High-fidelity Distributed Simulation of Local Area Networks*

Livio Ricciulli
Computer Science Laboratory
SRI International
Menlo Park, California, 94025

Abstract

The Adaptable Network COntrol and Reporting System (ANCORS) project merges technology from network management, active networking, and distributed simulation in a unified paradigm to assist in the assessment, control, and design of computer networks. After motivating our approach to network engineering, we describe an initial ANCORS prototype system. In particular, we describe a high-fidelity model of a Unix-based networking protocol stack, and characterize and compare two different distributed simulation synchronization mechanisms that were used to simulate an Ethernet-based Local Area Network.

1 Introduction

The Internet will become increasingly dynamic. Changes in the Internet will affect both its control mechanisms and the nature of information exchanged. New trends in network design [12, 7, 4, 13, 11, 1] seek to render network protocols more flexible and extensible, and to thus improve their overall usefulness. Configuration changes can be as dynamic as interpreting and executing a few predefined instructions as a network packet is received, causing new protocols to be loaded on demand, or modifying, deleting, or adding more permanent objects that implement application-specific network services. In addition to changing how data is transmitted, the introduction of new technologies as they become available may change the nature of network traffic. The Internet phone and the increasing interest of cellular phone companies in accessing services on the Internet are examples of future technologies that may greatly affect Internet traffic.

The current state of the art in network engineering, monitoring, and control must improve dramatically. It is becoming increasingly apparent that effective management of large, ever-changing networks depends on sophisticated monitoring to help understand the way a network changes. As new interdependencies arise in sharing resources beyond the domain level, monitoring capabilities, like application-specific protocols, should be able to change over time, should adapt to new conditions as they develop, and should be scalable.

In addition to sophisticated and adaptable monitoring, future networks would greatly benefit from simulation services so that network engineers can experiment with new network technologies without compromising network operations. Current network engineering tools can scale only to small and relatively simple networks and are not inter-operable. Tools will be required to scale far beyond current capabilities and will need to promote inter-operability and model reuse. In addition to evaluating performance metrics to compare one design with another, network engineering tools should implement a development environment for validating new designs.

Hardware design tools have reached a very high level of sophistication and can assist hardware designers in all phases of design and development. Such tools can support a wide spectrum of levels of abstraction, from high-level purely behavioral specifications, to increasingly finer detailed structural layouts, all the way down to the actual design of the transistors on the silicon. Simulation is used throughout all phases of this design process, and it is the main mechanism that guides design choices. As for hardware, network design should also be carried out in an environment that can offer a variable degree of abstraction and that can offer simulation as a pivoting technology to guide development. We argue that, because of the organic nature of current networks and their fast evolution pace, design should be carried out in the real network itself. Future networks will need tools that can adapt their functionality and scope, and that can grow and change with the network itself. To generate results that accurately predict network behavior and performance, simulation and analysis must be closely tied to the actual, rather than on artificially generated, network traffic conditions. To that end, the tools should run on the network itself, taking the actual observable

* This work was supported by DARPA contract number DABT63-97-C0040.

traffic conditions into consideration. Before committing network-wide changes such as the alteration of the network routing algorithm, an operator may want to conduct simulation experiments that can predict the behavior of the network under the new algorithm without affecting network reliability. That is, analysis and design tools should be available to a wide range of network operators, who could act independently or in collaboration with one another.

ANCORS leverages network management and introduces simulation as an additional network service. Integrating distributed simulation with network management has four main advantages: (1) it naturally supports reuse of both simulation software and network models, (2) the simulated models can use real network data produced by the monitoring agents, thus improving fidelity, (3) the consumers of the data (the simulation models) are placed close to the origin of the data to reduce overhead, and (4) the monitoring and control capabilities of network management can be reused to monitor and control the simulations.

We have primarily focused on implementing (1) a prototype of a root ANCORS agent that dynamically accepts and instantiates simulations and/or monitoring processes, and (2) a representative example of deployable engineering service that can be used to conduct very accurate, end-to-end quantitative network experimentation.

In the following sections, we will discuss some of the key design ideas and implementation details of our prototype network engineering system and will provide experimental results that characterize its performance. In the process of describing the designs, we will emphasize a novel, scalable, and efficient synchronization technique used to coordinate the distributed simulation of virtual LANs.

