National University of Ireland, Maynooth
Maynooth, Co. Kildare, Ireland.

Department of Computer Science
Technical Report Series

An Introduction to NS, Nam and OTcl scripting

Paul Meeneghan and Declan Delaney

NUIM-CS-TR-2004-05

National University of Ireland, Maynooth
Maynooth, Co. Kildare, Ireland

DEPARTMENT OF COMPUTER SCIENCE

TECHNICAL REPORT SERIES

NUIM-CS-TR-2004-05

An Introduction to NS, Nam and OTcl scripting

Paul Meeneghan & Declan Delaney

April 2004

Contents Page

Chapter 1: Overview of NS-2

1.1 Introduction to NS-2

This report deals with Network Simulator Version 2, also known as NS-2 [20]. NS-2 is an event driven packet level network simulator developed as part of the VINT project (Virtual Internet Testbed)[1]. 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) [2]. It is an open source software package available for both Windows 32[3] and Linux [4] platforms.

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

This report will discuss in chapter one how to install and operate NS-2 on a Windows 32 and Linux operating system. Chapter 2 explains the architecture of NS-2. Chapter 3 discusses the software tools that are used with NS-2, such as NAM [5, 27], a Network ANimation and visualization tool. An introduction to OTcl coding is given in chapter 4. OTcl is the language that is used to write network topologies for simulation on NS-2. Chapter 5 concludes this report with three samples OTcl scripts describing network protocols and agents discussed in chapter 4.

1.2 Downloading and Installing NS-2 and NAM

Installing on Win 32 platform

The Windows 32 version of NS-2 is around 50 MB in size. The Windows 2000 version of this software is available to download (as of April 2004 at www.isi.edu/nsnam/dist/). Microsoft Visual Studio version 5.0 [6] is required.

The disk contains the following files

1. Tcl8.3.2.tar.gz
2. tk8.3.2.tar.gz
3. ns-2.1b9a-win32.exe
4. nam-1.0a11a-win32.exe
5. compile_dll_D_Win2000_1.00.zip

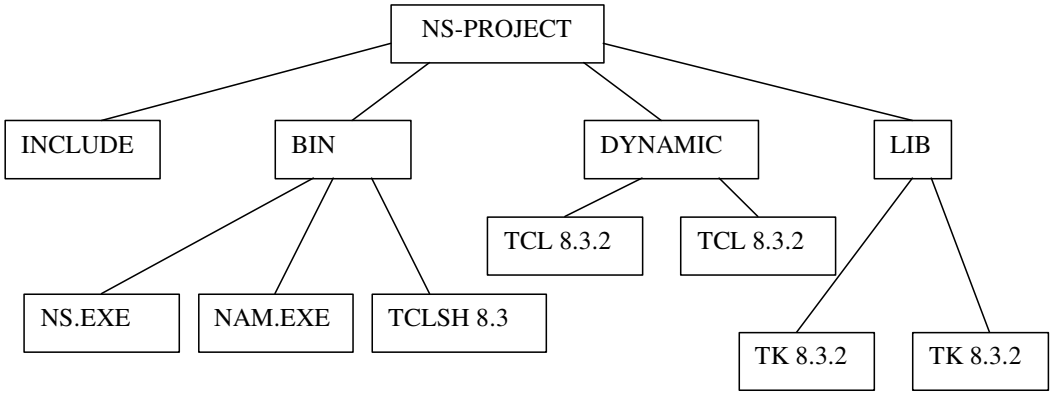

Figure 1: Directory structure of a NS-2 installation on the Windows 32 platform

The following steps are required for successful completion of this installation

1. Create a folder and copy the above files into it, e.g. "C:\PROJECT"

2. Create two subfolders under the folder created in step 1 and use WinZip [7], WinRar [8] or some other decompression software to decompress the archives into the them according to the following table:

Subfolder	**Archives to decompress in it**
C:\PROJECT\DYNAMIC	Tcl8.3.2.tar.gz tk8.3.2.tar.gz
C:\PROJECT\TEMP	compile_dll_D_Win2000_1.00.zip
C:\PROJECT\BIN	ns-2.1b9a-win32.exe nam-1.0a11a-win32.exe

3. Rename ns-2.1b9a-win32.exe and nam-1.0a11a-win32.exe to ns.exe and nam.exe respectively
4. Find the file vcvars32.bat on your system (somewhere in the C++ installation folders). Copy this file to "C:\PROJECT\TEMP". Edit the file using a text editor to add/modify and, if necessary, delete the quotes from the following lines:

   set VSCommonDir = Root of Visual Developer Studio Common files

   e.g. set VSCommonDir=C:\PROGRA~1\MICROS~3\Common

   set MSDevDir = Root of Visual Developer Studio installed files

   e..g. set MSDevDir=C:\PROGRA~1\MICROS~3\Common\msdev98

   set MSVCDirDir = Root of Visual C++ installed files

   e.g. set MSVCDir=C:\PROGRA~1\MICROS~3\VC98

5. Open a command prompt shell and type in the following commands (in bold).

   C:\>**cd C\PROJECT\TEMP** (or equivalent directory)

   C:\PROJECT\TEMP>**vcvars32.bat**

   C:\PROJECT\TEMP>**compile-dll-win32.bat c:\PROJECT**

6. Finally, add the line "C:\PROJECT\BIN" to the environmental variable PATH (right click 'my computer' on the desktop and go to properties).

Installing under Unix

Under Linux or Unix the following components can be installed. (Note that installation and compilation under Cygwin is not recommended).

- Tcl release 8.4.5 (required component)
- Tk release 8.4.5 (required component)
- OTcl release 1.8 (required component)
- TclCL release 1.15 (required component)
- Ns release 2.27 (required component)
- Nam release 1.10 (optional component)
- Xgraph version 12 (optional component)
- CWeb version 3.4g (optional component)
- SGB version 1.0 (?) (optional component, builds sgblib for all UNIX type platforms)
- Gt-itm gt-itm and sgb2ns 1.1 (optional component)

- Zlib version 1.1.4 (optional, but required should Nam be used)

These files can all be obtained as a single package called *ns-allinone* (see: http://www.isi.edu/nsnam/ns/ns-build.html#pieces).

Once the source code for the various components has been obtained, unpack the OTcl, TclCL and ns sources into the same top level directory and build them using the following commands:

- cd into the OTcl directory
- run *./configure*
- run *make*
- cd into the TclCL directory
- run *./configure*
- run *make*
- cd into the ns directory
- run *./configure*
- run *make*

Figure 2 shows the directory structure of NS-2/NAM on the Linux platform.

