

Maya: a Multi-Paradigm Network Modeling Framework for Emulating Distributed Applications*

Junlan Zhou, Zhengrong Ji, Mineo Takai, Rajive Bagrodia
University of California, Los Angeles
Parallel Computing Laboratory
3809 Boelter Hall, Los Angeles, CA 90095-1596
{zjl, jizr, mineo, rajive}@cs.ucla.edu

Abstract

This paper presents Maya, a multi-paradigm, scalable and extensible network modeling framework for emulating distributed applications. A novel three-tier architecture is proposed to integrate three disparate modeling paradigms, namely, discrete event models, analytical models and physical network interfaces into one unified framework of Maya. As the first effort to integrate all three paradigms into one framework, this paper discusses the implementations of Maya using Qualnet, fluid flow based TCP model and physical network interface. It addresses the performance issues involved in attaining the real time constraints imposed by distributed applications and demonstrates the effectiveness of using analytical models in Maya. Furthermore, it identifies the negative impact on real time performance through the computation intensive ordinary differential equation (ODE) solver in the fluid flow model. A new approach to interleaved executions of the fluid flow model is proposed to hide ODE solver turnaround time. As a result, the percentage of packets missing their deadlines has been reduced from more than 6% to less than 0.2%.

1 Introduction

Fast networks and low-cost processors are enabling the advent of a new wave of distributed applications ranging from client-server web services to highly distributed peer-to-peer computing environments. Unlike traditional standalone applications, developing distributed applications is remarkably harder due to the fundamental characteristics of the underlying network, i.e. failure, heterogeneity, errors and so on. It is essential to be aware of the dynamics of the underlying network when developing a distributed applica-

tion and validate the major design decisions prior to the actual deployment. However, a real test bed is generally not available, especially as the number of inter-connected computers in the network on which the distributed applications reside can be arbitrarily large.

The combination of real application codes and models for the underlying networks offers a promising solution to assess the impact of network dynamics on the performance of distributed applications. The network models mimic the behaviors of the underlying networks and provide programmers with a synthetic configurable environment to conduct repeatable experiments with the distributed applications. There are numerous challenges in building such a comprehensive environment that satisfies disparate needs for distributed applications. First, it must resemble the target network as closely as possible. Second, it needs to run in real time as the validity of the results depends on its ability to accommodate real time constraints. Third, it must be scalable in order to enable study and evaluation of distributed applications under large networks with high throughput of both foreground and background traffic. Finally, it must be transparent to applications and be able to support evaluation of precompiled applications.

Many previous efforts have addressed similar objectives. Studies in [25, 24, 8] simulate links among nodes in the target network with regular computers and higher-speed network devices while real codes are running at high layers. A module is introduced between the network layer and device driver in each computer, to introduce effects such as propagation delay, packet loss or bandwidth throttling, to emulate dynamic network conditions. As this approach requires the same number of networked machines or devices as in the target network, the amount of hardware required by these studies grows proportionally with the size of the target network, with its scalability limited. The works presented in [9], [17], [14], [20, 6, 21] and [11] utilize network simulators as their cores and develop an interface in

*This work is supported by the US Department of Defense/DARPA under Contract N66001-00-1-8937 "Maya: Next Generation Performance Prediction Tools for Global Networks".

the simulators for real world traffic to traverse the simulated network and interact with the simulated traffic. Using a parallel discrete event simulation (PDES) kernel, the studies in [20, 6, 21] target construction of wide area network environments. However, the runtime performance of discrete event models is highly dependent on network size and amount of traffic. This inherently limits scalability since the PDES techniques can speed up simulation executions only by the number of processors used.

Therefore, out of the many challenges in building an emulation environment for distributed applications, scalability has become a major concern as the number of distributed applications that target the Internet has grown significantly. Many research efforts have been dedicated to analysis of not only the statistical behavior of traffic generated by Internet-scale systems or applications [10, 7], but also the expected transient behavior of networks supporting such traffic flows [16]. The analytical models emerging from these efforts predict the network dynamics quite accurately and scale well to a large number of flows, and thus are of direct benefit in developing a framework for emulating distributed applications. Note, however, that the application of analytical models is typically limited for specific network and traffic types. Therefore, the framework needs to be flexible within which different types of analytical models can be integrated to provide a wide range of networking scenarios needed for evaluation of distributed applications.

