Improving Simulation for Network Research *

Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall,
Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy,
John Heidemann, Polly Huang, Satish Kumar, Steven McCanne,
Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu,
Haobo Yu, Daniel Zappala

USC Computer Science Department Technical Report 99-702b

March 1999 (revised September 1999)†

Abstract

New protocols and algorithms are being developed to meet changing operational requirements in the Internet. Simulation is a vital tool to quickly and inexpensively explore the behavior of these new protocol across the range of topologies, cross-traffic, and interactions that might occur in the Internet. This paper describes ns, a widely used, multi-protocol network simulator designed to address the needs of networking researchers. Ns provides multiple levels of abstraction to permit simulations to span a wide-range of scales, emulation, where real-world packets can enter the simulator. We describe the ns architecture and examine the range of ways simulation and ns are used in networking research.

Keywords: network protocol design, simulation, Internet protocols, split-language programming, ns, nam

1 Introduction

In recent years, the Internet has grown significantly in size and scope, and as a result new protocols and algorithms are being developed to meet changing operational requirements in the Internet. Examples of such requirements include quality of service support, multicast transport, security, mobile networking, and policy management. Development and evaluation of protocols and algorithms for these domains requires answering many design questions. Although small-scale evaluation in a lab, wide-area testbeds, and custom simulators can all be valuable, each has significant shortcomings. These approaches often lack the wide mix of traffic and topologies found in real networks, they can incur substantial expense, and repetition of experiments under controlled conditions can be difficult.

Multi-protocol network simulators can provide a rich environment for experimentation at low cost. A common simulation environment used across disparate research efforts can provide substantial benefits to the networking community. These benefits include improved validation of the behavior of existing protocols, a rich infrastructure for developing new protocols, the opportunity to study large-scale protocol interaction in a controlled environment, and easier comparison of results across research efforts.

The VINT project is attempting to facilitate the design and deployment of new wide area Internet protocols by providing network researchers with an improved set of simulation tools. This paper presents the VINT simulation framework and describes how it aims to meet many of the simulation needs of the network research community. We begin by identifying the requirements of a multi-protocol network simulator, after which we describe how VINT's ns simulator addresses these requirements. We then present the software architecture of ns, which provides an extensible framework within which new protocols can be developed. We then show several examples of ways in which ns has been used in protocol design and development, and

*This research is supported by the Defense Advanced Research Projects Agency (DARPA) through the VINT project at LBL under DARPA Order E243, at USC/ISI under DARPA grant ABT63-96-C-0054, at Xerox PARC under DARPA grant DABT63-96-C-0105.

†Originally published in March, 1999, this technical report was updated in September, 1999 (one section was moved and a number of typos were fixed). This technical report has been accepted to appear in IEEE Computer Magazine.

we evaluate the success and shortcomings of the VINT effort. We conclude by discussing previous work on network simulation and related topics, and by describing future challenges. A companion paper describes nam, the network animation companion to ns [12].

Ns is publicly available at `http://www-mash.cs.berkeley.edu/ns/` and has been widely used by network researchers.

Alternatives to a Common Simulator (sidebar)

Testbeds and laboratory experiments are also important approaches to network research. Since they use real code, experiments run in testbeds or labs automatically capture important details that might be missed in a simulation. This approach also has drawbacks; testbeds are expensive to build, testbeds and labs can be difficult to reconfigure and share, and they have limited flexibility. In addition, some networking phenomena such as wireless radio interference can be difficult to reproduce experimentally, thus making it difficult to compare or evaluate protocol designs.

Protocol design using simulation usually begins with an individual investigator's simulations of isolated protocol elements using small-scale topologies and simplified/static assumptions about higher and lower level protocols. Because the startup costs are so high, no individual group has the resources to create a comprehensive and advanced networking simulation environment, leading to a lack of standardization and reproducibility of simulations constructed by different groups of designers. In the current paradigm, directly comparable data would be available only if each individual designer implemented, within their own simulator, all of the competing mechanisms. Very few research groups have the resources to do this, and it is often most effective to have a simulation component constructed by those who know most about the particular protocol represented by the component.

Related Work (sidebar)

Network Simulators Network simulation has a very long history. Ns itself is derived from REAL [22], which is derived from NEST [11]. Although we cannot list all relevant network simulators here, this section describes distinguishing features of network simulators and compares prominent examples with ns.

Simulators have widely varying focuses. Many target a specific area of research interest such as a particular network type or protocol like ATM or PIM multicast. Others, including ns, REAL, OPNET [10], and INSANE [25] target a wider range of protocols. The most general of these provide a simulation language with network protocol libraries (e.g. Maisie [3] and OPNET [10]). Very focused simulators model only the details relevant to the developer.

The engine of ns and other network simulators is a discrete event processor. Several complementary approaches have been taken to improve accuracy, performance, or scaling. Some simulators augment event processing with analytic models of traffic flow or queueing behavior (for example, OO [29] and fluid network approximations [23]) for better performance or accuracy.

Parallel and distributed simulation is a second way to improve performance. Several simulators support multiprocessors or networks of workstations [22, 3, 31]. Although ns is focused only on sequential simulation, the TeD effort has parallelized some ns modules [31].

