Virtual InterNetwork Testbed: Status and Research Agenda*

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, Scott Shenker, Kannan Varadhan,
Haobo Yu, Ya Xu, Daniel Zappala

USC Computer Science Department Technical Report 98-678

July 17, 1998

Abstract

Simulation is an important tool in network protocol development, providing an effective way to perform controlled experiments, consider alternative designs, understand protocol interactions, and examine scales and topologies that are difficult to create in the laboratory. However, the scale and heterogeneity of today's networks create challenges for network simulation. In particular, configuring simulation inputs, properly modeling the myriad of interacting protocols, and processing and visualizing simulation output is becoming increasing difficult. The VINT project is developing a set of tools centered around ns simulator and nam network animator to meet these challenges. This paper describes the current status and future directions of this ongoing work.

1 Introduction

The rapid diffusion of internetworking technology brings two major sources of stress to the underlying protocol mechanisms and associated design methods: scale and heterogeneity. Scale must be considered in evaluating both the correctness and performance of wide area internet protocols at every level (from routing, to transport and application protocols). Heterogeneity of applications translates into a larger number of interacting protocols, service requirements, and traffic patterns.

Simulation is a valuable tool in designing and evaluating protocols for large heterogeneous networks. It allows repeatable and controlled experimentation while avoiding the cost and complexity of network testbeds. The VINT (Virtual InterNetwork Testbed) project is developing the simulation tools and supporting infrastructure to address the problems of scale and heterogeneity across a range of networking protocols and studies.

The software platform for our efforts is *ns* , a discrete event simulator composed of a C++ core, an object-oriented extension to the Tcl scripting language called OTcl, and a growing framework of networking protocols and building blocks. Upon this base we are developing the following techniques and tools to enable systematic studies of network protocol performance as scale and protocol interactions grow.

Libraries of network protocols Ns provides a rich set of protocols with which to build simulations. Due to its object framework, customization of the existing protocols and addition of new protocols is straightforward. The framework en-

*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.

courages extension and modification by users to promote investigation of variants to existing protocols and testing of new alternatives.

Libraries of topologies and traffic generators The load mix of the Internet is shifting rapidly (now to WWW, and expanding to audio and video), and the topology has always been in flux; it is important to provide researchers with the ability to explore the implications of these topology and load shifts. We are developing a library of network topologies and traffic generators for ns to facilitate protocol performance testing over a wide range of conditions. This common library of simulation environments will also allow better comparison between different simulations, easing the verification of the simulation results.

Abstraction techniques and tools It is hard to identify the relevant phenomena buried within the mountain of data generated by a detailed network simulation. Furthermore, resource limitations often constrain the number of network objects (i.e., nodes and links) that can be simulated. A crucial aspect of making simulations both practical and valuable is to provide the ability to vary the level of abstraction, in both the analysis of the data and in the simulation itself. Abstraction techniques will allow users to identify relevant phenomena using high-level simulations and then use detailed simulations to study the phenomena more extensively. Key to this approach is cross-validation of detailed and high-level simulations.

Visualization techniques Many relevant network phenomena remain invisible when only aggregate statistics are collected. Visualization techniques are crucial in enabling users to identify "interesting" aspects of the simulation. The visualization tools we are developing have proven to be extremely useful for debugging simulations and understanding protocol behavior, as well as for helping users frame their design questions.

Emulation interface Ns includes an emulation interface which provides a gateway between real-world network nodes and the simulator. Emulation enables a running simulator to send packets to, and receive packets from, operational network hosts and routers. This allows more thorough testing of implementations prior to deployment, and will allow simulations to be driven by more accurate network inputs.

The purpose of this paper is twofold. First, we want to describe the kinds of problems network researchers use simulation to study, and in doing so, demonstrate some of the functionality in ns. Second, we want to present the techniques we are developing to address the challenges of using simulation to study these algorithms and protocols in large heterogeneous networks.

We begin with an overview of the motivations and design decisions resulting in the current software architecture of the simulation environment. We then proceed to illustrate a wide range of network research areas in which investigators have employed ns in producing important research results. We touch upon scenario generation, scaling, and visualization, each active areas of development. We conclude with a brief discussion of prior work in network simulation and challenges for the future.

2 Composable simulation framework

The fundamental abstraction provided by the ns 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 [60].

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 is 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 the 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 [77] 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 [57].

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.

3 Protocol Design and Interaction

The ns simulator has a large population of users, and it and its ancestors have been used in many successful research efforts.[1] In this section, we review four areas of research activity that have used ns and describe existing functionality of the simulator. These areas include TCP congestion control, queue management, multicast routing, and reliable multicast transport. This is merely a sample of research using ns and is by no means an exhaustive list. In addition, we describe a recently developed emulation interface in ns.

3.1 TCP Congestion Control

The ns simulator and its ancestors have been used to study a number of algorithmic changes to the TCP protocol. Investigations of TCP error and congestion control algorithms have led to the development of several new algorithms for TCP including selective acknowledgments [30, 54], forward acknowledgments [53], and explicit congestion notification (ECN) [32]. Simulation studies using ns reveal how common TCP algorithms perform poorly when subjected to moderate to heavy packet loss, and how algorithms in end nodes such as "New Reno" and selective repeat can help to improve behavior significantly under such conditions. In addition, TCP can also be modified to avoid packet "bursts". Such bursts, which can have harmful effects on the network, often result after the successful reconstruction of the data stream following a series of packet losses [73] or after idle connections [74]. The availability of a public-domain simulator including several variants of TCP greatly facilitates the evaluation of proposed protocol enhancements.

[1] Ns is derived from REAL [46], which is derived from NEST [23]. The current version of ns (version 2) is available at http://www-mash.cs.berkeley.edu/ns.

3.2 Queue Management and Scheduling Policies

Ns has also been used to study router-based algorithms. Random Early Detection (RED) queue management [34], which was developed on one of the ancestors to ns, is supported as a standard queue management technique in ns. RED queue management reacts to congestion prior to overload. RED has been shown to achieve significant improvements over traditional FIFO ("drop-tail") queue management strategies by allowing for some bursty behavior but also providing implicit signaling to well-behaved network flows prior to buffer exhaustion. In addition, its random approach helps to avoid undesirable phase effects (i.e., throughput bias for particular connections) in networks of drop-tail queues [33].