2 ANCORS Agent

The current prototype of an ANCORS agent accepts commands to download a binary-compatible executable from a remote location. The binary executable is specified as a URL; the ANCORS agent, after downloading the code with an HTTP GET command, strips the HTML header from the received code and loads it as a dynamic library. As shown in Figure 1, the download command can either (1) trigger the ANCORS agent to duplicate itself by using a fork system call to run the downloaded code (as well to accept further commands), or (2) simply add a thread to an existing process. In either case, the downloaded code is essentially a shared library initially accessed through a universally predefined entry point (**init()**). This initial configuration function can simply transfer

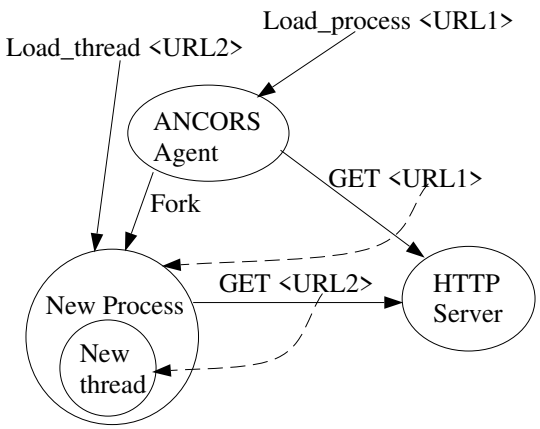

Figure 1: ANCORS agents can spawn a new process or a thread within a process

control to the downloaded code for execution, or it can first gather runtime configuration data in a manner specific to each instance [2] and then explicitly started. In this first prototype we have assumed that the configuration and monitoring functions are embedded inside the deployed code itself. These functions return HTML code that is fed to the network manager to gather some user-defined runtime parameters or for displaying usage data. The network engineer configures and monitors the downloaded code with HTML forms that are then pushed back through a CGI script to the created service. Each operation returns HTML forms that in turn may call other functions, thus allowing a hierarchical organization of HTML pages. Future extensions to our management system will allow the incorporation of existing NM software based on SNMP and Java [3] and the creation of new decentralized management solutions based on the concept of delegation (perhaps using Java as the delegation language).

The ANCORS agent offers a set of built-in primitives to the downloaded services that, in addition to standard native system functionality (I/O, memory management, networking), provide (1) multithreading (nonpreemptive), (2) LAN multicast emulation, and (3) global time synchronization. These support services were designed to facilitate distributed simulation network engineering applications, as well as some forms of sophisticated network monitoring.

[2] For example, it could use a MIME-encapsulated document to specify configuration and monitoring operations to be performed by the management system.

[3] All product and company names mentioned in this paper are the trademarks of their respective holders.

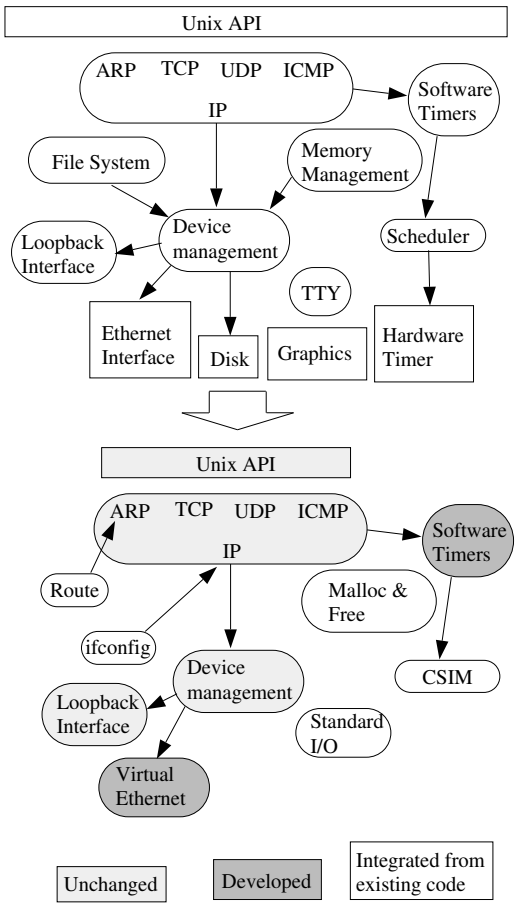

Figure 2: Linux transformation

3 Virtual Networking Using ANCORS

To date we have produced a representative example of an engineering network service that emulates a Unix kernel. The service was obtained by modifying a Linux operating system to allow its execution in user mode.

