Study of NS2: Computer Network Simulator

Introduction

Network simulators are tools used to simulate discrete events in a network and which helps to predict the behaviour of a computer network. Generally the simulated networks have entities like links, switches, hubs, applications, etc. Once the simulation model is complete, it is executed to analyse the performance. Administrators can then customize the simulator to suit their needs. Network simulators typically come with support for the most popular protocols and networks in use today, such as WLAN,UDP,TCP,IP, WAN, etc.

Most simulators that are available today are based on a GUI application like the NCTUNS while some others including NS2 are CLI based. Simulating the network involves configuring the state elements like links, switches, hubs, terminals, etc. and also the events like packet drop rate, delivery status and so on. The most important output of the simulations are the trace files. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble spots. Most of the simulation is performed in discrete time intervals where events that are in the queue are processed one after the other in an order.

Since simulation is a complex task, we cannot guarantee that all the simulators can provide exact or accurate results for all the different type of information. Examples of network simulators are: ns, NCTUNS, NetSim, etc.

ns2 is a name for series of discrete event network simulators like ns-1, ns-2 and ns-3. All of them are discrete-event network simulators, primarily used in research and teaching. ns2 is free software, publicly available under the GNU GPLv2 license for research, development, and use.

Installation of NS2 in Ubuntu

Download the all in one package for ns2 from

```
https://sourceforge.net/projects/nsnam/files/latest/download
```

The package downloaded will be named "ns-allinone-2.35.tar.gz". Copy it to the home folder. Then in a terminal use the following two commands to extract the contents of the package.:

```
cd ~/
tar -xvzf ns-allinone-2.35.tar.gz
```

All the files will be extracted into a folder called "ns-allinone-2.35". Now we are ready to install ns2. To do so we first require root privileges and then we can run the install script. ow go to ns-allinone-2.35/ns-2.35/linkstate sub folder. double click on "ls.h" file to open. go to line number 137 and change the below line

```
from
void eraseAll() { erase(baseMap::begin(), baseMap::end()); }
to
void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
```

Use the following two commands:

```
sudo su cd ~/ns-allinone-2.35/./install
```

Modify .bashrc file to set environment

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/your-username/ns-allinone-2.35/otcl-1.14
NS2_LIB=/home/your-username/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:
    $X11_LIB:$USR_LOCAL_LIB
# TCL_LIBRARY
TCL_LIB=/home/your-username/ns-allinone-2.35/tcl8.5.10/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB
# PATH
XGRAPH=/home/your-username/ns-allinone-2.35/bin:/home/akshay/ns-
    allinone-2.35/tcl8.5.10/unix:/home/akshay/ns-allinone-2.35/
    tk8.5.10/unix
#the above two lines beginning from xgraph and ending with unix
    should come on the same line
NS=/home/your-username/ns-allinone-2.35/ns-2.35/
NAM=/home/your-username/ns-allinone-2.35/nam-1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Running ns2

Once the system has restarted, open a terminal and start ns2 by using the following command:
ns

A simple example

Let us go through a simple example of a wired network that exchange packets between the nodes.

```
#create a new simulator object ns
set ns [new Simulator]
# creation of two nodes n0 and n1
set n0 [$ns node]
set n1 [$ns node]
#Open the files for tracing in write mode, if the file already
    exits, it will be overwritten, else a #new file will be
    created
set tracefile [open out.tr w]
$ns trace-all $tracefile
#Open the files for network animation
set namfile [open out.nam w]
```

```
$ns namtrace-all $namfile
#Create a duplex link between nodes n0 and n1
$ns duplex-link $n0 $n1 2MB 10ms DropTail
#Creates TCP Agent and attach it to node 0
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
#creates TCP Sink Agent and attach that agent to node 1
set tcpsink0 [new Agent/TCPSink]
$ns attach-agent $n1 $tcpsink0
# Set the Traffic for FTP and attach it to TCP Agent
set ftp [new Application/FTP]
$ftp attach-agent $tcp0
#connect the source and the sink
$ns connect $tcp0 $tcpsink0
#Start the traffic at 1.0seconds.
$ns at 1.0 "$ftp start"
#call the finish procedure at 4.0 sec
$ns at 4.0 "finish"
#Procedure finish{}
proc finish {} {
    global ns tracefile namfile
         exec nam out.nam &;
        #execute the animation within the procedure
        exit 0
}
#Run the simulation
$ns run
```

Experiment - Finding number of dropped packets

Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packet drops due to congestion

```
set ns [new Simulator]
set tf [open lab2.tr w]
$ns trace-all $tf
set nf [open lab2.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
$n0 label "Ping0"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$n2 label "Router"
$ns color 1 "red"
$ns color 2 "green"
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
```

```
$ns duplex-link $n3 $n2 1Mb 300ms DropTail
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail

$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n2 $n6 2
$ns queue-limit $n5 $n2 5

#The below code is used to connect between the ping agents to the
    node n0,
#n4 , n5 and n6.
set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0

set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4


set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5
set ping6 [new Agent/Ping]
$ns attach-agent $n6 $ping6

$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001
$ping5 set packetSize_ 60000
$ping5 set interval_ 0.00001

$ping0 set class_ 1
$ping5 set class_ 2
$ns connect $ping0 $ping4
$ns connect $ping5 $ping6

#Define a 'recv' function for the class 'Agent/Ping'
#The below function is executed when the ping agent receives a
    reply from the destination
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts " The node [$node_ id] received an reply from $from with
    round trip time of $rtt"
}
proc finish {} {
global ns nf tf
exec nam lab2.nam &
$ns flush-trace
close $tf
close $nf
exit 0
}

#Schedule events
$ns at 0.1 "$ping0 send"
$ns at 0.2 "$ping0 send"
$ns at 0.3 "$ping0 send"
$ns at 0.4 "$ping0 send"
$ns at 0.5 "$ping0 send"
```

```
$ns at 0.6 "$ping0 send"
$ns at 0.7 "$ping0 send"
$ns at 0.8 "$ping0 send"
$ns at 0.9 "$ping0 send"

$ns at 1.0 "$ping5 send"
$ns at 1.1 "$ping5 send"
$ns at 1.2 "$ping5 send"
$ns at 1.3 "$ping5 send"
$ns at 1.4 "$ping5 send"
$ns at 1.5 "$ping5 send"
$ns at 1.6 "$ping5 send"
$ns at 1.7 "$ping5 send"
$ns at 1.8 "$ping5 send"


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

We can use AWK - the text processing language to print the number of packets from files generated by NS2 while running the ns2 simulator. The listing for awk is given below.

