

Large-Scale Network Simulation: How Big? How Fast?

Richard M. Fujimoto, Kalyan Perumalla, Alfred Park, Hao Wu, Mostafa H. Ammar,
College Of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280
{fujimoto,kalyan,park,wh,ammar}@cc.gatech.edu

George F. Riley

Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30032-0280
riley@ece.gatech.edu

Keywords: parallel discrete event simulation communication networks

ABSTRACT

Parallel and distributed simulation tools are emerging that offer the ability to perform detailed, packet-level simulations of large-scale computer networks on an unprecedented scale. The state-of-the-art in large-scale network simulation is characterized quantitatively. For this purpose, a metric based on the number of Packet Transmissions that can be processed by a simulator per Second of wallclock time (PTS) is used as a means to quantitatively assess packet-level network simulator performance. An approach to realizing scalable network simulations that leverages existing sequential simulation models and software is described. Results from a recent performance study are presented concerning large-scale network simulation on a variety of platforms ranging from workstations to cluster computers to supercomputers. These experiments include runs utilizing as many as 1536 processors yielding performance as high as 106 Million PTS. The performance of packet-level simulations of web and ftp traffic, and Denial of Service attacks on networks containing millions of network nodes are briefly described, including a run demonstrating the ability to simulate a million web traffic flows in near real-time. New opportunities and research challenges to fully exploit this capability are discussed.

1. Introduction

Simulation is widely recognized as an essential tool to analyze networks. Although analytic methods are useful in many situations, the complexity of modern networks combined with the inability to apply simplifying assumptions in many analysis problems (e.g., it is well-known that Markovian traffic assumptions are often inappropriate and can lead to misleading results) limit the applicability of purely analytic approaches. Even when analytic methods can be used, simulation is often used to validate the analysis.

Simulation is often used to gain insight into the behavior

of particular protocols and mechanisms under a variety of network conditions. Here, simulations of small to medium sized networks may be sufficient to gain critical insights into the behavior of the network. However, in other cases, simulation is used to understand the true effect of a new protocol, mechanism, network service, attack, or application when it is widely deployed on a large network such as the Internet. Interactions between the new proposed mechanism and the large amount and variety of competing traffic on the Internet are important considerations in gaining such an understanding. For example, if one wished to understand the effect of enabling multicast on a large Internet Service Provider (ISP), much larger simulations with a variety of competing traffic is needed to gain credible evidence regarding the effect of this proposed change. This paper examines the tools necessary for this latter class of research questions that require simulations of large networks. Specifically, we are interested in quantitatively characterizing the ability of modern parallel simulation systems to perform detailed simulations of computer networks.

Here, we focus primarily on packet-level simulations that model the transmission and queuing of individual packets as they travel through the network. Packet-level simulation is used extensively for protocol design and evaluation. It is widely used by many network simulation tools such as ns2 [1], GloMoSim and its commercial version Qualnet [2], and Opnet [3], among many others.

Most studies performed today using packet level simulations are limited to experiments modeling hundreds to thousands of network nodes (e.g., routers and end host computers). A critical reason is the amount of computation time and memory required to perform simulations of much larger networks is prohibitive. The goal of this paper is to explore the limitations of packet level simulation using parallel computation techniques.

2. Scalability Limits of Network Simulators

In practice, two factors that often limit the scale of packet

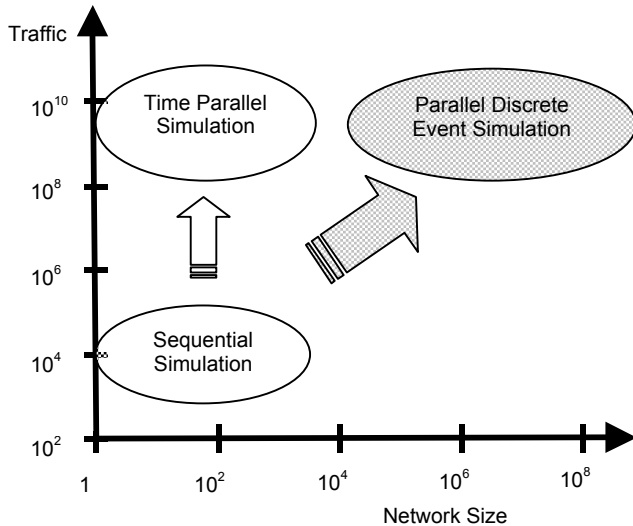


Figure 1. Notional diagram depicting network size (nodes and traffic) that can be simulated in real time.

level simulations that can be performed are the amount of memory required and the amount of computation time needed to complete each simulation run. Because the simulator requires a certain amount of memory to represent the state of each node, memory requirements increase at least linearly with the number of nodes. Memory requirements may increase faster than linearly in some simulators, e.g., to represent routing tables [4]. Thus, the amount of memory on the computer performing the simulation limits the number of nodes that can be represented.