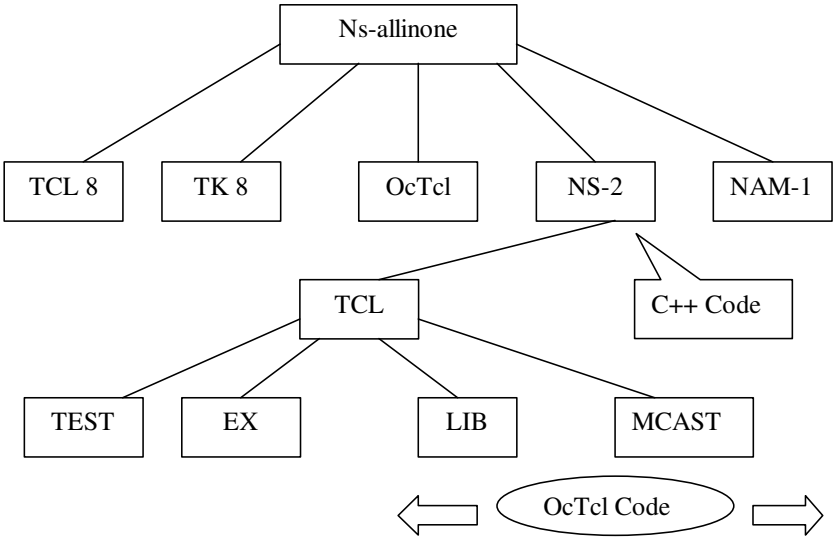

Figure 2: Directory structure of a NS-2 installation on the Linux platform

1.3 Running NAM/NS-2

To run NS-2 in Windows

Change directory to the location where ns is installed (bin directory) e.g. "C:\PROJECT\BIN." To run the ns executable type 'ns <Tclscript>'
Alternatively to run NS-2 through the Tcl interpreter Wish83.exe type 'ns <Tclscript>' where Tclscript is the name of the Tcl script written i.e. file.Tcl

To run NS-2 in Linux

Ensure that the location of *ns* is added to the environmental variable PATH and then type: 'ns <Tclscript>'

To run NAM

Change directory to the directory where ns is installed (bin directory) e.g. "C:\PROJECT\BIN". To run the NAM executable type 'nam <tracefile>' where tracefile is the name of the trace file produced i.e. file.tr

Chapter 2 Architecture of NS-2

2.1 Design of NS-2

As shown in the simplified user's view of Figure 3, 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.

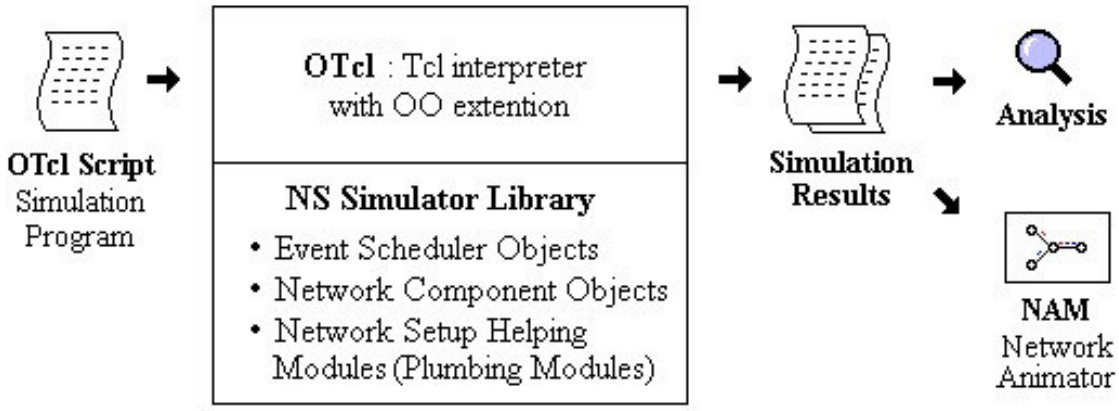

Figure 3. Simplified User's View of NS-2 [9]

To use NS-2, a user programs in the OTcl script language.

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.

A user can add OTcl modules to NS-2 by writing a new object class in OTcl. These then have to be compiled together with the original source code.

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:

- Organises 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 [10] or TraceGraph [11].

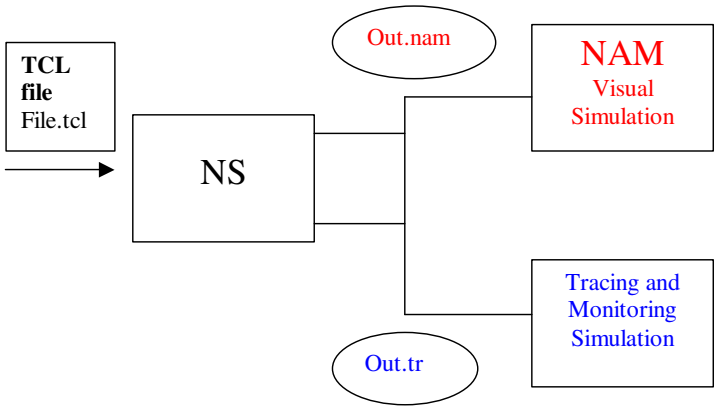

Figure 4: Flow of events for a Tcl file run in NS

2.2 C++/OTcl linkage

NS–2 is written in C++ with OTcl interpreter as a front end. For efficiency reason, NS separates the data path implementation from control path implementations.

What Languages are used with NS-2?

- Split-Language programming is used
    1. Scripting Language (Tcl - Tool Command Language and pronounced 'tickle')
    2. System Programming Language (C/C++)
- Ns is a Tcl interpreter to run Tcl Scripts

- By using C++/OTcl, the network simulator is completely Object-oriented.

In terms of lines of source code, NS-2 was written with 100k lines of C++ code, 70k lines of Tcl code and 20k of documentation.