```
BEGIN
{
    count=0;
}
{
    if ($1== "d")
        count++;
}
END
{
    printf("The total number of packets dropped : %d \n \n", count
    );
}

```

Run awk from command prompt:
awk -f lab2.awk lab2.tr

Simulate Link State Protocol using NS2

To simulate link state protocol we will go through the following steps

```
#step1: Initialise network
set ns [new Simulator]
$ns rtproto LS
#Step-2: Creating number of nodes:
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]
set node7 [$ns node]
#Step-3: Creating the trace file:
```

Step-1: Initializing the network: The first step is to initialize the network simulator, and we do so by creating a network simulator object. After that, we initialize rtproto (routing protocol) to Link State (LS).
Step-2: Creating number of nodes : We next create a random number of nodes, let's say 7. We use the node instance to create these nodes as follows.
Step-3: Creating the trace file : Our next step is to create the trace file and nam file. The nam file is used to view simulator output whereas the trace file traces all the routing information in the process. For this we create trace file and nam file objects and then open the files in write mode. The trace-all instance is used to trace all routing information into the trace file and similarly namtrace-all for the nam file.
Step-4: Labeling the nodes : In the next step we can label the nodes if we wish to. Here we are labeling them from node 0 to node 6. We can also customize the labels by assigning different colors to them and thus viewing the simulation much more

```
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf
#Step-4: Labeling the nodes :
$node1 label "node 1"
$node1 label "node 2"
$node1 label "node 3"
$node1 label "node 4"
$node1 label "node 5"
$node1 label "node 6"
$node1 label "node 7"
$node1 label-color blue
$node2 label-color red
$node3 label-color red
$node4 label-color blue
$node5 label-color blue
$node6 label-color blue
$node7 label-color blue

#Step-5: Creating duplex links :
$ns duplex-link $node1 $node2 1.5Mb 10ms DropTail
$ns duplex-link $node2 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node3 $node4 1.5Mb 10ms DropTail
$ns duplex-link $node4 $node5 1.5Mb 10ms DropTail
$ns duplex-link $node5 $node6 1.5Mb 10ms DropTail
$ns duplex-link $node6 $node7 1.5Mb 10ms DropTail
$ns duplex-link $node7 $node1 1.5Mb 10ms DropTail

#Step-6: Orient the links between the nodes :
$ns duplex-link-op $node1 $node2 orient left-down
$ns duplex-link-op $node2 $node3 orient left-down
$ns duplex-link-op $node3 $node4 orient right-down
$ns duplex-link-op $node4 $node5 orient right
$ns duplex-link-op $node5 $node6 orient right-up
$ns duplex-link-op $node6 $node7 orient left-up
$ns duplex-link-op $node7 $node1 orient left-up
#Step-7: Attaching TCP agents :
set tcp2 [new Agent/TCP]
$ns attach-agent $node1 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $node4 $sink2
$ns connect $tcp2 $sink2
# Step-8: Creating FTP traffic :
set traffic_ftp2 [new Application/FTP]
$traffic_ftp2 attach-agent $tcp2
#Step-9: Adding a finish procedure :
proc finish{} {

global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0

}
# Step-10: Scheduling the FTP :
$ns at 0.5 "traffic_ftp2 start"
```

```
$ns rtmodel-at 1.0 down $node2 $node3
$ns rtmodel-at 2.0 up $node2 $node3
$ns at 3.0 "traffic_ftp2 start"
$ns at 4.0 "traffic_ftp2 stop"
$ns at 5.0 "finish"
$ns run
```

Step-6: Orient the links between the nodes : Now we need to orient the links between the nodes appropriately to obtain proper alignment. The duplex-link-op instance is used for the same.
Step-7: Attaching TCP agents : The next step is to attach TCP agents (using attach-agent) at two nodes let's say node 1 and node 4. We can do this creating the source and sink objects and connecting them using connect instance. Step-8: Creating FTP traffic : Our next step is to create FTP traffic and attach to TCP source. The traffic then flows across node 1 and node 4. We can do this by creating an FTP agent and attaching it to tcp2.
Step-9: Adding a finish procedure : The next step is to add a finish procedure to flush all data into trace file and then and then run the nam file.
Step-10: Scheduling the FTP : The final step is to schedule the FTP traffic at the required time intervals. We can also disable the link between any pair of nodes at a certain timestamp using rtmodel-at instance and then enable it after a certain time. This is majorly done for testing purposes. Here we have disabled the link between nodes 2 and 3. The program ends with the run command.