As shown in Figure 2, the modifications of the operating system replaced all lower-level, hardware-dependent procedures and interfaces with user-level counterparts. We deleted the file system support and incorporated all necessary configuration procedures (like ifconfig and route) as additional system calls. Memory management and standard I/O were completely deleted and replaced by user-level functions (malloc, free, and printf) contained in the standard *c library*. The scheduling and the software timers were completely replaced and implemented on top of the nonpreemptive threading offered by the simulation package (CSIM [10]).

The resulting service executes in a virtual timescale, offers the identical networking behavior of a real Linux kernel, and can therefore be used as a vehicle to instantiate high-fidelity distributed simulations of virtual networks. One of the model's configuration functions accepts different timing configurations to approximate the protocol stack timing behavior of four different kernels (SunOS 4.13, SunOS 5.5, Linux 2.02, and BSD 2.2). To date we have made some gross approximations for SunOS 4.13, Linux 2.02, and BSD 2.2 but we have refined the models for SunOS 5.5 to yield very accurate timing estimates to be used in the first set of experiments documented in this paper in Section 5.

The virtual kernel offers the network application programming interface (API) of the real Linux counterpart and therefore can be used to reproduce a wide range of loading conditions. ANCORS's ability to add and delete threads can be used in this application to dynamically change loading conditions (by adding or deleting user-defined loading threads) or by injecting user-defined monitoring probes into the kernel so that specific parameters can be observed. For the time being, we have implemented some simple load models borrowed from classic queuing theory. As shown in Figure 3, the user-definable loads may be produced by either closely mimicking real load conditions recorded by network monitoring services or by linking some real applications to the virtual kernel to generate application-specific loads (for example, originating from a real-time video stream).

The virtual kernels communicate with each other through TCP, and automatically configure themselves to participate in emulated multicast sessions that parallel the behavior of virtual Ethernet segments. Initially, all real hosts are aware of all other real hosts that may share the same virtual Ethernet segment. Each virtual Ethernet segment network address assigned to a virtual host is transformed through a hash function into a port number. When the virtual kernel initializes its virtual interfaces, the multicast emulation initialization procedure tries to connect to all known peers that may share a virtual Ethernet segment, using the port associated with each virtual interface. Thus, if two or more virtual hosts share a virtual network address, and therefore use the same port, they establish a TCP connection used to tunnel virtual Ethernet packets. When a virtual host sends a simulation packet pertaining to a particular virtual Ethernet segment, it sends it to all virtual hosts that have connected to the associated port.

The deployment of a virtual network is achieved by downloading and configuring several virtual kernels through ANCORS agents. All these operations can be performed either through a standard HTML