Abstraction is a final common approach to improving simulator performance. All simulators adopt some level of abstraction when choosing what to simulate. FlowSim was the first network simulator to make this trade-off explicit [2]. As discussed in "Abstracting Simulation", ns supports several levels of abstraction.

A number of different simulation interfaces are possible, including programming in a high-level scripting language, a more traditional systems language [3], or sometimes both [10]. Some systems focus on allowing the same code to run in simulation and a live network (for example, x-Sim [6] and Maisie [3]). Most systems augment programming with a GUI shell of some kind Ns provides a split-level programming model where packet processing is done in a systems language while simulation setup is done in a scripting language. Nam [12] provides visualization output and is currently being enhanced to support simple scenario editing.

Network Emulation. Early work in network emulation included the use of "flakeways" (gateways that could alter or drop packets) and were used for early TCP/IP tests. More recent work has included special purpose stand-alone network emulators supporting packet delays and drops [1, 33]. These systems are usually implemented as kernel drop-in modules that intercept the IP layer packet forwarding path and thus appear to end stations as routers. Their capabilities are generally limited to simple packet manipulations and don't provide for interference from simulated cross traffic. Moreover, these systems do not include a general simulation capability as provided by ns.

2 Simulation Needs of Researchers

Simulation allows the evaluation of network protocols under varying network conditions. Studying protocols, both individually and as they interact with other protocols, under a wide range of conditions is critical to explore and understand the behavior and characteristics of these protocols. The VINT project, through the ns simulator and related software, provides several critical innovations that broaden the range of conditions under which protocols can be evaluated while making this experimentation tractable:

- **Abstraction:** Varying simulation granularity allows a single simulator to accommodate both detailed and high-level simulations. Networking protocols are studied at many levels, both at the detail of an individual protocol, and in the aggregation of many data flows and interaction of many protocols. The abstraction mechanisms in ns allow researchers to examine both of these issues without changing simulators, and to validate abstractions by comparing detailed and abstract results.

- **Emulation:** Most simulation experiments are confined to a single simulated world including only those protocols and algorithms included in the simulator. However, emulation, which allows a running simulator to interact with operational network nodes, can be a powerful tool in protocol design.

- **Scenario generation:** Testing protocols under an appropriate set of network conditions is critical to achieve valid and useful results. Automatic creation of complex traffic patterns, topologies, and dynamic events (i.e., link failures) can help generate such appropriate scenarios.

- **Visualization:** Tools that allow researchers to understand more easily the complex behavior in a network simulation are needed. Given the complex range of behaviors, and the large scale of the networks involved, merely providing tables of summary performance numbers does not adequately describe the behavior of the network. Visualization adds a dynamic representation to network behaviors, allowing researchers to develop better protocol intuition and aiding protocol debugging. Nam, a network animation tool, is described in a companion paper [12].

- **Extensibility:** The simulator must be easy to extend in order to add new functionality, explore a range of scenarios, and study new protocols. Ns employs a split programming model designed to make scripts easy to write and new protocols efficient to run.

In addition to these innovations, several engineering issues have substantial impact on a simulator's usability. First among these is the availability of a wide range of protocol modules in the simulator. This allows easy comparison of different approaches. It also reduces simulation development time enabling the researcher to focus on those aspects of the simulation relevant to the design question being studied. Second, validated protocols against which new variants can be compared are needed. Validation of TCP is illustrated in a separate paper [15]. Other protocols are validated in ns to the degree warranted by their maturity. Finally, given the significant number of protocol modules in ns and the interactions among them, mechanisms to prevent modifications in one module from breaking functionality in another are needed. To this end, ns includes many automated test suites that keep unintentional changes in behavior from creeping into the simulator.

In the following sections we expand on the innovations in ns and we describe its innovative software architecture.

3 VINT and the *ns* Simulator

The VINT project aims to bring a change in current protocol engineering practices by enabling the study of protocol interactions and scaling using a common simulation framework with advanced features. The public distribution of our system has helped to reduce the duplication of effort expended in the networking research and development community.

As mentioned above, the ns simulator includes several special features targeted at supporting large scale, multi-protocol simulations. These features include an alternative configuration for large-scale simulations, a capability to interface the simulator to a live network, automated simulation scenario generation facilities, and visualization. In the remainder of this section we describe the first three of these features. Visualization is described in a companion paper [12].

3.1 Abstracting Simulation

Computer resource limitations such as memory and processing time often constrain the number of net-

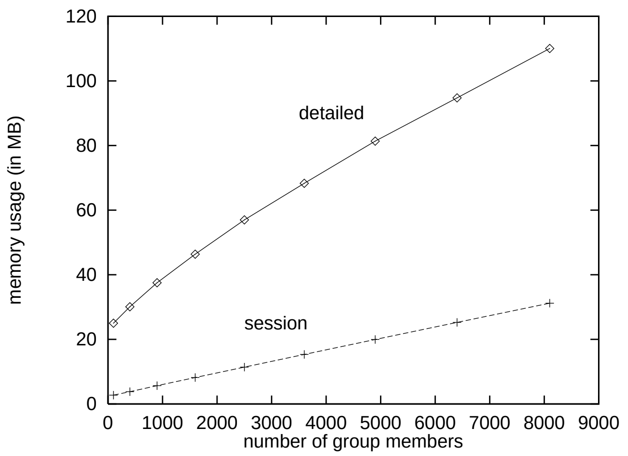

Figure 1: Session-level abstraction allows substantially larger numbers of multicast group members in the same amount of memory.