At the same time, the amount of time required to complete a packet level simulation usually increases in proportion with the amount of traffic that must be simulated. This is because packet-level simulators are usually based on a discrete event paradigm where the computation consists of a sequence of event computations, so the execution time is proportional to the number of events that must be processed, assuming the average amount of time to process each event remains constant. Events in a packet-level simulation represent actions associated with processing a packet such as transmitting the packet over a link. Thus the execution time in a packet level simulation is proportional to the number of packets that must be processed. Assuming the modeler is only willing to wait a certain amount of time for the simulation to complete, this places a limit on the amount of traffic that can be simulated.

These observations place specific limits on the scale (size and amount of traffic) that can practically be modeled by a given simulator using a specific hardware platform. A notional diagram illustrating these scalability limitations is

shown in Figure 1. In this figure we normalize the execution time constraint by requiring that the simulation *execute in real time*, i.e., to simulate S seconds of network operation requires no more than S seconds of wallclock time. Each shaded area in Figure 1 represents a class of network simulators. Sequential simulation methods provide a baseline. As discussed later, our experiments indicate that contemporary network simulators such as ns2 with suitable optimizations to reduce memory consumption are able to simulate on the order of tens to hundreds of thousands of packet transmissions per second of wallclock time for networks containing thousands to tens of thousands of nodes on a modern workstation or personal computer.

Concurrent execution of the simulation computation can improve the scalability of the simulator both in terms of network size and execution speed, depending on the method that is used. Time parallel simulation methods such as that described in [5-12] parallelize a fixed sized problem by partitioning the simulation time axis into intervals, and assigning a processor to simulate the system over its assigned time interval. These methods increase the execution speed of the simulator enabling more network traffic to be simulated in real time, but do not increase the size of the network being simulated, (see Figure 1). For example, performance on the order of 10^{10} simulated packet transmissions per second are reported in [12]. Parallel discrete event methods utilize a space-parallel approach where a large network is partitioned and the nodes of each partition are mapped to a different processor, offering the potential for both network size and execution speed to scale in proportion to the number of processors. As discussed later, several parallel simulators using this approach have been developed. This approach is the primary focus of this paper. Finally, in addition to these “pure” time and space parallel approaches, some approaches combine both time and space parallelism, e.g., see [13].

3. Simulator Performance

Just as computer hardware designers utilize metrics such as the number of instructions of floating point operations a machine can process per second of wallclock time, it is useful to have a quantitative metric to characterize the performance of packet-level network simulators. Since the bulk of the simulation computation in a packet level simulator involves simulating the transmission and processing of packets as they travel from the source, through intermediate nodes (routers), to its destination, it is convenient to use the number of *Packet Transmissions that can be simulated per Second of wallclock time* (or PTS) as the metric to specify simulator speed. This metric is useful because given the PTS rate of a simulator, one can estimate the amount of time that will be required to

complete a simulation run if one knows the amount of traffic that must be simulated, and the average number of hops required to transmit a packet from source to destination.

Specifically, the execution time T for a simulation run can be estimated by the equation

$$T = N_T / PTS_S$$

where N_T is the number of packet transmissions that must be simulated and PTS_S is the number of simulated packet transmissions that can be simulated per second of wallclock time by simulator S . N_T depends on several factors. A first order estimate is

$$N_T = N_F * P_F * H_F$$

where N_F is the number of packet flows that must be simulated, P_F is the average number of packet transmissions per flow, and H_F is the average number of hops a packet must traverse in traveling from source to destination. It is important to note, however, that this is an approximation because it does not consider packet losses, nor other non-user traffic packets that must be sent by the protocol being used, e.g., acknowledgement packets using TCP. Nevertheless, when these considerations are accounted for, these equations provide a reasonable means to estimate execution time.