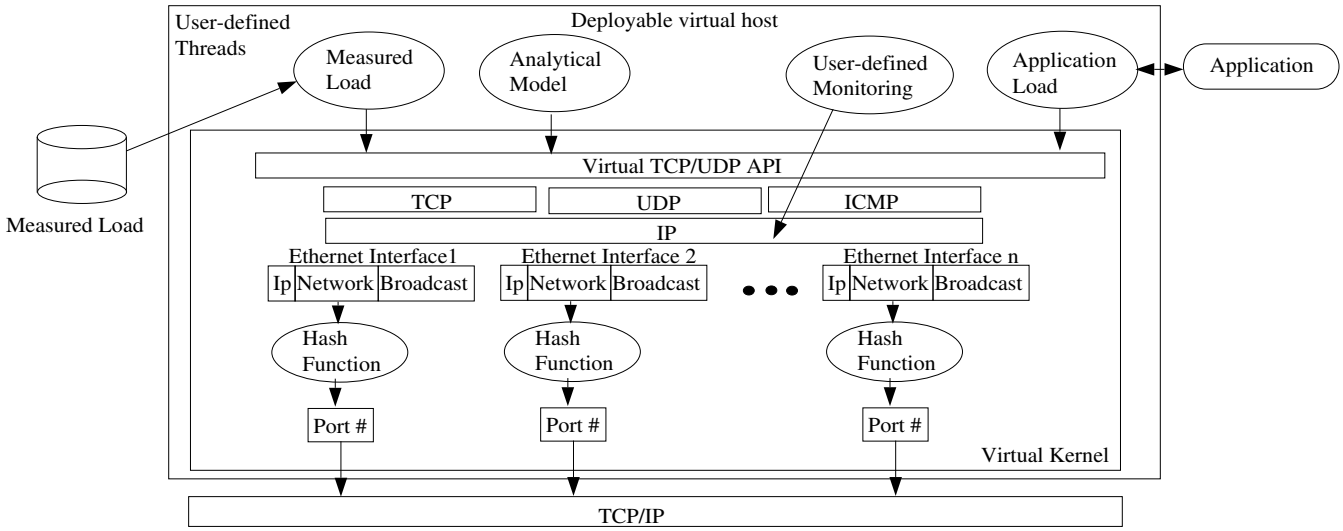

Figure 3: Deployable virtual host

browser or by using a script. We have so far instantiated several virtual networks running on a network of workstations including Sun SPARCstation 20s, UltraSPARCs, and Intel-based machines running BSD and Linux operating systems.

4 Synchronization

A practical distributed network engineering tool should allow varying degrees of modeling refinement and be able to trade off speed, level of abstraction, and estimation accuracy in a flexible way. Traditionally, simulation techniques have been able to offer good flexibility for modeling systems at different levels of abstraction but have not been able to cope well with high-fidelity simulation of large systems in a scalable way. Traditional distributed simulation techniques can offer mechanisms to effectively serialize the event ordering of distributed nodes, thus increasing simulation power by employing more computing resources. Although these techniques are very useful for performing analytical experiments in a distributed way, they can impose high levels of overhead to keep the distributed nodes synchronized.

We are developing a novel technique to trade off the accuracy of simulated time with the speed and scalability of distributed simulation while retaining the flexibility of being able to offer an arbitrary level of abstraction for modeling the target system. We have derived some of the ideas from the theory of Lamport's virtual time [6], and therefore we have named our technique Lamport Synchronization (LS). LS distributed simulation is based on the idea of decoupling the simulation into a behavioral component that emulates the semantic behavior of the modeled system (i.e., preserve the causal relationship among the distributed events according to the semantics of the system being modeled) and a timing component that estimates the virtual time in which the executed behavior could have been executed. We have successfully applied these ideas to the efficient distributed simulation of a massively parallel architecture [8, 9] and, as we will see in the following sections, we have also recently applied our techniques to simulate Ethernet LANs within the scope of the ANCORS project.

In addition to LS, for comparison purposes we have also implemented a distributed simulation synchronization algorithm derived from the algorithms proposed by Chandy and Misra (CM)[2]. We have not yet implemented a synchronization scheme based on Time-warp [5, 3] but we plan to do so in the near future. In the following sections we will describe in detail and compare the two synchronization mechanisms implemented so far (LS and CM) when applied to the distributed simulation of local area networks, using ANCORS.

4.1 Common Assumptions

Both the CM and the LS synchronization protocols use TCP to transmit messages between the logical processes (LPs) to guarantee that messages are reliably delivered in program order.

We model contention by broadcasting virtual Ethernet packets not only to the destination node but to all other nodes sharing the same Ethernet segment. A more efficient solution would be to centralize the modeling of the Ethernet in a single host, thus reduc-

ing the ratio of the number of real messages sent for each virtual message from n to two[4] (where n is the number of hosts sharing the same Ethernet segment). We plan to experiment with this solution in the near future even though we realize that careful mapping should be used to avoid creating a bottleneck in the host modeling the Ethernet.

Another common design methodology was to encapsulate the simulation synchronization mechanisms into the software module that implements the virtual Ethernet without involving either the application or the particular protocol stack used. Although this requirement heavily penalizes the CM mechanism, it permits reuse of the synchronization codes with different applications and/or protocols in a flexible and modular way.

