CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CHANDUBHAI S PATEL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

Subject Name: Computer Networks Semester: 5th (B.Tech)

Subject Code: IT344 Academic Year: 2018-2019

Lab Manual

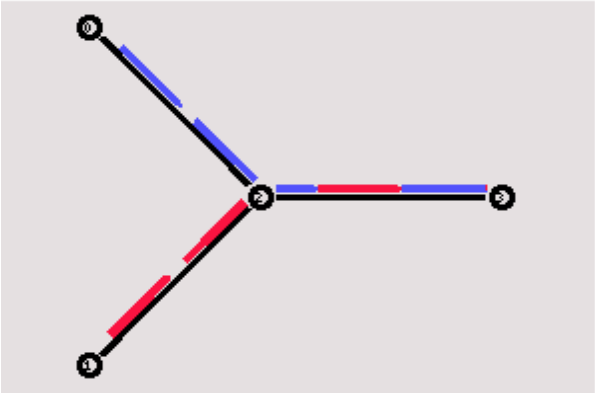
Basics & Compilation of NS-2

Practical: 1

A.

AIM:

Design simple tcl script for Wired topology of 4 nodes in NS-2 and analyze various tcl parameters like network nodes, links, queues and topology. Queue Size :- 5 ,Duplex Link, Queue Type Drop tail



Introduction

The network simulator is discrete event packet level simulator. The network simulator covers a very large number of application of different kind of protocols of different network types consisting of different network elements and traffic models. Network simulator is a package of tools that simulates behavior of networks such as creating network topologies, log events that happen under any load, analyze the events and understand the network. Well the main aim of our first experiment is to learn how to use network simulator and to get acquainted with the simulated

objects and understand the operations of network simulation and we also need to analyze the behavior of the simulation object using network simulation.

Features of NS2

1. It is a discrete event simulator for networking research.
2. It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, HTTP and DSR.
3. It simulates wired and wireless network.
4. It is primarily Unix based.
5. Uses TCL as its scripting language.
6. Otcl: Object oriented support
7. Tclcl: C++ and otcl linkage
8. Discrete event scheduler

Platform required to run network simulator

- Unix and Unix like systems
- Linux (Use Fedora or Ubuntu versions)
- Free BSD
- SunOS/Solaris
- Windows 95/98/NT/2000/XP

Backend Environment of Network Simulator

Network Simulator is mainly based on two languages.They are C++ and OTcl. OTcl is the object oriented version of Tool Command language.The network simulator is a bank of of different network and protocol objects. C++ helps in the following way:

- It helps to increase the efficiency of simulation.
- Its is used to provide details of the protocols and their operation.
- It is used to reduce packet and event processing time.

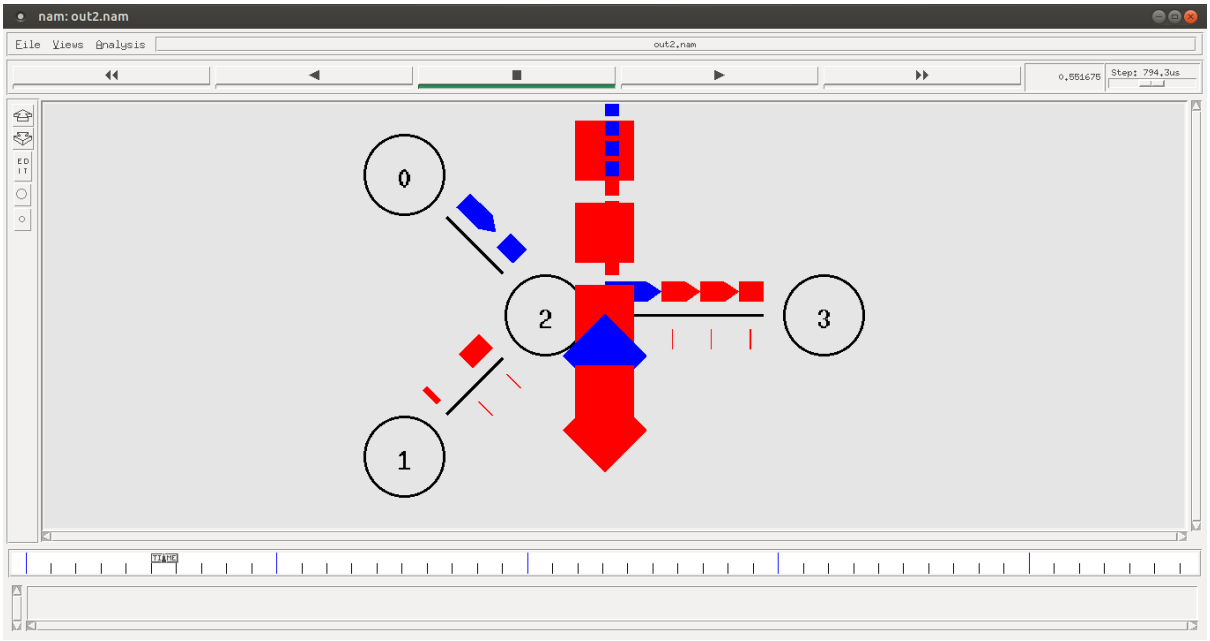
PROGRAMS:

```
#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 out2.nam w]
$ns namtrace-all $nf
```

```
set tr [open out2.tr w]
$ns trace-all $tr

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

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

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

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

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$tcp0 set class_ 2
$ns attach-agent $n0 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp0 $sink1
$tcp0 set fid_ 1
```

```
#Setup a FTP over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP

set tcp1 [new Agent/TCP]
$tcp1 set class_ 2
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
$tcp1 set fid_ 2

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
$ns at 0.1 "$ftp1 start"
$ns at 0.2 "$ftp0 start"
$ns at 4.0 "$ftp0 stop"
$ns at 4.5 "$ftp1 stop"

$ns at 5.0 "finish"

#Run the simulation
$ns run
```