When the objective is to achieve real-time performance, the execution time constraint is PTS_S must be at least as large as $N_F * R_F * H_F$ where R_F is the average rate of packets transmitted (packets per second) per flow, again with the same caveats concerning losses and protocol-generated packets. For example, consider a simulation of 500,000 active UDP flows in real time where each flow produces traffic at a rate of 1 Mbps. Assuming 1 KByte packets, this translates to 125 packets/second. If the average path length is 8 hops, the simulator must execute at a rate of 500 Million PTS to achieve real time performance.

4. Parallel Network Simulation

Several parallel discrete event simulation systems have been developed to improve the scalability of network simulations. The traditional approach to realizing such a system is to create the parallel simulator “from scratch,” where all the simulation software is custom designed for a particular parallel simulation engine. The simulation engine provides, at a minimum, services for communication and synchronization. Examples of parallel network simulators using this approach include GloMoSim [2], TeD [14, 15], SSFNet [16], DaSSF [17], TeleSim [18], and the ATM simulator described in [19], among others. One advantage of this approach is the software can be tailored to execute efficiently in a specific

environment. Because new models must be developed and validated, this approach requires a significant amount of time and effort to create a useable system.

Another approach to parallel/distributed simulation involves interconnecting existing simulators. These federated simulations may include multiple copies of the *same* simulator (modeling different portions of the network), or entirely *different* simulators. The individual simulators that are to be linked may be sequential or parallel. This approach has been widely used by the military to create simulation systems for training or analysis, and several standards have been developed in this regard [20-24]. An approach linking multiple copies of the commercial CSIM simulator to create parallel simulations of queues is described in [25]. The federated approach offers the benefits of model and software reuse, and provides the potential of rapid parallelization of existing sequential simulators. It also offers the ability to exploit models and software from different simulators in one system [26].

Here, we focus on this latter approach. Specifically, we are concerned with two parallel simulators: Parallel / Distributed Network Simulator (PDNS) based on the widely used ns2 network simulation tool, and GTNetS, a recently developed tool designed for scalable federated network simulation. Both systems use the same underlying runtime infrastructure (RTI) software that provides services for communication and synchronization (see Figure 2). Each of these is described next.

4.1. RTI-Kit

RTI-Kit is a component of the Federated Simulations Development Kit (FDK). It provides a set of libraries for realizing RTI software. Specifically, the MCAST library provides group communication services for implementing publication/subscription communication among simulators (federates). The current version uses reliable, point-to-point communication to implement group communications. The TM-Kit library includes software to synchronize the computation, as will be discussed in greater detail momentarily. The other modules shown in Figure 2 implement mechanisms such as buffer management, priority queues, and low level communications support.

Synchronization mechanisms for parallel discrete event simulation can be broadly classified as conservative or optimistic. The principal responsibility of the synchronization mechanism is to ensure that each simulator (called a *federate*) processes events in time stamp order. Conservative mechanisms use blocking to ensure an event (message) is not processed until it can be

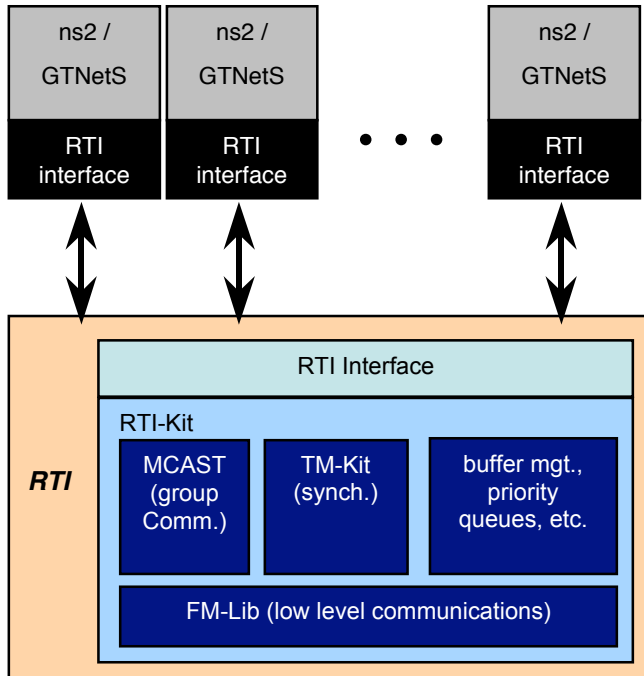


Figure 2. PDNS/GTNetS architecture.

guaranteed no event with a smaller time stamp will later be received. Optimistic mechanisms allow events to be processed out of time stamp order, but use some mechanism (e.g., rollback) to recover from such errors. These synchronization paradigms are discussed in greater detail in [27]. Conservative synchronization is better suited when federating existing sequential simulators because one need not add a rollback mechanism to the original simulator.

