

 Open access • Journal Article • DOI:10.1109/2.841785

Advances in network simulation — Source link





Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd ...+7 more authors

Published on: 01 May 2000 - IEEE Computer (IEEE Computer Society Press)

Topics: Network simulation, Network topology, Communications protocol, The Internet and Internet Protocol

Related papers:

- [Ad-hoc on-demand distance vector routing](#)
- [A performance comparison of multi-hop wireless ad hoc network routing protocols](#)
- [Dynamic Source Routing in Ad Hoc Wireless Networks](#)
- [Highly dynamic Destination-Sequenced Distance-Vector routing \(DSDV\) for mobile computers](#)
- [Scalability and accuracy in a large-scale network emulator](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/advances-in-network-simulation-2zy8yfyp2w>

Advances in Network Simulation

Network researchers must test Internet protocols under varied conditions to determine whether they are robust and reliable. The Virtual InterNetwork Testbed (VINT) project has enhanced its network simulator and related software to provide several practical innovations that broaden the conditions under which researchers can evaluate network protocols.

*Lee Breslau
Deborah Estrin
Kevin Fall
Sally Floyd
John Heidemann
Ahmed Helmy
Polly Huang
Steven McCanne
Kannan Varadhan
Ya Xu
Haobo Yu*
The VINT Project

The Internet's rapid growth has spurred development of new protocols and algorithms to meet changing operational requirements—such as security, multicast transport, mobile networking, policy management, and quality-of-service support. Development and evaluation of these operational tools requires answering many design questions.

Despite their value, custom simulators, wide-area testbeds, and small-scale lab evaluations all have drawbacks. Because they use real code, experiments run in testbeds or labs automatically capture important details that might be missed in a simulation. However, building testbeds and labs is expensive, reconfiguring and sharing them is difficult, and they are relatively inflexible. Further, reproducing some networking phenomena, such as wireless radio interference, can be difficult, complicating efforts 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 or static assumptions. High start-up costs prevent an individual group from acquiring the resources needed to create a comprehensive and advanced networking simulation environment. This limitation often causes simulations constructed by different groups to lack standardization and reproducibility.

In the current paradigm, directly comparable data would be available only if designers implemented all competing mechanisms within every simulator. Because few research groups have the resources to do this, it is often most effective to have those who know the most about the particular protocol construct the component.

ADVANTAGES OF A COMMON SIMULATOR

Multiprotocol network simulators provide a rich opportunity for efficient experimentation. Disparate research efforts using a common simulation environment can yield substantial benefits, including

- 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 Virtual InterNetwork Testbed (VINT) project provides improved simulation tools for network researchers to use in the design and deployment of new wide-area Internet protocols. An understanding of the VINT simulation framework and its principal tool, ns, the network simulator, can help researchers determine how best to proceed with their efforts.

SIMULATION NEEDS OF NETWORK RESEARCHERS

Simulation evaluates network protocols under varying network conditions. Studying protocols—both individually and interactively—under varied conditions is critical to understanding their behavior and characteristics. The VINT project has used the ns simulator and related software to provide several practical innovations that broaden the conditions under which researchers can evaluate protocols:

- *Abstraction.* Varying simulation granularity allows a single simulator to accommodate both

detailed and high-level simulations. Researchers study networking protocols at many levels, ranging from the detail of an individual protocol to the aggregation of multiple data flows and the interaction of multiple protocols. The abstraction mechanisms in ns allow researchers to examine protocols without changing simulators and to validate abstractions by comparing detailed and abstract results.

- *Emulation.* Most simulation experiments are confined to a single simulated world that employs only the protocols and algorithms in the simulator. In contrast, emulation allows a running simulator to interact with operational network nodes.
- *Scenario generation.* Testing protocols under appropriate network conditions is critical to achieving valid, useful results. Automatic creation of complex traffic patterns, topologies, and

Ns and Other Related Simulators

Network simulation has a long history. Ns itself derives from REAL (Realistic and Large),¹ which derives from NEST (Network Simulation Testbed).² Although we cannot list all relevant network simulators here, we can describe the distinguishing features of network simulators and compare prominent examples with ns.