OUTPUT:



B.

Design simple tcl script for Wired topology of 6 nodes in NS-2 and analyze various tcl parameters like network nodes, links, queues and topology.

Link	Bandwidth	Delay	Queue Type	Queue Size
no-n2	10Mbps	10ms	RED	10
n1-n2	10Mbps	10ms	RED	10
n2-n3	5Mbps	???	RED	???
n3-n4	10Mbps	10ms	RED	10
n3-n5	10Mbps	10ms	RED	10

ftp0:
Packet Size: 1000
Rate: 1
Interval: 150

cbr0:
Packet Size: 1500
Rate: 0.05
Interval: 150
Total Simulation Time: 60sec

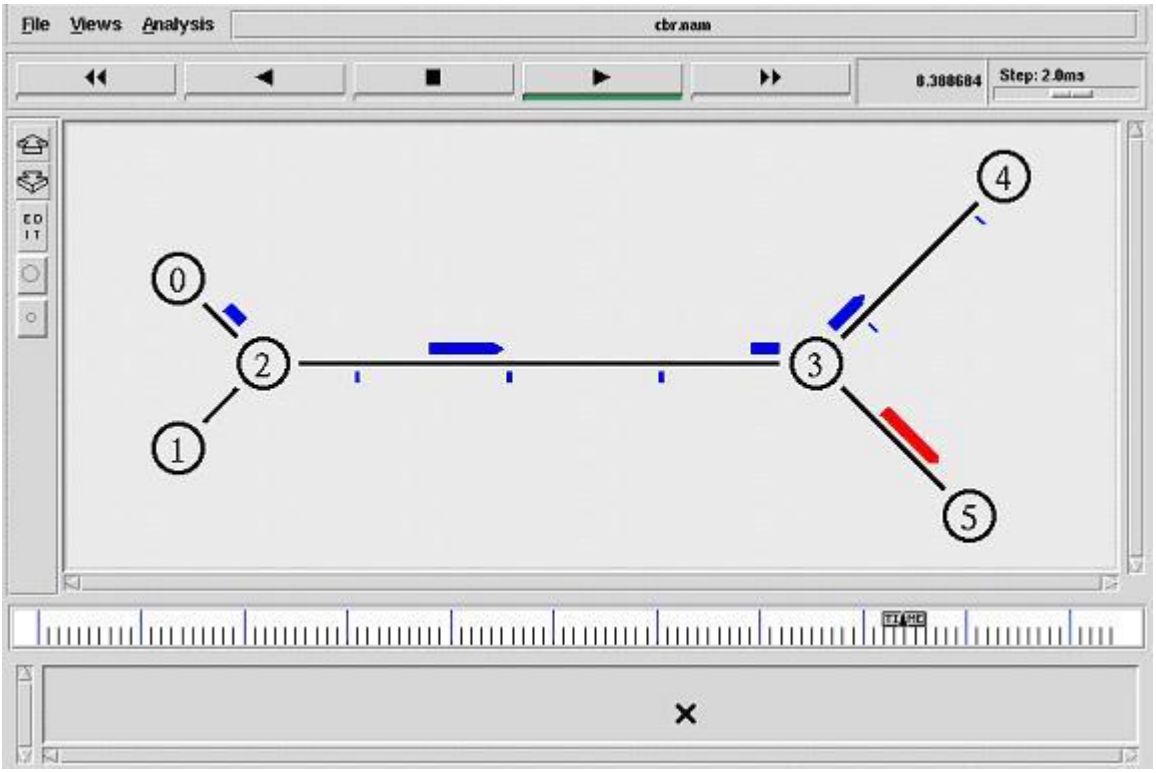
Tcl Script:

```
set ns [new Simulator]

$ns color 1 Green
$ns color 2 Red

set nf [open P6_RED.nam w]
$ns namtrace-all $nf

set tf [open P6_RED.tr w]
```

```
$ns trace-all $tf

#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 P6_RED.nam &
            exec awk -f DropPacket.awk P6_RED.tr &
        exit 0
}

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

#Create links between the nodes
$ns duplex-link $n0 $n2 10Mb 10ms RED
$ns duplex-link $n1 $n2 10Mb 10ms RED

$ns duplex-link $n2 $n3 5Mb 10ms RED

$ns duplex-link $n3 $n4 10Mb 10ms RED
$ns duplex-link $n3 $n5 10Mb 10ms RED


#Set Queue Size of link (n2-n3)
$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10
$ns queue-limit $n2 $n3 15
$ns queue-limit $n3 $n4 10
$ns queue-limit $n3 $n5 10

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

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

$ns duplex-link-op $n2 $n3 orient right

$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$tcp0 set class_ 2
$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set rate_ 1
$ftp0 set interval_ 150
$ftp0 set type_ FTP
$ftp0 set packetSize_ 1000

set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4
$ns connect $tcp0 $sink4
$tcp0 set fid_ 1


#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0

set sink5 [new Agent/Null]
$ns attach-agent $n5 $sink5
$ns connect $udp0 $sink5
$udp0 set fid_ 2

set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set type_ CBR
$cbr0 set packetSize_ 1500
```

```
$cbr0 set rate_ 0.05
$cbr0 set interval_ 150

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr0 start"
$ns at 58.0 "$cbr0 stop"
$ns at 0.2 "$ftp0 start"
$ns at 59.0 "$ftp0 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 60 "finish"

puts "CBR packet size = [$cbr0 set packetSize_]"
puts "CBR interval = [$cbr0 set interval_]"
puts "FTP packet size = [$ftp0 set packetSize_]"
puts "FTP interval = [$ftp0 set interval_]"

#Run the simulation
$ns run
```