The TCL interpreter:

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

The following is written and compiled with C+

- Event Scheduler
- Basic network component objects

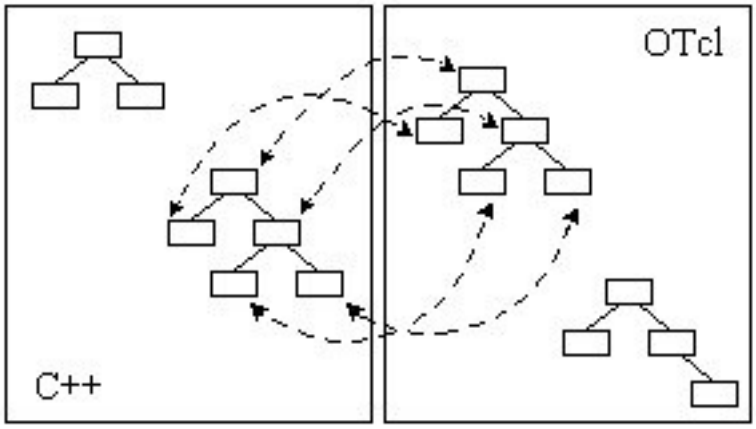

Figure 5. C++ and OTcl: The Duality [9]

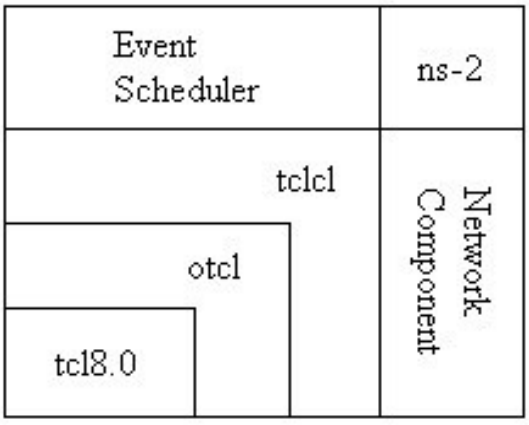

Figure 6. Architectural View of NS [9]

These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as

member functions and member variables of the corresponding OTcl object. It is also possible to add member functions and variables to a C++ linked OTcl object.

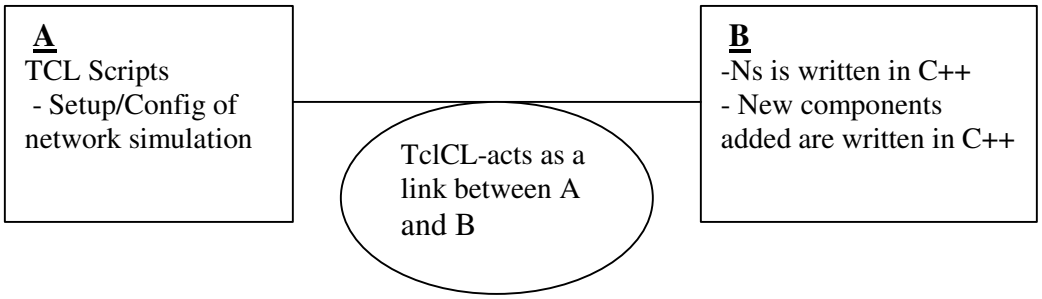

Figure 7: TclCL provides the linkage between C++ and OTCL

2.3 Characteristics of NS-2

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

Chapter 3 Software Tools used with NS-2

3.1 NAM

NAM [5, 27] provides a visual interpretation of the network topology created. The application was developed as part of the VINT project. Its features are as follows. Figure 8 displays the NAM application and its components.

- 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.

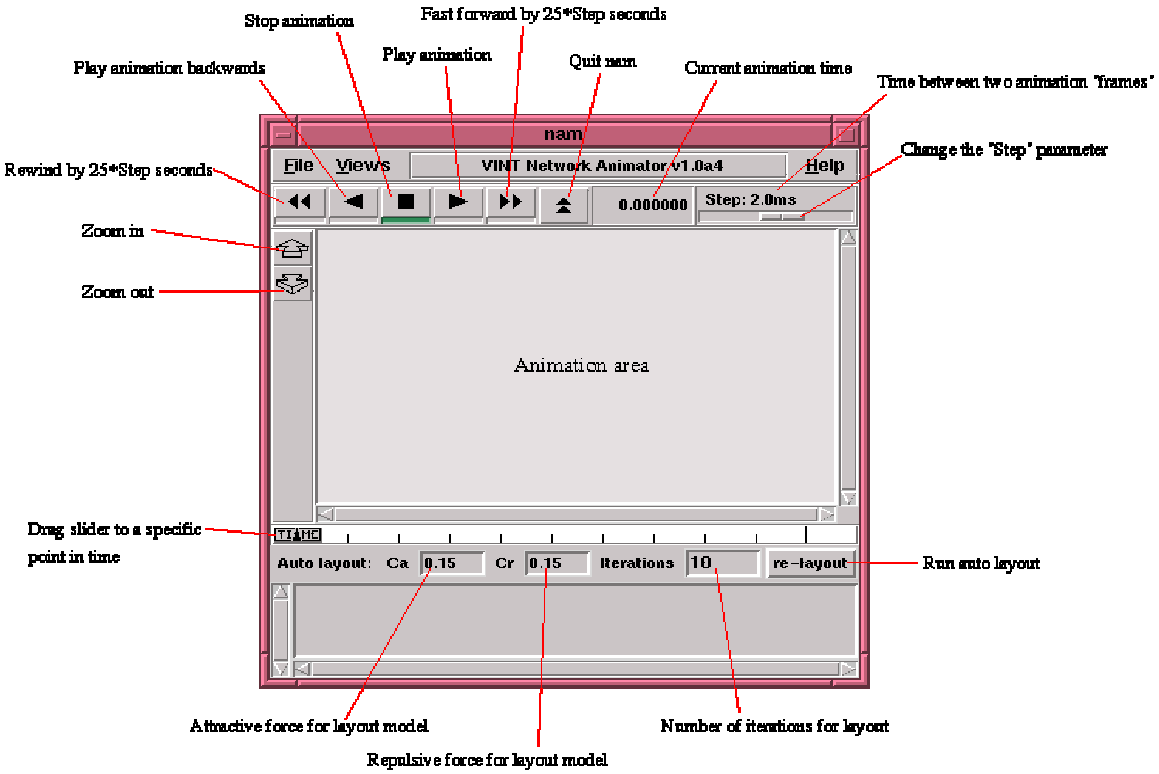

Figure 8:NAM tool description

3.2 Nscript

- A Graphical User Interface for building ns-TCL scripts
- The topology is built by drawing it. Nodes/Agents can be added with drag-and-drop to the edit screen. The TCL code is automatically created in the Tcl/Tk Script screen.
- Nscript is written in Java [13].

Uses of nscript:

1. Create complete topologies by adding nodes and links
2. Add and create transport agents i.e. UDP, TCP
3. Schedule simulation events i.e. sending/queuing packets
4. Create user defined libraries i.e. PING agent
5. The scripts created can be exported and run in NS.