This paper presents Maya, a multi-paradigm network modeling framework for testing and evaluating distributed applications. Maya provides an environment where many instances of three distinct paradigms (i.e., discrete event models, analytical models, and interfaces to the physical networks) can be integrated seamlessly and interact with each other transparently to distributed applications. While past studies have integrated two of these three paradigms (discrete event and analytical models[23], or discrete event models and physical network interfaces), to the best of our knowledge, this is the first effort to integrate all three paradigms in a single framework. In fact, while analytical models may run faster than the corresponding discrete event models, they have unique problems when integrated with a physical network interface due to its turnaround time to advance the virtual time. This paper discusses the issues involved in integrations of these different paradigms, and presents techniques to improve the time synchronization among instances of different paradigms. Experimental results show that the proposed optimizations improve the accuracy of time synchronization significantly, having Maya deliver data packets to the distributed applications in a timely manner even if the underlying network is highly heterogeneous and scales substantially.

The remainder of this paper is organized as follows. Section 2 gives an overview of the Maya framework and de-

scribes the approach to integrate heterogeneous paradigms into this unified framework. Section 3 analyzes Maya’s performance and presents an approach to the interleaved execution of the fluid flow model that significantly improves the performance of Maya. Section 4 concludes and discusses future works.

2 Maya

Our design philosophy for Maya is to have the framework open to existing and future network modeling and simulation tools. First, we present the conceptual design or architecture that captures the essential interoperations among different components in the Maya framework.

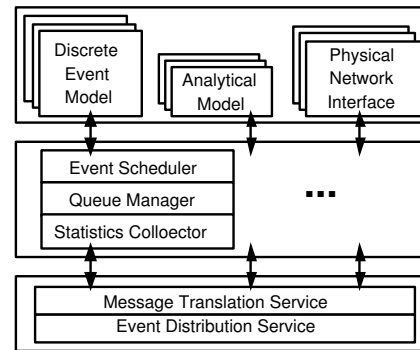


Figure 1. Overview of Maya Architecture

2.1 Architectural overview

The architecture of Maya is depicted in Figure 1. It uses a three-tier approach where components at the upper tier utilize functionalities provided at the lower tier(s). When developing the components in each tier, we try to leverage from existing tools or libraries if possible. Organization and functionality of each tier are described as follows:

Top tier: The top tier of Maya provides the building blocks for constructing underlying networks for emulating distributed applications. Three types of components are incorporated in this tier, namely discrete event model, analytical model and the physical network interface, corresponding to the three modeling paradigms. In contrast to the other components, the physical network interface captures and creates real IP packets, through which physical networks can be attached to simulated networks. Multiple instances of these components can be juxtaposed; each corresponds to a portion of the underlying networks on which distributed applications can be executed. The combination of modeling paradigms offer users the flexibility to tradeoff fidelity with scalability of network modeling and abstract portions of the networks with analytical models. These components exchange global events via services provided at lower tiers to model packets flowing among different portions of the

networks. Each generated event is tagged with a global timestamp and carries a packet to be forwarded.

Middle tier: The middle tier enables interoperability among the heterogeneous components in the top tier. It handles incoming and outgoing global events, ensuring that top tier components exchange meaningful information despite their semantic differences. When needed, it is also responsible for scheduling global events and synchronizations with a wall clock time, as required by emulation of distributed application. The components providing the above functionalities include event schedulers, statistics collectors, and queue managers.

Bottom tier: The bottom tier provides basic services such as time synchronization and global event distribution among the disjoint components in upper tiers. There have been many research studies on event distribution and time management among processes in distributed environments, such as the High Level Architecture proposed by the Department of Defense(DoD)[1], PARSEC[19], RTI-KIT[12] etc, all of which conceptually can be leveraged by Maya. We choose RTI-KIT for its open design and good performance in high-speed LAN environments as well as SMP machines. A dynamic simulation backplane [18] is utilized to allow components in the top tier to exchange protocol information and dynamically convert packets between native formats and a common format.

2.2 Implementations

There are many simulation and modeling tools falling into the paradigms of either discrete event model (such as NS-2[4], Glomosim[2], Qualnet[5], Opnet[3], PDNS[22] etc.) or analytical model (such as fluid flow based TCP model[16]). We have included one from each group to represent the discrete-event simulation and analytical models respectively: Qualnet and the fluid flow based TCP model in the initial implementation of Maya. We have also implemented a Physical Network Interface in Linux and integrated it into MAYA. The next section describes how each of them has been integrated into the framework.