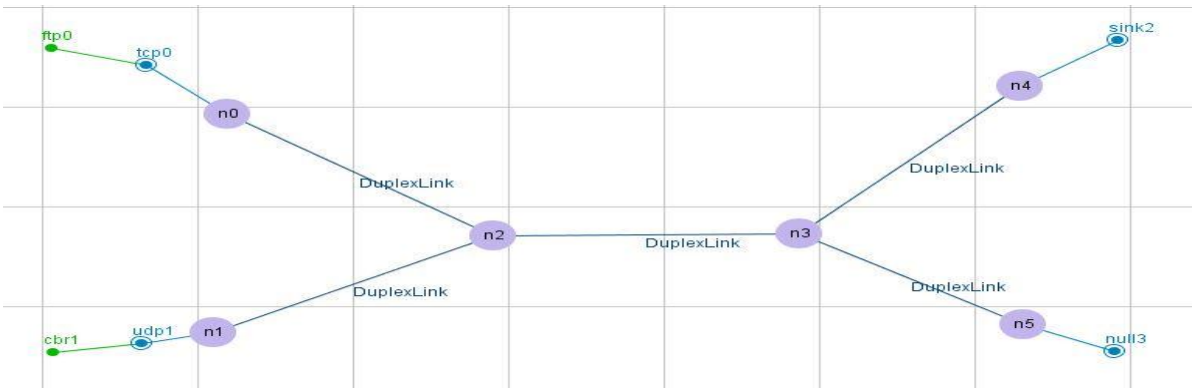


Practical: 2

A.

AIM:

To demonstrate various queuing mechanisms and make comparative analysis of various queuing techniques. (using trace file) (Droptail, RED,SFQ,FQ,FIFO)



Set the following parameters for Duplex Link:

Link	Bandwidth	Delay	Queue Type	Queue Size
no-n2	10Mbps	10ms	RED	10
n1-n2	10Mbps	10ms	RED	10
n2-n3	5Mbps	???	RED	???
n3-n4	10Mbps	10ms	RED	10
n3-n5	10Mbps	10ms	RED	10