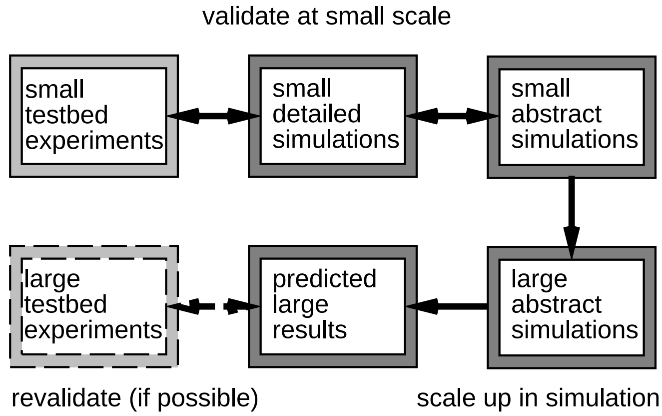

Figure 2: Validation of abstract simulations.

work objects (nodes, links and protocol agents) that can be simulated in a packet-level simulation. A scalable network simulator accommodates wide ranges of variation in each kind of network object, data in transit, and information collected. There are three complementary approaches to scaling a simulator: tuning the implementation, removing unnecessary simulation detail, and supporting parallelism. Other researchers have successfully explored parallel network simulation, and multiple efforts to parallelize ns are currently underway elsewhere (see "Related Work" for references to these approaches). Our efforts are focused on tuning our implementation and providing multiple levels of protocol abstraction. By eliminating less important details, substantial savings can be realized while preserving the basic validity of the model.

VINT provides several levels of abstraction in ns. The default simulator provides a *detailed* model with hop-by-hop packet forwarding and dynamic routing updates. *Centralized routing* replaces routing messages with a centralized computation, saving processing time and memory in exchange for slightly different timing in routing changes. *Session-level* packet forwarding replaces hop-by-hop packet flow with a precomputed propagation delay [21]. *Algorithmic routing* replaces shortest-path routing with tree-based routing, transforming $O(n \log n)$ memory requirements to $O(n)$. Each abstraction sacrifices some details to save memory, so abstractions must be applied only when appropriate.

By adjusting the simulation abstraction level, a user is able to trade off simulator performance versus packet-level accuracy. Increasing the level of abstraction provides the ability to perform increasingly large simulations, while decreasing the level of abstraction provides for a more realistic simulation. The session level simulator can abstract many details of links, nodes, and cross-traffic. Simulations can be run in both detailed and session level mode side-by-side to compare the performance and accuracy across the different levels of abstraction. Figure 1 shows the memory savings possible from session-level simulations for a particular scenario with large multicast groups.

The cost of abstraction is simulation accuracy. The degree to which accuracy is sacrificed, and the impact of this sacrifice on the validity of the results, varies greatly between simulation scenarios. For example, while the details of a particular media's approach to segmentation and reassembly are important for LAN simulations, they can be reflected adequately in the link's packet loss rate for higher-level WAN simulations.

To insure that abstraction does not substantially alter simulation results, Figure 2 shows how we validate simulations at small scale before projecting results at larger scales [21]. A quantitative analysis of SRM performance across detailed and session simulations suggests that while the timing of individual SRM events does vary, average aggregate behavior changes by only 3–9% in the cases we examined. Finally, we are also working on hybrid abstractions in which different portions of the same simulation operate in detailed and session levels of abstraction.

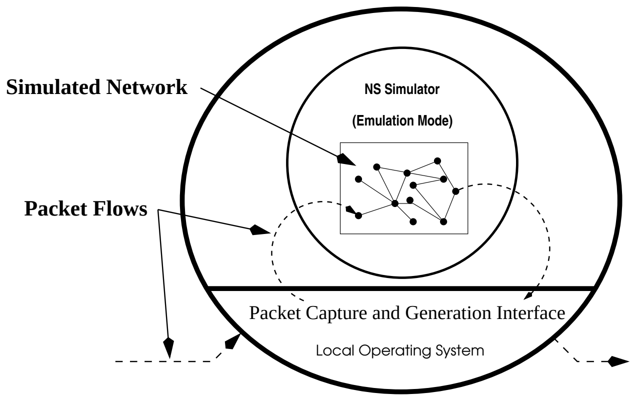

Figure 3: Emulation: live network traffic passes through simulated topology and cross-traffic.

3.2 Emulation interface

Ns includes an *emulation interface* which provides a method for network traffic to pass between real-world network nodes and the simulator. In combination with the simulator's tracing and visualization facilities, emulation provides a powerful analysis tool for evaluating the dynamic behavior of protocols and their implementations in end systems. An emulation scenario is constructed by placing the simulator as an intermediate node (or end node) along an end-to-end network path, as illustrated in Figure 3. The simulator contains a simulated network, and passes live network traffic through the simulation, subjecting it to the dynamics of the simulated network. The simulator's scheduler is synchronized with real-time, allowing the simulated network to emulate its real-world equivalent so long as the simulated network can keep pace with the real world events.