Other investigations into router traffic management employing the ns simulator include Class Based Queueing (CBQ) [35], a technique in which packets are treated as members of "classes". Classes are assigned a maximum bandwidth allocation and priority level relative to other classes. The CBQ scheduler limits classes to their assigned bandwidth unless some other classes are using less than their allocated bandwidth, in which case "bandwidth borrowing" may be allowed.

In each of these cases, simulations provided a convenient, repeatable environment in which to study and compare algorithms. Furthermore, the existence of a platform with which to study these new algorithms has facilitated a much wider understanding of their behavior, thereby lowering barriers to deployment in operational networks.

Finally, it should be noted that alternative queue management and scheduling policies are useful for other kinds of simulation studies as well. For example, ns was used in a study of the effects of service priority on the performance experienced by adaptive audio applications [6]. A comparison of uniform and priority dropping mechanisms on the performance of

layered video also used ns extensively [7]. In both these cases, while the scheduling and dropping algorithms were themselves not the focus of the study, they were a critical part of the infrastructure needed to carry out the simulation experiments.

3.3 Multicast routing

More recently, ns has been used in the study of multicast routing protocols. Several multicast routing protocols, which establish distribution trees for delivering datagrams from a single sender to all the members of a multicast group, have been proposed for the Internet. These protocols can be classified as either broadcast-and-prune or explicit join protocols. In the former, which include DVMRP [69] and PIM-DM [25], a multicast packet is transmitted to all leaf subnetworks in a distribution tree rooted a the source. Leaf subnetworks with no local members of the group send *prune* messages towards the source of the packet. This prevents future packets from being transmitted to these subnetworks and limits packet distribution to those subnetworks with group members. In contrast, in explicit join protocols, such as CBT [8] or PIM-SM [26, 27], routers send hop-by-hop *join* messages for the groups for which they have local members. These control messages build forwarding state in routers and are sent upstream towards the source to establish a distribution tree. Ns currently has implementations of both broadcast-and-prune (based on PIM-DM) and explicit join (based on PIM-SM) protocols.

These protocols provide an example of the benefits of the split programming model described in Section 2. Low-level mechanisms for *forwarding* multicast packets, common to all multicast protocols, are implemented in C++, while the routing protocols themselves are implemented in OTcl. This allows rapid design and experimentation with the routing protocol implementation with minimal performance degradation, as only control messages are subjected to the higher overhead of the interpreted language.

The above-mentioned protocols provide internetwork forwarding of multicast packets. Multicast is also used in LAN environments to exchange routing updates and control messages, or to bootstrap protocol mechanisms [28]. Ns has been extended to support multiaccess links connecting more than two nodes. Facilities to support packet tracing (for off-line analysis) and selective loss have been implemented. Other than packet replication and forwarding, all the support for LAN multicast was done in OTcl to provide flexibility. This facility has been used to study multicast routing protocol robustness in the presence of control-packet loss and node failure. In particular, this work identified several pathological cases in PIM-SM and led to corresponding fixes to the protocol specification.

The existence of multicast routing protocols in ns is not just of value to researchers interested in the design and evaluation of such protocols. Rather, other experimenters depend on multicast as necessary infrastructure for their simulations. We next discuss examples of such simulations.

3.4 Multicast Transport

Aspects of the many-to-many communication paradigm of multicast forwarding complicate the design of transport protocols. These include the anonymity of membership, membership dynamics, and the heterogeneity of the members. Ns, through infrastructure consisting of many different link and network layers, as well as unicast and multicast routing protocols, provides a unique environment to develop multicast transport protocols. In this section, we describe some of the work on reliable transport, congestion control, and application development in which simulation has been used.

Reliable Transport Scalable Reliable Multicast (SRM) [31] was designed originally for real-time whiteboard applications. It uses a NACK-based protocol to achieve reliability. Receivers detecting a loss multicast a negative acknowledgement to the group. These negative acknowledgements are multicast in order to suppress duplicates. In addition, receivers delay their negative acknowledgements for a random time to avoid inundating the network with synchronized requests and retransmissions. While the original simulations of SRM were done in a stand-alone simulation tool, an SRM implementation has been

added to ns. This is valuable to other researchers investigating reliable multicast transport. For example, Routing Policy Multicast (RPM) [38] is an application-specific, SRM-like protocol for the reliable delivery of routing policy objects. These objects are large but there is less of a time constraint on delivery than with SRM. Given the common mechanism shared between SRM and RPM, RPM was easily implemented by deriving a new class in the ns object framework. This implementation was used to evaluate rapidly different timer mechanisms, and determine the optimal parameter settings for RPM.

The periodic session messages in SRM lead to bandwidth overhead that impacts protocol scalability when the group membership is large. Scalable Session Messages (SSM) [72] algorithms use hierarchy mechanisms to reduce this overhead. ns was used to investigate the different mechanisms by which representatives are chosen, and to quantify the scaling benefits that can be achieved and the impact on the protocol's loss detection and recovery mechanisms.

As stated above, the existence of a public-domain simulator with an extensive set of protocol modules facilitates comparison of research results. As an example, Hänle [40] used ns to compared the Multicast File Transfer Protocol (MFTP) [68], a protocol specifically designed for bulk data transfer, to SRM under different network conditions in a variety of different topologies. By subjecting the protocols to identical test conditions, their behavior could be compared across a range of conditions.

Congestion Control Multicast congestion control is an active area of investigation. The challenge is to manage the feedback from a large set of homogeneous receivers in a timely and scalable manner. Receiver-driven Layered Multicast (RLM) [56] is an example of a multicast congestion control protocol for layered video transmission. Much of the design of RLM used ns; join-experiment heuristics, congestion measurements, and other features were prototyped through OTcl scripting.