ftp0:n0
Packet Size: 1000
Rate: 1
Interval: 150

cbr0:n1
Packet Size: 1500
Rate: 0.05
Interval: 150
Total Simulation Time: 90sec

CODE

(Note: - For other queue change name only)

```
puts -nonewline "Enter Delay      : "
flush stdout
set Delay [gets stdin]

puts -nonewline "Enter Queue Size : "
```

```
flush stdout
set QueueSize [gets stdin]

set ns [new Simulator]

$ns color 1 Green
$ns color 2 Red

set nf [open P6_RED_14IT120.nam w]
$ns namtrace-all $nf

set tf [open P6_RED_14IT120.tr w]
$ns trace-all $tf

#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 P6_RED_14IT120.nam &
           exec awk -f DropPacket.awk P6_RED_14IT120.tr &
        exit 0
}

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

#Create links between the nodes
$ns duplex-link $n0 $n2 10Mb 10ms RED
$ns duplex-link $n1 $n2 10Mb 10ms RED

$ns duplex-link $n2 $n3 5Mb $Delay RED

$ns duplex-link $n3 $n4 10Mb 10ms RED
$ns duplex-link $n3 $n5 10Mb 10ms RED


#Set Queue Size of link (n2-n3)
$ns queue-limit $n0 $n2 10
```

```
$ns queue-limit $n1 $n2 10
$ns queue-limit $n2 $n3 $QueueSize
$ns queue-limit $n3 $n4 10
$ns queue-limit $n3 $n5 10

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

$ns duplex-link-op $n2 $n3 orient right

$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$tcp0 set class_ 2
$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set rate_ 1
$ftp0 set interval_ 150
$ftp0 set type_ FTP

set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4
$ns connect $tcp0 $sink4
$tcp0 set fid_ 1


#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0

set sink5 [new Agent/Null]
$ns attach-agent $n5 $sink5
$ns connect $udp0 $sink5
$udp0 set fid_ 2

set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set type_ CBR
```

```
$cbr0 set packetSize_ 1500
$cbr0 set rate_ 0.05mb
$cbr0 set interval_ 150

$ns at 0.1 "$cbr0 start"
$ns at 59 "$cbr0 stop"

$ns at 0.1 "$ftp0 start"
$ns at 59 "$ftp0 stop"

$ns at 60 "finish"

puts "CBR packet size = [$cbr0 set packetSize_]"
puts "CBR interval = [$cbr0 set interval_]"

#Run the simulation
$ns run
```

B.

AIM:

To demonstrate the use of AWK script with NS2 trace file of scenario A. Find Out Throughput, Packet delivery ratio, Number Drop Packets for all Queues.

Throughput.awk

```
BEGIN{
        file_size = 0
        start_time = 0
        finish_time = 0
        got_start_time = 0
        latency = 0
        throughput = 0
        printf("\n Enter the Target Node for Throughput = ");
        getline node < "-"
}
{
```

```
        if($1 == "r" && $4== node){
                file_size += $6
                if(got_start_time == 0){
                        start_time = $2
                        got_start_time = 1
                }
                finish_time = $2
        }

}

END{

        latency = finish_time - start_time
        throughput = file_size*8/latency

        printf("\n File Size   : %d Bytes",file_size)
        printf("\n Start Time  : %f sec",start_time)
        printf("\n Finish Time : %f sec\n",finish_time)
        printf("\n Throughput  : %f bps \n",throughput)



}
```

Droprate.awk

```
BEGIN{
        drop=0
        rec=0
        eq=0
        deq=0
}
```

```
{
        if($1=="d"){
                drop++
        }
        if($1=="r"){
                rec++
        }
        if($1=="+"){
                eq++
        }
}
END{
        printf("\n\t----- DRR Queue ----")
        printf("\nTotal Number of Packet Dropped in Network = %d",drop)
        printf("\nTotal Number of Packet Send in Network    = %d",eq)
        printf("\nTotal Number of Packet Receive in Network = %d",rec)
        printf("\nTotal Number of Packet in Network        = %d\n\n",drop+rec)
}
```

AIM:

C

As well as Change the parameters of scenario A in such a way that, packet loss becomes Zero, having all the parameters fixed except the queue size, and we have to change queue size and attain minimum packet loss.(For All Queue types) and generate Xgraph for Number of packet drop in each queue.

CODE

```
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

set nf [open P5_14IT120.nam w]
$ns namtrace-all $nf

set tf [open P5_14IT120.tr w]
```

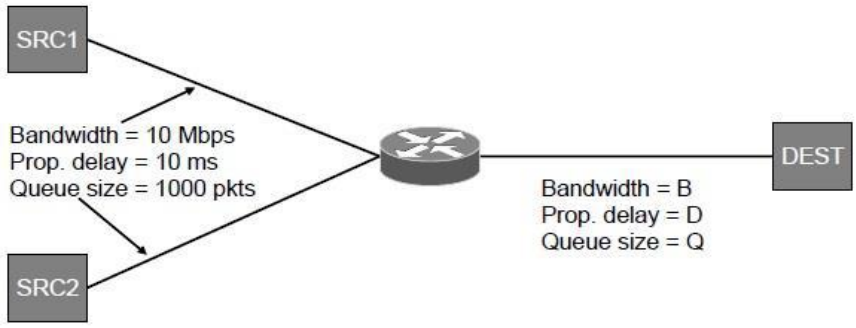
```
$ns trace-all $tf

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
#       exec nam P5_14IT120.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 $n3 5Mb 5ms DropTail
$ns duplex-link $n1 $n3 10Mb 5ms DropTail
$ns duplex-link $n2 $n3 15Mb 5ms DropTail

$ns duplex-link $n3 $n4 7Mb 5ms DropTail

$ns queue-limit $n3 $n4 43

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

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

$ns duplex-link-op $n3 $n4 queuePos 0.5

set tcp0 [new Agent/TCP]
$tcp0 set class_ 2
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP

set tcp1 [new Agent/TCP]
$tcp1 set class_ 2
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
```

```
set tcp2 [new Agent/TCP]
$tcp2 set class_ 2
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

set sink04 [new Agent/TCPSink]
$ns attach-agent $n4 $sink04
$ns connect $tcp0 $sink04
$tcp0 set fid_ 1

set sink14 [new Agent/TCPSink]
$ns attach-agent $n4 $sink14
$ns connect $tcp1 $sink14
$tcp1 set fid_ 2

set sink24 [new Agent/TCPSink]
$ns attach-agent $n4 $sink24
$ns connect $tcp2 $sink24
$tcp2 set fid_ 3

$ns at 0.1 "$ftp1 start"
$ns at 4.0 "$ftp1 stop"

$ns at 0.1 "$ftp0 start"
$ns at 4.0 "$ftp0 stop"


$ns at 0.1 "$ftp2 start"
$ns at 4.0 "$ftp2 stop"

$ns at 5.0 "finish"

$ns run
```

