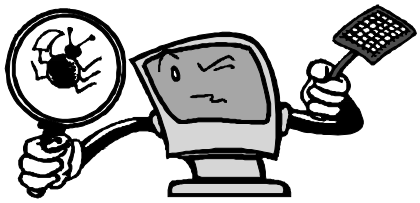

Network Simulation and Protocol Implementation Using Network Simulator 2

Written by:

Keijo Harju and Susanna Korventausta

Table of Contents

Glossary

ACIRI	AT&T Center for Internet Research at ICSI
CMU	Carnegie Mellon University
CONSER	Collaborative Simulation for Education and Research
DARPA	Defense Advanced Research Projects Agency
ICSI	International Computer Science Institute
LBL	Ernest Orlando Lawrence Berkeley National Laboratory
SAMAN	Simulation Augmented by Measurement and Analysis for Networks
UCB	University of California, Berkeley

1 Network Simulator 2

Network Simulator 2 (NS2) is a discrete event simulator targeted at networking research. It provides support for simulation of TCP, routing and multicast protocols over wired and wireless networks. Currently the development of NS2 is supported by DARPA, NSF and ACIRI. NS2l can be used for non-commercial use free of charge. Further information can be viewed via Network Simulator 2 web page at http://www.isi.edu/nsnam/ns/ .

1.1 Architecture

Network Simulator 2's architecture is object-oriented(C++/OTcl). It tries to achieve scalability as well as extensibility through separation of control and detailed protocol functionality. C++ is used for detailed protocol implementation for which runtime efficiency is important. OTcl on the other hand is used for the control of simulations in which runtime efficiency is not that important, but the possibility to quickly modify simulations is very useful. The main drawbacks resulting from this approach are that the user has to know C++ as well as Otcl, and debugging of simulations becomes more difficult. In the tool *TclCL* class hierarchy is used to link together C++ and OTcl. General architecture is depicted in Figure 1.