Distinguishing features

Simulators have widely varying focuses. Many target a specific area of research interest—a particular network type or protocol, for example, such as ATM or PIM multicast. Others, including ns, REAL, Opnet, and Insane,³ target a wider range of protocols. The most general of these simulators provide a simulation language with network protocol libraries (for example, Maisie⁴ and Opnet). Very focused simulators model only the details relevant to the developer.

Ns and other network simulators use a discrete-event processor as their engine. Researchers have adopted several complementary approaches to improve accuracy, performance, or scaling. Some simulators augment event processing with analytic models of traffic flow or queuing behavior for better performance or accuracy (for example, fluid network approximations).⁵

Parallel and distributed simulation provide a second way to improve performance. Several simulators support multiprocessors or networks of workstations.^{1,4,6} Although ns focuses only on sequential simulation, the TeD effort has parallelized some ns modules.

Abstraction

Abstraction is a 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.⁷ Ns supports several levels of abstraction.

Numerous simulation interfaces are possible, including programming in a high-level scripting language, a more traditional systems language, and sometimes both. Some systems, such as x-Sim⁸ and Maisie,⁴ focus on allowing the same code to run in simulation and on a live network. Most systems have augmented programming with a GUI shell of some kind. Ns provides a split-level programming model in which packet processing is done in a systems language while simulation setup is done in a scripting language. Nam,⁹ currently being enhanced to support simple scenario editing, provides visualization output.

Network emulators

Early work in network emulation included the use of “flakeways” (gateways that could alter or drop packets) for early TCP/IP tests. More recent work includes special-purpose stand-alone network emulators supporting packet delays and drops.¹⁰ Developers usually implement these systems as kernel drop-in modules that intercept the IP layer packet-forwarding path, and thus look like routers to end stations. Their capabilities are generally limited to simple packet manipulation and do not provide for interference from simulated cross traffic. Moreover, these systems do not include the general simulation capability that ns provides.

References

1. S. Keshav, *REAL: A Network Simulator*, tech. report 88/472, Univ. California, Berkeley, 1988.
2. A. Dupuy et al., “NEST: A Network Simulation and Prototyping Testbed,” *Comm. ACM*, Oct. 1990, pp. 64-74.
3. B.A. Mah, *Insane Users Manual*, The Tenet Group Computer Science Division, Univ. California, Berkeley, 1996.
4. R.L. Bagrodia and W.-T. Liao, “Maisie: A Language for the Design of Efficient Discrete-Event Simulations,” *IEEE Trans. Software Eng.*, Apr. 1994, pp. 225-238.
5. G. Kesidis and J. Walrand, “Quick Simulation of ATM Buffers with On-Off Multiples Markov Fluid Sources,” *ACM Trans. Modeling and Computer Simulations*, July 1993, pp. 269-276.
6. K. Perumalla, R. Fujimota, and A. Ogielski, “TED: A Language for Modeling Telecommunication Networks,” *ACM SIGMETRICS Performance Evaluation Rev.*, Mar. 1998, pp. 4-11.
7. J.-S. Ahn et al., “Hybrid Technique for Simulating High-Bandwidth Delay Computer Networks,” *Proc. ACM SIGMETRICS*, ACM Press, New York, 1993, pp. 260-261.
8. L. Brakmo and L. Peterson, “Experiences with Network Simulation,” *Proc. ACM SIGMETRICS 96*, ACM Press, New York, 1996, pp. 80-90.
9. D. Estrin et al., *Network Visualization with the VINT Network Animator Nam*, tech. report 99-703, Dept. Computer Science Univ. Southern California, Los Angeles, 1999.
10. J. Ahn et al., “Evaluation of TCP Vegas: Emulation and Experiment,” *Proc. ACM SIGCOMM 95*, ACM Press, New York, 1995, pp. 185-195.

dynamic events (link failures) can help generate such scenarios.