The synchronization mechanism used here is based on using a distributed snapshot algorithm to compute the lower bound on time stamp of future message that might be received. In addition to computing a global minimum, this algorithm must account for messages that have been sent, but have not yet been received, i.e., *transient messages*. Specifically, a variation of Mattern’s algorithm [28] is used for this purpose. At the heart of TM-Kit is the computation of reductions (global minimums) that account for transient messages using message counters. This computation and performance optimizations that have been developed are described in [29].

4.2. PDNS

PDNS is based on the ns2 simulation tool, and is described in greater detail in [30]. Each PDNS federate differs from ns2 in two important respects. First, modifications to ns2 were required to support distributed execution.

Specifically, a central problem that must be addressed when federating sequential network simulation software in this way is the global state problem. Each PDNS federate no longer has global knowledge of the state of the system. Specifically, one ns2 federate cannot directly reference state information for network nodes that are instantiated in a different federate. In general, some provision must be made to deal with both static state information that does not change during the execution (e.g., topology information), and dynamic information that does change (e.g., queue lengths). Fortunately, due to the modular design of the ns2 software, one need only address this problem for static information concerning network topology, greatly simplifying the global state problem.

To address this problem, a naming convention is required for an ns2 federate to refer to global state information. Two types of remote information must be accessed. The first corresponds to link end points. Consider the case of a link that spans federate boundaries, i.e., the two link endpoints are mapped to different federates. Such links are referred to in PDNS as *rlinks* (remote links). Some provision is necessary to refer to end points that reside in a different federate. This is handled in PDNS by using an IP address and network mask to refer to any link endpoint. When configuring a simulation in PDNS, one creates a TCL script for each federate that instantiates the portion of the network mapped to the federate, and instantiates rlinks to represent the “edge-connections” the span federates. The second situation where the global state issue arises concerns reference to endpoints for logical connections, specifically, the final destination for a TCP flow may reside in a different federate. Here, PDNS again borrows well-known networking concepts, and uses a port number and IP address to identify a logical end point.

The second way that PDNS differs from ns2 is it uses a technique called Nix-Vector routing. An initial study of ns2 indicated that routing table size placed severe limitations on the size of networks that could be simulated because the amount of memory required to store routing table information increased $O(N^2)$ where N is the number of network nodes. To address this problem, message routes are computed dynamically, as needed. The path from source to destination is encoded as a sequence of (in effect) output ports numbers call the Nix-Vector, leading to a compact representation. Previously computed routes are cached to avoid repeated re-computation of the same path. This greatly reduces the amount of memory required, and greatly increases the size of network that can be simulated on a single node. The NixVector technique is also applicable to the sequential version of ns2 and is used in PDNS.

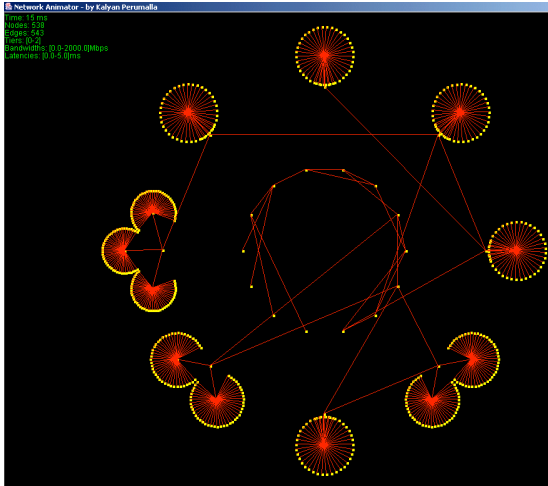


Figure 3. Topology of a single campus network.

4.3. GTNetS

GTNetS is tool recently develop at Georgia Tech for scalable network simulation [31]. Network models are written in C++. Like PDNS, the parallel version of GTNetS was realized using a federated simulation approach. Unlike PDNS, the sequential version of GTNetS was designed with parallel processing in mind. As such, it benefits from the experiences derived from creating PDNS, and it overall is more scalable and easier to use than PDNS.