DeLucia [18] proposed a representative based congestion control algorithm for multicast bulk data transfer applications. ns was used to verify correct behavior in the face of congestion, and evaluate the performance of the protocol in the presence of competing traffic in the network. Work is now focused on applying the congestion control schemes to Multicast Dissemination Protocol (MDP) [16]. This has been accomplished by retrofitting an existing implementation of MDP into ns, enabling study of the integrated protocol and congestion control scheme in the simulator.

Application Real-time Transport Protocol (RTP) [39] is designed for unreliable, but timely delivery of datagrams for real-time audio or video, or other multimedia or real-time applications. ns implements the control aspects of RTP (Real-time Transport Control Protocol—RTCP) in OTcl. The implementation is useful for further experimentation in the development of other transport and application protocols.

Reddy [65] proposed a multicast-based application of a network of dynamically adaptive measurement servers to gather localized information about the network. ns is being used in the design of the protocol, to determine the scope of each measurement server so that every node or link is monitored by a server, and to make the algorithm robust in the face of the addition, deletion and failure of servers.

McCanne et al [55] are using ns to investigate the minimal set of mechanisms that a router could implement to simplify the design and improve the performance of multicast transport protocols. One example of this is sub-tree multicast (or "subcast"), where the retransmit of a lost data packet is only sent to the subtree that experiences the loss.

Sidebar: Dynamics

Ns supports the study of network dynamics. That is, how are protocols affected by links and nodes that fail and recover? This support includes dynamic routing protocols and models of link failures. Study of network dynamics is important to characterize the behavior of end-to-end protocols in the context of a variety of network anomalies, including route flapping, routing loops and network partitions [62]. We have

used these approaches to study TCP and multicast transport protocols [73].

One effect of topology change is the reordering of packets in transit. Such interleaving of acknowledgements or data packets can have harmful consequences for a TCP session and the network. In particular, the sender can see a sudden and large increase in the amount of acknowledged data. The sender responds by opening its send window, and sends a large burst of packets back to back. The resulting congestion reduces the throughput for the session. We have observed these effects through simulation studies using ns; it remains to future work to recreate the effect of packet interleaving in an operational network.

Topology change has even greater impact on multicast transport protocols. Our analysis of the timer mechanisms in SRM revealed the need for accurate and timely distance estimation by the group members; collecting such estimates in the face of dynamically changing topology is problematic. We also identified some undesirable operating regions of the protocol. Such characterizations are useful both in the design and evaluation of the protocol, and in enhancing its manageability when deployed in operational networks.

The experience gained studying TCP and characterizing the behavior of SRM during topology change has contributed to an evolving systematic methodology for the study of a protocol behavior under a range of network conditions including packet loss, routing transients, and node and link failure.

3.5 Emulation

Ns supports an *emulation* facility, allowing simulations to interact with actual network traffic. 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 1. The simulator contains a simulated network, and passes live network traffic through the simulation, subjecting it to the dynamics of the simulated network. The

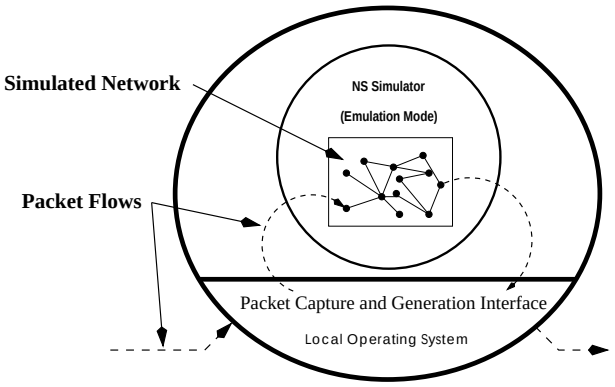

Figure 1: Emulation: live network traffic passes through simulated topology and cross-traffic.

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 predicting 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 a shared white board application using a version of the SRM protocol supported in the MASH toolkit [57]. 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 helps to pinpoint time-dependent behaviors of adaptive protocols which are very difficult to diagnose otherwise.

3.6 Other examples

Although we have focused primarily on the study of transport- and router-level network issues, ns has been used for studies of other network layers and problems. For example, Faber uses ns to examine active approaches to congestion control [29]. Portions of ns also have been used to study multicast address allocation [48], new rate based congestion control algorithms [66], and self-organizing clustering algorithms applied to network monitoring and reliable multicast session message aggregation.

4 Scenario generation

In general, a simulation scenario is defined by the components and parameters that comprise the simulation. In ns, scenarios are represented by simulation scripts which describe the *network topology*, including the physical interconnects between nodes and the static characteristics of links and nodes; *traffic models* for both unicast and multicast senders; and *test generation*, which creates events such as multicast group distribution (receivers joining and leaving) and network dynamics (node and link failures) designed to stress test an implementation.

4.1 Topology

The topology for a simulation is specified as part of the simulation script. For small simulations, the user will usually specify the topology "by hand". For larger topologies, generating topologies manually is less practical. Hence, topology generation tools, which create topologies according to a set of user specified parameters, are often used. Our goal here is to provide users with a library of topology generation tools to use with ns. In addition, we are accumulating a library of sample topologies for use in simulations. Rather than recreate previous work, we have chosen to leverage the work of others on automatic topology generation. For the two topology generation tools described below, we have written and made available conversion programs that allow the generated topologies to be used in ns simulations.

The Georgia Tech Internetwork Topology Models (GT-ITM) software package [12, 13, 79] can create flat random networks using a variety of edge distribution models, including pure random, exponential, locality, several variations of Waxman's model [75] and the Doar-Leslie model [22]. Given the size of a grid and the number of nodes desired, GT-ITM randomly places nodes on the grid and connects them according to the probability given by the edge model.

The GT-ITM software package can also create different types of hierarchical networks. It can create a multi-level hierarchy by first creating the top-level network (using one of the flat random models) and then recursively replacing each node in the current level with a connected graph. Edges between levels are resolved by randomly selecting a node within each replacement graph. GT-ITM can also create transit-stub hierarchies like those found in the Internet. Parameters control the average number of transit domains, the average number of stubs per transit, and the average size of transit and stub domains.

