken from Link State Advertisements (LSAs) sent by other routers. LSAs are packets that detail information about how many resources a given path would take.
4. **Exterior Gateway Protocol (EGP):**
Exterior Gateway Protocol or EGP is a protocol that is used to exchange data between gateway hosts that neighbor each other within autonomous systems. In other words, EGP provides a forum for routers to share information across different domains. The most high-profile example of an EGP is the internet itself. The routing table of the EGP protocol includes known routers, route costs, and addresses of neighboring devices. EGP was widely-used by larger organizations but has since been replaced by BGP.
5. **Border Gateway Protocol (BGP):**
Border Gateway Protocol or BGP is the routing protocol of the internet that is classified as a distance path vector protocol. BGP was designed to replace EGP with a decentralized approach to routing. The BGP Best Path Selection Algorithm is used to select the best routes for packet transfers. If you don't have any custom settings then BGP will select routes with the shortest path to the destination.

Exp 8

```
[mycomputer@compute-0-1 ~]# tcpdump -tttt -r ex.pcap
reading from file enp0s3-26082018.pcap, link-type EN10MB (Ethernet)
2018-08-25 22:03:17.249648 IP compute-0-1.example.com.ssh > 169.144.0.1.39406: Flags [P.], seq
1426167803:1426167927, ack 3061962134, win 291, options
[nop,nop,TS val 81358717 ecr 20378789], length 124
2018-08-25 22:03:17.249840 IP 169.144.0.1.39406 > compute-0-1.example.com.ssh: Flags [.], ack 124,
win 564, options [nop,nop,TS val 20378791 ecr 81358
717], length 0
2018-08-25 22:03:17.454559 IP controller0.example.com.amqp > compute-0-1.example.com.57836:
Flags [.], ack 1079416895, win 1432, options [nop,nop,TS v
al 81352560 ecr 81353913], length 0
2018-08-25 22:03:17.454642 IP compute-0-1.example.com.57836 > controller0.example.com.amqp:
Flags [.], ack 1, win 237, options [nop,nop,TS val 8135892
2 ecr 81317504], length 0
2018-08-25 22:03:17.646945 IP compute-0-1.example.com.57788 > controller0.example.com.amqp:
Flags [.], seq 106760587:106762035, ack 688390730, win 237
, options [nop,nop,TS val 81359114 ecr 81350901], length 1448
2018-08-25 22:03:17.647043 IP compute-0-1.example.com.57788 > controller0.example.com.amqp:
Flags [P.], seq 1448:1956, ack 1, win 237, options [nop,no
p,TS val 81359114 ecr 81350901], length 508
2018-08-25 22:03:17.647502 IP controller0.example.com.amqp > compute-0-1.example.com.57788:
Flags [.], ack 1956, win 1432, options [nop,nop,TS val 813
52753 ecr 81359114], length 0
```

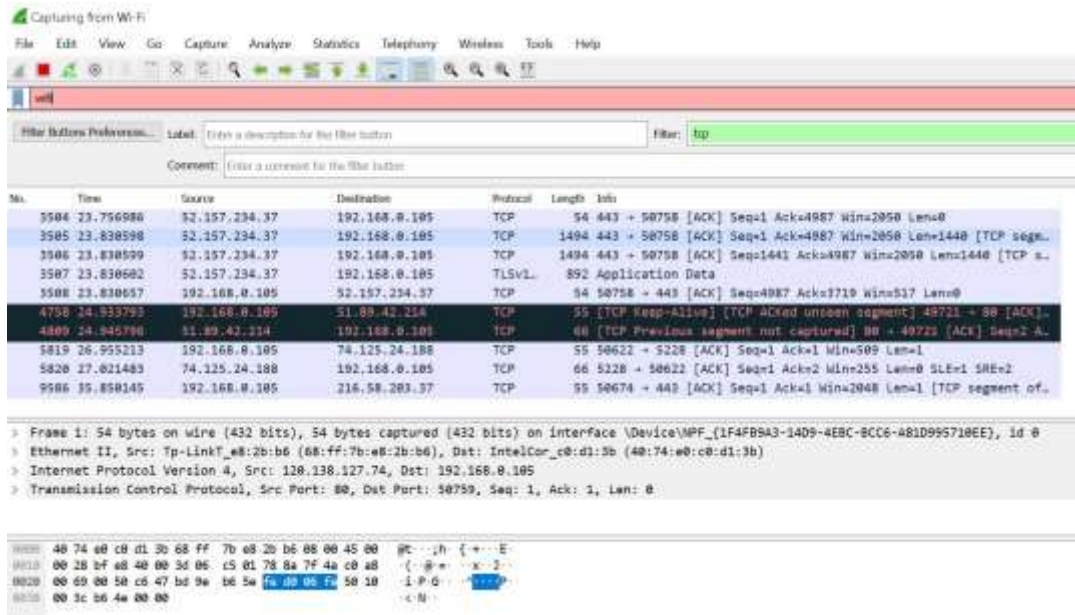
Approach:

1. Capture packets from interface with GUI or TCPDUMP command as follows
tcpdump -i eth0 -w ex.pcap
(Permission denied on machine for above command)
2. Analyze o/p directly on comsol or on GUI of wireshark.

I have chosen to capture packets on GUI of (Wireshark) as follows:

Wireshark packets analysis as follows:





Conclusion:

From above experiment we learn to capture packets using Wireshark tool and get know exactly which packets are getting transferred in our system.

Postlabs:

1. What are the next hop router's IP Address and MAC Address? How did you get this information?

A: The router will use ARP to lookup the MAC Address of the default gateway. Then the router will do the same to find its next hop. The router finds the IP address of the next hop by a lookup of its routing table to find optimal path.

2. What are local DNS server's hostname and IP Address? How did you get this information?

A: DNS server hostname: UnKnown

DNS server IP Address: 192.168.29.1 / 2405:201:800:1507::c0a8:1d01

This information can be looked up using "ipconfig/all" command in a terminal or command prompt and nslookup "IP Address" to find the hostname of the DNS Server IP.

Exp 9

Server Side:

```
#!/usr/bin/python                                     # This is server.py file

import socket                                         # Import socket module

s = socket.socket()                                  # Create a socket object
host = socket.gethostname()                          # Get local machine name
port = 1234                                          # Reserve a port for your
service.                                             # Bind to the port
s.bind((host, port))

s.listen(5)                                          # Now wait for client

#while True:
c, addr = s.accept()                                # Establish connection with
client.
print('Got connection from', addr)
while(True):
    #print("Receiving.....")
    str=c.recv(1024)
    print("Receiving.....",str)
    if str==b"Bye":
        print("Server Breaking connection\n")
        break
    str=input("enter ur message")
    if not str :
        break
c.send(str.encode())
    #print("Received from client.....")
    #print("Receiving from client....",c.recv(1024))
c.send(b"Bye")
c.close()                                           # Close the connection
```

Client Side:

```
#!/usr/bin/python                                     # This is client.py file

import socket                                         # Import socket module

s = socket.socket()                                  # Create a socket object
host = socket.gethostname()                          # Get local machine name
port = 1234                                          # Reserve a port for your
service.

s.connect((host, port))
while(True):
```



```

str=input("Enter your message")
if not str :
    break
s.send(str.encode())
print("Receiving.....")
str=s.recv(1024)
print(str)
if str==b"Bye":
print("Client Breaking connection\n")
    break
s.send(b"Bye")
s.close

```

Close the socket when done

Outputs :

Server Output :

```

PS C:\programs\python> python chatserver.py
Got connection from ('192.168.29.219', 50092)
Receiving..... b' Checking 001'
enter ur message Checked 001
Receiving..... b' Checking 002'
enter ur message checked 002 .. bye
Receiving..... b''

```

Client Output :

```

PS C:\programs\python> python chatclient.py
Enter your message Checking 001
Receiving.....
b' Checked 001'
Enter your message Checking 002
Receiving.....
b' checked 002 .. bye

```

Conclusion:

From above experiment we learn to establish communication between two processes by implementing socket programming in python (client and server communication).

Postlabs:

1. What is a socket?

A socket is a core endpoint of a two way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP Address and a port number.