GTNetS uses many of the same techniques used in PDNS to enable parallel execution. In particular, it uses rlinks, IP addresses, and address masks to identify link end points, and an IP address and port number to identify a remote end host. GTNetS also uses NIX-Vector routing, as described earlier, to economize on memory required to store routing table information.

5. Performance Study

The objective of this performance study was to stress test the PDNS and GTNetS simulations to quantitatively characterize the scale and performance of network simulations that can be done today. In the following we describe the methodology used to create and validate the network simulators. Specifically, we describe the networks that were used for this study. We then describe both sequential and parallel performance results on a Sun workstation, a Linux-based cluster and a large-scale supercomputer.

5.1. Methodology

The network topology, traffic, and parameters were based on a benchmark specification developed by the research group at Dartmouth College [16]. The benchmarks were

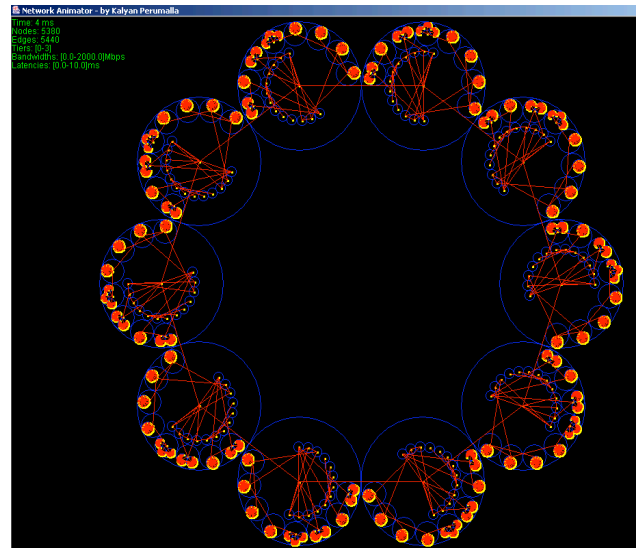


Figure 4. Topology of 10 campus networks configured in a ring.

developed as a set of baseline models for the network modeling and simulation community. The benchmark configurations were developed with the intention of facilitating the demonstration of network simulator scalability.

The basic “building block” used to construct large network configurations is referred to as a Campus Network (CN). The topology of a typical CN is shown in Figure 3. Each CN consists of 4 servers, 30 routers, and 504 clients for a total of 538 nodes. The CN is comprised of 4 separate networks. Net 0 consists of 3 routers, where one node is the gateway router for the CN. Net 1 is composed of 2 routers and 4 servers. Net 2 consists of 7 routers, 7 LAN routers, and 294 clients. Net 3 contains 4 routers, 5 LAN routers, and 210 clients.

Net 0 is connected to Net 2 and Net 3 via standalone routers. Net 1 is connected directly to Net 0 through a single link. All non-client links have a bandwidth of 2Gb/s and have a propagation delay of 5ms with the exception of the Net 0 to Net 1 links, which have a propagation delay of 1ms. Clients are connected in a point-to-point fashion with their respective LAN router and have links with 100Mb/s bandwidth and 1ms delay.

Multiple CNs may be instantiated and connected together to form a ring topology. This aspect of the network allows the baseline model to be easily scaled to arbitrarily large sizes. Multiple CNs are interconnected through a high latency 200ms 2Gb/s link via their Net 0 gateway router. A network consisting of 10 CNs is shown in Figure 4.

In our performance study, we focus on pure TCP traffic

requested by clients from server nodes. All TCP traffic is “external” to the requesting CN clients, *i.e.*, all the clients generate TCP traffic to/from servers in an adjacent CN in the ring (CN i communicates with CN $i+1$, etc.). Also, we use the short transfer case of the baseline model, where clients request 500,000 bytes from a random Net 1 server. The TCP sessions start at a time selected from a uniform distribution over the interval from 100 and 110 seconds of simulation time from the start of the run.

5.2. Validation

The simulators were validated by creating three separate implementations using different network simulation packages, and comparing the resulting statistics that were produced. Specifically, the campus network configuration was realized using PDNS, GTNetS, and Opnet; Opnet is a widely used commercial network simulation package.