Downloads:

1. Nscript-103 is available at [12]

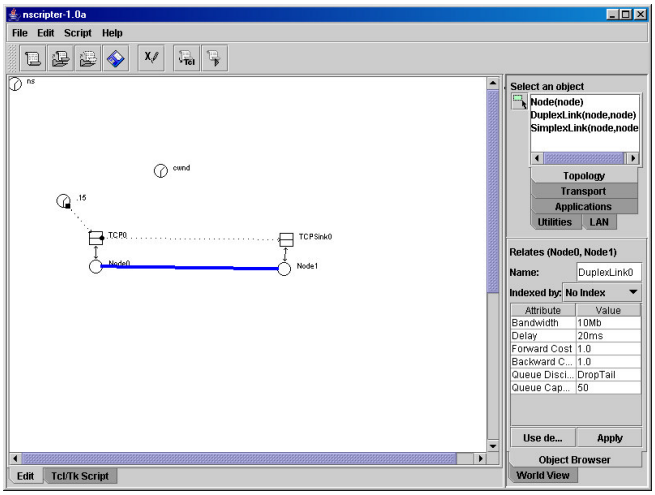

Figure 9:Nscript application

3.3 Topology Generators

Topology Generators [24,26] are used with NS-2 to create a network topology to simulate a certain network model. Each topology generator provides a Graphical User Interface.

The user can then choose the structure of the topology e.g. number of nodes. When this is complete the generator can be run to produce TCL code depicting the topology for use with NS-2. The four most common topology generators are as follows.

GT-ITM [13]

This generator focuses on reproducing the hierarchical structure of the topology of the Internet based on the TS (Transit Stub)

The steps are as follows

- A connected random graph is first generated (using the Waxman model)
- Each node in the graph represents an Entire Transit Domain.
- For each node in the Transit domain- a number of random graphs are generated representing a Stub Domain that are attached to that node.

Tiers [14]

This is based on a three level hierarchy aimed at reproducing the differentiation between WAN, MAN and LAN compromising the Internet.

Brite [15]

This is a single generation model providing several degrees of freedom with respect to how nodes are placed in the plane. The properties of an interconnection method are used.

Inet [16]

This generator initially assumes node degrees from a power law distribution. The steps of this generator are as follows

- Forms a spanning tree using nodes of degree greater than two.
- Attaches nodes with degree one to the spanning tree.
- Matches the remaining unfulfilled degrees of all nodes with each other.

3.4 Trace Data Analyzers

This section describes XGraph and TraceGraph, two applications used to analyse trace files produces from a simulation

XGraph [10]

. 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

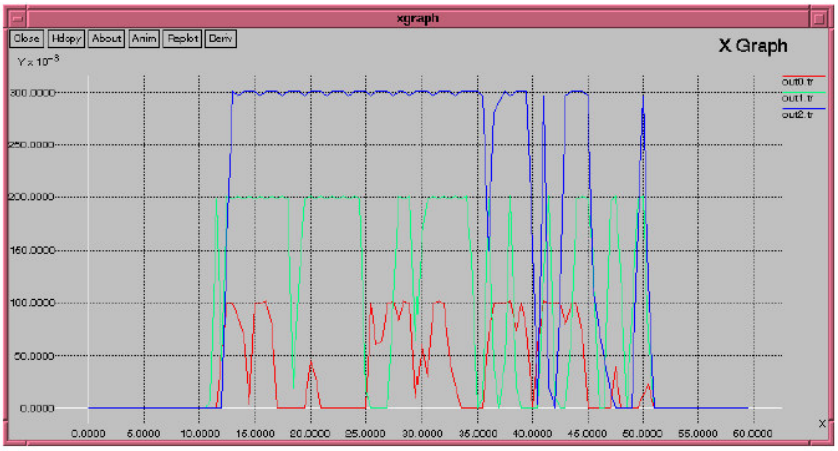

Figure 10: XGraph running comparing three trace files in a graph

To call XGraph within an nscript, the line indicated in **bold** in Figure 11 is used. It is worth noting that Figure 11 defines a procedure called finish(). In this case the trace file out0.tr is used. Multiple trace files can be taken is as arguments. This will produce a graph of size 800x400 displaying information on the traffic flow and time (Figure 10).

```
proc finish {} {
global f0 f1 f2
#Close the output files
close $f0
close $f1
close $f2
#Call XGraph to display the
results
exec xgraph out0.tr -geometry
800x400 &
exit 0
}
```

Figure 11: XGraph called within a OTcl script

TraceGraph

TraceGraph [11] is a trace file analyser that runs under Windows, Linux and UNIX systems and requires Matlab 6.0 [18] or higher. Two sample plots are shown in Figure 12.

TraceGraph supports the following trace file formats.

- Wired
- Satellite
- Wireless (old and new trace)

Version 2.02 has the following features/options

- 238 2D graphs
- 12 3D graphs
- Delays, jitter, processing times, Round Trip Times, number of intermediate nodes, throughput graphs and statistics
- The whole network, link and node graphs and statistics
- All the results can be saved to text files, graphs can also be saved as jpeg and tiff
- x, y, z axes information: minimum, mean, maximum, standard deviation, median

- Any graph saved in text file with 2 or 3 columns can be plotted
- Script files processing to do the analysis automatically.