Emulation is useful beyond conventional simulation in evaluating both end system and network element behavior. With emulation, end system protocol implementations can be subjected to packet dynamics (e.g. drops, re-ordering, delays) that are difficult to reproduce reliably in a live network. Furthermore, by capturing traffic traces of live traffic injected into the simulation environment, visualization tools may be employed to evaluate the end system's dynamic responses. In the converse situation, network element behavior (e.g., a queueing or packet scheduling discipline) may be evaluated in relation to live traffic generated by real-world end stations. Such simulations are useful in identifying undesirable network element behavior prior to deployment in live networks.

The ns emulation facility is currently under development, but an experimental version has already proven useful in diagnosing errors in protocol implementation. For example, researchers at UC Berkeley have developed the MediaBoard, a shared whiteboard application using a version of the SRM protocol supported in the MASH toolkit [27]. The simulator is placed between groups of live end stations communicating using SRM. Multicast traffic passing between groups must traverse the simulator, and is subject to the dynamics of its simulated network. Visualization of traces taken within the simulation environment reveals end station retransmissions triggered by packets dropped or delayed within the simulated network. This use of emulation has helped to pinpoint time-dependent behaviors of the MediaBoard that are otherwise very difficult to diagnose.

3.3 Scenario Generation

In ns, a *simulation scenario* defines the input configuration for a simulation run. Scenarios are made up of several components: a *network topology*, including the physical interconnects between nodes and the static characteristics of links and nodes, *traffic models* which define the network usage patterns and locations of unicast and multicast senders, and *test generation*, which creates events such as multicast group distributions (receivers joining and leaving) and *network dynamics* (node and link failures) designed to stress an implementation. Automated generation of scenarios is important in the evaluation of protocol robustness. It allows researchers to cover much larger portions of the operational space than is possible through manual reconfiguration. Furthermore, by subjecting competing protocols to identical scenarios, meaningful comparative studies can be performed.

Topology Ns supports both pre-defined and automatically generated network topologies. Pre-defined topologies may be created manually or chosen from a *topology library* ranging from simple topologies to the topologies of real operational networks. Tools that automatically generate topologies provide the ability to create random topologies according to a set of specified parameters such as degree of connectivity, levels of hierarchy, and other features. Rather than create our own topology generation tools from scratch, we support the Georgia Tech Internetwork Topology Models (GT-ITM) package which creates flat random networks using a variety of edge distribution models, as well as hierarchical and transit-stub networks. In addition, the *tiers* system can be used to create three-level hierarchical topologies similar to the transit-stub GT-ITM topologies [7].

The key challenge in topology generation is coming up with topologies that embody relevant characteristics of real networks. Once accomplished, the ns framework easily allows simulation of any generated topology. Hence, if new and better topology generation tools are developed in the future, using their output in ns likely requires at most a simple format conversion program.

Traffic Models Traffic generation support or *load libraries* provide a synthetic application workload model. For example, application traffic generation, call patterns, and multicast group membership dynamics may be included in a load library. As with the topology libraries, load libraries may be derived from empirical data, analytic models, or generated randomly to allow "what-if" investigation of particular parts of the operating region, even if that region is not currently observable in operational networks.

Ns provides a wide variety of source models that can be used in conjunction with both unicast and multicast transport protocols. At present, supported protocols include reliable delivery transport (e.g. several TCP variants, SRM), and unreliable transports with various semantics (e.g. RTP and UDP). For simulations of TCP, both bulk data and interactive sources are available. The former can model an FTP application while the latter, based in part on a model developed from traffic traces [8], models Telnet-like applications. We simulate web traffic with models based on Mah's measurements [24]. Other source models are available for non-flow controlled applications. These include a constant bit rate source, on-off sources using either exponential or Pareto distributions (the latter useful in generating self-similar traffic [37]), and a source that generates traffic from a trace file.

The composable framework of ns makes adding new traffic models fairly easy, and encourages construction of compound models out of the individual component. In simulations of Receiver-driven Layered Multicast (RLM), for example, a multi-layered video source was created by combining several CBR streams [28]. A similar approach was used to incorporate correlations of burstiness across layers in another study involving layered video [4].

In creating a simulation scenario, specifying individual traffic sources generated by the source models provided by ns is not always sufficient. Instead, in large network simulations, configuring a set of sources that in the aggregate generate suitable background traffic with desired characteristics (e.g., aggregate bandwidth, burstiness, self-similarity, etc.) is a challenge. Developing tools to help users synthesize simulation scenarios is an area of ongoing work in the VINT project.

Test Generation Choosing an appropriate set of test conditions for a simulation experiment is never simple, and evaluating the correctness of a protocol can be a daunting task. We developed a framework for *S*ystematic *T*esting of Protocol *R*obustness by *E*valuation of *S*ynthesized *S*cenarios (STRESS) [19, 20] in order to reduce the effort needed to identify pathological cases of protocol behavior. As the name implies, this framework integrates systematic synthesis of test scenarios with the VINT simulation environment of ns. We are in the process of developing automatic test generation algorithms for multicast protocols. These methods were applied to multicast routing protocol studies in ns. Several design errors were discovered and corrected with the aid of STRESS; the detailed results are presented in [19].