A single campus network configuration was used for validation purposes. The simulators were run, and processed over 4.5 million packet transmissions. Because the different simulators use different random number generators and the model implementations differed slightly (e.g., the different packages use different means to specify network traffic), statistical results from the different packages were not identical. However, we verified that average end-to-end delay statistics produced by the simulators differed by less than 3%.

5.3. Hardware Platforms

Experiments were performed across a variety of platforms. Initial sequential experiments to establish baseline performance were conducted on a 450 MHz Sun UltraSPARC-II machine running the Solaris Operating System.

Parallel measurements were performed on both a cluster computing platform at Georgia Tech and a supercomputer at the Pittsburgh Supercomputing Center (PSC). The cluster computing platform is a large Linux cluster consisting of 17 machines, or a total of 136 CPUs. Each machine is a Symmetric Multi-Processor (SMP) machine with eight 550MHz Pentium III XEON processors. The eight CPUs of each machine share 4 GB of RAM. Each processor contains 32KB (16KB Data, 16KB Instruction) of non-blocking L1 cache and 2MB of full-speed, non-blocking, unified L2 cache. An internal core interconnect utilizes a 5-way crossbar switch connecting two 4-way processor buses, two interleaved memory buses, and one I/O bus. The operating system is Red Hat Linux 7.3 running a customized 2.4.18-10smp kernel. The 17 SMP machines are connected to each other via a Dual Gigabit Ethernet switch with EtherChannel aggregation. Our RTI software uses shared memory for communications within

an SMP, and TCP/IP for communication across SMPs.

Supercomputer experiments were conducted on the “Lemieux” machine at PSC. This machine includes 750 HP-Alpha ES45 servers. Each server contains 4 GBytes of memory and four 1.0 GHz CPUs. A high speed Quadrics switch is used to interconnect the servers.

5.4. Performance Measurements

Sequential performance of ns2 (version 2.1b9) and GTNetS in simulating a single CN on a Sun/Solaris machine and an Intel/Linux machine are shown in Table 1. As can be seen, both simulators execute on the order of 40K to 44K PTS on the Sun. A single Intel/Linux processor in the cluster machine executes ns2 simulations about 2.2 times faster than the same simulation running on the Sun/Solaris platform.

Table 1. Baseline Performance Measurements.

	ns2 (Sun)	GTNets (Sun)	ns2 (Intel)
Events	9,107,023	9,143,553	9,117,070
Packet Transmissions	4,546,074	4,571,264	4,552,084
Run Time (sec)	104	112.3	48
PTS	43,712	40,706	94,814

The parallel simulation experiments described here scale the size of the simulated network in proportion to the number of processors used. This is a widely accepted approach for scalability studies in the high performance computing community. It also circumvents the problem of having a sequential machine with enough memory to execute the entire model (necessary to compute speedup), which would not be possible for the large simulations that were considered here.

Figure 5 shows the performance of PDNS on the Linux cluster as the problem size and hardware configuration are increased in proportion. It can be seen that performance increases linearly as the scale is increased. The largest configuration simulates 645,600 network nodes on 120 processors, yielding performance of approximately 2 million PTS. Since these experiments were performed, optimizations to PDNS were later added, and found to yield aggregate performance of 5.5 Million PTS using 136 processors in simulating a campus network configuration containing 658,512 nodes and 616,896 traffic flows. A

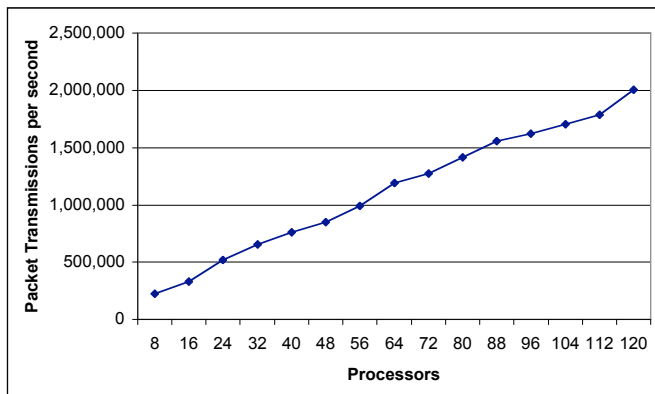


Figure 5. PDNS performance on cluster of Intel SMPs; each processor simulates 10 campus networks (up to 645,600 network nodes with 120 processors).