Practical: 3

AIM:

Run a set of simulations using B=10 Mbps, D=10 ms, Q= 20 and draw xgraph of the following advertised windows:
AW1=20, AW2=30
AW1=50, AW2=50
AW1=20, AW2=80
Consider the queue type = Drop tail




```
puts -nonewline "Enter Window Size AW1: "
flush stdout
set aw1 [gets stdin]

puts -nonewline "Enter Window Size AW2: "
flush stdout
set aw2 [gets stdin]

set ns [new Simulator]

$ns color 1 Green
$ns color 2 Red

set nf [open P7_DropTail_14IT069.nam w]
$ns namtrace-all $nf

set tf [open P7_DropTail_14IT069.tr w]
$ns trace-all $tf

#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 P7_DropTail_14IT069.nam &
           exec awk -f DropPacket.awk P7_DropTail_14IT120.tr &
        exit 0
}

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

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

$ns duplex-link $n2 $n3 5Mb 10ms DropTail


#Set Queue Size of link (n2-n3)
$ns queue-limit $n2 $n3 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 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#window
#set aw1 20
#set aw2 30

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$tcp0 set class_ 2
Agent/TCP set window_ $aw1
$tcp0 set window_ $aw1
$ns attach-agent $n0 $tcp0


set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
$ftp0 set type_ FTP
$ftp0 set packetSize_ 1000

set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp0 $sink3
$tcp0 set fid_ 1


set tcp1 [new Agent/TCP]
$tcp1 set class_ 2
Agent/TCP set window_ $aw2
$tcp1 set window_ $aw2
$ns attach-agent $n1 $tcp1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP3
$ftp1 set packetSize_ 1000
set sink4 [new Agent/TCPSink]
$ns attach-agent $n3 $sink4
$ns connect $tcp1 $sink4
$tcp1 set fid_ 2


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$ftp0 start"
$ns at 9.0 "$ftp0 stop"
$ns at 0.2 "$ftp1 start"
$ns at 9.0 "$ftp1 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 10 "finish"

puts "FTP packet size      : [$ftp0 set packetSize_]"

#Run the simulation
$ns run
```

Topological scenario using NS-3

Practical: 4

AIM:-

Briefing of Network Simulator

- Introduction , Features and Network supported by NS3 and platform required to run Network Simulator
- the waf build system
- Backend Environment of Network Simulator
- Installation steps of NS-3 in Ubuntu 14.04 or 16.04 LTS
- Installation and configuration of NetAnim

A comparison of ns-2 and ns-3(From Opensource)

ns-2 is network simulator version 2 and ns-3 is network simulator version 3. This wordplay somehow suggests that ns-3 is merely the next version of ns-2, a very common misconception held by many. There is nothing in common between ns-2 and ns-3 except that both are network simulators. ns-2 can act only as a simulator, whereas ns-3 has the ability to act as a simulator as well as an emulator. One tough decision made by the ns-3 development team was not to make ns-3 backward compatible with ns-2. If ns-3 was backward compatible with ns-2, it might have become more popular. But then the main advantage of ns-3 is that with freshly written code it has been possible to create a new core architecture capable of supporting long term development. So I believe the decision to scrap backward compatibility with ns-2 will benefit ns-3 in the long run.
Even though the cores of both ns-2 and ns-3 are in C++, the ns-2 source code didn't have much impact

on the development of the ns-3 source code. Another implementation level difference between ns-2 and ns-3 is regarding the binding language. The binding language (scripting language) associated with ns-2 is OTcl, whereas for ns-3, it is Python. With today's faster computers, it is not mandatory to have a separate scripting language to save compilation time. So in ns-3, the use of Python scripting is limited or can be avoided altogether. The dependence on a bi-language platform is another difficulty while learning ns-2, because half your time is spent on understanding the binding between the compiled hierarchy (C++ code) and the interpreted hierarchy (OTcl code). This is no longer a problem with ns-3 because C++ alone is sufficient to simulate different networking scenarios.