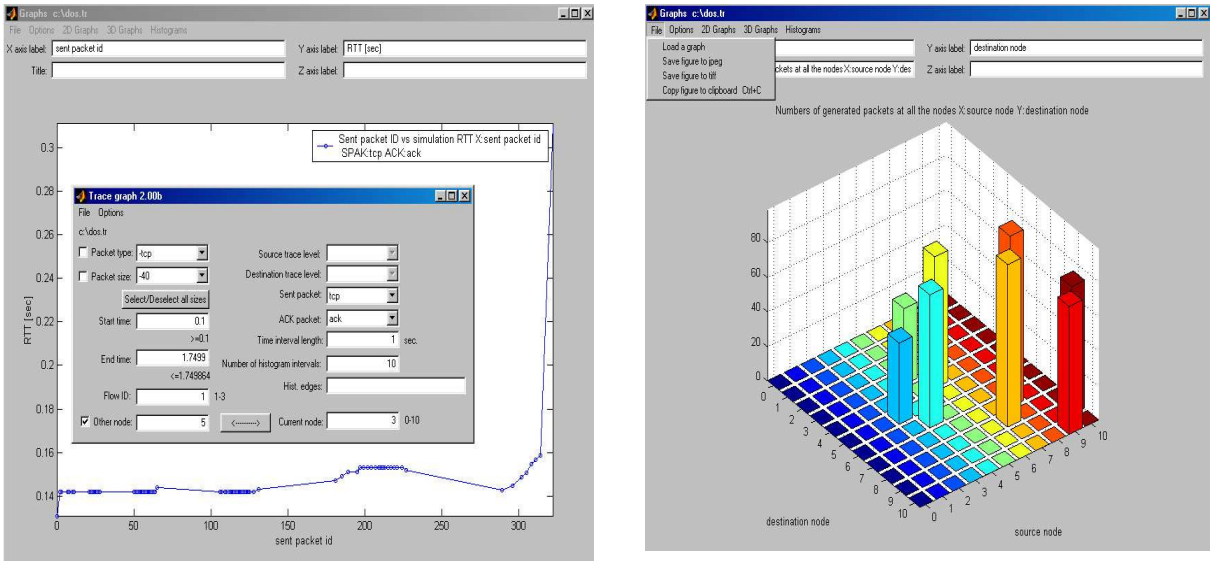

Figure 12: TraceGraph analysing a trace file to produce numerous graphs

Chapter 4 OTcl Scripting with NS-2 [19,21,23,29]

4.1 Node Creation

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 is illustrated in Figure 13.

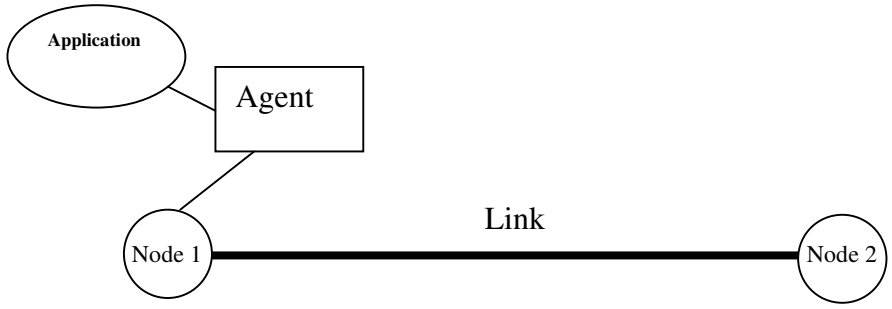

Figure 13: NS-2 is very structured. This diagram shows two nodes, a link, an agent and an application.

The overall structure of an OTcl script is as follows

Event scheduling

- **Create scheduler ->** `set ns [new Simulator]`
- **Schedule Event -> ** `$ns at <time> <event>`
- **Start scheduler -> ** `$ns run`

To create a node the simulator node object is used. The following two lines create two node objects and assigns them the handles “n0” and “n1” respectively using the command ‘set.’

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

The following creates 5 nodes, with handles n0-n4

```
   for {set i 0} {$i<5} {incr i} {
               Set n($i) [$ns node]
}
```

To set the colour of a node, the following code is used.

```
$n0 color <colour>
```

where <colour> is black, re, blue, seaGreen.

4. 2 Node links

A unidirectional link between the two nodes is created as follows

- **A Simplex link (one way) ->** `$ns simplex-link $n0 $n1 <bandwidth> <delay> <queue_type>`

A bi-directional link between the two nodes is created as follows

- **A duplex link (both ways) ->**`$ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>`

Note: In this example the link is between node 0 and node 1. Sample values for bandwidth and delay could be 1Mb and 10ms respectively.

NS-2 supports numerous queue types including FIFO, RED (Random Early Detection), Drop Tail, FQ (Fair Queuing), SFO(Stochastic)

4.3 Network Agents

Traffic generation in NS-2 is based on the objects of two classes, the class Agent and the class Application. Every node in the network that needs to send or receive traffic has to

have an agent attached to it. On top of an agent runs an application. The application determines the kind of traffic that is simulated.

There are two types of agents in NS-2: UDP and TCP agents

- UDP

```
set udp0 [new Agent/UDP]
set null [new Agent/NULL]

$ns attach-agent $n0 $udp0    # attach the udp0 agent
#to node 0
$ns attach-agent $n1 $null

$ns connect $udp $null        # connect the 2 agents
```

This code first creates a UDP agent and attaches it to n0 using the attach-agent procedure. It then creates a Null agent, which will act as a traffic sink and attach it to n1. The two agents are connected using the simulator method connect.

To add a Loss Monitor to the agent the following OTcl code is used. The Agent/LossMonitor can monitor number of packets transferred, as well as packets lost. A procedure can be scheduled to poll the LossMonitor every *T* seconds and obtain throughput information.

- `set lossMonitor [new Agent/LossMonitor]`
- `$ns_ connect $udp0 $lossMonitor`

- TCP

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

```
$ns attach-agent $n0 $tcp          # attach the tcp
agent to node 0
$ns attach-agent $n1 $tcp_sink

$ns connect $tcp $tcp_sink    # connect the 2 agents
```

This code first creates a TCP agent and attaches it to the tcp node using the attach-agent procedure. It then creates a TCPSink agent, which will act as a TCP sink, and attaches it to the node tcp_sink. The two agents are connected using the simulator method connect. The following types of TCP are available in NS-2:
TCP, TCP/Reno, TCP/Vegas, TCP/Sack1, TCP/Fack, TCPSink.

4.4 Traffic Applications

This section will discuss four traffic applications that go on top of a UDP agent to simulate network traffic.

CBR (Constant Bit Rate)

A CBR traffic object generates traffic according to a deterministic rate. Packets are a constant size. The OTcl code to implement a CBR traffic source in a simulation is as follows:

- `set my_cbr [new Application/Traffic/CBR]`
- `$my_cbr attach-agent $udp`
- `$ns at <time> "$my_cbr start"`

Parameters:

- `start`: starts sending packets according to the configuration parameters
- `stop`: stops sending packets

Configuration parameters:

- `PacketSize_`: constant size of packets generated e.g 48

- `rate_`: sending rate e.g. 64kb
- `interval_`: (optional) interval time between packets e.g 0.05
- `random_`: Flag to introduce noise in the departure times, default is off, 1 for on
- `maxpkts_`: the maximum number of packets to send e.g 1000

Exponential

Traffic is determined by an exponential distribution. Packets are a constant size. This produces an on/off distribution. Packets are sent at a fixed rate during on periods. No packets are sent during off periods.

TCL code to implement a CBR traffic source in a simulation:

- `set my_exp [new Application/Traffic/Exponential]`

Configuration parameters

- `PacketSize_`: constant size of packets generated e.g 210
- `burst_time_`: average on time for the generator e.g. 500ms
- `idle_time_`: average off time for the generator e.g 500ms
- `rate_`: sending rate during the "on" time e.g. 100k

Pareto

The distribution for traffic generation is taken from a pareto on/off distribution. This is generally used to generate aggregate traffic that exhibits long range dependency. The following is OTcl code to implement a Pareto traffic source in a simulation. Idle times are taken from a pareto distribution.