2.2.1 Integration of discrete event models

Figure 2 shows the operations of Qualnet and how it interacts or exchanges events with other components in the top tier of Maya via the services provided by the components in the lower tier, namely, the queue manager and event scheduler.

The queue manager exports and imports global events, employing the utilities available in the backplane and RTI-KIT at the bottom tier. It converts a packet between native and common format and fills up the extra data fields of a native format packet in the global event appropriately, before delivering the packet to Qualnet. The event scheduler

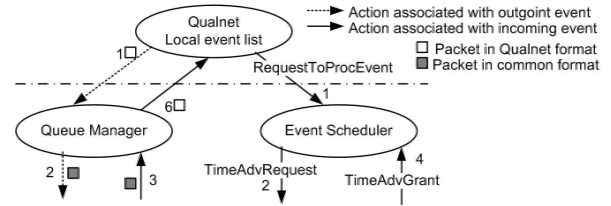


Figure 2. Integration of Qualnet

ensures that Qualnet processes all events in strict timestamp order. Before scheduling a local event, Qualnet consults the event scheduler (see Figure 2), which in turn makes a time advancement request to the RTI-KIT. Inbound global events with timestamps smaller than the requested time are passed to the queue manager by the RTI-KIT. After all the eligible global events are delivered, the RTI-KIT grants the request and guarantees that no more events earlier than the granted time will arrive later. The event scheduler then informs the queue manager to forward all received packets to Qualnet, allowing Qualnet to schedule the outstanding event with the smallest timestamp for processing.

One challenge to be addressed by the event scheduler is that Qualnet must tightly synchronize with real time during its execution. Suppose another top-tier component is tightly synchronized with real time. As long as the event scheduler ensures Qualnet is time constrained by that component, Qualnet will be transitively synchronized with real time. In this way, direct wall clock time synchronization of Qualnet is avoided, resulting in better efficiency. A typical candidate for bootstrapping the real time synchronization is the physical network interface, since it needs to timestamp the real IP packets with the wall clock anyway.

2.2.2 Integration of analytical models

The fluid flow based TCP model[16] is the representative analytical model integrated into the initial implementation of Maya. As its name suggests, the fluid flow model treats TCP traffics as fluids. It derives a set of ordinary differential equations (ODE) to model the rates of traffic changes and queuing process at routers. It excels in modeling wired backbone networks with high volumes of traffic. Unlike discrete event model and physical network interface, there are no notions of “packet” or “event” in the fluid flow model. A queue manager, statistics collector, and event scheduler have been developed in the middle tier to enable the fluid flow model to process global events and interact with its peers.

Figure 3 shows how packets originated from a physical network interface or discrete event model traverse a wired backbone network represented by the fluid flow model. Information (such as packet size etc.) of the packets entering

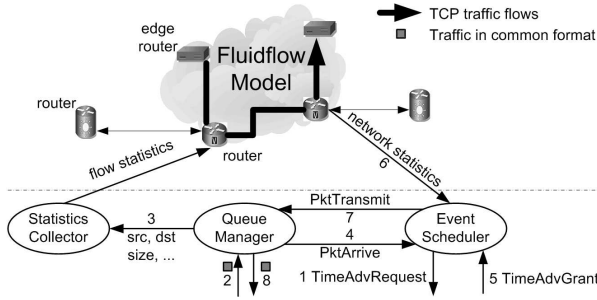


Figure 3. Integration of the Fluid Flow Model

the wired network will be extracted by the queue manager and used by the statistics collector to compute statistics of the traffic flows, which is the input to the ODE solver of the fluid flow model. Since the amount of traffic entering the wired network varies over time, the statistics of traffic flows are averaged within a certain time interval (called time averaging interval) and fed into the ODE solver periodically. As an output, the ODE solver produces network statistics, such as queue length, TCP window size and so on, for the next time averaging interval. The results will be used to estimate the delay of packets traveling across the wired network during that time interval.

