GUI based NS-2 application in Linux

Akash I. Mecwan (akashnrec@yahoo.com, am_elect.it@nirmauni.ac.in)
Vijay G. Savani (savani1979@yahoo.com, vgs_ec.it@nirmauni.ac.in)
Lecturer in Electronics & communication Department
Institute of Technology, Nirma University

Abstract:This project is based on working with NS-2 (Network Simulator-2). The task is implemented in Linux .A detailed study of NS2 and making a user friendly GUI working in the front end and NS-2 working in the back ends in Linux. This project also has a comparison of Windows and Linux. NS-2 is a script, which is supported by Linux. It also works in Windows. But, for that, software called 'cygwin' working, as the shell is required. There are many tools in Linux for creating GUIs such as KDE GUI Development, Qt, Glade, Real Basic, Ruby etc. A GUI is created using Qt. An NS script (a .tcl file) is written in a text editor and saved with the extension mentioned above. The link between GUI and NS script is then created. So when a GUI button is clicked, at the back end, NS script runs and give the output. This GUI helps in learning NS2 easily.

1. Linux

Linux is an operating system that was initially created as a hobby by a young student, Linus Torvalds, at the University of Helsinki in Finland.

Apart from the fact that it's freely distributed, Linux's functionality, adaptability and robustness, has made it the main alternative for proprietary Unix and Microsoft operating systems.

1.1 Introduction

To have Linux in a system, it needs a system that's at the Pentium III/AMD Athlon class or betters with at least 256 MB of RAM. In order to install all the nice programs that will help you get your work done, you'll need at least 4 GB of space in your hard drive. You'll also need some additional space to store your files.

Versions (also known as *distributions*) of Linux, that are well suited for this, are: 1) Mandriva 2) SUSE 3) Linspire 4) Xandros 5) MEPIS 6) Fedora Core 7) Ubuntu

There are many different ways one can install Linux in PC: 1) Distribution or Version. A "distribution" is the compiled Linux source code, usually combined with extra features and software. 2) By downloading through the Internet. 3) Using CD-ROM from Linux retailers. 4) Purchasing a book that contains the CD Resource. However in some PCs and laptops, it is preinstalled.

1.2 Comparison between Linux and Windows

Comparison of Linux and Windows for different factors shows the following results:

1) **Price:** Majority of Linux versions is available on Internet for free, while Microsoft Windows versions are not that economical ($50.0- $150.0 US dollars per license copy).
2) **Ease:** It's very clear that when the point comes about ease, definitely it's Windows that's much easier than Linux.
3) **Reliability:** However, Microsoft has improved Windows versions a lot, Linux is still reliable.
4) **Hardware:** Windows is the OS that is being widely used worldwide and it is supported by the most of the hardware.
5) **Security:** When it comes to security point, Linux is safer than Windows.

2. NS-2 (Network Simulator-2)

A **Network Simulator** is a piece of software that predicts the behavior of a network, without an actual network being present. NS (version 2) is an object-oriented, discrete event driven network simulator written in C++ and OTcl.

It is also an open source like Linux and can run on both the platforms, Windows as well as Linux. However, in Windows, it needs a shell prompt (software called 'crygwin') like Linux for NS-2.

2.1 Introduction

NS is primarily useful for simulating local and wide area networks. Although NS is fairly easy to use once you get to know the simulator, it is quite difficult for a first time user, because there are few user-friendly manuals.

The purpose of this project is to give a new user some basic idea of how the simulator works, how to setup simulation networks, where to look for further information about network components in simulator codes, how to create new network components, etc., mainly by giving simple examples and brief explanation.

As shown in Figure 1, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object).

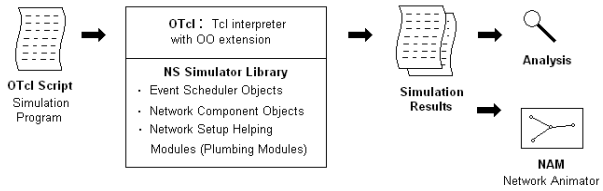

Figure 1. Simplified user's view of NS

In other words, to use NS, you program in OTcl script language. To setup and run a simulation network, a user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler.

The term "plumbing" is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the "neighbor" pointer of an object to the address of an appropriate object.

When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object.

This may sound like complicated job, but the plumbing OTcl modules actually make the job very easy. The power of NS comes from this plumbing. Now we are going to write a 'template' that you can use for all of the first Tcl scripts.

