**Part A Experiments**

1. **Overview of NS-2**

**1.1 Introduction to NS-2**

NS-2 is an event driven packet level network simulator developed as part of the VINT project (Virtual Internet Testbed). This was a collaboration of many institutes including UC Berkeley, AT&T, XEROX PARC and ETH. Version 1 of NS was developed in 1995 and with version 2 released in 1996. Version 2 included a scripting language called Object oriented Tcl (OTcl). It is an open source software package available for both Windows and Linux platforms. It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

NS-2 has many and expanding uses including:

* To evaluate the performance of existing network protocols.
* To evaluate new network protocols before use.
* To run large scale experiments not possible in real experiments.
* To simulate a variety of ip networks

**1.2 Downloading/Installing ns**

You can download the package from <http://www.isi.edu/nsnam/ns/ns-build.html>. There are two ways to build ns: from the various packages or ‘all-in-one’ package. For simplicity, it is recommended to start with the ‘all-in-one’ package. Please refer <http://www.isi.edu/nsnam/ns/ns-problems.html> for any installation problems.

**1.3 Starting ns**

You start ns with the command 'ns <tclscript>' (assuming that you are in the directory with the ns executable, or that your path points to that directory), where '<tclscript>' is the name of a Tcl (Tool Command Language) script file which defines the simulation scenario (i.e. the topology and the events). You can also just start ns without any arguments and enter the Tcl commands in the Tcl shell, but that is definitely less comfortable**.**

**1.4 Starting nam (Network Animator)**

You can either start nam with the command 'nam <nam-file>' where '<nam-file>' is the name of a nam trace file that was generated by ns, or you can execute it directly out of the Tcl simulation script for the simulation which you want to visualize.

**1.5 Architecture of NS-2**

As shown in the simplified user's view of Figure, NS is an Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network set-up (plumbing) module libraries.



An OTcl script will do the following.

* Initiates an event scheduler.
* Sets up the network topology using the network objects.
* Tells traffic sources when to start/stop transmitting packets through the event scheduler

Another major component of NS besides network objects is the event scheduler. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. The event scheduler in NS-2 performs the following tasks:

* Organizes the simulation timer.
* Fires events in the event queue.
* Invokes network components in the simulation.

Depending on the user’s purpose for an OTcl simulation script, simulation results are stored as trace files, which can be loaded for analysis by an external application:

1. A NAM trace file (file.nam) for use with the Network Animator Tool
2. A Trace file (file.tr) for use with XGraph or TraceGraph [11].



TclCL is the language used to provide a linkage between C++ and OTcl. Toolkit Command Language (Tcl/OTcl) scripts are written to set up/configure network topologies. TclCL provides linkage for class hierarchy, object instantiation, variable binding and command dispatching. OTcl is used for periodic or triggered events.

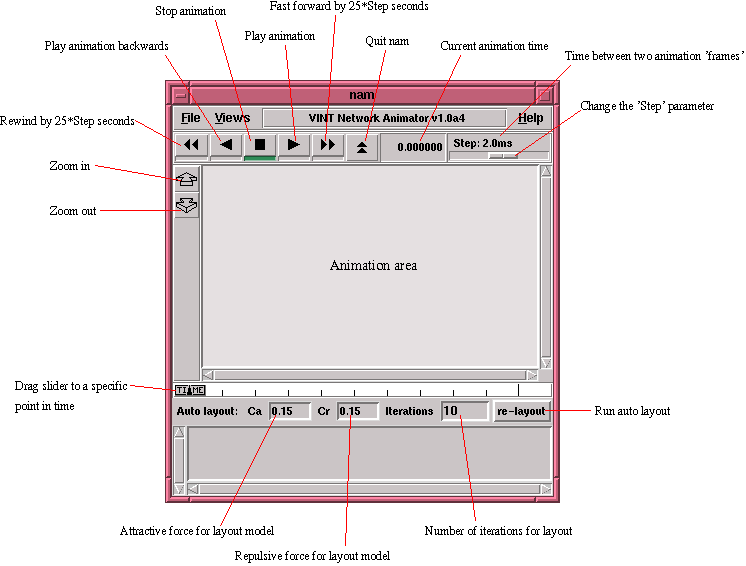
**1.6 NS-2 features**

NS-2 implements the following features

1. Router queue Management Techniques DropTail, RED, CBQ,
2. Multicasting
3. Simulation of wireless networks
   * Developed by Sun Microsystems + UC Berkeley (Daedalus Project)
   * Terrestrial (cellular, adhoc, GPRS, WLAN, BLUETOOTH), satellite
   * IEEE 802.11 can be simulated, Mobile-IP, and adhoc protocols such as
   * DSR, TORA, DSDV and AODV.
4. Traffic Source Behaviour- www, CBR, VBR
5. Transport Agents- UDP/TCP
6. Routing
7. Packet flow
8. Network Topology
9. Applications- Telnet, FTP, Ping
10. Tracing Packets on all links/specific links

**1.7 NAM (Network Animator)**

NAM provides a visual interpretation of the network topology created. Below you can see a screenshot of a nam window where the most important functions are being explained.



Its features are as follows:

* Provides a visual interpretation of the network created
* Can be executed directly from a Tcl script
* Controls include play, stop ff, rw, pause, a display speed controller and a packet monitor facility.
* Presents information such as throughput, number packets on each link.
* Provides a drag and drop interface for creating topologies.

**1.8 XGraph**

XGraph is an X-Windows application that includes:

* Interactive plotting and graphing
* Animation and derivatives

To use XGraph in NS-2 the executable can be called within a TCL Script. This will then load a graph displaying the information visually displaying the information of the trace file produced from the simulation.