The queue manager handles importing, exporting and queuing of global events. It extracts information such as source IP address, destination IP address, packet size, etc., and feeds it to the statistics collector. The event scheduler maintains a local event list comprising two types of events: those related to packet arrivals or transmissions and those responsible for periodically invoking the ODE solver of the fluid flow model. An event of “PktArrive” is triggered via reception of a global event carrying a packet. To process the “PktArrive” event, the event scheduler estimates the delay of the packet indicated by the “PktArrive” event to traverse the wired network based on the network statistics produced by the ODE solver, and schedules the transmission of the packet accordingly, as if the packet has traveled across the wired network. The event scheduler will not be able to process an event until its time advances to the event’s timestamp, via which the fluid flow model is synchronized with its peers.

2.2.3 Integration of physical network interfaces

Contemporary large-scale networks are usually comprised of multiple LANs or even autonomous systems (AS) interconnected via gateways. When emulating distributed applications, it is logical and convenient to partition the underlying networks into LANs or autonomous systems, each of which is either physical or modeled by an instance of discrete event or analytical model. The physical network interface should reside at the gateways of the physical net-

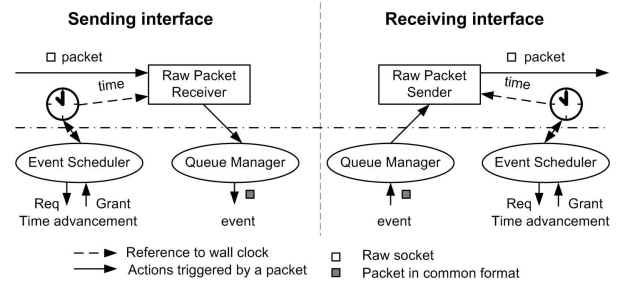


Figure 4. Integration of Physical Network Interfaces

works so that it can hookup the physical networks with the modeled portions of the underlying network when emulating distributed applications.

As shown in Figure 4, the physical network interface has two modules, a sending interface and a receiving interface. The raw packet receiver of the sending interface captures the real IP packets originated from distributed applications running on machines in the physical networks using the Berkeley Packet Filter (BPF)[15]. The timestamp of a captured packet is set to the wall clock time when captured by the raw packet receiver. The timestamped packet is sent to the queue manager for immediate event delivery. The event scheduler of the sending interface tries to synchronize the sending interface with the wall clock time by requesting its virtual time to be equal to current wall clock time. As the real packet can come at any time, the raw packet receiver runs as a separate thread polling the BPF kernel for newly captured packets.

The event scheduler of the receiving interface keeps requesting for time advancement. Its queue manager will import the events received from the lower tier during the time advancement requesting process. When the time advancement request is granted, the event scheduler will instruct the queue manager to convert these events into timestamped real packets and ship them to the raw packet sender. The raw packet sender will then send the packets at the exact time specified by the timestamp. The timer for the Linux operating system running on the Intel PC has a granularity of 10ms, which may not be accurate enough. Thus, the raw packet sender is implemented as a separate thread that polls the system clock between times when it should send packets, which improves accuracy down to microseconds.

A look-ahead[13] is needed at the sending interface in order for the discrete event model and the analytical model to run faster than real time. As Maya aims to emulate distributed applications running on top of large-scale worldwide networks, the queuing and propagation delays between autonomous systems can range from a few milliseconds to tens of milliseconds, which makes them a good choice for look-ahead. Even longer delays may be seen for satellite links. As a matter of fact, similar look-ahead val-

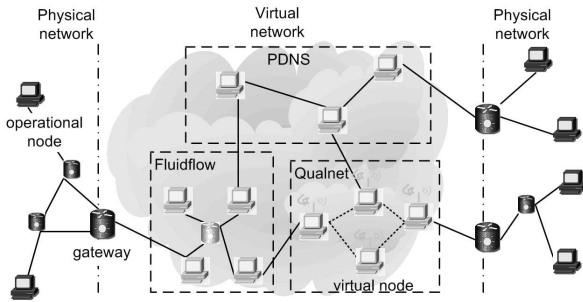


Figure 5. An Example of Configuring Underlying Network

ues can be determined for all the components in the top tier. However, an exception can be made for the receiving interface module of the physical network interface, because it does not time-constrain[13] the sending interface module, and other components in the top tier.

2.3 Configuring underlying network for emulating distributed application

Figure 5 illustrates how a target network is partitioned and mapped to top-tier components of Maya for emulating distributed applications. In the example, the network is partitioned into virtual subnets and physical subnets. A virtual subnet is modeled by a discrete event simulator (such as Qualnet or PDNS in this example) or an analytical model. The choice of mapping is based on a combination of factors that may include applicability, desired level of fidelity, and execution speed. A physical subnet comprises regular computers (referred to as operational nodes) inter-connected by network devices, on which the distributed applications reside. The physical network interfaces, which execute on the gateways of physical subnets, allow live traffic generated by the distributed applications to traverse virtual subnets and be subjected to the network conditions present there.

