# Modeling 100,000 Nodes and Beyond: Self-Validating Design

**DARPA/NIST Workshop on Validation of Large Scale Network Simulation Models**

**May 25 – 26, 1999, Reston, VA**

James H. Cowie
Cooperating Systems Corp.
cowie@cooperate.com

David M. Nicol
Dartmouth College
nicol@cs.dartmouth.edu

Andy T. Ogielski
DIMACS and WINLAB, Rutgers
ato@winlab.rutgers.edu

After several years of research, we now have the capability of modeling the Internet at scales on the order of 100,000 multi-protocol hosts (IP, TCP, client/server applications) and routers (OSPF, BGP-4), with long-range correlated IP packet traffic [1]. Getting to this point required the design of a scalable Internet modeling framework (SSFNET), a scalable simulation framework (SSF), and a high–performance parallel discrete-event simulation basis.

Even after factoring in acceptable simplifications, the large size of these Internet models has exposed intrinsic model validation challenges that must be addressed in the design of a suitable modeling framework. Because we plan to explore alternative-future scenarios, in this report we do not address validation of simulated networks against empirical data. Instead, we describe techniques used in SSF for designs that lend themselves to *self-validation*.

**Configuration, Composition, Validation.** Configuring a large heterogeneous network model is very much like bringing up a real network *and* configuring all its hosts and routers at the same time: installing the protocols and applications, and setting all parameters. The difference is that it has to be done quickly, correctly, and automatically, by one modeler instead of by a team of engineers. As a result, network models are traditionally built programmatically – an executable script or program grows a model from its components, line by line, supplying appropriate parameters along the way via constructor arguments and initialization functions.

This is a natural approach; however, in our experience, it scales very poorly beyond small projects. Programmatic model construction breaks down with size, because configuration changes inevitably result in code modification. Applying existing source code control mechanisms to the problem is unsatisfactory and brittle. Changing configurations of interdependent class instances is hard to automate and harder to verify.

What works instead is a modeling framework that decouples configuration data from configuration code. A model is built from a hierarchy of self-configurable classes with assistance of a database. The goal at each design stage is to simplify the class code so that it is both verifiably and intuitively correct. If we can verify the pieces, and verify the methodology used to glue the pieces together into a large model, then we can inductively validate even very large, complex models.

**Self-configuration promotes self-validation.** In SSFNET, for example, each component class (such as a ProtocolSession or ProtocolGraph) is self-configurable via a standard interface method, `config()`, that takes a reference to a queriable configuration database as its argument. Each

component self-configures from a fragment of information retrieved from the database, and can check its correctness. Compositional relations among configurable class instances then naturally mirror the hierarchical organization of the configuration database.

When configurable classes are kept separate from configuration data we find that we gain significant benefits over programmatic model construction. In particular, a configuration database (non-executable data) can be crafted as an independent product, with its own distinct versioning and derivation policies. The modeler need not modify or recompile configurable component code unless the underlying database schema changes. An experimental series takes the form of a sequence of incremental modifications to the configuration database, while the configurable classes stay fixed. Without this basis for comparison, it's difficult to formally compare the output of two successive models in an experimental series.

## Overview: Scalable Simulation Framework

**The Scalable Simulation Framework** is a new open standard for discrete-event simulation of large, complex systems. SSF models are standard-compliant C++ or Java programs; they hide all details of simulator internals (threads, processors, event queues, and synchronization) from the modeler. SSF implementations are available for SGI IRIX, Sun Solaris, DEC/OSF, Linux, and Windows, as well as the Java Virtual Machine. SSF modelers use five base classes: Entity, inChannel, outChannel, Process, and Event. These five classes form a self-contained design pattern for constructing process-oriented, event-oriented, and hybrid simulations. Object frameworks like SSF have significant advantages over new full-custom programming languages: they can be compiled using existing compilers, debugged using existing debuggers, and extended simply by deriving new classes. Additional component layers can be built atop SSF to model specific domains such as radio propagation, factory automation, game theory, or IP networking. These *derivative frameworks* add their own, more specific metaphors to those offered by SSF.

**SSFNET** is the first collection of SSF-based models for simulating Internet protocols and networks. Available in C++ and Java forms, the SSFNET models are open source software, distributed under the GNU General Public License. SSFNET models are *self-configuring* — that is, each SSFNET class instance can autonomously configure by querying a parameter database, which may be locally resident or available over the Web. Configuration data is hierarchically structured, meaning that parent entities pass a fragment of their own configuration information to their children. The configuration database contains its own hierarchical *schemas* – information about the acceptable values and nested structures that may be stored in and retrieved from the database. This strategy automatically gets SSFNET models past the thorniest verification hurdle for large models: the task of applying parameterization consistently and efficiently over the course of an experimental series, while automatically checking for bugs in the structure and typing of configuration data.

**Scalability Experiments** To evaluate scalability, we constructed a sequence of SSFNET models consisting of 10, 20, 50, and 100 Autonomous Systems and a transit domain; they contain, 8,100, 16,100, 40,100, and 80,100 hosts and routers, respectively. The transport is UDP rather than TCP, routing of IP packets is by BGP-4 and OSPF. We considered a simple measure of performance that matters to network modelers: the network event rate, defined as the number of any of packet

production, packet forwarding, or packet reception simulator actions executed per second of wallclock time. Figure 1 shows that SSF achieves scalability in the dimensions that matter: model size and model speed, as implied by a mathematical analysis [2]. Importantly, we found memory consumption, rather than processor cycles, to be the usual limiting factor; increased node complexity actually lends itself to improved parallel performance.