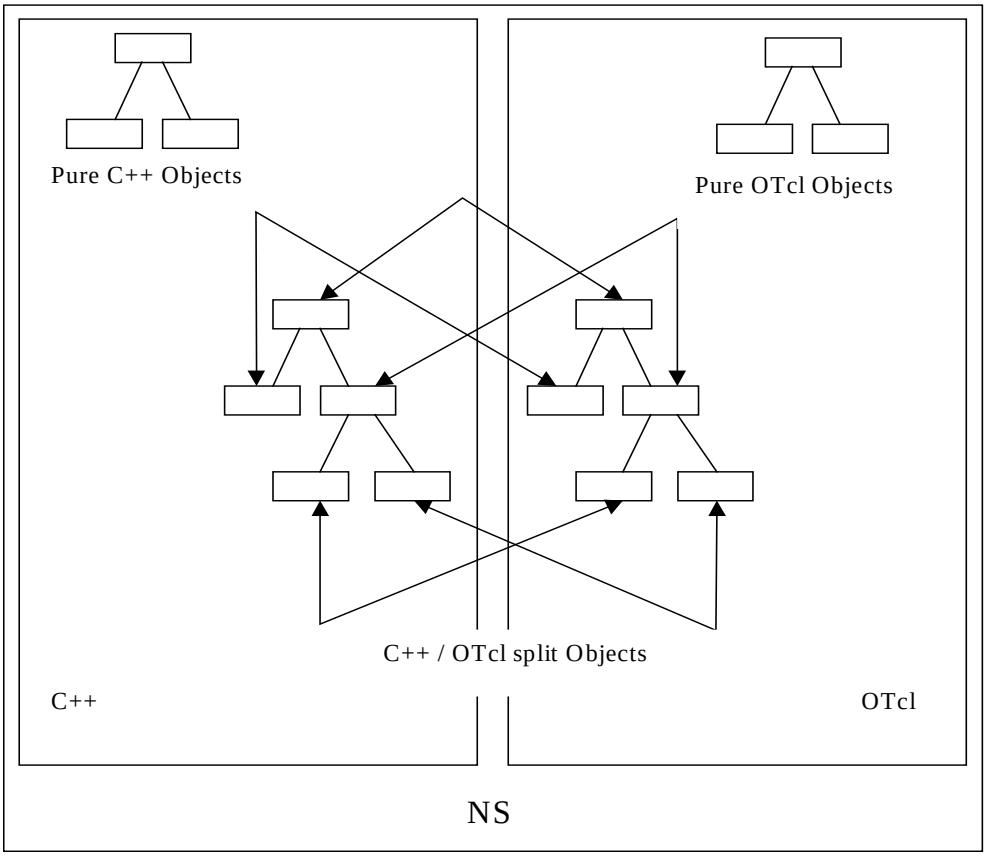

Figure 1 : General architecture

After linking of C++ member variables to OTcl object variables by using the *bind* member function, C++ variables can be modified through Otcl (e.g. in initialization) directly. Other commands from OTcl are delivered to C++ object through member function *TclObject::command(int argc, const char*const*argv).* The procedure is illustrated in Figure 2.

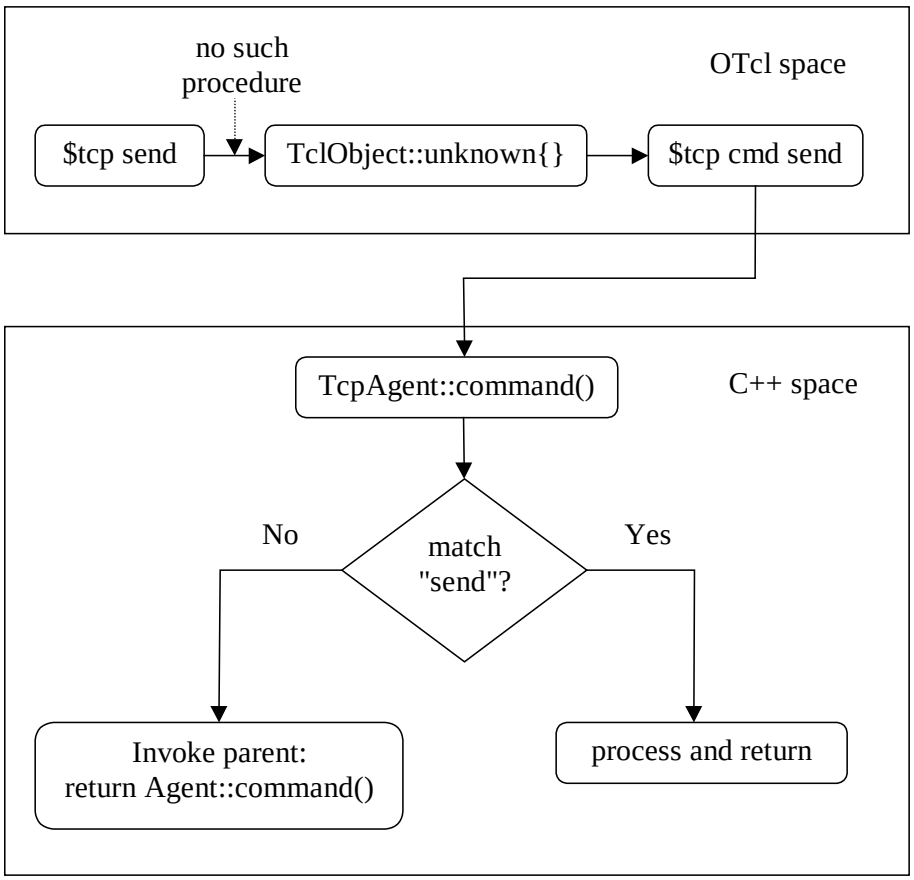

Figure 2 : Procedure for interpretation of command from OTcl to C++

Class *Tcl* can be used to invoke OTcl procedures from C++ code. This can be useful for example in passing a result string to OTcl.