3 Performance Analysis and Optimizations

In this section, we describe a set of experiments conducted to evaluate the performance of Maya. First, we demonstrate the performance advantage of Maya over the pure discrete event simulator based framework. We then evaluate the ability of Maya to conform to real time constraints so as to emulate distributed applications. Finally, we propose an approach to the interleaved execution of fluid flow models and show that it significantly improves the performance of Maya.

3.1 Improving Execution Speed with Fluid Flow Model

Compliance with real time constraints is one of the key requirements for Maya to emulate distributed applications.

While it is intuitive that performance of a discrete event network simulator degrades as the size of the target network grows, its performance also degrades with capacity of heavily loaded links. On the other hand, a fluid flow based analytical model should be able to model large number of flows with constant execution time.

A common scenario depicted in Figure 6(a) is created to compare the performance of these two modeling methods quantitatively. Nodes S and D are edge routers and nodes X and Y are core routers. Each solid line in the graph represents a full-duplex wired link with $50ms$ propagation delay. There are 2000 identical TCP flows from S to D sharing the network path $S - X - Y - D$. Both Qualnet and fluid flow model simulate the above scenario for 500 seconds duration and the ratios of actual execution time over the simulation time (500 seconds in this case) are shown in Figure 7(a). As the link capacity increases from $10Mbps$ up to $70Mbps$, the execution time of Qualnet increases almost linearly while the execution time of the fluid flow model remains rather constant and is much smaller than that of Qualnet. It indicates that even if the network topology is simple, the growing number of background traffic flows will prevent discrete event simulators from keeping up with real time. Thus, it justifies the need to integrate fluid flow model into Maya for emulating distributed applications in large-scale heterogeneous networks.

Next we augment the previous scenario with an extra subnet and extra applications, to evaluate how fast Maya can run. As shown in Figure 6(b), there is an $802.11b$ wireless subnet of 60 nodes connected to the wired network via a gateway in this scenario. The nodes are uniformly distributed in a 300 square-meter terrain. There is moderate traffic among the nodes in the wireless subnet. There are 2000 background TCP flows between S and D in the wired network. At the same time, one wireless node S' has a synthetic application sending CBR traffic using UDP protocol to destination D . These UDP packets are of the same size (1000 bytes) and evenly distributed in the time scale. Given the Maya framework, a natural way to configure the underlying network is to partition it according to the wireless subnet and the wired subnet. The wireless subnet is modeled by a Qualnet instance while the wired subnet is modeled by a fluid flow model instance. As no physical network is involved in this experiment, Maya can run as fast as possible without synchronizing with wall clock time, which is exactly what we want in order to compare it with the execution speed of Qualnet. The experiment has been conducted on a DELL SMP machine with eight $550MHz$ Intel CPUs. Default process scheduling policy of the system is used. The execution time of Maya to model the network for 500 seconds duration is measured and the results are shown in Figure 7(b).

As packets flow between the wireless subnet and the

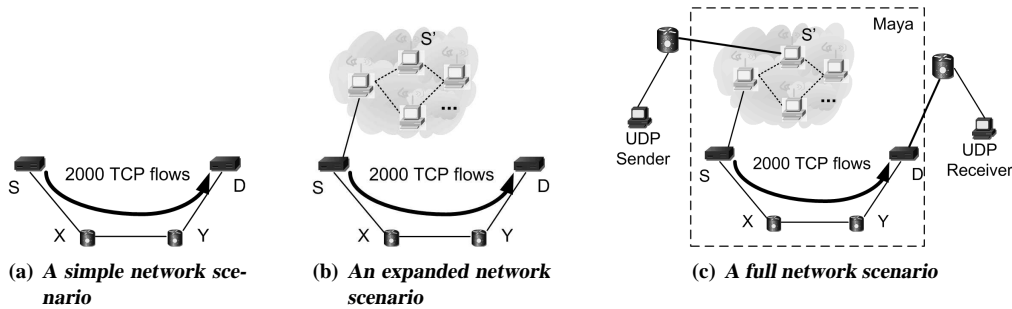
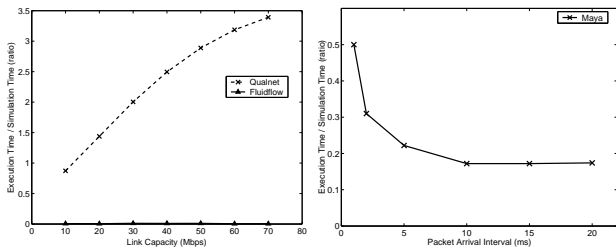