2.2 A tcl script for NS-2

You can write your Tcl scripts in any text editor like joe or emacs. Let's call this first example as 'example1.tcl'. 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
    }
```

You don't really have to understand all of the above code yet. It will get clearer to you once you see what the code does.

The next line tells the simulator object to execute the 'finish' procedure after 5.0 seconds of simulation time.

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

You probably understand what this line does just by looking at it. NS provides you with a very simple way to schedule events with the 'at' command. The last line finally starts the simulation.

```
$ ns run
```

You can actually save the file now and try to run it with 'ns example1.tcl'. You are going to get an error message like 'nam: empty trace file out.nam' though, because until now we haven't defined any objects (nodes, links, etc.) or events. Let's define a very simple topology with two nodes that are connected by a link. 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. So here we have made two nodes and connected them with the above command declaring their bandwidth and delay. This is a duplex link between the two nodes.

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 figure 2.

Of course, this example isn't very satisfying yet, since you can only look at the topology, but nothing actually happens, so the next step is to send some data from node n0 to node n1.

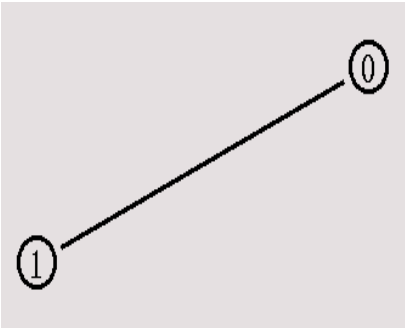

Figure 2. A connection between n0 and n1

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 n0s

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 packetSize 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. Note: 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”
```

This code should be self-explaining again. 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.

Another major component of NS beside 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.

In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event.

Network components communicate with one another passing packets, however this does not consume actual simulation time.

All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet.

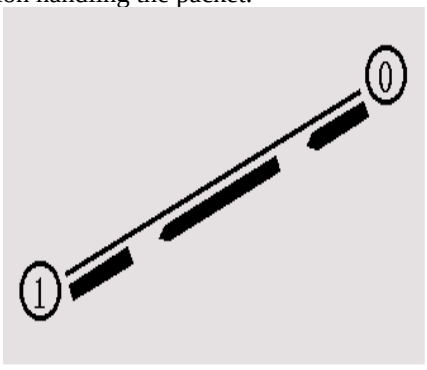

Figure 3. Data Transfer through n0 and n1

For example, a network switch component that simulates a switch with 20 microseconds of switching delay issues an event for a packet to be switched to the scheduler as an event 20 microseconds later.

The scheduler after 20 microseconds dequeues the event and fires it to the switch component, which then passes the packet to an appropriate output link component. Another use of an event scheduler is timer.

For example, TCP needs a timer to keep track of a packet transmission time out for retransmission (transmission of a packet with the same TCP packet number but different NS packet ID). Timers use event schedulers in a similar manner that delay does. The only difference is that timer measures a time value associated with a packet and does an appropriate action related to that packet after a certain time goes by, and does not simulate a delay.

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++.

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.

In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object. The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTcl. Likewise, an object (not in the data path) can be entirely implemented in OTcl.

3. NAM Editor

The NAM Window gives a graphical display of the code written in the tcl format. We 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 we can execute it directly out of the Tcl simulation script for the simulation which we want to visualize. The figure below gives somewhat idea of how the Nam window looks like.

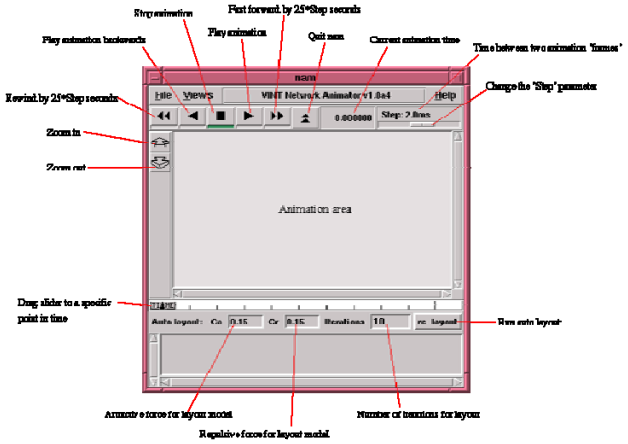

Figure 4. The Network Animator Window

The Nam Window has the option of recording the simulation, edit the nodes, zoom in and out of the view, relayout the nodes if they are formed without orientation and starting and stopping the simulation at any time. The analysis also can be done and help is given in it.

4. Making a GUI

Comparing all the softwares mentioned in the abstract Qt was the best among them and very user friendly.