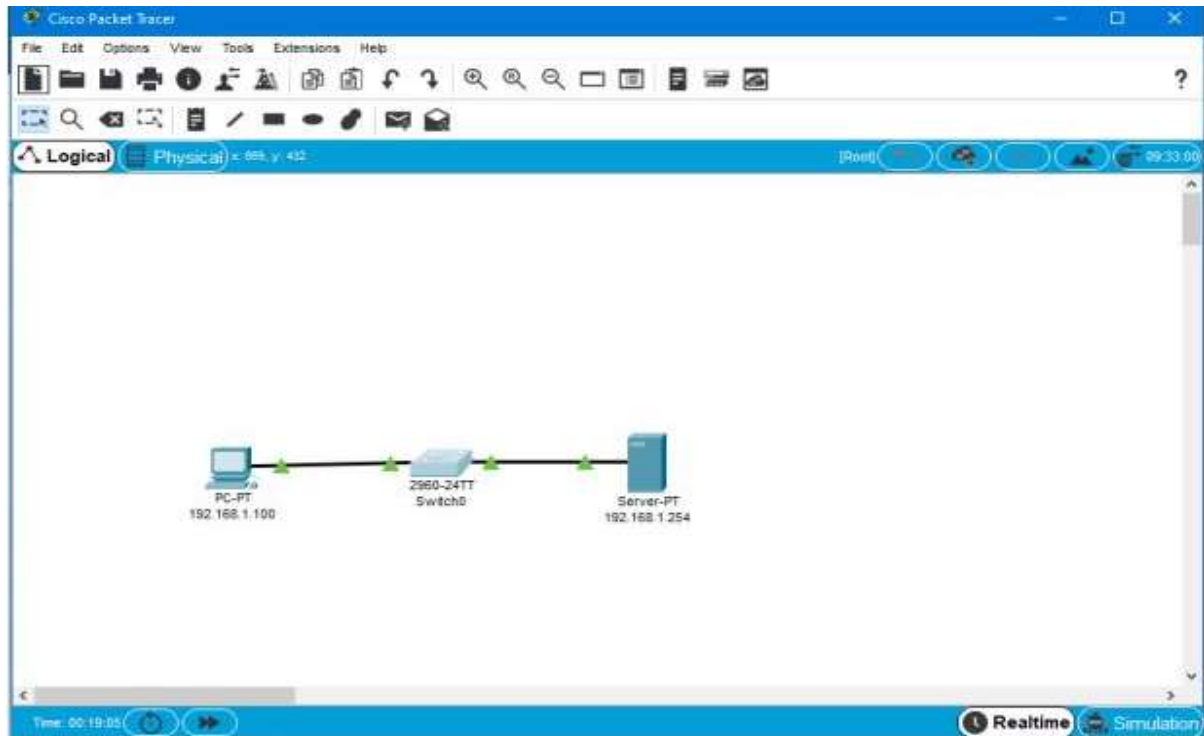
2. What is a stream socket?

In computer operating systems, a stream socket is a type of inter-process communications socket or network socket which provides a connection-oriented, sequenced, and unique flow of data without record boundaries, with well-defined mechanisms for creating and destroying connections and for detecting errors.

3. Differentiate between datagram and stream socket

Datagram Socket	Stream Socket
Datagram socket is a type of network socket which provides connection-less point for sending and receiving packets	Stream socket is a type of inter-process communications socket or network socket which provides a connection-oriented, sequenced, and unique flow of data
Every packet is individually routed and delivered	It provides sequenced delivery of packets
It is an unreliable mechanism with no error detection or prevention mechanisms	It is a reliable mechanism with fixed steps to creating and destroying connections and detecting errors
It uses UDP protocol for transmission	It uses TCP protocol for transmission

Exp 10



```
192.168.1.100
Physical Config Desktop Programming Attributes
Command Prompt
C:\>ping 192.168.1.254

Pinging 192.168.1.254 with 32 bytes of data:

Reply from 192.168.1.254: bytes=32 time<1ms TTL=128
Reply from 192.168.1.254: bytes=32 time=3ms TTL=128
Reply from 192.168.1.254: bytes=32 time<1ms TTL=128
Reply from 192.168.1.254: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.254:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 3ms, Average = 0ms

C:\>ftp 192.168.1.254
Trying to connect...192.168.1.254
Connected to 192.168.1.254
220- Welcome to PT Ftp server
Username: student
331- Username ok, need password
Password:
230- Logged in
(passive mode On)
ftp>quit

221- Service closing control connection.
C:\>
```