- *Visualization.* Researchers need tools that help them understand the complex behavior in network simulation. Merely providing tables of summary performance numbers does not adequately describe a network's behavior. Visualization using the network animation tool *nam* provides a dynamic representation that allows researchers to develop better protocol intuition and aids in protocol debugging.¹
- *Extensibility.* The simulator must be easy to extend if its users are 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.

Engineering issues also bear on a simulator's usability. A wide range of protocol modules must be available in the simulator. This breadth allows easy comparison of different approaches and reduces simulation development time, enabling the researcher to focus on the simulation aspects relevant to the design question being studied.

The need to compare new network variants demands validated protocols. *Ns* validates other protocols to the degree their maturity warrants. The many protocol modules in *ns* and the interactions among them mandate mechanisms that prevent modifications in one module from breaking functionality in another. To this end, *ns* includes many automated test suites that keep unintentional changes in behavior from creeping into the simulator.

VINT AND NS

The VINT project has developed *ns* as a common simulator with advanced features to change current protocol engineering practices by enabling the study of protocol interactions and scaling. Public distribu-

tion of our system has helped reduce duplication of effort within the networking research and development community. The "Ns and Other Related Simulators" sidebar puts *ns* into perspective. *Ns* is publicly available at <http://www-mash.cs.berkeley.edu/ns/>.

Abstracting simulation

Computer resource limitations, such as memory and processing time, often limit the number of network objects (nodes, links, and protocol agents) that designers can simulate at the packet level. A scalable network simulator accommodates wide variations in each kind of network object, data in transit, and information collected. Designers use three complementary approaches in 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 under way (see the "Protocols Investigated with *Ns*" sidebar). VINT's efforts focus on a complementary approach, tuning implementation, and providing multiple levels of protocol abstraction. Eliminating less important details can yield substantial savings while preserving the model's basic validity.

Ns provides several levels of abstraction:

- 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.²
- Algorithmic routing replaces shortest-path routing with tree-based routing, transforming $O(n \log n)$ memory requirements to $O(n)$.

Protocols Investigated with Ns

Researchers have used *ns* to develop and investigate the following protocols:

- *TCP behavior:* selective acknowledgment, forward acknowledgment, explicit congestion notification, rate-based pacing, asymmetric links (satellite)
- *Router queuing policies:* random early detection (RED), explicit con-

gestion notification (ECN), class-based queuing

- *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, audio and video quality-of-service, transcoding

- *Wireless networking:* Snoop and split-connection TCP, multihop routing protocols
- Protocol response to topology changes
- *Application-level protocols:* Web cache consistency protocols

For more details about this research, see <http://www-mash.cs.berkeley.edu/ns/ns-research.html>.

Each abstraction sacrifices some details to save memory, so the user must apply abstractions only when appropriate. Users can trade simulator performance for packet-level accuracy by adjusting the level of simulation abstraction. Increasing the simulation abstraction level permits increasingly large simulations, while decreasing it yields a more realistic simulation. The session-level simulator can abstract many details of links, nodes, and crosstraffic. Users can run simulations side by side in both detailed and session-level modes to compare performance and accuracy across different levels of abstraction. Figure 1 shows the memory savings possible from session-level simulations for a particular scenario with large multicast groups.

The risk in abstraction is simulation accuracy. The degree to which abstraction sacrifices accuracy and the impact of this sacrifice on the validity of the results vary greatly among simulation scenarios. For exam-

ple, although the details of a particular medium's approach to segmentation and reassembly are important for LAN simulations, the link's packet-loss rate for higher-level WAN simulations can reflect the details adequately.

Figure 2 shows how VINT validates small-scale simulations before projecting results at larger scales to ensure that abstraction does not substantially alter simulation results.² A quantitative analysis of scalable reliable multicast (SRM) performance across detailed and session-level simulations suggests that although the timing of individual SRM events varies, average aggregate behavior changes by only 3 to 9 percent. We are also working on hybrid abstractions in which different portions of the same simulation operate in detailed session levels of abstraction.