Doar has written *tiers* [21, 13, 20] to create three-level hierarchical topologies similar to the transit-stub GT-ITM topologies. In *tiers*, the three levels correspond to wide-area, metropolitan-area, and local-area networks. A network at any level is created by randomly placing nodes within a grid and then connecting them with a minimum spanning tree. Edges between levels are created by attaching a child network to a parent network, for example by attaching a local-area network to a metropolitan-area network. Topology generation is controlled by specifying the number and size of networks at each level, plus the connectivity between levels.

The key challenge in topology generation is coming up with topologies that embody relevant characteristics of real networks. Once this is done, the ns

framework easily allows simulation of any topology that is generated. We have developed an API to the GT-ITM flat random and transit-stub topologies. Ns simulation scripts are automatically created by a simple format conversion program. We plan to extend the API for other types of topologies and topology generation packages. Similarly, actual maps of (parts of) the Internet topology can also be used as simulation input.

4.2 Traffic Models

Ns provides a wide variety of source models, all of which are configurable in Tcl allowing flexible creation and parameterization. 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 [17], models Telnet-like applications. To simulate web traffic, a traffic generator based on the model described in [51] has also been implemented in ns. 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 distribution (the latter useful in generating self-similar traffic [49, 63]), and a source that generates traffic from a trace file. The composable framework of ns makes adding new traffic models easy, and allows construction of compound models out of the individual ones. For example, in simulations of RLM a multi-layered video source was created by combining several CBR streams [56]. A similar approach was used to incorporate correlations of burstiness across layers in another study involving layered video [7].

Ns provides an extensive set of models for individual traffic sources. However, in creating a simulation scenario, a network researcher is often more interested in background traffic with desired characteristics (i.e., aggregate bandwidth, burstiness, self-similarity, etc.). Recently, we have developed an API to help users easily create random background traffic with various characteristics (unicast or multicast and with various distributions). We are continuing to develop this API.

4.3 Test Generation

Choosing an appropriate set of test conditions for a simulation experiment is often straightforward. For example, the performance of an algorithm such as RED can be evaluated by using a reasonable set of parameter values as inputs to the simulation. Evaluating the correctness of a protocol, on the other, can be a much more daunting task. We developed a framework for *S*ystematic *T*esting of Protocol *R*obustness by *E*valuation of *S*ynthesized *S*cenarios (STRESS) [42, 44] 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. Currently, we are comparing three methods for testing multicast routing robustness in the presence of selective message loss on a LAN. We call these methods: a) heuristic test generation (HTG), b) fault-independent test generation (FITG), and c) fault-oriented test generation (FOTG).

HTG uses domain-specific heuristics and topological equivalence relations to limit the number of simulated scenarios. Simulation of these scenarios in ns is then conducted to explore the behavior of the protocol in the presence of message loss. This method does not require a formal model of the protocol.

In contrast, FITG and FOTG process a finite state machine model of the protocol. Using a forward search technique, the FITG method investigates an equivalent subset of the protocol state space for a given topology to generate the scenario events leading to erroneous behavior. FOTG starts from the target message and synthesizes a topology necessary to trigger and be affected by this message. Then, using a backward search technique, it generates scenario events leading to protocol error.

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 [42, 44, 43].

Future work in this area will consider the effect of a wider range of 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.

5 Scaling and Simulation Abstraction

Ns has shown its utility in a number of research projects (some of which were described in Section 3). Extending this success to other research problems requires the simulator to scale to thousands of nodes and links, far beyond ns's original design goals. Custom simulators built for a given problem can tailor simulation abstraction to the problem at hand, but often omit many of the services present in a general purpose simulator like ns (for example, tracing and debugging support and existing loss, traffic models, and dynamic topologies). More importantly, it is often difficult to validate these simulators against more detailed simulations and reality. In ns, we are working on approaches which allow simulations to abstract away unnecessary details while still using general services [45]. By supporting an adjustable level of abstraction we will allow users to select abstract models when needed, but also focus in on the details to investigate interesting phenomena.

We are taking two complementary approaches to scaling ns: carefully tuning the implementation, and allowing the user to abstract out details unnecessary for some simulations. The former improves the efficiency of the simulator without changing the simulation abstraction. The latter changes aspects of the simulated model to achieve performance gains. As an example implementation improvement we have made changes to our binding mechanism between C++ and Tcl to improve memory consumption when there are large numbers of shared objects. Use of centralized route computation is an example abstraction. Although it produces slightly different transient behavior, for many simulations these details are unimportant while savings in time and memory is [45]. These approaches often interplay, for example our *session-level* simulator abstracts away cross-traffic network interference and uses a very lightweight node and link representation (an implementation change).

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 from model to model. 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 loss rate for higher-level WAN simulations. An analysis of how abstractions change simulation results accompanies our development of new abstractions. An analysis of SRM performance across detailed and session-level simulations suggests that while individual SRM events do vary, average aggregate behavior changes by only a few percent in the cases we examined [45].

Abstraction techniques that improve ns's ability to run large simulations is an area of ongoing effort. Particularly promising among the techniques we are pursuing is two-phase topology generation, in which a large topology is generated and populated with a few agents, and then replaced with a more abstract but equivalent topology through node aggregation. We are also developing hybrid abstractions where one part of the simulator runs in detailed mode and another uses session-level mode or some other appropriate abstraction.[2]

In addition to enabling large simulations, data analysis in large simulations is an important issue. Hierarchical visualization tools are needed for large networks.

Sidebar: What is Scaling?

In the abstract, scaling a simulator is running bigger simulations. But simulations have many dimensions of "big": large numbers of nodes, links, senders and receivers, multicast groups, multicast group members, and packets-in-transit, as well as a large amount

[2] It should be noted that parallel simulation on large multiprocessing hardware can provide a complementary technique to abstraction when additional performance gains are needed.

of data collected. A simulator that supports 100 simultaneous TCP flows may work well with a 10Mb/s LAN-style sending rate but may consume much more memory when simulating 45Mb/s cross-country traffic, since many more packets are concurrently in flight.