Figure 6. Network Scenarios



(a) Execution Speed of Qualnet v.s. Fluid Flow (b) Execution Speed of Maya

Figure 7. Execution Speeds

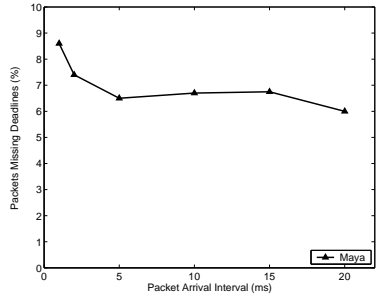
wired subnet belong to UDP traffic, we are able to vary the packet arrival interval at the source of the traffic and determine how Maya will perform under various traffic volumes across the wireless subnet and the wired subnet. As shown in Figure 7(b), when such traffic volume is small, the execution time of Maya is less than one fifth of the simulation time. The execution time reaches its lower bound when the packet arrival interval is about $10ms$ because time synchronization is dominant when the traffic volume is relatively low. The execution time will increase as the traffic volume further increases. However, Maya performs quite well as it only takes about 250 seconds to run the 500-second long scenario even when packet arrival interval is as small as $1ms$.

3.2 Time synchronization at finer granularity

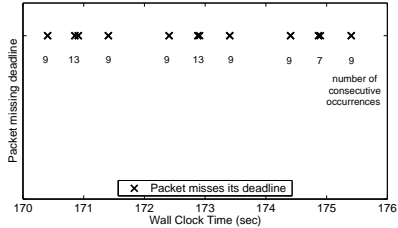
In previous experiments, we evaluate whether Maya can keep up with real time in the long run using synthetic application traffic and we measure how fast the framework can execute relative to the time duration it models. But it is important to see how Maya performs at finer time granularity as well, as the objective is to use the framework to emulate distributed applications. Experiments are conducted to model a so-called full network scenario shown in Figure 6(c). In this scenario, two physical subnets are added into the previous scenario depicted in Figure 6(b). Two real applications are running on two machines, one in each physi-

cal subnet. For easy measurement and analysis, we implement a UDP sender application capable of sending fixed-sized UDP packets at a fixed time interval as accurate as $1ms$. A UDP receiver application is used to consume these packets. The previously mentioned DELL SMP machine is configured as the gateway between the two physical subnets, with Maya executed entirely on this SMP machine. While the simulated network portions are modeled in the same way as that of the previous scenario, two physical network interface instances are added to hookup with the physical subnets. This configuration is used to emulate the above UDP applications.

The percentage of packets missing their deadlines has been used as the performance metric of this experiment. In this context, a packet moving from simulated subnet to physical subnet is considered missing its deadline if the wall clock has bypassed its timestamp. Due to tight time synchronization, Maya might not always run ahead of real time. When that happens, packets going to physical subnets may miss their deadlines. As shown in Figure 8(a), the percentage of packets missing their deadlines is smaller than 10% for packet arrival intervals ranging from 1 millisecond to 20 milliseconds. This ratio remains rather constant within the range of the packet arrival interval shown in the figure, which motivates us to take a chronological view of the packets going to the UDP receiver. Figure 8(b) visualizes a small portion of the trace on finding a packet that misses its deadline. The snapshot is extracted from the trace of the experiment in which the packet arrival interval is 2 milliseconds. The graph reveals that late packets occur in bursts (each cross in Figure 8(b) actually denotes a sequence of consecutively late packets in the trace) and appear periodically. A number below a cross tells the number of packets missing their deadlines in the single burst. In fact, similar behavior can be observed on traces of all experiments conducted. This behavior will affect the accuracy of emulating distributed applications and is not desirable. In the next subsection, we will discuss the cause of this problem and present a solution.