- `set my_pareto [new Application/Traffic/Pareto]`

Configuration parameters

- `PacketSize_`: constant size of packets generated e.g. 210
- `burst_time_`: average on time for the generator e.g. 500ms
- `idle_time_`: average off time for the generator e.g. 500ms
- `rate_`: sending rate during the "on" time e.g. 100k

- `shape_`: the shape parameter used by the pareto distribution e.g. 1.5

TrafficTrace

Traffic is generated according to a trace file. The binary file must contain 2 x 32 fields in network (big-endian) byte order. The first field contains the time in ms until next packet is generated. The second field contains the length in bytes of the next packet. The method filename of the Tracefile class associates a trace file with the Tracefile object.
The following is OTcl code to implement a Trace file traffic source in a simulation:

- `set t_file [new Tracefile]`
- `$t_file filename <file>`
- `set src [ new Application/Traffic/Trace]`
- `$src attach-tracefile $t_file`

where $t_file is a binary file and the two fields in the file contain inter-packets times in milliseconds and packet size in bytes.

4.5 Telnet, FTP, HTTP

Two simulation applications exist to send traffic on top of a TCP object: Application/FTP and Application/Telnet

File Transfer Protocol (FTP- for simulating bulk data transfer)

OTCL Code for using FTP in a simulation:

- `set ftp [new Application/FTP]`
- `$ftp attach-agent $tcp`
- `$ns at <time> "$ftp start"`

Parameters

- `attach-agent`: attach-agent: attaches an Application/FTP agent to an agent

- `start:` start the Application/FTP to send data
- `stop:` stop sending data
- `produce n:` where n is the counter of packets to be sent
- `producemore n:` where n is the new increased value of packets to be sent
- `send n:` similar to producemore, but sends n bytes instead of packets

Telnet

OTcl code for using Telnet in a simulation:

- `set telnet [new Application/Telnet]`
- `$telnet attach-agent $tcp`

Parameters

- `start;` start producing packets
- `stop:` stop producing packets
- `attach-agent:` attaches a Telnet object to an agent

Configuration Parameters

- `interval_:` The average inter-arrival time in seconds for packets generated by the Telnet object
  if(interval_ == 0) Inter arrival times taken from the tcplib distribution.
  if(interval_ != 0) Inter arrival times taken from the exponential distribution, average is set to what interval_ is.

HTTP

The following is OTcl code for implementing HTTP(server and client) in a simulation.

- Application HTTP
  –Client node:
  `set client [new HTTP/Client $ns $node0]`
  `$client connect $server`

–Server node:

```
set server [new HTTP/Server $ns $node1]
$server set-page-generator $pgp
```

4.6 Ping

The following OTcl code is added to a simulation script to create the Ping agent:

```
#Define a 'recv' function for the class 'Agent/Ping'
Agent/Ping instproc recv {from rtt} {
        $self instvar node_
        puts "node [$node_ id] received ping answer from \
              $from with round-trip-time $rtt ms."
}
```

To ping a node from another node, ping agents must be set up on both nodes

```
set PingAgent1 [new Agent/Ping]
$ns attach-agent $node0 $ PingAgent1
set PingAgent2 [new Agent/Ping]
$ns attach-agent $node1 $ PingAgent2
```

The two ping agents are connected as follows

```
$ns connect $pB $p3
```

To ping a node the following code is used

```
$ns at 0.1 "$PingAgent1 send"
```

The following could be added to draw a square box around the pinging node in NAM

```
$ns at 0.1 "$node0 add-mark m0 red box"
```

The following could be added to print out to the screen in NAM

```
$ns at 0.1 "$ns trace-annotate \"Pinging node 1 from node 0
""
```

4.7 Tracing

A Trace file contains all information needed for animation purposes- both on a static network layout an on dynamic events such as packet arrivals, departures, drops and link failures.

Tracing in NS-2 is implemented with the following OTcl code.

To Trace packets on all links

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

An example of a standard trace file in NS-2 follows and its format is shown in Figure 14:

```
+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
- 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600
r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602
+ 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
- 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
r 1.84609 0 2 cbr 210 ------- 0 0.0 3.1 225 610
+ 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
d 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
- 1.8461 2 3 cbr 210 ------- 0 0.0 3.1 192 511
r 1.84612 3 2 cbr 210 ------- 1 3.0 1.0 196 603
+ 1.84612 2 1 cbr 210 ------- 1 3.0 1.0 196 603
- 1.84612 2 1 cbr 210 ------- 1 3.0 1.0 196 603
+ 1.84625 3 2 cbr 210 ------- 1 3.0 1.0 199 612
```

This trace file exert contains five enqueue operations(‘+’), four dequeue operations(‘-’), four receive events (‘r’) and one drop event(‘d’). The columns

1	Operation performed in the simulation
2	Simulation time of event occurrence
3	Node 1 of what is being traced
4	Node 2 of what is being traced
5	Packet type
6	Packet size
7	Flags
8	IP flow identifier
9	Packet source node address
10	Packet destination node address
11	Sequence number
12	Unique packet identifier

Figure 14:Trace file data explanation

Depending on the simulation, different trace file formats are produced. A full list of these formats can be found at http://www.cs.toronto.edu/~wbiao/ns2/traceformats/

To trace a specific link

- `ns trace-queue $node0 $node1 $trace_file`

To trace up variable tracing in NS-2

- `set cwnd_chan_ [open all.cwnd w]`
- `$tcp trace cwnd_ # tcp tracing its own variable cwnd_chan_`
- `$tcp attach $ cwnd_chan_`

The variable sstthresh of $tcp is traced by a generic $tracer

- `Set tracer [new Trace/Var]`
- `$tcp trace ssthresh_ $tracer`

4.8 Routing and Network dynamics

NS-2 implements three different routing strategies: static routing, session routing and DV routing. To specify the routing strategy used the rtproto method in the class Simulator uis used.

- `$ns rttproto Static`
- `$ns rttproto Session`
- `$ns rttproto DV`

The rtmodel-at provides the facility to dynamically bring links down or bring them up.

- `$ns rtmodel-at 1.0 down $node1 $node2`
- `$ns rtmodel-at 2.0 up $node1 $node2`

This model can be extended using an exponential distribution for manipulating links.

- `$ns rtmodel Exponential 0.7 2.0 2.0 down $node1 $node2`

More information on the above Tcl scripting can be found on the NS-2 documentation page [20] and the NS-Manual [19].