Pushing these dimensions of scaling can tax a computer's memory, CPU, and I/O resources. Although we feared our use of Tcl might make CPU usage a bottleneck, in our simulations we have found memory to be the most frequent problem. It is understandably difficult for a single machine to keep track of most of the network state for thousands of simulated hosts.

To make matters worse, many researchers are currently investigating the properties of multicast protocols (both routing and reliable transport) when there are large number of members per group. Memory required for these protocols grows $O(n^2)$ as the number of group participants increases, so a detailed simulation of very large groups is often impossible. Our session-level abstraction for multicast abstracts away the details of cross-traffic interference, allowing a much more compact link and node representation. This allowed simulations of twice as many group members with the same amount of memory (Figure 2).

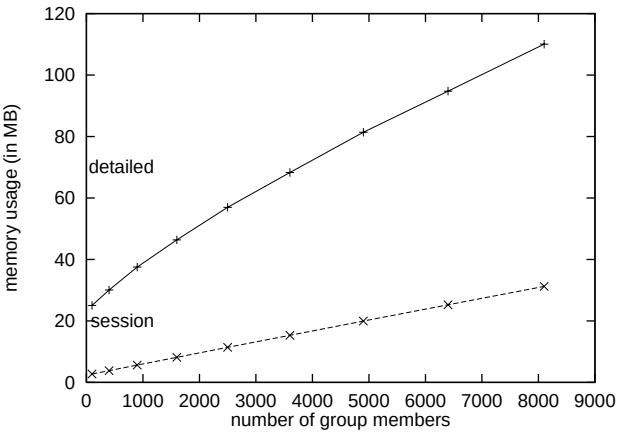

Figure 2: Session-level abstraction allows substantially larger numbers of multicast group members in the same amount of memory.

6 Visualization

Usually, the output of ns is a trace file consisting of packet events and parameters. These data are difficult to interpret by direct inspection, so graphs and aggregate statistics are often employed (for example, time/sequence number plots common in TCP studies). While these approaches are helpful, they can miss interesting details of the simulation.

Visualization has been found to be extremely powerful in helping understand characteristics of large complex data sets. The VINT project is exploring this technique through *nam* (the *n*etwork *an*imator). Like ns, nam is built with a C++ core and support of OTcl/Tk, which allows nam to be extensible and easily customized. Using trace files generated by ns, nam provides animation, inspection, and synchronized view-based, drill-down analysis of the traces.

Animation The animation tool displays the network topology, animates packet transmission across links and displays node and link dynamics. The unit of animation is the *event*, which is read from the trace file. In general, there are four types of events. *Nodes* and *links* define the topology of the simulation. *Packet-related events* denote packets leaving or reaching a node, entering or leaving a queue, and being dropped. *Protocol-related events* describe the position of protocol instances (*agents*) and their internal states (*variables*). *Annotations* serve as indices to the trace file for easier browsing.

Nam defines different animation models for different events. Packets are displayed as filled, colored rectangles with a tapered end to indicate the direction they travel. Dropped packets are displayed as rolling squares falling out of the window. Queues are shown as stacked packets on a link. Nodes and links can be shown with different shapes and colors to highlight simulation events. For example, link color can indicate its status (up or down), or node color can indicate membership in a multicast group. Agents (data senders and receivers on a node) are displayed

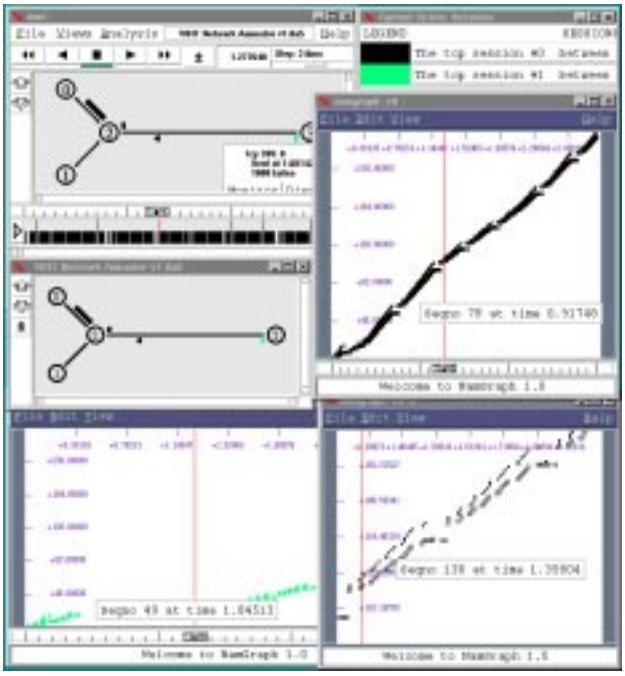

Figure 3: Nam screen snapshot.

as small labeled rectangles attached to nodes. These visual representations make it easy to inspect data (see below).

Nam provides two ways to browse the traces. First, a `time` slider is provided to start animation from any point in the trace file. Second, annotations are provided as indices to the trace files. They are displayed in the bottom pane in nam main window (Figure 3). Double clicking on any of them brings nam to the time indicated by that annotation. Annotations may be edited interactively during animation. Nam provides VCR-like buttons (e.g., Play, Fast Forward, Rewind, etc) to control the animation..

Sometimes it is useful to compare several animations side-by-side, for example, when one is studying behaviors of the SRM protocol [31] with different parameters in the same scenario. In order to support this, several instances of nam running on the same machine can control each others' animations. It is assumed that only one nam instance will be in control at one time, therefore no concurrency control mechanism is provided.

Inspection The animation tool only displays some aspects of the simulation traces; much of the information, such as packet headers or protocol state variables, in the traces cannot be shown directly by the animation tool. For large data sets, it may be beneficial to have multiple views emphasizing different aspects of the data. The inspection tool and statistics tool of nam are designed with this in mind.

When an interesting point is reached during an animation, it is often desirable to examine certain data, such as the source and destination of a packet, state variables in an agent, and link parameters, in more detail. The inspection tool provides two ways to do this. First, clicking on any of the displayed objects, e.g., packets and agents, will bring out a panel showing additional information about that object. Second, *monitors* can be used to continuously display state about a link or a node. Monitors display and update all available information about the object. All monitors are displayed in a pane in nam's main window, as shown in Figure 3.