4.2 LS Mechanism

Each logical process (LP) has two separate clocks —a local clock T_L and a global clock that measures global simulated time (T_g) (the estimation of what would be the physical time if the simulated system was real). In the LS scheme, T_g does not order the execution of the simulation, but it is derived simply to obtain a performance measurement. We start from an interleaving of distributed events that is guaranteed to be legal by the simulated system synchronization semantics, and we derive a T_g estimate in which the observed execution could have been carried out. We always execute all local simulation mechanisms with respect to T_L and perform all global operations with respect to T_g. Events executing with respect to T_L typically enforce the behavioral correctness of the distributed system, while events executed with respect to T_g tend to be executed purely for measurement purposes. The separation of the T_L and T_g clocks is only a logical one and does not need two separate event lists because scheduling T_g events can be piggybacked on the scheduling of the T_L events. For example, for scheduling an event with respect to T_g to time T one would execute $while(T_g < T)hold(x)$ where x is a reasonable amount of T_L time during which T_g may change. T_g in general is explicitly adjusted during the execution of the simulation by modeling the synchronization interactions of a distributed system as a series of requests, acquisitions, and releases of global resources (i.e., the Ethernet segment). Once the resource becomes available the execution is resumed with T_g adjusted to an appropriate value. For an in-depth description of the LS technique applied to the distributed simulation of parallel computers see [8, 9].

The LS distributed simulation technique applied to simulating LANs is best described with a simple example. In Figure 4 two virtual nodes Va and Vb that share an Ethernet segment are executing on two different physical processors. In this example, node Va sends a message to Vb. After Va accounts for the protocol stack overhead, the message is time-stamped with T_g (time T_1 in Figure 4). When the message reaches Vb, the timestamp is updated by adding the latencies of the network being simulated. In particular, in this case Vb would add the transmission latency of the Ethernet and its receiving latency calculated at the moment the message is received (time $T_3 - T_2$ in Figure 4).

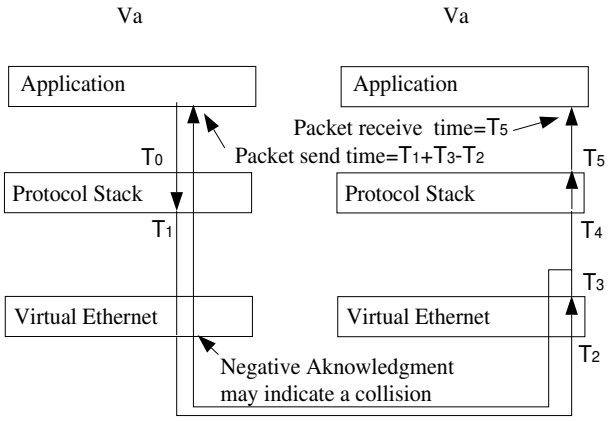

Figure 4: LS scheme timing measurement. The protocol stack latency is measured locally while the Ethernet timing is measured at the destination. If a collision is detected, a negative acknowledgment will cause the source to back off and try again.

Depending on whether or not a collision has been detected, Vb then sends back a positive or negative acknowledgment to the originator. If positive, the acknowledgment bears a timestamp that reflects the time the message sending should have completed in Va's time reference ($T_1 + T_3 - T_2$). The timestamp of the acknowledgment is then used by Va for adjusting the estimate of T_g. If the acknowledgment is negative (Vb detected a collision), Va triggers the standard Ethernet back-off and retry mechanism. On the receiving end, if the message sent by Va did not trigger a collision, the application receives the message at T_g T_5 and the timestamp is discarded.

4.3 Chandy and Misra

In this scheme, all LPs keep only one clock that orders both local and global events in a synchronous way. With this well-known synchronization scheme, each distributed node waits for receipt of a message

[4] In the LS case from n+1 to 3

from each of the other hosts that could possibly send a message to compute the latest time at which it is safe to send a message. Deadlock is avoided by broadcasting null messages that advance time artificially when necessary.

In our experiments each host waits for messages transmitted by all other hosts that are connected on the same virtual Ethernet segment and sends messages on the virtual Ethernet only if the clock plus the Ethernet latency is less than the timestamp of the latest message received.

Each time an LP receives on all the input queues, it broadcasts a null message to all other LPs bearing a timestamp with the local time plus the local estimate of lookahead time. If the LP is wanting to transmit, the lookahead time is set to the timestamp of the next message to be sent out; otherwise, the lookahead time is set to the transmission latency of the smallest possible Ethernet packet (60 bytes). Our lookahead estimate is not very sophisticated because we did not want to specialize this technique to a particular application. A more intelligent use of lookahead would look into the protocol stack or, even better, use information embedded in the application for increasing the lookahead time and consequently improve performance.

4.4 Qualitative Comparison

Because of the nondeterminism introduced by the runtime behavior of the physical hosts, in the LS scheme different scheduling behaviors might be observed for the same simulated application. The applications, if correct, for each run, will always produce the same results but might arrive at those results in different ways following different scheduling behaviors. This critical aspect about the LS scheme might inspire skepticism. We try to show why we think this nondeterminism is an acceptable attribute of LS and can in some cases be desirable.