Another major difference between ns-2 and ns-3 is the way they have reached their present states, over a period of time. The difference is that ns-2 is software that has evolved, whereas ns-3 is planned software. Let me explain. REAL simulator was initially a single persons venture and later on, more code was added to it to create ns-1. Then ns-1 was remodeled and modified to ns-2. In the beginning, nobody had any idea about the eventual fate of ns-2. Source code was added, bugs were fixed, patches were released and, eventually, the present day ns-2 evolved. But with ns-3, this process was altogether different. ns-3 was developed with clear goals and deadlines in mind. Consider the fact that any planned software worth its salt ought to have a mascot or an emblem, but you can't find one for ns-2. Figure 2 shows the emblem for ns-3. So the moral of the story is, Eventually planned software is going to be far more useful than software evolved over time. Yes, you have to trust me on this one—ns-3 is much better than ns-2. Moreover, nowadays ns-2 is only lightly maintained whereas ns-3 is maintained very fervently. To confirm this fact, please check the mailing list of network simulator (ns); topics related to ns-2 are mostly ignored whereas ns-3 topics often lead to heated debates.

Prerequisite packages for Linux are as follows:

1. 1.Minimal requirements for Python: gcc g++ python
2. Debugging and GNU Scientific Library (GSL) support: gdbpython-dev
3. valgrind gsl-bin libgsl0-dev libgsl0ldbl Network Simulation Cradle (nsc): flex bison
4. Reading pcap packet traces: tcpdump
5. 4.Database support for statistics framework: sqlite sqlite3
6. Xml-based version of the config store: libxml2
7. 6.A GTK-based configuration system: libgtk2.0-0
8. Experimental with virtual machines and ns-3: vtun lxc

Detail steps are as follows:

1. sudo apt-get update / dnf update
2. sudo apt-get upgrade / dnf upgrade
3. Once ubuntu/fedora is installed run following command opening the terminal(ctrl+alt+T) window.

4. To install prerequisites dependancy packages- Type the following command in terminal window.

   sudo apt-get/ dnf install gcc g++ python python-dev mercurial bzr gdb valgrind gsl-bin libgsl0-dev libgsl0ldbl flex bison tcpdump sqlite sqlite3 libsqlite3-dev libxml2 libxml2-dev libgtk2.0-0 libgtk2.0-dev uncrustify doxygen graphviz imagemagick texlive texlive-latex-extra texlive-generic-extra texlive-generic-recommended texinfo dia texlive texlive-latex-extra texlive-extra-utils texlive-generic-recommended texi2html python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev python-pygccxml
5. After downloading NS3 on the drive, extract all the files in the NS3 folder, which you have created.
6. Then you can find build.py along with other files in NS3 folder. Then to build the examples in ns-3 run :

   ./build.py --enable-examples –enable-tests

   If the build is successful then it will give output
   "Build finished successfully".
7. Now run the following command on the terminal window,to configure with waf(build tool)

   ./waf -d debug --enable-examples --enable-tests configure

   To build with waf(optional)

   ./waf

8. To test everything allright run the following command on the terminal window,

   ./test.py

   If the tests are ok the installation is done
9. Now after installing ns3 and testing it run some programs first to be ns3 user:
   make sure you are in directory where waf script is available then run

OR

sudo apt-get install gcc g++ python python-dev mercurial qt4-dev-tools tcpdump wireshark gnuplot

Go to following link to download ns-allinone-3.20 tarball:

https://www.nsnam.org/release/ns-allinone-3.20.tar.bz2

cd Home

tar xjf ns-allinone-3.20.tar.bz2

Go in ns-allinone-3.20 folder and give the following command for installation:

./build.py --enable-examples --enable-tests

Go to ns-allinone-3.20/ns-3.20 folder and give the following directory:

./test.py -c core

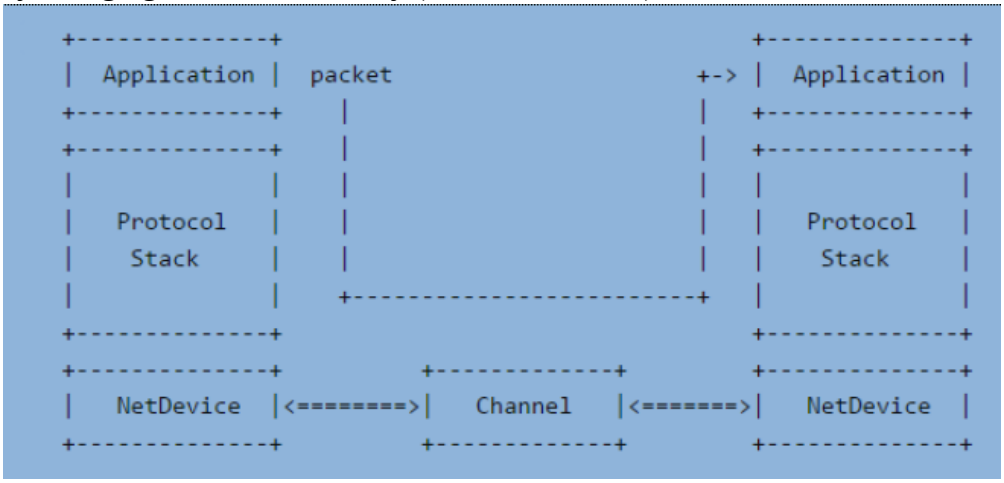
Go to ns-allinone-3.20/netanim-3.105 and give the following command:

./NetAnim

Practical: 5

AIM :-

Design simple program for two nodes client server wired topology and analyze behavior of this topology by changing data rate and delay.(Point – to - Point)



Node

Because in any network simulation, we will need nodes. So ns-3 comes with NodeContainer that you can use to manage all the nodes (Add, Create, Iterate, etc.).

```
// Create two nodes to hold.
NodeContainer nodes;
nodes.Create (2);
```

Channel and NetDevice

In the real world, they correspond to network cables (or wireless media) and peripheral cards (NIC). Typically these two things are intimately tied together. In the first example, we are usingPointToPointHelper that wraps the Channel and NetDevice.

```
// Channel: PointToPoint, a direct link with `DataRate` and `Delay` specified.
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

Then we need to install the devices. The internal of Install is actually more complicated, but for now, let’s just skip the magic behind the scene.

```
// NetDevice: installed onto the channel
NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);
```

Protocols

Internet and IPv4. Since Internet is the current largest network to study, ns-3 has a particular focus on it. The InternetStackHelper will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.

```
// Protocol Stack: Internet Stack
InternetStackHelper stack;
stack.Install (nodes);
```

To assign IP addresses, use a helper and set the base. The low level ns-3 system actually remembers all of the IP addresses allocated and will generate a fatal error if you accidentally cause the same address to be generated twice.

```
// Since IP Address assignment is so common, the helper does the dirty work!
// You only need to set the base.
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
// Assign the address to devices we created above
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

Applications

Every application needs to have Start and Stop function so that the simulator knows how to schedule it. Other functions are application-specific. We will use UdpEchoServer and UdpEchoClientfor now.

```
// Application layer: UDP Echo Server and Client
// 1, Server:
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
// 2, Client:
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Simulation

```
// Start Simulation
Simulator::Run ();
Simulator::Destroy ();
return 0;
```

CODE

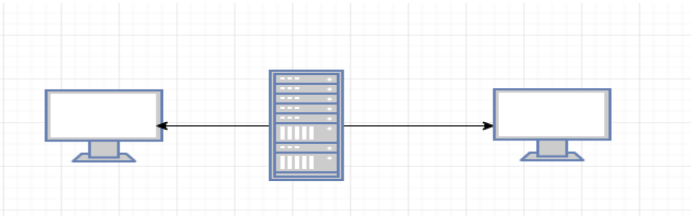
```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
```

```
int
main (int argc, char *argv[])
{
Time::SetResolution (Time::NS);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
NodeContainer nodes;
nodes.Create (2);
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);
InternetStackHelper stack;
stack.Install (nodes);
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign (devices);
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
Simulator::Run ();
Simulator::Destroy ();
return 0;
```

Practical: 6

AIM :-

Program in NS3 for connecting three nodes considering one node as a central node and generate trace file.



CODE

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
```

```
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
int
main (int argc, char *argv[])
{
Time::SetResolution (Time::NS);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
NodeContainer nodes;
nodes.Create (3);
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer devices, devices1;
devices = pointToPoint.Install (nodes.get(0),nodes.get(1));
devices1 = pointToPoint.Install (nodes.get(2),nodes.get(1));
InternetStackHelper stack;
stack.Install (nodes);
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign (devices);
Ipv4InterfaceContainer interfaces1 = address.Assign (devices1);
UdpEchoServerHelper echoServer (90);
ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 90);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
UdpEchoClientHelper echoClient (interfaces1.GetAddress (1), 90);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps1 = echoClient.Install (nodes.Get (2));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
Simulator::Run ();
Simulator::Destroy ();
return 0;
```

Practical: 7

AIM :-

A Star Network Topology is best suited for smaller networks and works efficiently when there is limited number of nodes. One has to ensure that the hub or the central node is always working and extra security features should be added to the hub because it s the heart of the network.
Program in NS3 to implement star topology and generate trace file as well as animation file.

