To appear, ACM SIGCOMM Workshop on Computer Networking: Curriculum Designs and Educational Challenges, pp. 45--46. August 20, 2002. Pittsburgh, PA, USA.

Using Ns in the Classroom and Lab[*]

Christos Papadopoulos and John Heidemann[†]

1 Introduction

The ns-2 network simulator is widely used in research to evaluate new networking protocols (see `http://www.isi.edu/nsnam/ns/ns-research.html`). Even though ns has been used by researchers, it has seen relatively little use in the classroom and laboratory. Yet network simulation is a good fit for classroom and laboratory use because simulation allows capturing and dissecting all aspects of protocol operation in a much simpler way that is possible with real code and experiments. On the other hand, ns is a large piece of software, portions of it can be complex, and start-up overhead makes it difficult to bring to bear by a new student or busy professor.

This white paper describes how we are using ns in networking education. We seek to apply simulation to two different areas: the classroom and the laboratory. For *classroom* use, we seek to augment lectures with animations that show specific protocol behavior. In *laboratory* use, students modify simulation scenarios to explore protocol design choices. Here Ns is used directly by students, who submit their work for grading. We give a brief summary of how we have used ns in these roles and changes we have made to ns to make it more amenable to such purposes.

2 Classroom Use

Ns' companion tool nam provides packet-level animations of ns simulations. Animations have been used before to show algorithms such as sorting. We believe that animations are also useful to illustrate network protocols, by visualizing packet exchanges and state distributed in different nodes.

We are building a library of animations that illustrate several networking protocols, including transport-level issues (stop and wait, the effect of various back-off strategies, and TCP-specific issues such as slow-start and fast retransmit), router queueing policies (drop-tail, RED, etc.), multicast routing (flood and prune, PIM shared- and source-specific-trees), and reliable multicast (SRM, PGM).

Experiences: Our experience with animations in the classroom have been very positive. Since they play-out at some fraction of real-time, animations are particularly good at demonstrating time-dependent concepts such as the delay in TCP's reaction to congestion or loss. Animations are also good at representing distributed state since one animation shows several nodes at the same time. For example, nodes are annotated with SRM timer values to illustrate the benefits RTT-biased and randomized delay intervals, and colors are used to distinguish nodes waiting to send a repair request from those whose requests have been suppressed.

We are also gaining experience authoring animations. Important lessons learned include the need to have separate animations, each focused on a specific concept, rather than tackling multiple concepts in a single animation. It's also important that the animation be completely self-contained. Initially we anticipated animations to be augmented with web pages (particularly for self-guided animations) In practice, however, it has proven difficult for an observer to know when to focus on the web page and when on the animation. Finally, as a practical matter, it should be very easy to get started using animations. It should be easy to find animations, understand the context of the work, and put together the necessary pieces quickly. The ability download a binary version of nam and animation scripts are important simplifications (as opposed to compiling the software and running simulations) if they are to be integrated into a busy semester.

Approaches: Our initial experiences with classroom annotation have prompted several developments. First, we established a web-based repository of educational ns scripts at `http://www.isi.edu/nsnam/repository/`. This database stores scripts in a uniform format and allows anyone to contribute new scripts via a simple electronic form. As of this writing, the database contains about two dozen modules contributed from four different institutions.

Second, we have refactored some of or early animations into smaller, better focused modules to clarify the concepts.

Finally, we plan to gradually improve nam's annotation capabilities. Although nam currently provides packet-level animations with a fair amount of control over node labeling, color, shape, and packet color, more work is needed to add text annotations for packets (for example, to label a packet "3rd dup-ack" for TCP fast-retransmit), and make

[*]This work was supported as part of the CONSER project, funded by NSF as grant number ANI-9986208.

[†]Christos Papadopoulos and John Heidemann are with the USC Comptuer Science Department and the USC Information Sciences Institutes. E-mail: {christos,johnh}@isi.edu.

these capabilities easier to use from ns scripts.

3 Laboratory Use

An important complement to classroom lectures are laboratory experiments. In networking, this often implies programming, protocol design, experiments and measurement. We believe that simulation has an important role here, since it allows students to examine problems with much less work and of much larger scope than are possible with experiments on real hardaware. Simulation can be easier than experimentation because simulators do not need to reproduce all the details of the real world and they can be easily instrumented. In addition, simulations of dozens or hundreds of nodes are easy on limited hardware, many more than is affordable if physical hardware was required.

We have used ns to do several types of laboratory experiments. The simplest are of the form "run this script" and examine the trace output or nam animation, asking students to identify TCP behavior. The next step up is to have them modify the simulation script in simple ways that require some or little understanding of the script details. Examples include "change the router queueing policy from drop-tail to RED", "vary the link propagation delay and observe the results", or "observe this scenario and describe what to change to improve throughput". We have also assigned simple protocol implementations in the context of a message passing framework (described below) or modifying an existing C++ implementation.

Experiences: Our experiences with experiments as homework problems have also been positive, but clearly such problems must be designed with care. If more complicated assignments are to be assigned (such as those requiring new coding), it's best to introduce the simulator gradually.

One observation that initially surprised us is that many students were not familiar with the concepts behind discrete event simulation. Confusion between real-time and simulation-time, and multithreading and event-driven programming can be prevented with a brief review of the concepts (typically a half-hour to hour of lecture time).

We were pleased to discover that students adapted quite quickly to using either Tcl to specify new scenarios, or C++ to changing existing modules, and many were able to use Tcl as an scripting language to specify the scenario. Efforts that require them to work in both languages simultaneously are probably best reserved for more advanced classes. We have been hesitant, however, to give students a blank slate. The framework of an existing script is necessary to avoid stumbling over initialization details that are irrelevant to protocol design.

Approaches: Our experience has suggested several helpful approaches. First, we are developing a graphical editor based on nam that allows strictly GUI-based creation of simple scenarios. With the editor, topology and traffic design can be done by point, drag and click operations. Configuration of parameters is done with dialog boxes, and the simulation can be launched directly from the editor window. The editor hides irrelevant details such as initialization and scripting, allowing undergraduates to do simple experiments from scratch. While, we do not believe a GUI editor can encompass the whole range of simulations possible in ns (there are simply too many options to make that feasible), the editor exposed a subset of the ns functionality without any traditional programming. (Our experiences with lab use of the editor so far are limited to one semester.)

If ns is to be used for coding complete new protocols, the amount of background students require must be minimized. We are developing a *message passing* module in ns to allow simple protocols to be developed within a subset of the simulator. This includes simple ways to add headers and process messages at each node, with alternative implementations either completely in Tcl and completely in C++. Early experience in one semester has been promising: as an example homework, students successfully simulated scenarios showing the synchronization of periodic routing messages, as described by Floyd and Jacobson. As a side benefit, the message passing model may also be useful for researchers who want to quickly prototype a new protocol.

Finally, as a practical matter, an ns installation can sometimes be difficult and by default it consumes a large amount of disk space. We recommend installing ns on personal machines using the "allinone" package, which provides a simple download and installation process. Variations in Windows development environments have encouraged us to provide a pre-compiled binary for that platform.

To mitigate the size of an ns installation, in systems where many accounts have a shared file system we use a shared installation of ns. For assignments where students need to modify and recompile ns, we have a procedure where students create symbolic links to the source. Students then remove the symbolic links and make copies of the specific files they would like to change.

4 Conclusions

We have been happy with our use of ns in the class and lab, although we plan to continue to refine the tools and lessons. Perhaps ns and nam are now able to serve not just as tools for researchers, but also as tools for education.