(a) **Percentage of Packets Missing Deadlines**



(b) **A Chronological Snapshot**

Figure 8. Effects of Realtime Constraints on Maya

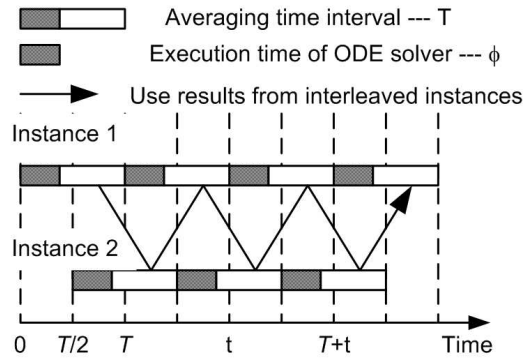
3.3 Interleaved executions of fluid flow model

As explained earlier in Section 2, the fluid flow model computes the instantaneous network statistics such as queuing delay and packet drop rate for the next time averaging interval at the beginning of such an interval. If incoming packets arrive at the fluid flow model before the analytical results for that interval are ready, the processing of these packets must be delayed. Time profiling on the fluid flow model indicates that this delay could be very costly because it takes up to 100ms for the ODE solver of the fluid flow model to compute the results for the next 1 second interval on the DELL SMP machine. Furthermore, running as a single process, the fluid flow model will not be responsive to time synchronization events of the RTI-KIT until the ODE solver completes the computation. This in turn hinders the time advancements of other top tier components as well. Hence there are packets missing their real time deadlines periodically due to the periodic execution of ODE solver.

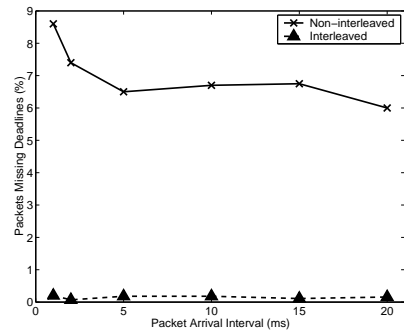
To tackle this problem, we propose an approach to interleaved executions of the fluid flow model. Assume the time averaging interval is T . The ODE solver takes time ϕ to execute each time it is invoked, where $\phi < T/2$. Normally there is only one fluid flow model instance modeling a portion of the underlying network. If the ODE solver is invoked at time t , the output is meant for use in time interval $[t, t + T]$. However, the result is only ready after time $t + \phi$. In the new approach depicted in Figure 9(a), a second instance of the fluid flow model modeling the same portion of the network is interleaved with the original instance of

the fluid flow model. The two intertwined instances are essentially the same except that the computations performed by the second instance are shifted by $T/2$ from the first instance in the time scale. Thus, results produced by the first instance will be ready for use at the same time that the second instance is computing results for its next interval, and vice versa. In this interleaved execution approach, invocations to the ODE solver will be executed in a worker thread, while the main thread of the fluid flow model can process time synchronization events and schedule packet transmissions using results obtained earlier. This method should eliminate the penalties of invoking the ODE solver except for the first time when there are no previous results ready for use, which should be tolerable in most cases.

The experiments described in previous subsection are repeated with the optimization presented above. Figure 9(b) shows the percentage of packets missing their deadlines in the experiments with the full network scenario, both with and without the new approach to interleave the execution of fluid flow models in Maya. It is encouraging to see that the average ratio of packets missing their deadlines is reduced very significantly to less than 0.2%, with the interleaved execution of fluid flow model, which proves the effectiveness of the optimization.



(a) **Interleaved Execution of Fluid Flow Model**



(b) **Its Effect**

Figure 9. Interleaved Execution and its Effect

4 Conclusions

In this paper, we have presented Maya, a scalable and extensible network modeling framework for emulating distributed applications. It spans three distinct modeling paradigms: discrete event models, analytical models and physical network interfaces. In Maya's three-tier architecture, components at the upper tier utilize the functionalities offered by those in the lower tiers. Multiple instances of modeling paradigms reside in the top tier. Implementation issues of integrating heterogeneous modeling paradigms into this unified framework have been discussed with specific examples. A series of experiments have been conducted to evaluate Maya's ability to conform to real time constraints. The results show that the fluid flow model executes much faster than Qualnet and its execution time remains constant as network capacities increase. As a result, Maya can keep up with real time in the long run. However, periodical invocations of the computation intensive ODE solver in the fluid flow model causes packets to miss their real time deadlines repeatedly. An approach to interleaved executions of the fluid flow model has been proposed to mask the turnaround time of the single instance fluid flow model. Experiment results show that the new approach significantly reduces the ratio of packets missing deadlines.