Nam provides multiple views of the same topology

(the lower smaller window in Figure 3), thus allowing simultaneous animation and inspection of different parts of the topology. All views are zoomable for inspection of details. They are useful for examining in detail multiple objects in a large topology.

Synchronized view-based, drill-down analysis Besides examining packet flows and node/link dynamics, we would also like to have high-level representations (or "overviews") of the entire trace. For example, when we are studying TCP's behavior, it is useful to have TCP sequence number plots in addition to the animations. These views are synchronized in time; time change in any view (e.g., by moving a time slider or by clicking on an interesting event) will change time in all views. The views also enable the user to discover interesting and useful results by selecting a subset of trace events, continuously zooming in for drill-down analysis. Each trace event is displayed in the same way (i.e., color, shape, etc.) across views to help the user coordinate events across views.

Currently, two types of analysis views are supported. The first displays link-specific information. Specifically, utilization and packet loss over time is shown for the entire simulation (see the two bars starting with an arrow in Figure 3). The second kind of view is a high-level protocol analysis tool. Figure 3 shows a plotting and analysis tool for TCP. The window in the upper right corner shows the number of active TCP sessions in the simulation (two in this case). The window below it shows both the sequence number and acknowledgement number as a function of time for the first tcp session. The lower left window shows the same plots for the second tcp session. The lower right window is a zoomed in view of a portion of the first tcp connection. The vertical line inside each view denotes current time in the animation. A message view can be opened for each plot, for instance, to show the sequence number of a particular packet in the graph. Note that the same color is used for the view of the first tcp session, its zoomed view, and its packets in the animation. The tcp sequence plot is marked as dot while the ACK plot is marked as a square. Currently, protocol plotting and analysis is provided only for TCP. Similar tools for other protocols, such as SRM, are being developed and will be integrated into nam in the same manner.

Network Layout Before animating a simulation trace a layout for the network topology must be specified. This can be done manually by the user or automatically by nam. Much work has been done in automatic layout for arbitrary networks [15, 24, 50]. Nam adopts an algorithm based on a spring-embedder model [36] for its simplicity and efficiency. It assigns attractive forces on all links and repulsive forces between all nodes, and tries to achieve balance through iteration. Experience has shown that it works very well for topologies with less than 100 nodes. Manual layout can be specified in a trace file or by interactively editing an existing layout.

7 Related Work

Network Simulators. Network simulation has a very long history. Ns itself is derived from REAL [46], which is derived from NEST [23]. 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 network type (ATM Simulator [37]) or protocol (PIM-SIM [76]). Others, including ns, REAL, OPNET [19], INSANE [52] target a wider range of protocols. The most general provide a general simulation language with network protocol libraries (for example, Maisie [5]). Very focused simulators model only the details relevant to the developer. The differences between network-targeted and general simulators is much less clear.

The core of ns and most 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 [58] and fluid network approximations [47]) for better performance or accuracy.

Parallel and distributed simulation is a second way to improve performance. Several simulators support multiprocessors or networks of workstatations [46, 5, 64]. Although ns is focused only on sequential simulation, the TeD effort has parallelized some ns modules [64]; we see parallel simulation as complementary to abstraction.

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 [1]. As discussed in Section 5, 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 [5], or sometimes both [19]. Some systems focus on allowing the same code to run in simulation and a live network (for example, x-Sim [11] and Maisie [5]). Most systems augment programming with a GUI shell of some kind [23, 46, 19, 59, 10, 3, 76, 70, 5]. Ns provides a split-level programming model (see Section 2) where packet processing is done in a systems language while simulation setup is done in a scripting language. Nam provides visualization output and we plan to add some ability to edit topology.

Network Emulation. Previous work in network emulation has included special purpose stand-alone network emulators supporting packet delay (Yan's Hitbox [2]). While these systems modify an existing packet stream, more sophisticated emulation systems such as Sun's Packet Shell [61] have allowed generation of new streams, typically for protocol testing. By linking a general purpose simulator to live network traces ns promises to accomplish both of these aims.

Visualization and animation Like network simulation, visualization has a long history. Becker, Eick and Wilks described the SeeNet system for 2-D visualization of network data [9]. They display network traffic, overload and idle capacity on geographic maps. A matrix display of network overload is provided as an alternative. Animation is supported for inspecting time-varying characteristics of data. Lamm *et al* uses 3D display to show web server loads on a geographic map [71]. Different traffic types are colored coded. Scullin *et al* uses Scattercube Matrix metaphor to display web server performance in a virtual reality environment [78]. The above systems mainly focus on high-level statistics of network data, while nam currently focuses on packet-level animation and inspection.

Many researchers have tackled the problem of visualization of complex data. Antis *et al*'s SeeData generates 2-D visualization of database structure [4]. It provides multiple colored views, each of which focuses on a different aspect of database structure, ranging from abstract overview of the entire structure to detailed associations among relations. Cat-a-Cone focuses on search and browsing of very large hierarchical data [41]. It uses ConeTree [67] for browsing and overview of search structure, and WebBook [14] to display search results. These systems share a common principle, i.e., multiple linked views are essential in visualizing complex data. Nam adopts this principle. It organizes visualization around the main topology view, from which a number of specialized views may be derived.

8 Conclusions

The goal of network simulation is exploration of new protocols and of old protocols in new environments. Distribution of early versions of ns has resulted in better understanding of TCP and router queueing mechanisms. Currently ns is seeing widespread use in the development of reliable multicast protocols. Finally, we described our current and future efforts in scenario generation and protocol testing, simulation scaling through the use of abstraction, and improved visualization tools. By offering an expanding set of network protocols suitable across a wider range of scenarios and scales we hope to make future protocol development and comparison easier.

References

[1] Jong-Suk Ahn, P.B. Danzig, D. Estrin, and B. Timmerman. Hybrid technique for simulating high bandwidth delay computer networks. In *Proceedings of the ACM SIGMETRICS*, pages 260–261, Santa Clara, CA, USA, May 1993. ACM.