Chapter 5 NS-2 OTcl Sample Scripts

This chapter presents three sample Tcl scripts. These scripts should be entered using a standard text editor and saved with the extension **tcl**. If ns is installed correctly and entered in the environmental PATH variable, it is only necessary to open a command console, navigate to the directory containing the Tcl script and type:

ns scriptname.tcl

This will generate a trace file according to the instructions included in the Tcl script. This trace file can then be analysed and inspected using a visualisation or analysis tool such as Nam or Tracegraph.

5.1 OTcl Sample Script 1

This OTcl Script does the following

- Creates 2 nodes, add a duplex link between the two with relevant parameters.
- Adds a UDP agent and UDPSink agent to nodes 0 and 1 respectively.
- Adds a CBR traffic application to the UDP agent.
- Connects the agents and run the simulation for 5 seconds.
- Calls NAM to animate this topology (Figure 15).

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

# Open the nam trace file, associated with nf, log
everything as nam output in nf
set nf [open out.nam w]
$ns namtrace-all $nf

# Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the trace file
        close $nf
        #Execute nam on the trace file
```

```
        exec nam out.nam &
        exit 0
}

#Create two nodes
set n0 [$ns node]
set n1 [$ns node]

#Create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

#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_ 0.1
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node
n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0


# Schedule events for the CBR agent
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation
time
$ns at 5.0 "finish"

# Run the simulation
$ns run
```

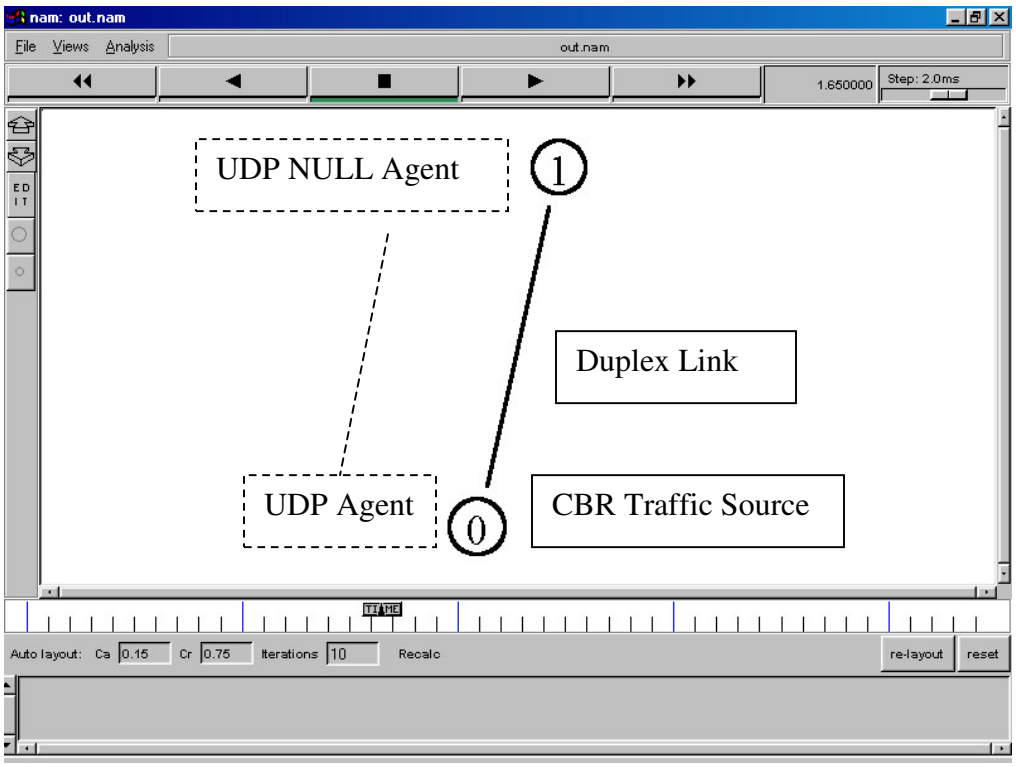

Figure 15: NAM output of OTcl script 1

5.2 OTcl Sample Script 2

This script describes four types of internet traffic. Four functions/procedures are displayed describing the types of traffic generators that are available. Two nodes are created for this purpose.

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

#Open a 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 $nf
        exec nam out.nam &
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]

#Connect the nodes with two links
$ns duplex-link $n0 $n1 1Mb 10ms DropTail


proc www_traffic { node0 node1 } {
     global ns
     set www_UDP_agent [new Agent/UDP]
     set www_UDP_sink [new Agent/Null]

     $ns attach-agent $node0 $www_UDP_agent
     $ns attach-agent $node1 $www_UDP_sink

     $ns connect $www_UDP_agent $www_UDP_sink

     set www_CBR_source  [new Application/Traffic/CBR]
     $www_CBR_source  attach-agent $www_UDP_agent
     $www_CBR_source  set packetSize_ 48
     $www_CBR_source  set interval_ 5000ms
     $ns at 0.0 "$www_CBR_source start"
```

```
}

proc smtp_traffic {node0 node1 } {
     global ns
     set smtp_UDP_agent [new Agent/UDP]
     set smtp_UDP_sink [new Agent/UDP]

     $ns attach-agent $node0 $smtp_UDP_agent
     $ns attach-agent $node1 $smtp_UDP_sink

     $ns connect $smtp_UDP_agent $smtp_UDP_sink

     set smtp_UDP_source  [new
Application/Traffic/Exponential]
     $smtp_UDP_source  attach-agent $smtp_UDP_agent
     $smtp_UDP_source  set packetSize_ 210
     $smtp_UDP_source  set burst_time_ 50ms
     $smtp_UDP_source  set idle_time_ 50ms
     $smtp_UDP_source  set rate_ 100k
     $ns at 0.0 "$smtp_UDP_source start"
}


proc ftp_traffic {node0 node1 } {
     global ns
     set ftp_TCP_agent [new Agent/TCP]
     set ftp_TCP_sink [new Agent/TCPSink]

     $ns attach-agent $node0 $ftp_TCP_agent
     $ns attach-agent $node1 $ftp_TCP_sink

     $ns connect $ftp_TCP_agent $ftp_TCP_sink

     set ftp_FTP_source  [new Application/FTP]
     $ftp_FTP_source  attach-agent $ftp_TCP_agent
     #$ns at 0.0 "$ftp_FTP_source start"
     #$ns at 0.1 "$ftp_FTP_source send 1"
     $ns at 0.1 "$ftp_FTP_source produce 3"
     $ns at 0.9 "$ftp_FTP_source stop"
     $ns at 1.2 "$ftp_FTP_source producemore 6"
     #$ns at 2.0 "finish"
}


proc telnet_traffic {node0 node1 } {
     global ns
     set telnet_TCP_agent [new Agent/TCP]
```

```
     set telnet_TCP_sink [new Agent/TCPSink]

     $ns attach-agent $node0 $telnet_TCP_agent
     $ns attach-agent $node1 $telnet_TCP_sink

     $ns connect $telnet_TCP_agent $telnet_TCP_sink

     set telnet_TELNET_source  [new Application/Telnet]
     $telnet_TELNET_source attach-agent $telnet_TCP_agent
     $telnet_TELNET_source set interval_ 20
     $ns at 0.0 "$telnet_TELNET_source start"
     $ns at 0.8 "$telnet_TELNET_source stop"
}


proc generate_traffic {node0 node1 } {
    puts "function calls in here, Choose one at a time per
simulation to use"
    #smtp_traffic $node0 $node1
    #telnet_traffic $node0 $node1
    #www_traffic $node0 $node1
    #ftp_traffic $node0 $node1
}


generate_traffic $n0 $n1

$ns at 2.0 "finish"

$ns run
```