Maya has been successfully used in our lab for emulating distributed applications like NetMeeting and multimedia streaming applications. In the future, we will conduct experiments emulating more sophisticated distributed applications on larger-scale networks. We would also like to evaluate the performance of distributed execution of Maya over the NOW (network of workstations) infrastructure, especially when wall clock synchronization between multiple machines becomes necessary. A third dimension of our future works will be integrating application traffic models such as web traffic models[7, 10] into the framework so as to generate more accurate background traffic when emulating distributed applications.

References

- [1] *High Level Architecture*. <http://www.hla.mil>.
- [2] <http://pcl.cs.ucla.edu/projects/gloimosim/>.
- [3] <http://www.opnet.com>.
- [4] *Ns-2 user manual*. www.isi.edu/nsnam.
- [5] *Qualnet user manual*. www.scalable-networks.com.
- [6] R. Bradford, R. Simmonds, and B. Unger. A parallel discrete event ip network emulator. In *Proceedings of the Ninth International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'00)*, San Francisco, 2000.
- [7] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic evidence and possible causes. *IEEE/ACM Transactions on Networking*, 1997.
- [8] H. Daniel and R. Kurt. *A Dynamic Network Scenario Emulation Tool*. Institute of Parallel and Distributed High-Performance Systems (IPVR), University of Stuttgart.
- [9] N. Davies, G. S. Blair, K. Cheverst, and A. Friday. A network emulator to support the development of adaptive applications. In *Proceedings of USENIX Symposium on Mobile and Location Independent Computing*, 1995.
- [10] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223.
- [11] K. Fall. Network emulation in the vint/ns simulator. In *the Fourth IEEE Symposium on Computers and Communications (ISCC'99)*, July 1999.
- [12] R. Fujimoto and et al. *FDK User Guide*, February 2001.
- [13] R. M. Fujimoto. Time management in the high level architecture. *Simulation*, 71(6):388–400, December 1998.
- [14] Q. Ke, D. Maltz, and D. B. Emulation of multi-hop wireless ad hoc networks. In *Proceedings of 7th International Workshop on Mobile Multimedia Communications (MoMuC'00)*, Oct 2000.
- [15] S. McCanne and V. Jacobson. The bsd packet filter: a new architecture for user-level packet capture. In *Winter USENIX Technical Conference*, pages 25–29, San Diego, CA, January 1993.
- [16] V. Misra, W. Gong, and D. Towsley. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *Proceedings of ACM/SIGCOMM*, 2000.
- [17] B. D. Noble and et al. Trace-based mobile network emulation. In *Proceedings of SIGCOMM '97*, September 1997.
- [18] G. Riley, M. Ammar, R. Fujimoto, K. Perumalla, and D. Xu. Distributed network simulations using the dynamic simulation backplane. In *Proceedings of the International Conference on Distributed Computing Systems*, April 2001.
- [19] G. F. Riley, R. M. Fujimoto, and M. H. Ammar. A generic framework for parallelization of network simulations. In *MASCOTS*, 1999.
- [20] R. Simmonds, R. Bradford, and B. Unger. Applying parallel discrete event simulation to network emulation. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS 2000)*, Bologna, Italy, May 2000.
- [21] R. Simmonds and B. Unger. Towards scalable network emulation. In *SPIE ITCOM 2001*, Denver, August 2001.
- [22] H. Wu, R. Fujimoto, and G. Riley. Experiences parallelizing a commercial network simulator. In *the Winter Simulation Conference*, December 2001.
- [23] T. Yung, M. Takai, and R. Bagrodia. Mixed fluid flow and packet level simulation models for large scale networks. In *Proceedings of SPIE*, volume 4523, August 2001.
- [24] P. Zheng and L. M. Ni. Empower: A scalable framework for network emulation. In *Proceedings of the 2002 International Conference on Parallel Processing (ICPP '02)*, Vancouver, Canada, 2002.
- [25] P. Zheng and L. M. Ni. Emwin: Emulating a mobile wireless network using a wired network. In *Proceedings of The Fifth International Workshop on Wireless Mobile Multimedia (WoWMoM 2002)*, Atlanta, Georgia, U.S.A., 2002.