1.2 Simulator

Network Simulator 2 is a discrete event simulator. It contains three types of discrete event schedulers : list, heap and hash-based calendar. NS2 also provides default implementations for network nodes, links between nodes, routing algorithms, some transport level protocols (especially UDP and many different variants of TCP) and some traffic generators. The simulator can be extended by adding functionality to these objects.

NS2 also contains some useful utilities which include e.g. Tcl debugger, simulation scenario generator and simulation topology generator. Tcl debugger is used to debug Tcl scripts and it might become necessary if one is using large scripts to control a simulation. Tcl-debug is not however installed automatically with NS2 but it can be installed later. One drawback of using Tcl-debug is that it is dependent on used Tcl version and also NS2 version.

For topology generation there are four choices : NTG, RTG, GT-ITM and TIERS packages. At least GT-ITM is part of the NS2 distribution. It requires Knuth's cweb and SGB software to work together with NS2, but these software packages are also part of the NS2 distribution. With these topology generators one can create large network topologies without the need to define the whole topology by hand.

Simulation scenario generator can be used to create traffic between nodes. When simulating wireless networks, the scenario generator can also be used to generate files that define the movement of nodes.

1.3 Emulator

It is also possible to use Network Simulator 2 as an emulator. Currently emulator support is available for FreeBSD operating system only. Emulator can be used to connect the tool to a live network. When the emulator is used one must use realtime scheduler instead of schedulers mentioned in the preceding chapter. It connects the simulation time to real time. If the scheduler falls behind emulation fails. The emulator has two modes. In *protocol mode* the emulator interprets received traffic (e.g. ICMP Echo). In *opaque mode* received data is not interpreted. The emulator is connected to live network on the IP level.

1.4 Support software

1.4.1 Nam – VINT/LBL Network Animator

Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet trace data. It supports topology layout, packet level animation, and various data inspection tools.

Nam began at LBL. It has evolved substantially over the past few years. Nam development effort is now an ongoing collaboration with the VINT project.

The first step to use nam is to produce a trace file. The trace file should contain topology information, for example nodes, links and packet traces. During an NS2 simulation, user can produce topology configurations, layout information and packet traces using tracing events in NS2.

When the trace file is generated, it is ready to be animated by nam. Upon startup, nam will read the trace file, create topology, pop up a window, do layout if it is necessary

and then pause at the time of the first packet in the trace file. Nam provides control over many aspects of animation through its user interface.

Nam does animation using the following building blocks: node, link, queue, packet, agent and monitor. Nam user manual can be found from http://www.isi.edu/nsnam/nam/ This manual page is mentioned to be incomplete.

1.4.2 XGraph – an animating plotting program

One part of the ns-allinone package is *xgraph*, a plotting program which can be used to create graphic representations of simulation results.

XGraph with animation is a modification of the popular XGraph plotting program written by David Harrison at UC Berkeley. The original program supported line plots (and restricted surface plots) on any X11 display, and had several useful features, including being able to zoom on a region with the mouse.

Paul Walker added two features to the original code. First, it does crude animation of data sets. The animation only pages through data sets in the order in which they are loaded. It is quite crude, but useful if all your data sets are in one file in time order, and are output at uniform times. Also, the code will take derivatives of your data numerically and display those derivatives in a new XGraph window.

These new features have been made a couple of years ago, and has no longer been supported.

There are a few annoying problems which haven't been cleaned up yet, the most aggravating of which is a problem with handling expose events, which sometimes forces the window to animate or refresh four or five times when it is exposed. Also, the derivative routines have not been rigorously tested. Finally, the labels are occasionally wrong when doing log plots.

Further information of XGraph can be found via
http://jean-luc.ncsa.uiuc.edu/Codes/xgraph .