[2] J.S. Ahn, Peter B. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: Emulation and experiment. In *Proceedings of the ACM SIGCOMM*, pages 185–195, Cambridge, Massachusetts, August 1995. ACM.

[3] C. Alaettinoglu, A. U. Shankar, K. Dussa-Zieger, and I. Matta. Design and implementation of mars: A routing testbed. *Journal of Internetworking Research and Experience*, 5(1):17–41, mar 1994. ftp://ftp.isi.edu/pub/cengiz/publications/MaRS:Design.ps.gz.

[4] J.M. Antis, S.G. Eick, and J.D. Pyrce. Visualizing the structure of large relational databases. *IEEE Software*, 13(1):72–9, January 1996.

[5] Rajive L. Bagrodia and Wen-Toh Liao. Maisie: A language for the design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering*, 20(4):225–238, April 1994.

[6] Sandeep Bajaj, Lee Breslau, and Scott Shenker. Is service priority useful in networks. In *ACM SIGMETRICS*, June 1998.

[7] Sandeep Bajaj, Lee Breslau, and Scott Shenker. Uniform versus priority dropping for layered video. *To appear in ACM Sigcomm*, 1998.

[8] A. J. Ballardie, P. F. Francis, and J. Crowcroft. Core Based Trees. In *Proceedings of the ACM SIGCOMM*, San Francisco, 1993.

[9] Richard A. Becker, Stephen G. Eick, and Allan R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):16–28, March 1995.

[10] Bones, 1998. http://www.cadence.com/alta/produces/bonesdat.html.

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

[12] Ken Calvert and Ellen Zegura. Georgia tech internetwork topology models. http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html.

[13] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, June 1997.

[14] Stuart K. Card, George G. Robertson, and William York. The WebBook and the Web Forager: An information workspace for the World-Wide Web. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 416–417, Vancouver, Canada, 1996.

[15] Michael K. Coleman and D. Stott Parker. Aesthetics-based graph layout for human consumption. *Software - Practice and Experience*, 26(12):1415–38, December 1996.

[16] W. Dang and Joseph Macker. The Multicast Dissemination Protocol (MDP) Framework. *Internet Draft: NONE working group*, June 06, 1997. Work in Progress.

[17] Peter B. Danzig, Sugih Jamin, Ramón Cáceres, Danny J. Mitzel, and Deborah Estrin. An empirical workload model for driving wide-area TCP/IP network simulations. *Journal of Internetworking: Research and Experience*, 3(1):1–26, March 1992.

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

[19] F.H. Desbrandes, S. Bertolotti, and L. Dunand. Opnet 2.4: an environment for communication

network modeling and simulation. In *Proc European Simulation Symposium*, October 1993.

[20] Matthew Doar. tiers. ftp://ftp.nexen.com/pub/papers/tiers1.1.tar.gz.

[21] Matthew Doar. A better model for generating test networks. In *IEEE Global Telecommunications Conference / GLOBECOM '96*, November 1996.

[22] Matthew Doar and Ian Leslie. How bad is naive multicast routing? In *IEEE INFOCOM*, 1993.

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

[24] E.R. Gansner, E. Koutsofios, S.C. North. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–30, March 1993.

[25] D. Estrin, D. Farinacci, A. Helmy, V. Jacobson, and L. Wei. Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification. *Proposed Experimental RFC. URL http://netweb.usc.edu/pim/pimdm/PIM-DM.{txt,ps}.gz*, September 1996.

[26] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Motivation and Architecture. *Proposed Experimental RFC. URL http://netweb.usc.edu/pim/pimsm/PIM-Arch.{txt,ps}.gz*, October 1996.

[27] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification. *RFC 2117. URL http://netweb.usc.edu/pim/pimsm/PIM-SMv2-Exp-RFC.{txt,ps}.gz*, March 1997.

[28] D. Estrin, M. Handley, A. Helmy, P. Huang, and D. Thaler. A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing. *Submitted to IEEE/ACM Transactions on Networking. URL http://www.usc.edu/dept/cs/technical_reports.html*, May 1997.

[29] Theodore Faber. Optimizing throughput in a workstation-based network file system over a high bandwidth local area network. *ACM Operating Systems Review*, 32(1):29–40, January 1998.

[30] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe, reno, and sack tcp. *ACM Computer Communication Review*, 26(3), July 1996.

[31] S. Floyd, V. Jacobson, S. McCanne, C-G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *ACM/IEEE Transactions on Networking*, 1997. To appear.

[32] Sally Floyd. Tcp and explicit congestion notification. In *ACM Computer Communication Review*. ACM, October 1994.

[33] Sally Floyd and Van Jacobson. On traffic phase effects in packet-switched gateways. *Journal of Internetworking: Research and Experience*, 3(3), September 1992.

[34] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. In *ACM/IEEE Transactions on Networking*. ACM, August 1993.

[35] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. In *ACM/IEEE Transactions on Networking*. ACM, August 1995.

[36] T.M.J. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, November 1991.

[37] Nada Golmie, Alfred Koenig, and David Su. *The NIST ATM Network Simulator Operation and Programming Version 1.0*. U.S. Department of

Commerce Technology Administration National Insitute of Standards and Technology Computer System Laboratory Advanced Systems Division, Gaithersburg, MD 20899, aug 1995. ftp://isdn.ncsl.nist.gov/atm-sim/sim_man.ps.Z.

[38] R. Govindan, H. Yu, and D. Estrin. Scalable non-transactional replication in the Internet. submitted for publication.

[39] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889 edition, 1996.

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

[41] Marti Hearst and Chandu Karadi. Cat-a-cone: An interactive interface for specifying searches and viewing retrieval results using a large category hierarchy. In *Proceedings of the 20th Annual International ACM/SIGIR Conference*, pages 246–255, Philadelphia, PA, July 1997.

[42] A. Helmy and D. Estrin. Simulation-based 'STRESS' Testing Case Study: A Multicast Routing Protocol. *Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98)*, July 1998.