“SynFlood” Denial of Service attack scenario with 25,000 attacking hosts and background traffic was also executed on the simulator, yielding performance of 1.5 Million PTS on 136 processors. This version of PDNS was based on ns2 version 2.26, which we observed to be somewhat slower than version 2.1b9 that was used for the previous experiments.

Performance measurements of PDNS executing on the PSC are shown in Figure 6. The largest configuration contains approximately 4 million network nodes, and yields a simulator performance of 106 Million PTS on 1536 processors.

Experiments with GTNetS on PSC also demonstrated scalable executions could be obtained for that simulator. An execution containing 5.5 million nodes and 5.2 million flows on 512 processors yielded a performance of 12.3 Million PTS. Another experiment demonstrated that 1.0 million web browsers generating http traffic using the model described in [32] on a network containing 1.1 million nodes executed in near real time, requiring 541 seconds of wallclock time to simulate 300 seconds of network operation. This experiment demonstrates that real-time packet level simulation of million node networks is within reach today.

6. The Future

The goal of this paper is to try to characterize quantitatively the capability of parallel simulation tools to simulate large-scale networks, and to point out that the ability now exists to simulate large networks. This is by no means to imply scalable network simulation is a “solved problem”! Much additional research and development is required to effectively exploit these capabilities.

On the modeling side, creating realistic models of the

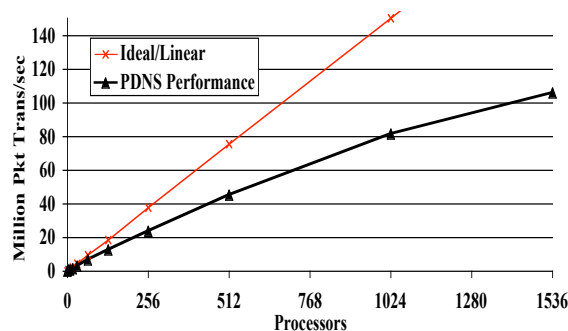


Figure 6. PDNS Performance on Pittsburgh Supercomputer.

Internet remains an extremely challenging problem. Some of the key issues that must be addressed are described in [33]. For example, the topology, configuration, and traffic of the Internet *today* is not well understood and is constantly changing, let alone the Internet of tomorrow that is targeted by most simulation studies. Methodologies to effectively validate and experiment with large-scale network simulations must be developed. Much work is required to make the parallel simulation tools easily accessible and usable by network researchers who are not experts in parallel processing. Robust parallel performance is needed; modest changes to the configuration of the network being simulated may result in severe performance degradations. These and many other issues must be addressed before large-scale network simulation tools can reach their fullest potential.

Acknowledgments

This research was supported under DARPA contract N66001-00-1-8934 and NSF grants ANI-0136939 and ANI-9977544. The assistance of the staff at the Pittsburgh Supercomputing Center is also gratefully acknowledged.

References

1. Breslau, L., et al., *Advances in Network Simulation*. IEEE Computer, 2000. **33**(5): p. 59-67.
2. Zeng, X., R. Bagrodia, and M. Gerla, *GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks*, in *Proceedings of the 1998 Workshop on Parallel and Distributed Simulation*. 1998. p. 154-161.
3. Chang, X., *Network Simulations with OPNET*, in *Proceedings of the 1999 Winter Simulation Conference*. 1999.
4. Riley, G.F., R.M. Fujimoto, and M.A. Ammar, *Stateless Routing in Network Simulations*, in *Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 2000.
5. Lin, Y.-B. and E.D. Lazowska, *A Time-Division algorithm for Parallel Simulation*. ACM Transactions on Modeling and Computer Simulation, 1991. **1**(1): p.