Emulation interface

Ns includes an emulation interface that permits network traffic to pass between real-world network nodes and the simulator. Together with the simulator's tracing and visualization facilities, emulation provides a powerful analytical tool for evaluating the dynamic behavior of protocols and their implementations in end systems.

Figure 3 shows how emulation scenarios interpose the simulator as an intermediate node (or end node) along an end-to-end network path. This passes live network traffic through the simulation, allowing it to experience complex dynamics, such as crosstraffic. The simulator's scheduler is synchronized in real time and allows the simulated network to emulate its real-world equivalent.

Beyond conventional simulation, emulation is also useful in evaluating both end-system and network-element behavior. Researchers use emulation to introduce packet dynamics—for example, drops, reordering, and delays—to end-system protocol implementations. Reproducing these conditions reliably in a live network is difficult.

Furthermore, researchers can capture traces of live traffic injected into the simulation environment and use visualization tools to evaluate the end system's responses. Conversely, researchers can evaluate network element behavior (for example, a queuing or packet-scheduling discipline) in relation to live traffic that real-world end stations generate. Such simulations help identify undesirable network-element behavior before deployment in live networks.

The VINT 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 that uses a version of the SRM protocol that the MASH toolkit supports.³ The simulator is

Figure 1. Memory saving occurs through session-level abstraction as the number of multicast group members changes.

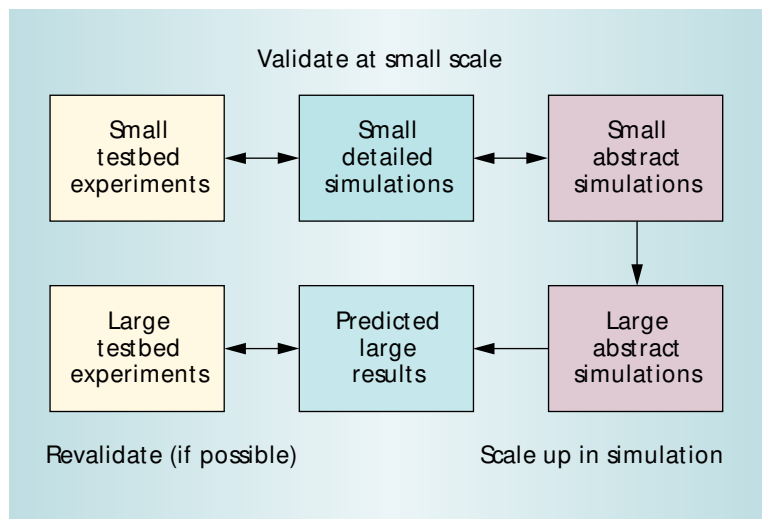
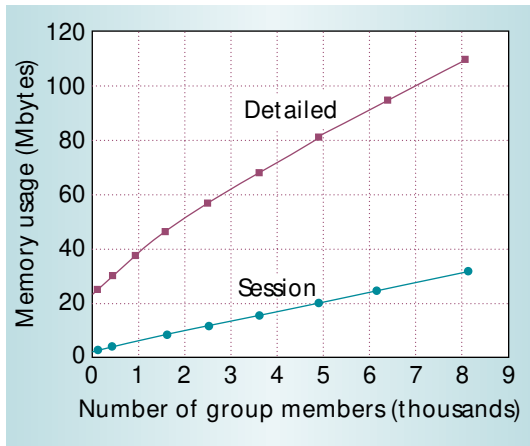


Figure 2. VINT validates small-scale simulations to ensure that abstraction does not substantially alter results at larger scales.

placed between groups of live end stations communicating via SRM.

Multicast traffic passing between groups must traverse the simulator and be exposed to the dynamics of its simulated network. Visualization of traces taken in the simulation environment reveals end-station retransmissions triggered by dropped or delayed packets. This method has helped pinpoint time-dependent MediaBoard behaviors that are otherwise difficult to diagnose.

Scenario generation

Simulation scenarios require

- network topologies that define links and their characteristics,
- traffic models that specify sender and receiver locations and demands, and
- network dynamics that include node and link failures.