2 Existing Protocol Testing Using NS2 (Minor assignment)

2.1 How to begin

The first step in the simulation is to acquire an instance of the Simulator class. Instances of objects in classes are created and destroyed in NS2 using the *new* and *delete* methods. For example, an instance of the Simulator object is created by the following command:

e.g. set ns [new Simulator]

A network topology is realized using three primitive building blocks : nodes, links, and agents. The Simulator class has methods to create/configure each of these building blocks.

Nodes are created with the *node* Simulator method that automatically assigns a unique address to each node. Links are created between nodes to form a network topology with the *simplex-link* and *duplex-link* methods that set up unidirectional and bidirectional links respectively.

Agents are the objects that actively drive the simulation. Agents can be thought of as the processes and/or transport entities that run on nodes that may be end hosts or routers. Traffic sources and sinks, dynamic routing modules and the various protocol modules are all examples of agents. Agents are created by instantiating objects in the subclass of class *Agent* i.e., *Agent/type* where type specifies the nature of the agent. For example, a TCP agent is created using the command:

set tcp [new Agent/TCP]

Once the agents are created, they are attached to nodes with the *attach-agent* Simulator method. Each agent is automatically assigned a port number unique across all agents on a given node (analogous to a tcp or udp port). Some types of agents may have sources attached to them while others may generate their own data. For example, you can attach *ftp* and *telnet* sources to tcp agents but *constant bit-rate* agents generate their own data. Sources are attached to agents using the *attach-source* and *attach-traffic* agent methods.

Each object has some configuration parameters associated with it that can be modified. Configuration parameters are instance variables of the object. These parameters are initialized during startup to default values that can simply be read from the instance variables of the object. For example, *$tcp set window_* returns the default window size for the tcp object. The default values for that object can be explicitly overridden by simple assignment either before a simulation begins, or dynamically, while the simulation is in progress. For example the window-size for a particular TCP session can be changed in the following manner:

$tcp set window_ 25

The default values for the configuration parameters of all the class objects subsequently created can also be changed by simple assignment. For example, we can say :

Agent/TCP set window_ 30

to make all future tcp agent creations default to a window size of 30.

Events are scheduled in NS2 using the Simulator method that allows OTcl procedures to be invoked at arbitrary points in simulation time. These OTcl callbacks provide a flexible simulation mechanism : they can be used to start or stop sources, dump statistics, instantiate link failures, reconfigure the network topology etc. The

simulation is started via the *run* method and continues until there are no more events to be processed. At this time, the original invocation of the run command returns and the Tcl script can exit or invoke another simulation run after possible reconfiguration. Alternatively, the simulation can be prematurely halted by invoking the *stop* command or by exiting the script with Tcl's standard *exit* command.

Packets are forwarded along the shortest path route from a source to a destination, where the distance metric is the sum of costs of the links traversed from the source to the destination. The cost of a link is 1 by default : the distance metric is simply the hop count in this case. The cost of a link can be changed with the *cost* Simulator method. A static topology model is used as the default in NS2 in which the states of nodes/links do not change during the course of a simulation.

Also static unicast routing is the default in which the routes are pre-computed over the entire topology once prior to starting the simulation.

2.2 About TCP

The TCP agent does not generate any application data on its own : instead, the simulation user can connect any traffic generation module to the TCP agent to generate data. Two applications are commonly used for TCP : FTP and telnet. FTP represents a bulk data transfer of large size, and telnet chooses its transfer sizes randomly from tcplib.

There are two major types of TCP agents : one-way agents and a two-way agent. One-way agents are further subdivided into a set of TCP senders (which obey different congestion and error control techniques) and receivers (sinks). The two-way agent is symmetric in the sense that it represents both a sender and receiver. It is still under development.

2.3 About the minor assingment

The existing protocol testing assingment is planned to be an easy one. The primary task is to write a simulation script for a specific case. Starting the use of NS2 is more difficult than some other simulation program because with NS2 you have to build the network you want to simulate by using a script. This is more difficult than with some tool where you have a good graphical user interface. The difficulty arises when you want to make some new experiments : The only error message you get from NS2 might be, that it will not start the simulation. With some graphical user interface you could really see where the correction is needed and concentrate on finding the right correction. With NS2 a lot of time goes to reading the script through and trying to solve what has gone wrong.

