NS2 – Network Simulator 2

NS2 Introduction:

- NS-2 stands for Network Simulator Version 2
- An event driven simulation tool
- Useful in studying the dynamic nature of communication networks
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors

NS Simulator Preliminaries:

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Structure of NS-2 Program: (Tcl scripting)

- Creating a Simulator Object
- Setting up files for trace & NAM
- Tracing files using their commands
- Closing trace file and starting NAM
- Creating LINK & NODE topology & Orientation of links

Working of NS-2:

- NS2 provides users with executable command ns which takes an input argument, the name of a Tcl simulation scripting file.
- Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.
- In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation.

Trace file and NamTrace file:

- Once the simulation is complete, we get to see two files – trace.tr and nam.out
- The trace file (trace.tr) is a standard format used by ns2.
- In ns2, each time a packet moves from one node to another, or onto a link, or into a buffer, etc., it gets recorded in this trace file.
- Each row represents one of these events and each column has its own meaning.

Step1: Declare Simulator and setting output file
Step2: Setting Node and Link
Step3: Setting Agent
Step4: Setting Application
Step5: Setting Simulation time and schedules
Step6: Declare finish.

Step 1: Declare Simulator and setting output files

An ns simulation starts with the command

```
set ns [new Simulator]
```

It is the first line in the tcl script. This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using "open" command:

#Open the Trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

```
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

The above creates a trace file called "out.tr" and a nam visualization trace file called "out.nam". Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called "tracefile1" and "namfile" respectively.

The termination of the program is done using a "finish" procedure.

#Define a 'finish' procedure

```
Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0
}
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method "**flush-trace**" will dump the traces on the respective files. The tcl command "**close**" closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will end the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that is a exitbecause something fails.

At the end of ns program we should call the procedure "finish" and specify at what time the termination should occur. For example,

```
$ns at 125.0 "finish"
```

will be used to call "**finish**" at time 125sec.Indeed,the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command

```
$ns run
```

Step 2: Setting Node and Link

The way to define a node is

```
set n0 [$ns node]
```

The node is created which is printed by the variable n0. When we shall refer to that node in the script we shall thus write $n0.

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

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

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a unidirectional link instead of a bi-directional one, we should replace "duplex- link" by "simplex-link".

The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

We should also define the buffer capacity of the queue related to each link.

set n0 [$ns node]	setting a node
$ns duplex-link $n0 $n2 3Mb 5ms DropTail	#bidirectional link between n0 and n2 is declared bandwidth 3Mbps and delay 5ms. DropTail is a waitingqueue type.
$ns simplex-link $n0 $n2 3Mb 5ms DropTail	#unidirectional link

Step 3: Setting Agent

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

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

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

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

Step 4: Setting Application

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

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packet_size_ 512
```

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$tcp set packet_size_ 512
```

Step 5: Setting time schedule for simulation

Time schedule of a simulation is set as below:

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

Step 6: Declare finish

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

$ns at 4.0 "finish"	The finish function is used to output data file at the end ofsimulation.

proc finish {} {
global ns file namfile tcpfile
$ns flush-trace
close $file
close $namfileclose
$tcpfile
exit 0
}	

Structure of Trace Files

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, The meaning of the fields are:

Event	Time	From Node	To Node	PKT Type	PKT Size	Flags	Fid	Src Addr	Dest Addr	Seq Num	Pkt id

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.

3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of "node.port".
10. This is the destination address, given in the same form.
11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes
12. The last field shows the Unique id of the packet.

AWK file:

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

Syntax:

```
awk –f file.awk file.tr
```

XGRAPH:

The xgraph program draws a graph on an x-display given data read from either data file or from standard input if no files are specified. It can display upto 64 independent data sets using different colors and line styles for each set. It annotates the graph with a title, axis labels, grid lines or tick marks, grid labels and a legend.

Syntax:

```
Xgraph [options] file-name
```