Ns meets these needs by providing a library of predefined topologies and using packages such as the Georgia Tech models (GT-ITM) to generate random topologies. Traffic models benefit from ns's rich library of protocols. VINT supports widely used topology models for Telenet and Web traffic. Ns composability is also important here; VINT has assembled several constant-bit-rate sources to simulate layered multicast video (<http://www-mash.cs.berkeley.edu/ns/ns-topogen.html>).

Automatic scenario generation in ns plays an important role in the STRESS (Systematic Testing of Robustness by Evaluation of Synthesized Scenarios) approach to systematic protocol testing.⁴ STRESS automatically generates test scenarios to explore protocol correctness. This approach has discovered several design errors in multicast routing and is now being used for performance evaluation.

SOFTWARE ARCHITECTURE

Ns software promotes extension by users. The fundamental abstraction the software architecture provides is "programmable composability." This model expresses simulation configurations 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. Adopting a full-fledged programming model for simulation configuration lets the experimentalist 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.

The split-programming model

Rather than adopting a single programming lan-

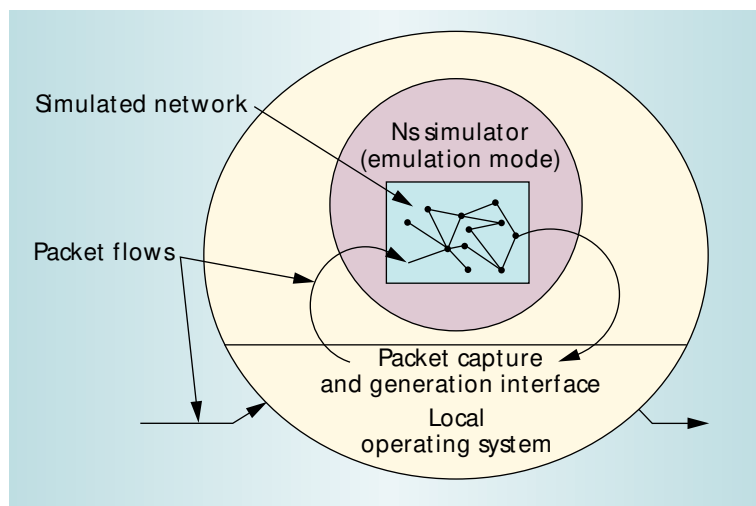


Figure 3. Emulation allows live network traffic to pass through the simulator.

guage that defines a monolithic simulation, we have found that different simulations require different programming models. The goal is to provide adequate flexibility without unduly constraining performance. In particular, tasks such as 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 expressing an implementation in a compiled language such as C++.

On the other hand, tasks such as 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 such as Tcl.⁵

To this end, ns exploits a split-programming model. C++ implements the simulation kernel—the core set of high-performance simulation primitives—but the Tcl scripting language expresses the definition, configuration, and control of the simulation.

This split-programming approach can be a boon to long-term productivity. It cleanly separates the burden of simulator design, maintenance, extension, and debugging from the simulation's goal—the actual research experiments—by providing the simulation programmer with an easy-to-use, reconfigurable, programmable simulation environment. Moreover, ns allows 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.

Virtues of split programming

Implementing fine-grained simulation objects in C++ and combining them with Tcl scripts yields more-powerful, higher-level macro-objects. For example, a

Ns is an ideal virtual testbed for comparing protocols because it offers a publicly available simulator with a large protocol library.

simulated router is composed of demultiplexers, queues, packet schedulers, and so forth. The split-programming approach allows faithful simulation of a range of routers. We can configure or arrange the low-level demultiplexers, queues, and schedulers to model an IP router, perhaps with multicast forwarding support, or arrange them instead into a configuration that models a high-speed switch with a new scheduling discipline. In the latter case, protocol agents (implemented entirely in Tcl) that model an experimental signaling protocol could easily extend the switch.