Future work in this area will consider the effect of a wider range of network failures on multicast routing. We will also investigate systematic methods for performance evaluation and sensitivity analysis of end-to-end protocols such as multicast transport. In addition, we plan to use the emulation interface in ns to conduct systematic conformance testing and performance profiling of actual protocol implementations.

4 Software Architecture

The ns software is constructed in a way intended to promote extension by users. The fundamental abstraction provided by the software architecture is "programmable composability". In this model, simulation configurations are expressed as a *program* rather than as a static configuration or through a schematic capture system. A simulation program *composes* objects dynamically into arbitrary configurations to effect a simulation configuration. By adopting a full fledged programming model for simulation configuration, the experimentalist is free to extend the simulator with new primitives or "program in" dynamic simulation "event handlers" that interact with a running simulation to change its course as desired.

Rather than adopt a single programming language that defines a monolithic simulation environment, we have found that different simulation functions require different programming models to provide adequate flexibility without unduly constraining performance. In particular, tasks like low-level event processing or packet forwarding through a simulated router require high performance and are modified infrequently once put into place. Thus, they are best served by an implementation expressed in a compiled language like C++. On the other hand, tasks like the dynamic configuration of protocol objects and the specification and placement of traffic sources are often iteratively refined and undergo frequent change as the research task unfolds. Thus, they are best served by an implementation in a flexible and interactive scripting language like Tcl [30].

To this end, ns exploits a *split programming model*, where the simulation kernel—i.e., the core set of high-performance simulation primitives—is implemented in a compiled language (C++) while simulations are defined, configured, and controlled by writing an "ns simulation program" expressed in the Tcl scripting language. This approach can be a boon to long-term productivity because it cleanly separates the burden of simulator design, maintenance, extension, and debugging from the goal of simulation itself—the actual research experiments—by providing the simulation programmer with an easy to use, reconfigurable, and programmable simulation environment. Moreover, it encourages a programming style that leads to an important separation of mechanism and policy: core objects that represent simple and pure operations are free of built-in control policies and semantics and can thus be easily reused.

In our split programming model, fine-grained simulation objects are implemented in C++ and are combined with Tcl scripts to effect more powerful, higher-level "macro-objects". For example, a simulated router is composed of demultiplexers, queues, packet schedulers, and so forth. By implementing each primitive in C++ and composing them using Tcl a range of routers can be simulated faithfully. We can string together the low-level demultiplexers, queues, and schedulers to model an IP router perhaps with multicast forwarding support, or instead arrange them into a configuration that models a high speed switch with a new scheduling discipline. In the latter case, the switch could be easily extended with protocol agents (implemented entirely in Tcl) that modeled an experimental signaling protocol. Performance also guides our split programming model. Low-level event-level operations like route lookups, packet forwarding, and TCP protocol processing are implemented in C++, while high-level control operations like aggregate statistics collection, modeling of link failures, route changes, and low-rate control protocols are implemented in Tcl. Careful design is necessary to obtain a desirable trade-off between performance and flexibility, and this division often migrates during the course of a protocol investigation.

This composable macro-object model is naturally expressed using object-oriented design, but unfortunately, at the time we designed ns, Tcl did not provide support for object-oriented programming constructs nor did it provide very effective programming constructs for building reusable modules. Thus, we adopted an object-oriented extension of Tcl. Of the several Tcl object extensions available at the time, we chose the Object Tcl (OTcl) system from MIT [36] because it required no changes to the Tcl core and had a particularly elegant yet simple design. We further adopted a simple extension to OTcl called *TclCL* (for Tcl with classes) that provides object scaffolding between C++ and OTcl and thereby allows an object's implementation to be split across the two languages in congruence with our split programming model [27].

With the OTcl programming model in place, each macro-object becomes an OTcl class and its complexity is hidden behind a simple-to-use set of object methods. Moreover, macro-objects can be embedded within other macro-objects, leading to a hierarchical architecture that supports multiple levels of abstraction. As an example, high-level objects might represent an entire network topology and set of workloads, while the low-level objects represent components like demultiplexers and queues. As a result, the simulation designer is free to operate at a high level (e.g., by simply creating and configuring existing macro-objects) at a middle level (e.g., by modifying the behavior of an existing macro-object in a derived subclass) or at a low level of abstraction (e.g., by introducing new macro-objects or split objects into the ns core). Finally, class hierarchies allow users to specialize implementations at any one of these levels, for example extending a "vanilla TCP" class to implement "TCP Reno". The net effect is that simulation users can implement their simulation at the highest level of abstraction that supports the level of flexibility required, thus minimizing exposure to and the burden associated with unnecessary details.

5 Research with Ns

Network research simulations can often be categorized into one (or more) of a few broad themes. These include selecting a mechanism among several options, exploring complex behavior, and investigating unforeseen multiple protocol interaction. This section uses examples from the broad base of ns-based simulations in the networking community to demonstrate instances of each theme.