The best way to start studying NS2 is to read Marc Greis’s tutorial available on the NS2 web page mentioned in the beginning of this document. This has also been mentioned in the work description togther with some other links. The idea is to get some picture about NS2 and learn the very basics about how to write the simulation scripts. The simulation concentrates on TCP Sliding Window Flow Control and the

Slow Start mechanism. These have been chosen because they are familiar features of TCP.

3 Protocol Implementation Using NS2 (Major assignment)

In the protocol implementation assignment the student is asked to implement a Go-Back-N ARQ(Automatic Repeat Request) protocol. A state machine description of the protocol agent is given as well as some example MSCs. The students should know at least the basic idea of the protocol.

Usually every software developer working with NS2 should have his/her own installation of the tool. Using this approach with Telecommunications Laboratory Course would mean that lots of disk space would be needed. That is why a modified Makefile has been written. When using this Makefile all the students can use same installation of NS2. Also modifications to files that need to be modified when adding a new protocol to NS2 have been done except a modification to file ns-default.tcl. This modification cannot be done when only one installation of NS2 is used.

Template files for the students working with the assignment contain an implementation of a needed timer. There is also *Hints* chapter in the assignment specification in order to help the students to find ways to do things with NS2 without a thorough inspection of Ns manual.

4 Conclusions

4.1 Installation

There were some problems during installation of the software from the all in one package. Two lines in the NS2 source code had to be edited manually due to differences in socket handling in Linux and for example in Sun. Modifications were done based on instructions represented in the NS2 mailing list. Both modifications were made to file *ns-2.1b6/indep-utils/webtrace-conv/ucb/utils.cc* .

Line 358

```
if((ns = accept(s,(struct sockaddr*)&from,&fromlen)) < 0)
```

was replaced by line

```
if((ns=accept(s,(struct sockaddr*)(&from),(socklen_t*)(&fromlen)))<0)
```

and line 477

```
if(getsockname (s, &sockname, &len) < 0)
```

was replaced by line

```
if(getsockname (s, &sockname, (socklen_t*)(&len)) < 0)
```

Validation scripts that are run after the installation reported also some minor warnings due to rounding up error. According to NS2 documentation these should not however be considered as fatal. One drawback of NS2 is that it is dependent of several independent software packages e.g. NS2 itself, Tcl and zlib libraries. This might cause problems if user's environment already contains some of these packages and their versions don't work together. However when NS2 is installed from the all in one package all the necessary software are installed under one directory. This results that one should be able to resolve potential problems with user's environment as well as keep track of the used versions of software packages.

4.2 Existing Protocol Testing

Learning of how to write tcl-scripts takes some time. Marc Greis's tutorial is a big help in the very beginning. The basics have been explained very clearly there. Ns manual provides reference documentation for NS2. It is helpful when you already know the basics. There is also some example files available in the *tcl/ex/* directory, but unfortunately they haven't been updated which means that most of them don't work. When you have something wrong with your script and you are going to start the simulation, you will not get any clear error message from NS2. This means that errors are quite tedious to find.

4.3 Protocol Implementation

Implementation of new protocol to NS2 is quite easy as long as the protocol's state machine is not complicated. NS2 does not support state machine like programming of protocols. Addition of new PDU structure to the tool is also quite easy. NS2 does not however give good support for PDU encoding and decoding on byte level because of the structure used in the tool to represent a PDU. Simple protocols are quite quickly implemented because the tool provides default implementations for e.g. timers and queues. Documentation of NS2 is quite poor and that is why one's first implementations using the tool could take some time. Ns manual is also a bit too general on several key subjects which are needed when implementing protocols. Some features are not documented at all. When compared to other simulation tools NS2 seems also quite primitive - it lacks graphical user interface and control of the simulations is done using Tcl which is sometimes quite hard to debug.