**XGraph running comparing three trace files in a graph**

**1.9 TraceGraph**

TraceGraph is a trace file analyser that runs under Windows, Linux and UNIX systems and requires Matlab 6.0 or higher.

TraceGraph supports the following trace file formats.

* + Wired
  + Satellite
  + Wireless

1. **OTcl Scripting with NS-2**

In NS-2, the network is constructed using nodes which are connected using links. Events are scheduled to pass between nodes through the links. Nodes and links can have various properties associated with them. Agents can be associated with nodes and they are responsible for generating different packets (e.g. TCP agent or UDP agent). The traffic source is an application which is associated with a particular agent (e.g. ping application).



This diagram shows two nodes, a link, an agent and an application.

**2.1 How to start ?**

First of all, you need to create a simulator object. This is done with the command

set ns [new Simulator]

Now we open a file for writing that is going to be used for the nam trace data.

set nf [open out.nam w]

$ns namtrace-all $nf

The first line opens the file 'out.nam' for writing and gives it the file handle 'nf'. In the second line we tell the simulator object that we created above to write all simulation data that is going to be relevant for nam into this file. The next step is to add a 'finish' procedure that closes the trace file and starts nam.

proc finish {} {

global ns nf

$ns flush-trace

close $nf

exec nam out.nam &

exit 0

}

The last line finally starts the simulation

$ns run

**2.2 Node creation and linking**

The following two lines define the two nodes. (Note: You have to insert the code in this section **before** the line '$ns run', or even better, before the line '$ns at 5.0 "finish"').

set n0 [$ns node]

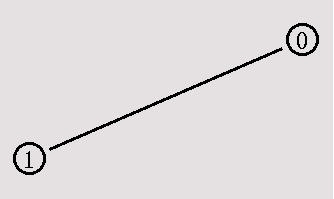
set n1 [$ns node]

A new node object is created with the command '$ns node'. The above code creates two nodes and assigns them to the handles 'n0' and 'n1'. The next line connects the two nodes.

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

This line tells the simulator object to connect the nodes n0 and n1 with a duplex link with the bandwidth 1Megabit, a delay of 10ms and a DropTail queue.

Now you can save your file and start the script with 'ns example1.tcl'. nam will be started automatically and you should see an output that resembles the picture below.



**2.3 Sending data**

The next step is to send some data from node n0 to node n1. In ns, data is always being sent from one 'agent' to another. So the next step is to create an agent object that sends data from node n0, and another agent object that receives the data on node n1.

#Create a UDP agent and attach it to node n0

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0

These lines create a UDP agent and attach it to the node n0, then attach a CBR traffic generator to the UDP agent. CBR stands for 'constant bit rate'. Line 7 and 8 should be self-explaining. The packet Size is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second). The next lines create a Null agent which acts as traffic sink and attach it to node n1.

set null0 [new Agent/Null]

$ns attach-agent $n1 $null0

Now the two agents have to be connected with each other.

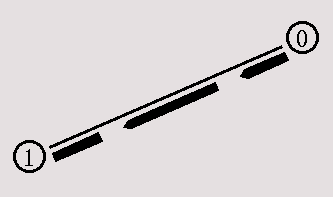
$ns connect $udp0 $null0

And now we have to tell the CBR agent when to send data and when to stop sending. It's probably best to put the following lines just before the line '$ns at 5.0 "finish"'.

$ns at 0.5 "$cbr0 start"

$ns at 4.5 "$cbr0 stop"

Now you can save the file and start the simulation again. When you click on the 'play' button in the nam window, you will see that after 0.5 simulation seconds, node 0 starts sending data packets to node 1. You might want to slow nam down then with the 'Step' slider.



Add the following two lines to your CBR agent definitions.

$udp0 set class\_ 1

$udp1 set class\_ 2

The parameter 'fid\_' stands for 'flow id'.

Now you can add the following piece of code to your Tcl script, preferably at the beginning after the simulator object has been created, since this is a part of the simulator setup.

$ns color 1 Blue

$ns color 2 Red

This code allows you to set different colors for each flow id.

You can add the following line to your code to monitor the queue for the link from n2 to n3.

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

You can see the packets in the queue now, and after a while you can even see how the packets are being dropped, though (at least on my system, I guess it might be different in later or earlier releases) only blue packets are being dropped. But you can't really expect too much 'fairness' from a simple Drop Tail queue. So let's try to improve the queuing by using a SFQ (stochastic fair queuing) queue for the link from n2 to n3. Change the link definition for the link between n2 and n3 to the following line.

$ns duplex-link $n3 $n2 1Mb 10ms SFQ

The queuing should be 'fair' now. The same amount of blue and red packets should be dropped.

**Experiment No. 1**

**Problem Statement:**

Simulate a three nodes point-to-point network with duplex links between them. Set the queue size vary the bandwidth and find the number of packets dropped.

set ns [new Simulator]

#Open a new file for NAMTRACE

set nf [open out.nam w]

$ns namtrace-all $nf

#Open a new file to log TRACE

set tf [open out.tr w]

$ns trace-all $tf

#Body of the finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

close $nf

close $tf

exec nam out.nam &

exit 0

}

#Create Nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

#Create Links between Nodes

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

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

#Set the queue limit - default is 50 packets

$ns queue-limit $n0 $n1 50

$ns queue-limit $n1 $n2 50

#Create Transport Agent

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

set null0 [new Agent/Null]

$ns attach-agent $n2 $null0

$ns connect $udp0 $null0

#Create Application to generate traffic

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0

#Start and Stop generating traffic

$ns at 0.5 "$cbr0 start"

$ns at 4.5 "$cbr0 stop"

#Stop the simulation

$ns at 5.0 "finish"

#Run the simulation

$ns run

**Nam output:**

**A black line with a white circle and a white circle

AI-generated content may be incorrect.**