Selecting a Mechanism As in most design activities, much of the time spent in protocol design, redesign, and debugging concerns evaluation of the various alternatives to accomplishing a goal. Ns has seen broad use in developing TCP variants and extensions, exploring reliable multicast protocols, and in considering packet scheduling algorithm in routers.

As an example, ns has been used to explore several TCP variants and extensions such as selective acknowledgments [13], forward acknowledgments [26], explicit congestion notification (ECN) [14], and pacing [35]. These efforts were aided by the existence of a simulator-specific TCP implementation. By omitting application-specific baggage such as memory management and IP fragmentation, ns users were able to focus on the research issues such as packet retransmission policies and throughput.

Exploring Complex Behavior Complex behavior often takes the form of unexpected self-organization of dynamic systems. Examples include synchronization of periodic network traffic such as routing updates, TCP "ACK compression" in asymmetric or congested networks, undesired or unpredicted differential treatment of TCP flows due to RTT variations, contention for bandwidth reservations, and "ACK implosion" for large-scale reliable multicast protocols. In each of these domains, simulation has been a useful tool in helping to identify and understand these phenomena.

Error recovery in the Scalable Reliable Multicast (S-RM) [17] is an example of exploration of complex behavior with ns. SRM was designed to support reliable group communication for large group sizes. It uses a probabilistic-based NACK protocol to achieve reliability. A receiver detecting a loss multicasts negative acknowledgement to the group. Each group member who has the missing data prepares to repair the error. To avoid repair implosion (everyone sending the repair at once), repairs are delayed by a random amount proportional to the estimated distance between the participants. While the original simulations of SRM were done in a stand-alone simulation tool, an SRM implementation has been added to ns, where it has been widely used to study SRM recovery behavior over a wide range of topologies [32] and variants [34]. This research was enabled by the public availability of of SRM in a well-documented simulator.

Comparing Research Results: A common research challenge is comparing a new protocol design against existing protocols. Comparisons of full protocols are often difficult because they may require a particular operating system or may not be widely available. By providing a publicly available simulator with a large protocol library, ns has become an ideal "virtual testbed" for comparing protocols.

The reliable multicast community have used ns widely for protocol comparison. In addition to the S-RM variants previously described, Hänle used ns to compared the Multicast File Transfer Protocol [18], and DeLucia considered representative-based congestion control [9].

Multi-protocol interactions Multiple protocol interactions include the impact of protocol operation at one layer upon another layer (e.g. http on TCP, reservations on datagram delivery) or the interaction of unrelated protocols (e.g. the effect of uncontrolled traffic sources on congestion-controlled traffic flows or routing stability on transport layer performance). The problem with studying protocol interactions is that it requires twice the effort of studying a single protocol: the designer must understand and implement protocols at all the relevant layers. Ns reduces this effort by providing a validated library of important protocols.

RED and TCP snooping are two examples where ns greatly aided protocol studies exploring interactions between TCP and router queueing policies (RED) and TCP and wireless networking (Snoop). Random Early Detection (RED) queue management suggests that routers should detects incipient congestion (before running out of buffer capacity) and signal the source [16]. Early work on RED began on an ancestor of ns; RED is now a standard part of the simulator. Snooping proposes that TCP performance can be improved if routers replay TCP segments lost due to transmission failure over a wireless hop [5]. Both of these approaches benefitted from the rich ns protocol library.

Protocols Investigated With Ns (sidebar)

Ns has been used to develop and investigate a number of protocols:

- TCP behavior: selective acknowledgements, forward acknowledgments, explicit congestion notification, rate-based pacing, over asymmetric links (satellite)
- router queuing policies: random early detection, explicit congestion notification, class based queueing

- multicast transport: Scalable Reliable Multicast (SRM) and variants (RPM, scalable session messages), PIM variants, router support for multicast, congestion control, protocol validation and testing, reliable multicast
- multimedia: layered video (RLM), audio and video quality-of-service, transcoding
- wireless networking: SNOOP and split-connection TCP, multi-hop routing protocols
- protocol response to topology changes
- application-level protocols: web cache consistency protocols

References to some specific papers can be found in the text and at the web page http://www-mash.cs.berkeley.edu/ns/ns-research.html. As an example of ns's use in networking community, it was the most commonly used simulator at SIGCOMM '98.

6 Evaluation

The VINT effort has benefited from the contributions of a wide number of users. The project itself spans four geographically-dispersed groups of developers, and the user community includes more than 200 institutions world-wide (based on messages posted to the mailing list). Ns includes a large amount of code contributed from this user community. Currently, we have two mechanisms for adding contributed code from users: we can point to the contribution on a "Contributed Code" web page, or we can incorporate the contributed code into the main ns distribution (typically with documentation and a validation test program). Code integrated into the main distribution will track ns as it evolves; experience stresses the importance of the automated validation tests in this process.

Although the ns user community has been steadily growing, there will always be times when a researcher finds it more convenient to write stand-alone code or to choose an alternative general-purpose simulator. A custom simulator can address exactly the problem faced by a researcher. Even though ns's abstraction techniques allow two orders of magnitude scaling, a researcher's custom simulator can get exactly the correct scaling behavior. Finally, a new simulator will avoid the cost of learning ns. However, we have found that researchers often underestimate the amount of infrastructure required to build a new simulator and interpret its results.