As detailed in Section 5.2, we have observed that the execution of a real distributed system exhibits nondeterministic behavior because of runtime system interferences. We therefore believe that it is not practical to introduce performance overhead in the simulation of a distributed system execution to make its behavior totally deterministic when in reality this does not happen. Furthermore, the inaccuracies introduced by a nondeterministic scheduling behavior in our judgment can be very small compared to other kinds of inaccuracies introduced by simulation. (Abstracting the behavior of certain components of the design and adopting inaccurate timing estimates for components that have never been built are examples of more important sources of inaccuracies introduced by simulation.)

Some metrics greatly depend on the scheduling behavior of an application (false sharing is a good example), a deterministic sequential simulation (or an equivalent distributed one) is the only alternative if one wants to obtain very accurate measurements. The CM scheme offers a deterministic scheduling behavior at the expense of higher synchronization cost and can be used in those cases in which repeatability of the results needs to be enforced at a very fine grain level. Another hidden cost that is unique to the CM scheme and that may become more and more relevant in the context of Web-based simulations is that if one of the LPs goes down, the simulation stops even if the LP is not semantically necessary for the simulation to proceed. In other nonconservative distributed simulation approaches the simulation would make forward progress even in the case of failures of some LPs.

5 Quantitative Comparison

Using ANCORS we have performed several distributed simulations aimed at evaluating the performance and scalability of the two synchronization schemes we have implemented (LS and CM) and comparing the timing accuracy of our models with the real system being modeled.

The experiments consist in simulating an Ethernet segment shared by four virtual hosts. The virtual hosts establish virtual TCP connections and exchange variable-length messages. The four virtual hosts can be spawned on four physical hosts or on a single host. Three of the four virtual hosts continuously send short messages [5] to the fourth host and block on the reception of a message of length derived from an exponential distribution with variable mean and standard deviation. This kind of simulation tries to emulate the dynamics of a small LAN in which a server is being queried by three clients for read-only data (like in HTTP).

5.1 Efficiency

We have compared the efficiency of both the LS and CM schemes for synchronizing the simulation of our high-fidelity virtual LAN. Table 1 summarizes the measurements we obtained. For both synchronization schemes we report the slowdown of the distributed simulation. The slowdown is calculated by dividing the physical time needed by the simulation by the simulated global time. Table 1 also reports the slowdowns for simulations running on four workstations [6] or run-

[5] Messages with mean length of 10 bytes and standard deviation of 10 bytes

[6] Sun Microsystems UltraSPARCs

Table 1: Simulation Efficiency

Simulation Efficiency						
Packet Size	1 Workstation Slowdown		4 Workstation Slowdown		Speedup	
	C&M	LS	C&M	LS	C&M	LS
10	5993	1830	2774	641	2.16	2.85
1024	3989	523	1426	177	2.8	2.96
2048	3921	401	1375	128	2.85	3.12
4096	3884	307	1295	98	3.0	3.13
8192	3844	262	1265	86	3.04	3.03
16,384	3834	237	1257	81	3.05	2.92
32,768	3836	225	1247	78	3.07	2.88
65,536	3838	219	1243	77	3.09	2.82
131,072	3828	216	1214	78	3.15	2.76

Table 2: Simulation Accuracy

Simulation Accuracy						
Packet Size	Real		C&M		LS	
	B/s	S.D.	B/s	S.D.	B/s	S.D.
10	13041	235	41385	0	14566	1055
1024	275397	2856	371477	0	299509	6280
2048	326715	8188	379014	0	328191	7155
4096	355251	2858	389770	0	346504	6288
8192	364256	3315	394557	0	355683	5919
16,384	364299	4468	397555	0	354758	5247
32,768	360321	2661	400154	0	361155	4122
65,536	357389	2721	401124	0	363721	3377
131,072	349829	7860	402827	0	365064	645

ning on a single workstation. In the single workstation experiments, the messages between the LPs (which are separate Unix processes) are exchanged using the loop-back device, thus eliminating any overhead due to the Ethernet; in this case the bottleneck is the host processing power. Because of the lack of lookahead information, the CM simulations are much slower and the slowdown is quite constant. The LS scheme is much faster and becomes more efficient as the packet size increases (this is because the simulation packets are at most 60 bytes long and as the real system saturates the Ethernet it is slowed down with respect to the simulation). The speedup is calculated by dividing the slowdown of 1 workstation by the slowdown of the 4 workstations. It is quite good for both simulation schemes, thus suggesting that this kind of high-fidelity simulation is a good candidate for parallelization.