Performance also guides our use of split programming. The model implements low-level, event-level operations such as route lookups, packet forwarding, and TCP protocols in C++, whereas it implements high-level control operations such as aggregate statistics collection, modeling of link failures, route changes, and low-rate control protocols in Tcl.

Obtaining a desirable trade-off between performance and flexibility requires careful design. This division often migrates during the course of investigating a protocol. Object-oriented design naturally expresses this composable macro-object model. However, when we designed ns, Tcl did not support object-oriented programming constructs and it did not provide effective programming constructs for building reusable modules.

Thus, VINT adopted an object-oriented extension of Tcl. Of the several Tcl object extensions available, we chose the Object Tcl (OTcl) system from MIT because it didn't require any changes to the Tcl core and it has a particularly elegant, yet simple, design.⁶ We also adopted Tcl with Classes (TclCL), a simple extension of OTcl that provides object scaffolding between C++ and OTcl. This extension facilitates use of our split-programming model to divide an object's implementation across the two languages.

With the OTcl programming model in place, each macro-object becomes an OTcl class; a simple-to-use set of object methods hides its complexity. Moreover, the model can embed macro-objects in 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, whereas the low-level objects represent components such as demultiplexers and queues. This approach frees the simulation designer to operate at various levels of abstraction:

- *high-level*—for example, by simply creating and configuring existing macro-objects;
- *mid-level*—for example, by modifying the behavior of an existing macro-object in a derived subclass; or

- *low-level*—for example, by introducing new macro-objects or splitting 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 users can implement their simulation at the highest level of abstraction that supports the required level of flexibility, thus minimizing exposure to unnecessary details and the burden associated with them.

TYPES OF NS RESEARCH

Network research simulation categories represent three broad themes:

- selecting a mechanism from among several options,
- exploring complex behavior, and
- investigating unforeseen multiple-protocol interaction.

The following examples from the broad base of ns-based simulations in the networking community illustrate each of these themes.

Selecting a mechanism

As in most design activities, much time is lost in selecting which alternative to use in accomplishing a goal. Researchers have used ns to develop TCP variants and extensions, explore reliable multicast protocols, and consider packet-scheduling algorithms in routers.

A simulator-specific TCP implementation aided the efforts to use ns to explore TCP variants and extensions, such as selective acknowledgments, forward acknowledgments, explicit congestion notification, and pacing. Omitting application-specific baggage such as memory management and IP fragmentation allows ns users to focus on research issues such as packet retransmission policies and throughput.

Exploring complex behavior

Complex behavior often appears as unexpected self-organization of dynamic systems, including examples such as

- 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 because of RTT variations,
- contention for bandwidth reservations, and
- “ACK implosion” for large-scale reliable multicast protocols.

Simulation has proved useful in helping to identify and understand each of these phenomena.

Error recovery in SRM technology is an example of ns's exploration of complex behavior.⁷ SRM supports reliable communication for large groups. It uses a probabilistic-based negative acknowledgment (NACK) protocol to achieve reliability. A receiver detecting a loss multicasts NACK to the group. Each group member with the missing data prepares to repair the error. To avoid repair implosion (everyone sending the repair at once), SRM delays repairs by a random time proportional to the estimated distance between the participants.

Although the original SRM simulations used stand-alone simulation, we added an SRM implementation to ns. Researchers have since used the implementation to study SRM recovery behavior over a wide range of variants and topologies.^{6,8}

Comparing research results

Comparing a new protocol design against existing protocols is a common research challenge. Comparisons of full protocols are often difficult because they require a particular operating system or they are not widely available. Ns is an ideal virtual testbed for comparing protocols because it offers a publicly available simulator with a large protocol library.

The reliable multicast community has used ns widely for protocol comparison. In addition to the SRM variants previously described, Christophe Hänle used ns to compare the multicast file transfer protocol,⁹ and Dante DeLucia used it to research representative-based congestion control.¹⁰

Multiple protocol interactions

Multiprotocol interactions include either

- the impact of protocol operation at one layer upon another layer (for example, HTTP on TCP), or
- the interaction of unrelated protocols (for example, the effect of uncontrolled traffic sources on congestion-controlled traffic flows).