Wide use of a common simulation platform provides some very serendipitous effects, however. By providing a rich collection of alternatives and variants for frequently used functionality (e.g. for TCP and queueing variants), ns encourages researchers to incorporate these alternatives into the parameter space of their own simulations. Without the infrastructure of ns or a similar environment, it seems unlikely researchers would be able to cover such a rich parameter space due to the additional cost of developing such infrastructure. This is particularly true of experimental new approaches. For example, RED queue management in ns has been widely used in a range of simulations well before it was standardized and available in products. This availability has helped understanding and acceptance of RED and helped other researchers anticipate how their protocols will behave in future networks.

A disadvantage of ns is that it is a large system with a relatively steep initial learning curve. Availability of a tutorial (contributed by Marc Greis) and continuing evolution of the ns documentation has improved the situation, but ns's split programming model remains a barrier to some developers. As described in "Software Architecture", the choice of the fine-grain object decomposition is intentional because it allows two levels of programming. Simple scripts, topology layout, and parameter variation can often be done exclusively in OTcl. Although C++ is required to implement most new protocols, ns's object-oriented structure makes it fairly easy to implement variants of existing protocols. For completely new protocols, the large set of existing modules promotes re-use by the advanced programmer as is evident in ns' existing protocols and classes.

7 Conclusions

Simulation in network research plays the valuable role of providing an environment in which to develop and test new network technologies without the high cost and complexity of constructing testbeds. While not a complete replacement for testbeds, a standard framework for simulation used by a diverse set of researchers increases the reliability and acceptance of simulation results. Despite the benefits of a common framework, the network research community has largely developed individual simulations targeted at specific studies due to the considerable effort required to construct a general-purpose simulator. Because of the special purpose nature of such simulators, studies based on them often do not reflect the richness of experience derived from experimentation with a more extensive set of traffic sources, queuing techniques, and protocol models.

The VINT project, using ns as its simulator base and nam as its visualization tool, has constructed a common simulator containing a large set of models for use in network research. By including algorithms still in the research phase of development, users of the simulator are able to explore how their particular work interacts with these future techniques. Furthermore, because of the many protocols and models included with the system, researchers are often able to modify and construct their own simulations based on the provided models with relative ease. In several cases, modules developed outside the VINT project have been incorporated as a standard component to the simulator. We intend to further foster such contributions, and expect them to increase in the future.

While the VINT project so far has been relatively successful in achieving its goals, it remains to be seen how well the VINT project and the ns simulator will address the challenges of building on this success. The VINT project is an ongoing experiment in providing and using a multi-protocol simulator that allows researchers in the network research community to more easily build on each others' work. Future challenges for the VINT project include the development of mechanisms for the successful integration of code contributed by the user community, reducing the learning curve for using ns, further developing tools for large-scale simulations with a diverse traffic mix, and providing tools for newer areas of research such as mobility and higher-level protocols.

References

[1] Ahn, J., Danzig, P. B., Liu, Z., and Yan, L. Evaluation of TCP Vegas: Emulation and experiment. In *Proceedings of the ACM SIGCOMM* (Cambridge, Massachusetts, Aug. 1995), ACM, pp. 185–195.

[2] Ahn, J.-S., Danzig, P., Estrin, D., and Timmerman, B. Hybrid technique for simulating high bandwidth delay computer networks. In *Proceedings of the ACM SIGMETRICS* (Santa Clara, CA, USA, May 1993), ACM, pp. 260–261.

[3] Bagrodia, R. L., and Liao, W.-T. Maisie: A language for the design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering 20*, 4 (Apr. 1994), 225–238.

[4] Bajaj, S., Breslau, L., and Shenker, S. Uniform versus priority dropping for layered video. In *ACM SIGCOMM* (Sept. 1998), pp. 131–143.

[5] Balakrishnan, H., Padmanabhan, V., Seshan, S., and Katz, R. A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of the ACM SIGCOMM '96* (Stanford, CA, Aug. 1996), ACM, pp. 256–269.

[6] Brakmo, L., and Peterson, L. Experiences with network simulation. In *Proceedings of the ACM SIGMETRICS* (1996), ACM.

[7] Calvert, K., Doar, M., and Zegura, E. W. Modeling Internet topology. *IEEE Communications Magazine 35*, 6 (June 1997), 160–163.

[8] Danzig, P. B., Jamin, S., Cáceres, R., Mitzel, D. J., and Estrin, D. An empirical workload model for driving wide-area TCP/IP network simulations. *Journal of Internetworking: Research and Experience 3*, 1 (Mar. 1992), 1–26.

[9] DeLucia, D., and Obraczka, K. A multicast congestion control mechanism using representatives. Tech. Rep. USC-CS TR 97-651, Department of Computer Science, University of Southern California, May 1997.

[10] Desbrandes, F., Bertolotti, S., and Dunand, L. OPNET 2.4: an environment for communication network modeling and simulation. In *Proceedings of the European Simulation Symposium* (Delft, Netherlands, Oct. 1993), Society for Computer Simulation, pp. 609–614.