5.2 Accuracy

Table 2 compares the estimation of the overall throughput obtained with our high-fidelity simulations using the LS and CM schemes with the throughput of the real system being modeled. In addition we report the standard deviation of the throughput estimates that were obtained over multiple experiments. Both the CM and the LS schemes give fairly good estimates of the overall throughput, with the CM requiring some adjustments in the timing model. CM offers a deterministic behavior and thus a zero standard deviation of the measurements. Notice that in both the real system and in the LS simulations the standard deviation is considerable and it does not seem to be related to the packet size. In both the real system and the LS simulations the standard deviation is about 10%, thus suggesting that a deterministic simulation may give an erroneous picture of the system's dynamics. The LS scheme, on the other hand, seems to offer the same degree of variability with respect to the real execution, thus offering more simulation fidelity. We plan to conduct more experiments modeling more complex topologies and being executed on more heterogeneous environments to better characterize the variability of the LS simulations as they are compared to the variability of the system being modeled.

6 Conclusion

As the Internet will become more dynamic both in its control mechanisms (protocols) and in the user requirements, new, efficient, and user-friendly network engineering tools will be required to go far beyond current capabilities. ANCORS offers a new paradigm for designing, deploying, and monitoring networks that integrates simulation as an additional component of network management. In this paper we have outlined an initial ANCORS prototype consisting of multiple distributed agents capable of deploying network engineering applications in a flexible and machine-independent way and a high-fidelity model of a Unix network protocol stack. We have used this initial ANCORS prototype to experiment with two synchronization protocols for the distributed simulation of a high-fidelity LAN. In comparing the two protocols we have both qualitatively and quantitatively described the two approaches and have concluded that the LS implementation, given our methodological assumptions, is more efficient. In the near future, using both simulation synchronization protocols, we plan to experiment with more complex network loads and larger and more interesting topologies. In particular, we plan to explore both simulation techniques and network design issues in the presence of self-similar traffic.

Acknowledgments

I would like to thank Nachum Shacham, Phillip Porras, José Meseguer, Patrick Lincoln and Chris Dodd, all with SRI International, and Caveh Jalali with Sun Microsystems for the helpful discussions and significant technical advice that contributed to the making of this paper.

References

[1] D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan M. Smith. Active bridging. *Proceedings of the ACM SIGCOMM'97 Conference*, Cannes, France, September 1997.

[2] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, April 1981.

[3] R. M. Fujimoto. Optimistic approaches to parallel discrete event driven simulation. *Trans. Society for Computer Simulation*, June 1990.

[4] U. Manber J. Hartman, L. Peterson, and T. Proebsting. Liquid software: A new paradigm for networked systems. Technical Report 96-11, University of Arizona, 1996.

[5] D. Jefferson. Virtual time. *ACM Trans. Programming Languages and Systems*, July 1985.

[6] L. Lamport. Time, clocks, and the ordering of events in distributed systems. *Communications of the ACM*, July 1978.

[7] U. Legedza, D. J. Wetherall, and J. V. Guttag. Improving the performance of distributed applications using active networks. *Submitted to IEEE INFOCOM'98*, 1998.

[8] L. Ricciulli. A technique for the distributed simulation of parallel computers. In *MASCOTS '95*, January 1995.

[9] L. Ricciulli, J. Meseguer, and P. Lincoln. Distributed simulation of parallel executions. In *29th Annual Simulation Symposium*, pages 15–24, 1996.

[10] H. Schwetman. Csim: A c-based, process-oriented simulation language. Technical report, MCC, 1989.

[11] Jonathan Smith, David Farber, Carl A. Gunter, Scott Nettle, Mark Segal, William D. Sincoskie, David Feldmeier, and Scott Alexander. Switchware: Towards a 21st century network infrastructure. *http://www.cis.upenn.edu/ switchware/papers/sware.ps*, 1997.

[12] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. Ants: A toolkit for building and dynamically deploying network protocols. *Submitted to IEEES OPENARCH'98*, 1998.

[13] Y. Yemini and S. da Silva. Towards programmable networks. *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October 1996.