CODE

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-layout-module.h"
// Network topology (default)
//
// n2 n3 n4 .
// \ | / .
// \|/ .
// n1--- n0---n5 .
// /|\ .
// / | \ .
// n8 n7 n6 .
//
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Star");
int
main (int argc, char *argv[])
{
//
// Set up some default values for the simulation.
//
Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue
(137));
// ??? try and stick 15kb/s into the data rate
Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue
("14kb/s"));
//
// Default number of nodes in the star. Overridable by command line argument.
//
uint32_t nSpokes = 8;
CommandLinecmd;
```

```
cmd.AddValue ("nSpokes", "Number of nodes to place in the star", nSpokes);
cmd.Parse (argc, argv);
NS_LOG_INFO ("Build star topology.");
PointToPointHelperpointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
PointToPointStarHelper star (nSpokes, pointToPoint);
NS_LOG_INFO ("Install internet stack on all nodes.");
```

```
InternetStackHelper internet;
star.InstallStack (internet);
NS_LOG_INFO ("Assign IP Addresses.");
star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"));
NS_LOG_INFO ("Create applications.");
//
// Create a packet sink on the star "hub" to receive packets.
//
uint16_t port = 50000;
Address hubLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", hubLocalAddress);
ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());
hubApp.Start (Seconds (1.0));
hubApp.Stop (Seconds (10.0));
//
// Create OnOff applications to send TCP to the hub, one on each spoke node.
//
OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());
onOffHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
onOffHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
ApplicationContainer spokeApps;
for (uint32_t i = 0; i < star.SpokeCount (); ++i)
{
AddressValue remoteAddress (InetSocketAddress (star.GetHubIpv4Address (i),
port));
onOffHelper.SetAttribute ("Remote", remoteAddress);
spokeApps.Add (onOffHelper.Install (star.GetSpokeNode (i)));
}
spokeApps.Start (Seconds (1.0));
spokeApps.Stop (Seconds (10.0));
NS_LOG_INFO ("Enable static global routing.");
//
// Turn on global static routing so we can actually be routed across the
star.
//
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
NS_LOG_INFO ("Enable pcap tracing.");
//
// Do pcap tracing on all point-to-point devices on all nodes.
//
pointToPoint.EnablePcapAll ("star");
NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
return 0;
```

Practical: 8

AIM :-

Program in NS3 to implement a bus topology and generate trace file as well as graph using gnu plot.

CODE

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
// Default Network Topology
// 10.1.1.0
// n0 ---------- n1 n2 n3 n4
// point-to-point | | | |
// ================
// LAN 10.1.2.0
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");
int
main (int argc, char *argv[])
{
bool verbose = true;
uint32_t nCsma = 3;
CommandLine cmd;
cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
cmd.Parse (argc,argv);
```

```
if (verbose)
{
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
}
nCsma = nCsma == 0 ? 1 : nCsma;
NodeContainer p2pNodes;
p2pNodes.Create (2);
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
InternetStackHelper stack;
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
address.SetBase ("10.1.2.0", "255.255.255.0");
```

```
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second", csmaDevices.Get (1), true);
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

```
[root@mta-192-180-1-110 ns-3.24.1]# ./waf --run udpro8
Waf: Entering directory `/home/mait/Downloads/ns-allinone-3.24.1/ns-3.24.1/build
'
Waf: Leaving directory `/home/mait/Downloads/ns-allinone-3.24.1/ns-3.24.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.566s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.0078s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.0078s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.01761s client received 1024 bytes from 10.1.2.4 port 9
[root@mta-192-180-1-110 ns-3.24.1]#
```

Practical: 9

AIM :-

Program in NS3 for connecting multiple routers and nodes and building a hybrid topology and generate trace file as well as graph using gnuplot.

Installing NetAnim
The website:
http://www.nsnam.org/wiki/index.php/NetAnim
1. Install Mercurial:
apt−get/dnf install mercurial
2. Install QT4 development package:
apt−get/dnf install qt4−dev−tools
3. You can use Synaptic too, to install both the above packages.
4. Download NetAnim: hg clone http :// code .nsnam. org/netanim
5. Build NetAnim:
cd netanim
make clean qmake NetAnim. pro make

Compiling code with NetAnim
So you will have to make the following changes to the code, in order to view the animation on NetAnim.

```
#include " ... "
#include "ns3/netanim−module .h" //1 Include. . .
int main ( int argc , char ∗argv [ ] )
{ std : : string animFile = "somename. xml"; //2 Name of f i l e for animation
. . .
AnimationInterface anim ( animFile ); //3 Animation interface
Simulator : : Run (); Simulator : : Destroy (); return 0;
}
```

4.1 To run the code:
1. Move the waf , waf.bat , wscript and wutils.py les in to the scratch folder (~/ns-allinone-3.24/ns-3.24/scratch/).
2. Move the example code to the scratch folder and make the changes required for NetAnim, as shown above.
3. Now cd to the scratch folder (cd ~/ns-allinone-3.24/ns-3.24/scratch/).
4. Run the code using the command:
./ waf −−run <filename>
Note: < lename> should not contain the extension .cc
4.2 To visualize on NetAnim:
1. cd to the netanim folder (cd ~/netanim/).
2. Run Netanim:
./NetAnim
3. Include the .xml le generated in the ns-3.24 folder (~/ns-allinone-3.17/ns3.24/).