- 73-83.
6. Andradottir, S. and T. Ott, *Time-Segmentation Parallel Simulation of Networks of Queues with Loss or Communication Blocking*. ACM Transactions on Modeling and Computer Simulation, 1995. **5**(4): p. 269-305.
 7. Greenberg, A.G., B.D. Lubachevsky, and I. Mitrani, *Algorithms for Unboundedly Parallel Simulations*. ACM Transactions on Computer Systems, 1991. **9**(3): p. 201-221.
 8. Fujimoto, R.M., I. Nikolaidis, and A.C. Cooper, *Parallel Simulation of Statistical Multiplexers*. Discrete Event Dynamic Systems: Theory and Applications, 1995: p. 115-140.
 9. Wu, H., R.M. Fujimoto, and M. Ammar, *Time-Parallel Trace-Driven Simulation of CSMA/CD*, in *Proceedings of the Workshop on Parallel and Distributed Simulation*. 2003.
 10. Jones, K.G. and S.R. Das, *Time-Parallel Algorithms for Simulation of Multiple Access Protocols*, in *Ninth International Symposium on modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 2001.
 11. Greenberg, A., et al., *Efficient Massively Parallel Simulation of Dynamic Channel Assignment Schemes for Wireless Cellular Communications*, in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*. 1994. p. 187-194.
 12. Fujimoto, R.M., I. Nikolaidis, and A.C. Cooper, *Parallel Simulation of Statistical Multiplexers*. Journal of Discrete Event Dynamic Systems, 1995. **5**: p. 115-140.
 13. Szymanski, B.K., Y. Liu, and R. Gupta, *Parallel Network Simulation under Distributed Genesis*, in *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*. 2003. p. 61-68.
 14. Perumalla, K., R. Fujimoto, and A. Ogielski, *TeD - A Language for Modeling Telecommunications Networks*. Performance Evaluation Review, 1998. **25**(4).
 15. Poplawski, A.L. and D.M. Nicol, *Nops: A Conservative Parallel Simulation Engine for TeD*, in *12th Workshop on Parallel and Distributed Simulation*. 1998. p. 180-187.
 16. Cowie, J.H., D.M. Nicol, and A.T. Ogielski, *Modeling the Global Internet*. Computing in Science and Engineering, 1999.
 17. Liu, J. and D.M. Nicol, *DaSSF 3.0 User's Manual*. 2001.
 18. Unger, B., *The Telecom Framework: a Simulation Environment for Telecommunications*, in *Proceedings of the 1993 Winter Simulation Conference*. 1993.
 19. Pham, C.D., H. Brunst, and S. Fdida, *Conservative Simulation of Load-Balanced Routing in a Large ATM Network Model*, in *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*. 1998. p. 142-149.
 20. Miller, D.C. and J.A. Thorpe, *SIMNET: The Advent of Simulator Networking*. Proceedings of the IEEE, 1995. **83**(8): p. 1114-1123.
 21. IEEE Std 1278.1-1995, *IEEE Standard for Distributed Interactive Simulation -- Application Protocols*. 1995, New York, NY: Institute of Electrical and Electronics Engineers, Inc.
 22. IEEE Std 1278.2-1995, *IEEE Standard for Distributed Interactive Simulation -- Communication Services and Profiles*. 1995, New York, NY: Institute of Electrical and Electronics Engineers Inc.
 23. Kuhl, F., R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture for Simulation*. 1999: Prentice Hall.
 24. IEEE Std 1516.3-2000, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) -- Interface Specification*. 2000, New York, NY: Institute of Electrical and Electronics Engineers, Inc.
 25. Nicol, D. and P. Heidelberger, *Parallel Execution for Serial Simulators*. ACM Transactions on Modeling and Computer Simulation, 1996. **6**(3): p. 210-242.
 26. Perumalla, K., et al., *Experiences Applying Parallel and Interoperable Network Simulation Techniques in On-Line Simulations of Military Networks*, in *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*. 2002. p. 97-104.
 27. Fujimoto, R.M., *Parallel and Distributed Simulation Systems*. 2000: Wiley Interscience.
 28. Mattern, F., *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation*. Journal of Parallel and Distributed Computing, 1993. **18**(4): p. 423-434.
 29. Perumalla, K.S., et al., *Scalable RTI-Based Parallel Simulation of Networks*, in *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*. 2003. p. 97-104.
 30. Riley, G., R.M. Fujimoto, and M. Ammar, *A Generic Framework for Parallelization of Network Simulations*, in *Proceedings of the Seventh International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 1999. p. 128-135.
 31. Riley, G.F., *The Georgia Tech Network Simulator*, in *Proceedings of the Workshop on Models, Methods, and Tools for Reproducible Network Research (MoMe Tools)*. 2003.
 32. Mah, B.A., *An Empirical Model of HTTP Network Traffic*, in *INFOCOM*. 1997. p. 592-600.
 33. Floyd, S. and V. Paxson, *Difficulties in Simulating the Internet*. IEEE/ACM Transactions on Networking, 2001. **9**(4): p. 392-403.