5.3 OTcl Sample Script 3

This script does the following

- Creates six nodes, 2 with Ping Agents attached and 2 with UDP agents attached
- These nodes will send data to node 3 via node 2
- Due to the bandwidth of the node2/3 link a queue will be formed and packets will be dropped

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

 # Open a trace file
set nf [open simu.tr w]
$ns namtrace-all $nf

# Define different colors for data flows
$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 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]

# Pinging Nodes
set n4 [$ns node]
set n5 [$ns node]
```

```
set p0 [new Agent/Ping]
$ns attach-agent $n4 $p0

set p1 [new Agent/Ping]
$ns attach-agent $n5 $p1

set p2 [new Agent/Ping]
$ns attach-agent $n3 $p2

set p3 [new Agent/Ping]
$ns attach-agent $n3 $p3

#Connect the two agents
$ns connect $p0 $p2
#Connect the two agents
$ns connect $p1 $p3

$n2 color red
$n3 color blue
$n4 color SeaGreen
$n5 color SeaGreen


# Define a 'recv' function for the class 'Agent/Ping'
Agent/Ping instproc recv {from rtt} {
        $self instvar node_
        puts "node [$node_ id] received ping answer from \
              $from with round-trip-time $rtt ms."
}

#Create links between the nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
$ns duplex-link $n4 $n2 1Mb 10ms DropTail
$ns duplex-link $n5 $n2 1Mb 10ms DropTail

$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 the link between node 2 and node 3
$ns duplex-link-op $n2 $n3 queuePos 0.5

# Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 set class_ 1
```

```
$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

# Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1

# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#Create a Null agent (a traffic sink) and attach it to node
n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0
$ns connect $udp1 $null0

#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"

$ns at 0.6 "$p0 send"
$ns at 0.8 "$p0 send"
$ns at 0.9 "$p0 send"
$ns at 0.6 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"

#Call the finish proc. after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

References

[1] http://www.isi.edu/nsnam/vint/ Virtual Internetwork Testbed collaboration (Valid 26/04/04)

[2] http://www.Tcl.tk TCL homepage (Valid 26/04/04)

[3] http://www.micosoft.com Microsoft website (Valid 26/04/04)

[4] http://www.linux.org Linux homepage (Valid 26/04/04)

[5] http://www.isi.edu/nsnam/nam/ NAM Network Animator (Valid 26/04/04)

[6] http://msdn.microsoft.com/visualc/ Visual Studio C++ homepage (Valid 26/04/04)

[7] http://www.winzip.com/ WinZip homepage (Valid 26/04/04)

[8] http://www.rarlab.com/ Winrar homepage (Valid 26/04/04)

[9] http://www.isi.edu/nsnam/ns/ns-tutorial/ucb-tutorial.html 5th UCB/LBNL Network Simulator (NS): June 1999 Tutorial

[10] http://www.isi.edu/nsnam/xgraph/ XGraph homepage (Valid 26/04/04)

[11] http://www.geocities.com/tracegraph/ TraceGraph homepage (Valid 26/04/04)

[12] http://home.gwu.edu/~ecamposn/software.html Nscript NS-2 scripting tool (Valid 26/04/04)

[13] http://java.sun.com/ Java Sun Microsystems homepage (Valid 26/04/04)

[14] http://www.cc.gatech.edu/projects/gtitm/. GT-ITM topology generator. (Valid 26/04/04)

[15] http://www.isi.edu/nsnam/ns/ns-topogen.html#tiers TIERS. Tiers Topology Generator. (Valid 26/04/04)

[16] http://www.cs.bu.edu/brite/. BRITE topology generator. (Valid 26/04/04)

[17] http://topology.eecs.umich.edu/inet/. Inet Topology Generator(Valid 26/04/04)

[18] http://www.mathworks.com/ Matlab Mathworks homepage (Valid 26/04/04)

[19] http://www.isi.edu/nsnam/ns/ns-documentation.html NS Manual (Valid 26/04/04)

[20] http://www.isi.edu/nsnam/ns/ NS-2 website (Valid 26/04/04)

[21] http://www.ee.surrey.ac.uk/Personal/L.Wood/ns/slides/IEC-workshop-June-2000/IEC NS Workshop slides June 2000. (Valid 26/04/04)

[22] http://www.cs.toronto.edu/~wbiao/ns2/traceformats/index.html Trace file formats (Valid 26/04/04)

[23] http://www.isi.edu/nsnam/iTcl/doc/tutorial.html OTcl tutorial (Valid 26/04/04)

[24] http://www.cs.bu.edu/brite/user_manual/node3.html Available Topology Generators (Valid 26/04/04)

[25] Fall. K, Floyd S et al *Advances in Network Simulation, 2000 IEEE Research feature*

[26] O. Heckmann, M Piringer, J. Schmitt, R. Steinmetz *Generating Realistic ISP-Level Network Topologies, IEEE COMMUNICATIONS LETTERS, VOL. 7, NO. 7, JULY 2003*

[27] Estrin D, Handley M, Heidemann J, McCanne, Xu, Y, Haobo Yu, USC/Information Sciences Institute

Network Visualization with Nam, the VINT Network Animator November 2000

[28] http://www.isi.edu/nsnam/ns/tutorial/index.html Marc Greiss NS Tutorial (Valid 26/04/04)