Studying protocol interactions requires twice the effort that studying a single protocol does: The designer must understand and implement protocols at all relevant layers. Ns reduces this effort by providing a validated library of important protocols.

Random early detection (RED) and TCP Snoop are two examples where ns greatly aided protocol studies. RED explored interactions between TCP and router queuing policies; Snoop explored interactions between TCP and wireless networking. RED queue management suggests that routers should detect incipient congestion (before running out of buffer capacity) and signal the source.¹¹ Early work on RED began on an ancestor of ns; RED is now a standard part of the

simulator. Connection snooping proposes that TCP performance can be improved if routers replay TCP segments lost because of transmission failure over a wireless hop.¹² Both approaches benefited from the rich ns protocol library.

EVALUATION

Many users have contributed to the VINT effort. The project spans four geographically dispersed developer groups. Messages posted to the mailing list indicate that the user community includes more than 200 institutions worldwide, and ns incorporates much code contributed from this user community.

Currently, users can contribute code in two ways: on a contributed-code Web page or through incorporation 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 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 exactly address the problem a researcher faces. Although ns's abstraction techniques allow two-orders-of-magnitude scaling, a custom simulator can get exactly the correct scaling behavior. Also, a new simulator will avoid the cost of learning ns. However, we have found that researchers often underestimate the infrastructure required to build a new simulator and interpret its results.

Wide use of a common simulation platform provides serendipitous effects, however. Ns encourages researchers to incorporate its rich collection of alternatives and variants for frequently used functions—for example, for TCP and queuing variants—into the parameters of their own simulations. Without ns or a similar environment, the additional cost of developing the required infrastructure would likely prevent researchers from delving so deeply.

This benefit particularly applies to experimental new approaches. For example, RED queue management in ns was widely used in many simulations well before it was standardized and available in products. This availability has helped develop understanding and acceptance of RED, and it has helped other researchers anticipate how their protocols will behave in future networks.

On the other hand, a disadvantage of ns is that it is a large system with a relatively steep initial learning curve. A tutorial contributed by Marc Greis and the continuing evolution of ns documentation have improved the situation, but ns's split-programming model remains a barrier to some developers.

Ns's object-oriented structure makes it fairly easy to implement variants of existing protocols.

We chose ns's fine-grain object decomposition intentionally because it allows two levels of programming. Simple scripts, topology layout, and parameter variation can often be done exclusively in OTcl. Although developers must use C++ 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 reuse by advanced programmers, as evidenced in ns's existing protocols and classes.

Simulation plays a valuable role in network research. A diverse set of researchers using a standard framework 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. Because of the focused nature of such simulators, studies that employ them often do not reflect the breadth of experience that can result if experimenters use 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 undergoing research, simulator users can explore how their particular work interacts with these future techniques. In several cases, we have incorporated modules developed outside the VINT project as standard simulator components.

Although the VINT project has so far been relatively successful, it and the ns simulator must address more challenges, such as

- developing mechanisms for the successful integration of code from the user community,
- reducing the ns user's learning curve,
- 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. *

Acknowledgments

We thank the following for their contributions to ns and this article: Sandeep Bajaj, Padma Halder, Mark Handley, Tom Henderson, Satish Kumar, Giao Nguyen, Reza Rejaie, Nader Saleki, Scott Shenker, Puneet Sharma, and Daniel Zappala, and the CMU Monarch project. A more complete list of ns contributors can be found at <http://www-mash.cs.berkeley.edu/ns/ns-contributors.html>. This research is supported by the Defense Advanced Research Projects Agency through the VINT project at the LBNL under

DARPA Order E243; at USC/ISI under DARPA Grant ABT63-96-C-0054; and at Xerox PARC under DARPA Grant DABT63-96-C-0105.