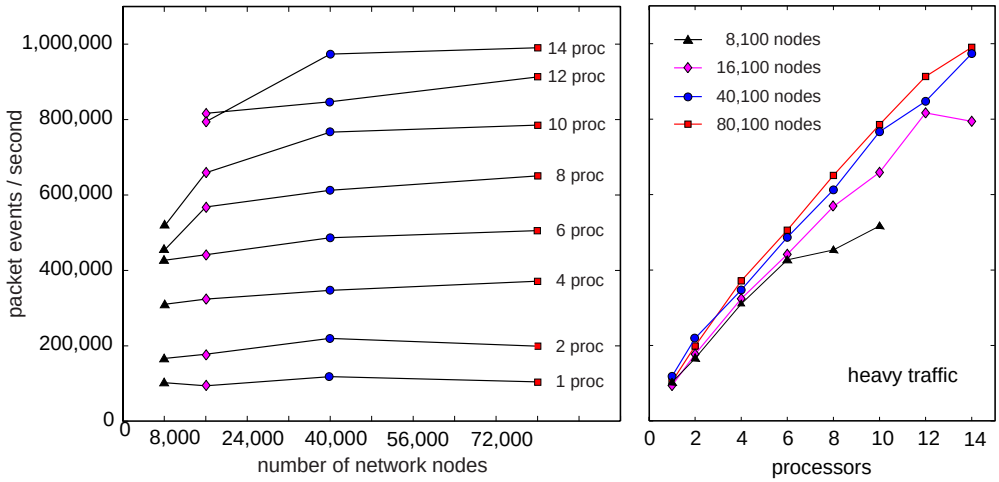


Figure 1: Two complementary views on scalability with model size and with the number of processors under heavy traffic conditions: (a) simulated packet events per wallclock second plotted versus the number of network nodes, for different number of processors used; note the stability of execution rates with increasing model size. (b) the same data plotted versus the number of processors, for different model sizes; note the linear speedup for larger models. Sun Enterprise 4000/Solaris 2.6.

## Future Plans and Open Problems

We hope that the network research community will find SSF and SSFNET useful in pushing the envelope of very large, very complex Internet modeling. Our experience to date suggests that hierarchical composition and configuration techniques are our best bet for inductive validation of models with more than a few dozen nodes, or with very complex node behavior (full network protocol graphs, radio terminals, etc.). However, some critical tools for model validation are still missing in action; in this section we describe some of them.

**Visualization challenges.** Big network models start out as graph problems; big graph problems require scalable design and visualization tools. To understand what's going on in models built from very large graphs, the simple visualizers available today will have to yield to hierarchical (multiresolution) navigation tools. The modeler, the tool builder, and the underlying parallel simulation kernel have to be on the same page when it comes to defining hierarchical strategies for decomposing a very large simulation. None of the available graph drawing packages work at such scales.

**Tools for generation and validation of network topologies.** For reproducing and verifying network simulation results, it's critical to derive a standard methodology for documenting the assumptions and origins of the topologies that were used. We are now designing the next generation

of network topology generators, based on prototyping, compositing and and reuse of "validated exemplars." Existing tools for generating network topologies [4] generate raw graphs without much semantic content. Our tools will annotate generated topologies with validated content (IP addresses, link specifications, router and host protocol configurations).

**Scalable data collection facilities.** SSF supports low-overhead probe processes that can be dynamically attached and detached to areas of interest. Future SSF tools will allow the user to set threshold-based alarms, install probe routines, sample activity at any point within the simulated network, and perform distributed checkpoint/restart with modified instrumentation. These are nontrivial goals in the presence of concurrent multi-timeline execution.

**Multiresolution tools for sensitivity analysis.** Standard statistical methods make assumptions that do not hold in the very large scale networks of interest. Experiments at these very large scales must account for and effectively deal with the main sources of variation at all levels and ignore those aspects of variations that have little or no impact [3]. *Multiresolution analysis* (MRA) formalizes the notion of coarse and fine approximation and the increment in information needed to pass from one resolution to another. Our aim is to demonstrate how ideas from multiresolution analysis can give rise to novel and practically useful approaches for measuring network-specific problems and for analyzing, understanding, and visualizing the temporal, spatial, or hierarchical structure of the resulting data.

## Acknowledgments

We wish to acknowledge the effort put into this project by our students: Xiaowen Liu, Anna Poplawski and Brian Premore at Dartmouth, and Philip Kwok and Hongbo Liu at DIMACS.

This work is partially supported by DARPA Contract N66001-96-C-8530 (Next Generation Internet) and by NSF Grant NCR-9527163 (Special Projects in Networking).

## References

1. James H. Cowie, David M. Nicol and Andy T. Ogielski. Modeling the Global Internet. *Computing in Science and Engineering*, January/February 1999, pp. 42–50.

2. D.M. Nicol. Scalability, Locality, Partitioning, and Synchronization in PDES. *Proc. Workshop Parallel and Distributed Simulation*, IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 4-11.

3. W. Willinger and V. Paxson. Where Mathematics Meets the Internet. *Notices of the American Mathematical Society*, Vol. 45, No. 8, Sept. 1998, pp. 961-970.

4. E. Zegura, K. Calvert, and M. Donahoo. A Quantitative Comparison of Graph-Based Models for Internet Topology. *ACM/IEEE Trans. Networking*, Vol. 5, No. 6, Dec. 1997, pp. 770-783.