4.1 Introduction to Qt

Qt (pronounced "cute") is a cross-platform application development framework, widely used for the development of GUI programs (in which case it is known as a *Widget toolkit*), and also used for developing non-GUI programs such as console tools and servers. Qt is most notably used in KDE, the web browser Opera, Google Earth, Skype, Qtopia and Photoshop Elements. It is produced by the Norwegian company Trolltech.

Qt is a cross-platform rich application development framework. It is a multi-platform toolkit. When you implement a program with Qt, you can run it on the X Window System (Unix/X11), Apple Mac OS X, and Microsoft Windows NT/9x/2000/XP by simply compiling the source code for the platform you want. One of the key design goals behind Qt is to make cross-platform application programming intuitive, easy and fun. Qt achieves this goal by abstracting low-level infrastructure functionality in the underlying window and operating systems, providing a coherent and logical interface that makes sense to programmers.

4.2 GUI building in Qt

We can start a C++ Project which would prompt us to give the name to the project and path of the directory where the project is to saved. After creating a project we can include dialogs, wizards, Main Window, etc and other things.

After we include the dialogs the forms will open that can be named and must be saved where the project is saved so that we know how many files have been made.
A header file is made every time we make a new form. Here a Filename.ui file is made for every dialog which stores all the widgets in it that have been included in it. The Filename.ui.h is the source code file of that form where we include the variables, libraries and accordingly form the code.

There is main cpp file which is normally connected to the main window and it can be made in the same way as we include a new dialog but this optoion will be visible only when we are making a project else while working on a dialog with buttons without a project would not have a main cpp file.

The Drag and Drop option helps to put the widgets from the ToolBox into the Form very easily and the Object Explorer, Property Editor/Signal Handler and the Project Overview Windows help the user in knowing what elements are present in the project and the functions, handles, variables, signals and slots.

The Slots are functions which when connected by some action like for eg. Clicking a Pushbutton can be executed to perform some action according to the code written in it.

In this way a simple GUIs can be made. I personally feel that one should first try to make a GUI on his own without reading the manual by experimenting so that one would be familiar about certain things before reading the manual and which would make it easy for the user to make the GUI later on.

No readymade examples are present in Qt but the Designer Manual helps in making one with the code and how to execute it in the terminal.
GUI building is thus easy in Qt.

5. Future Scope and Application

This Project was made with the prevision of making a simple user friendly GUI which helps the inexpert users to learn working with NS-2 and get familiarized with it. NS-2 is a network simulator software and is less obscure than other softwares. Hence to make it notable we have learned how to work on NS-2 and made a GUI.

This GUI made in Qt can be modified more for more options by the user once the user knows more about working with Qt. The more people know about NS-2 the more usage it will have and through this more complex scripting of NS-2 will start being built and we will see people coding in it so that more complex real-time network simulations can be implemented in it.

The future of NS-2 is bright and we might even see more and more GUIs being built for NS-2 to increase its popularity among people and hence its usage.

6. Conclusion

The aim of the project was to learn how NS-2 works and make a GUI for it so that the user finds it easy to work with it. We have learned how to install Linux and how to install NS-2 in it. We have learned the scripting language of NS-2 and how to run the tcl script in NS-2 and do the graphical analysis using the NAM (Network Animator). We have also made a GUI using Qtopia of Trolltech in Linux which is quite a user friendly GUI Builder and have successfully made a GUI for NS-2 so that the user can simple with the help of some buttons make an NS-2 script and run it in the NAM. We thus have implemented a GUI for NS-2 so that any new NS-2 user can work on NS-2 with ease. Many options have been kept in the GUI so that the user can modify according to the requirement of the simulation.

7. References

[01] Sumitabha Das, *Unix: concepts and applications*, TATA McGRAW HILL, New Delhi, 2nd Edition, 2001.

[02] http://www.linux.org/info/

[03] http://en.wikipedia.org/wiki/Linux

[04] http://www.isi.edu/nsnam/nam/index.html

[05] http://www.isi.edu/nsnam/ns/tutorial/index.html)

[06] http://cygwin.com/

[07] http://www.isi.edu/nsnam/ns/ns-build.html

[08] http://labs.nec.com.cn/tcpeval/tcpeval-manual.pdf

[09] http://dirt.cs.unc.edu/packmime/#Installation

[10] http://www.isi.edu/nsnam/ns/ns-tutorial/index.html

[11] http://nile.wpi.edu/NS/

[12] http://aplawrence.com/Linux/c_compiling_linux.html

[13] http://www.cs.binghamton.edu/~nael/cs528/proj1.html

[14] http://ftp.gnome.org/pub/GNOME/sources/glade3/3.4/

[15] http://linux.about.com/cs/softofficeutility/a/gui_cli.htm