References

1. D. Estrin et al., *Network Visualization with the VINT Network Animator Nam*, tech. report 99-703, Computer Science Dept., Univ. Southern California, Los Angeles, 1999.
2. P. Huang, D. Estrin, and J. Heidemann, "Enabling Large-Scale Simulations: Selective Abstraction Approach to the Study of Multicast Protocols," *Proc. Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 241-248.
3. S. McCanne et al., "Toward a Common Infrastructure for Multimedia-Networking Middleware," *Proc. 7th Int'l Workshop Network and Operating Systems Support for Digital Audio and Video*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 39-49.
4. A. Helmy and D. Estrin, "Simulation-Based 'STRESS' Testing Case Study: A Multicast Routing Protocol," *Proc. Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 36-43.
5. J.K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, Mass., 1994.
6. D. Wetherall and C.J. Linblad, "Extending Tcl for Dynamic Object-Oriented Programming," *Proc. Usenix Tcl/Tk Workshop*, Usenix, Berkeley, Calif., 1995, p. 288.
7. S. Floyd et al., "A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing," *ACM/IEEE Trans. Networking*, Dec. 1997, pp. 784-803.
8. S. Raman, S. McCanne, and S. Shenker, "Asymptotic Scaling Behavior of Global Recovery in SRM," *Proc. ACM SIGmetrics*, ACM Press, New York, pp. 90-99.
9. C. Hänle, *A Comparison of Architecture and Performance Between Reliable Multicast Protocols Over the Mbone*, master's thesis, Inst. Telematics, Univ. Karlsruhe, Germany, 1997.
10. D. DeLucia and K. Obraczka, *A Multicast Congestion Control Mechanism Using Representatives*, tech. report USC-CS TR 97-651, Computer Science Dept., Univ. Southern California, Los Angeles, 1997.
11. S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *ACM/IEEE Trans. Networking*, Aug. 1993, pp. 397-413.
12. H. Balakrishnan et al., "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links," *Proc. ACM SIGCOMM 96*, ACM Press, New York, 1996, pp. 256-269.

Lee Breslau was a co-principal investigator at Xerox PARC, where his work included admission control and Web cache consistency algorithms. He is currently at AT&T Research.

Deborah Estrin is a VINT co-principal investigator and a professor in the Computer Science Department at the University of Southern California. She is also known for her multicast protocol work.

Kevin Fall, currently at Intel, was a member of the Lawrence Berkeley National Laboratory technical staff, where his ns work included full TCP, emulation, and RED.

Sally Floyd was a co-principal investigator at Lawrence Berkeley National Laboratory. She was a co-developer of ns-1, and her work in ns-2 included TCP, ECN, RED, and the ns TCP test suites. She is currently at ACIRI.

John Heidemann is a computer scientist at the University of Southern California/Information Sciences Institute, where his research includes ns scaling and architecture.

Ahmed Helmy, formerly a graduate student at the University of Southern California, is now a professor in the USC Department of Electrical Engineering. He developed and now leads the STRESS approach to identifying protocol pathologies.

Polly Huang received a PhD from the University of Southern California, where her thesis examined simulation scaling through abstraction. She is currently pursuing postdoctoral studies at ETH Zurich.

Steven McCanne was a primary developer of ns-1 and the ns-2 approach to fine-grained composition. Formerly an assistant professor at UC Berkeley, where he used ns-2 to study reliable multicast, McCanne is now CTO of FastForward Networks.

Kannan Varadhan was formerly a graduate student at the University of Southern California, where his thesis work examined TCP and SRM behavior in the face of network dynamics. He is currently at Lucent Technologies, Bell Labs.

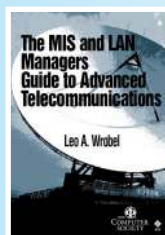
Ya Xu was a staff programmer at the University of Southern California/Information Sciences Institute, where his work on VINT included major enhancements to nam and the wireless protocol stack.

Haobo Yu is a graduate student at the University of Southern California, where his research includes Web and cache consistency protocols and application-level routing. Yu is responsible for the current ns Web model.



How will it all connect?

Find out in



The MIS and LAN Managers Guide to Advanced Telecommunications

\$40 for Computer Society members

Now available from the Computer Society Press



computer.org/cspress/