[43] Ahmed Helmy, Deborah Estrin, and Sandeep Gupta. Fault-oriented test generation for multicast routing protocol design. *Formal Description Techniques (FORTE XI) & Protocol Specification, Testing, and Verification (PSTV XVIII), 1998 IFIP TC6/WG6.1 Join International Conference, Paris, France.*, November 1998.

[44] Ahmed A-G. Helmy. Systematic Testing of Multicast Protocol Robustness. *Ph.D. Dissertation proposal. Submitted as Technical Report to Computer Science, University of Southern California.*, December 1997.

[45] Polly Huang, Deborah Estrin, and John Heidemann. 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. to appear.

[46] Srinivasan Keshav. REAL: a network simulator. Technical Report 88/472, University of California, Berkeley, December 1988.

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

[48] Satish Kumar, Pavlin Radoslavov, Dave Thaler, Cengiz Alaettinoğlu, Deborah Estrin, and Mark Handley. The MASC/BGMP architecture for inter-domain multicast routing. To appear, SIGCOMM, September 1998.

[49] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *ACM/IEEE Transactions on Networking*, 2(1):1–15, February 1994.

[50] P. Luders, R. Ernst, and S. Stille. An approach to automatic display layout using combinatorial optimization algorithms. *Software - Practice and Experience*, 25(11):1183–1202, November 1995.

[51] B. Mah. An empirical model of http network traffic. In *Proceedings of the IEEE Infocom*, Kobe, Japan, April 1997. IEEE.

[52] Bruce A. Mah. *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.

[53] Matt Mathis and Jamshid Mahdavi. Forward acknowledgement: Refining tcp congestion control. In *ACM SIGCOMM*. ACM, August 1996.

[54] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. Tcp selective acknowledgement options (rfc 2018). In *Internet Request For Comments*, October 1996.

[55] S. McCanne. Router forwarding services for reliable multicast. Note 199704141535.IAA10590@mlk.cs.berkeley.edu to the Reliable Multicast list rm@mash.cs.berkeley.edu, April 1997.

[56] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM*, pages 117–130, Stanford, CA, U.S.A., August 1996.

[57] Steven McCanne, Eric Brewer, Randy Katz, Lawrence Rowe, Elan Amir, Yatin Chawathe, Alan Coopersmith, Ketan Mayer-Patel, Suchitra Raman, Angela Schuett, David Simpson, Andrew Swan, Teck-Lee Tung, David Wu, and Brian Smith. Toward a common infrastructure for multimedia-networking middleware. In *Proceedings of the 7thInternational Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 39–49, St. Louis, Missouri, May 1997. IEEE.

[58] Armin R. Mikler, Johnny S. K. Wong, and Vasant Honavar. An object oriented approach to simualting large communication networks. *Journal of Systems Software*, 40:151–164, 1998. huang folder: general simulator.

[59] Robb Mills. Comnet iii: Object-oriented network performance prediction. In G.W. Evans, M. Mollaghasemi, E.C. Russell, and W.E. Biles, editors, *Proceedings of the 1993 Winter Simulation Conference*, pages 237–239, December 1993.

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

[61] Steve Parker and Chris Schmechel. The packet shell protocol testing tool. Software distribution at http://playground.sun.com/psh/, 1997.

[62] V. Paxson. End-to-end routing behavior in the internet. In *ACM SIGCOMM*, August 1996.

[63] Vern Paxson and Sally Floyd. Wide-area traffic: the failure of Poisson modeling. In *ACM SIGCOMM*, pages 257–268, London, United Kingdom, August 1994. ACM.

[64] K. Perumalla, R. Fujimota, and A. Ogielski. Ted - a language for modeling telecommunication networks. *ACM SIGMETRICS Performance Evaluation Review*, 25(4), March 1998.

[65] A. Reddy. A self organizing monitoring architecture. Thesis Proposal, Available from the authors, May 1997.

[66] Reza Rejaie, Mark Handely, and Deborah Estrin. Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. Submitted to ICNP '98, available at http://netweb.usc.edu/reza/icnp98.ps, 1998.

[67] George G. Robertson, Stuart K. Card, and Jock D. MacKinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):56–71, April 1993.

[68] K. Robertson, K. Miller, M. White, and A. Tweedly. StarBurst Multicast File Transfer Protocol (MFTP) Specification. *Internet Draft: NONE working group*, February 13, 1997. Work in progress.

[69] D. Waitzman S. Deering, C. Partridge. Distance Vector Multicast Routing Protocol, November 1988. RFC1075.

[70] Hussein Salama. Mcrsim user's manual, May 1995. huang folder: specialized simulator.

[71] W.H. Scullin S.E. Lamm, D.A. Reed. Real-time geographic visualization of world wide web traffic. *Computer Networks and ISDN Systems (Fifth International World Wide Web Conference)*, 28(7-11):1457–68, May 1996.

[72] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. Scalable timers for soft state protocols. In *IEEE Infocom*, 1997.

[73] Kannan Varadhan, Deborah Estrin, and Sally Floyd. Impact of network dynamics on end-to-end proocols: Case studies in TCP and reliable multicast. Technical Report USC CS TR 98-672, University of Southern California, March 1998.

[74] Vikram Visweswaraiah and John Heidemann. Improving restart of idle TCP connections. Technical Report 97-661, University of Southern California, November 1997.

[75] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9), December 1988.

[76] Liming Wei. The design of the USC PIM simulator (pimsim). Technical Report 95-604, University of Southern California Computer Science, Los Angeles, CA 90089-0781, aug 1995. http://catarina.usc.edu/lwei/TR-95-604.ps.gz.

[77] David Wetherall and Christopher J. Linblad. Extending Tcl for dynamic object-oriented programming. In *Proceedings of the USENIX Tcl/Tk Workshop*, page 288, Toronto, Ontario, July 1995. USENIX.

[78] D.A. Reed W.H. Scullin, T.T. Kwan. Real-time visualization of world wide web traffic. In *Proceedings of 1995 ICASE/LaRC Symposium on Visualizing Time-Varying Data*, pages 31–45, Williamsburg, VA, USA, September 1995.

[79] Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE INFOCOM*, 1996.