[11] Dupuy, A., Schwartz, J., Yemini, Y., and Bacon, D. NEST: A network simulation and prototyping testbed. *Communications of the ACM 33*, 10 (Oct. 1990), 64–74.

[12] Estrin, D., Handley, M., Heidemann, J., McCanne, S., Xu, Y., and Yu, H. Network visualization with the VINT network animator nam. Tech. Rep. 99-703, University of Southern California, Mar. 1999.

[13] Fall, K., and Floyd, S. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review 26*, 3 (July 1996).

[14] Floyd, S. TCP and explicit congestion notification. *ACM Computer Communication Review 24*, 5 (Oct. 1994), 10–23.

[15] Floyd, S. Simulator tests. From ftp://ftp.ee.lbl.gov/papers/simtests.ps.Z, Oct 1996.

[16] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *ACM/IEEE Transactions on Networking 1*, 4 (Aug. 1993), 397–413.

[17] Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S., and Zhang, L. A reliable multicast framework for light-weight sessions and application level framing. *ACM/IEEE Transactions on Networking 5*, 6 (Dec. 1997).

[18] Hänle, C. A comparison of architecture and performance between reliable multicast protocols over the MBone. Master's thesis, Institute of Telematics, University of Karlsrühe, 1997.

[19] Helmy, A., and Estrin, D. Simulation-based 'STRESS' Testing Case Study: A Multicast Routing Protocol. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Montreal, Canada, July 1998), IEEE, pp. 36–43.

[20] Helmy, A., Estrin, D., and Gupta, S. Fault-oriented test generation for multicast routing protocol design. *Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE/PSTV'98), IFIP TC6/WG6.1 Join International Conference* (Nov. 1998), 93–109.

[21] Huang, P., Estrin, D., and Heidemann, J. Enabling large-scale simulations: selective abstraction approach to the study of multicast protocols. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Montreal, Canada, July 1998), IEEE, pp. 241–248.

[22] Keshav, S. REAL: a network simulator. Tech. Rep. 88/472, University of California, Berkeley, Dec. 1988.

[23] Kesidis, G., and Walrand, J. Quick simulation of atm buffers with on-off multiclass markov fluid sources. *ACM Transactions on Modeling and Computer Simulations 3*, 3 (July 1993), 269–276.

[24] Mah, B. An empirical model of HTTP network traffic. In *Proceedings of the IEEE Infocom* (Kobe, Japan, Apr. 1997), IEEE.

[25] Mah, B. A. *INSANE Users Manual*. The Tenet Group Computer Science Division, University of California, Berkeley 94720, may 1996. `http://HTTP.CS.Berkeley.EDU/~bmah/Software/Insane/InsaneMan.ps`.

[26] Mathis, M., and Mahdavi, J. Forward acknowledgement: Refining TCP congestion control. In *Proceedings of the ACM SIGCOMM '96* (Stanford, CA, Aug. 1996), ACM, pp. 281–291.

[27] McCanne, S., Brewer, E., Katz, R., Rowe, L., Amir, E., Chawathe, Y., Coopersmith, A., Mayer-Patel, K., Raman, S., Schuett, A., Simpson, D., Swan, A., Tung, T.-L., Wu, D., and Smith, B. Toward a common infrastructure for multimedia-networking middleware. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (St. Louis, Missouri, May 1997), IEEE, pp. 39–49.

[28] McCanne, S., Jacobson, V., and Vetterli, M. Receiver-driven layered multicast. In *ACM SIGCOMM* (Stanford, CA, U.S.A., Aug. 1996), pp. 117–130.

[29] Mikler, A. R., Wong, J. S. K., and Honavar, V. An object oriented approach to simualting large communication networks. *Journal of Systems Software 40* (1998), 151–164. huang folder: general simulator.

[30] Ousterhout, J. K. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, 1994.

[31] Perumalla, K., Fujimota, R., and Ogielski, A. TED—a language for modeling telecommunication networks. *ACM SIGMETRICS Performance Evaluation Review 25*, 4 (Mar. 1998).

[32] Raman, S., McCanne, S., and Shenker, S. Asysmptotic scaling behavior of global recovery in SRM. In *Proceedings of the ACM SIGMETRICS* (Madison, WI, USA, June 1998), ACM, pp. 90–99.

[33] Rizzo, L. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review 27*, 1 (Jan. 1997).

[34] Sharma, P., Estring, D., Floyd, S., and Zhang, L. Scalable session messages in SRM. Sumbitted to infocom '98?, USC technical report, Aug. 1997.

[35] Visweswaraiah, V., and Heidemann, J. Improving restart of idle TCP connections. Tech. Rep. 97-661, University of Southern California, Nov. 1997.

[36] Wetherall, D., and Linblad, C. J. Extending Tcl for dynamic object-oriented programming. In *Proceedings of the USENIX Tcl/Tk Workshop* (Toronto, Ontario, July 1995), USENIX, p. 288.

[37] Willinger, W., Taqqu, M., Sherman, R., and Wilson, D. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. In *Proceedings of the ACM SIGCOMM* (Cambridge, Massachusetts, Aug. 1995), ACM, pp. 100–113.