12

Parallel and Distributed Immersive Real-Time Simulation of Large-Scale Networks

Jason Liu
Florida International University
United States

1. Introduction

Network researchers need to embrace the challenge of designing the next-generation high-performance networking and software infrastructures that address the growing demand of distributed applications. These applications, particularly those potential "game changers" or "killer apps", such as voice-over-IP (VoIP) and peer-to-peer (P2P) applications surfaced in recent years, will significantly influence the way people conduct business and go about their daily lives. These distributed applications also include platforms that facilitate large-scale scientific experimentation through remote control and visualization. Many large-scale science applications—such as those in the field of astronomy, astrophysics, climate and environmental science, material science, particle physics, and social science—depend on the availability of high-performance facilities and advanced experimental instruments. Extreme networking capabilities together with effective high-end middleware infrastructures are of great importance to interconnecting these applications, computing resources and experimental facilities. "When all you have is a hammer, everything looks like a nail." The success of advancing critical technologies, to a large extent, depends on the available tools that can help effectively prototype, test, and analyze new designs and new ideas. Traditionally, network research has relied on a variety of tools. Physical network testbeds, such as WAIL (Barford and Landweber, 2003) and PlanetLab (Peterson et al., 2002), provide physical network connectivity; these testbeds are designed specifically for studying network protocols and services under real network conditions. However, the network condition of these testbeds is by and large constrained by the physical setup of the system and therefore inflexible for network researchers to explore a wide spectrum of the design space.

To allow more flexibility, some of these testbeds, such as EmuLab (White et al., 2002) and VINI (Bavier et al., 2006), also offer emulation capabilities by modulating network traffic according to configuration and traffic condition of the target network. Physical and emulation testbeds currently are the mainstream for experimental networking research, primarily due to their capability of achieving desirable realism and accuracy. These testbeds, however, are costly to build. Due to limited resources available, conducting prolonged large-scale experiments on these platforms is difficult. Another solution is to use analytical models. Although analytical models are capable of bringing us important insight to the system design, dealing with a system as complex as the global network requires significantly simplified assumptions to be made to keep the models tractable. These simplified assumptions often

exclude implementation details, which are often crucial to the validity of the system design.
Simulation and emulation play an important role in network design and evaluation. While both refer to the technique of mimicking network operations in software, one major distinction is that simulation is purely virtual, whereas emulation focuses on interactions with real applications. A network simulation consists of software implementation of network protocols and various network entities, such as routers and links. Network operations (e.g., packet forwarding) are merely logical operations. As a result, the simulation time advancement bears no direct relationship to the wall-clock time. Emulation, on the other hand, focuses on interactions with real applications, such as distributed network services and distributed database systems. These real applications generate traffic; an emulator provides traffic shaping functions by adding appropriate packet delays and sometimes dropping packets. Emulation delivers more realism as it interacts with the physical entities. Comparatively, simulation is effective at capturing high-level design issues, answering what-if questions, and therefore can help us understand complex system behaviors, such as multi-scale interactions, self-organizing characteristics, and emergent phenomena. Unfortunately, simulation fairs poorly in many aspects, including notably the absence of operational realism. Further, simulation model development is both labor-intensive and error-prone; reproducing realistic network topology, representative traffic, and diverse operational conditions in simulation is known to be a substantial undertaking (Floyd and Paxson, 2001).
Real-time simulation combines the advantages of both simulation and emulation: it can run simulation and simultaneously interact with the physical world. Real-time network simulation, sometimes called immersive network simulation, can be defined as the technique of simulating computer networks and communication systems in real time so that the simulated network can interact with real implementations of network protocols, network services, and distributed applications. The word "immersive" suggests that the virtual network behavior should not be distinguishable from a physical network for conducting network traffic. That is, simulation should capture important characteristics of the target network and support seamless interactions with the real applications. Real-time network simulation is based on simulation, and therefore is fast in execution and flexible at answering what-if questions. It allows high-level mathematical models (such as stochastic network traffic models) to be incorporated into the system with relative ease. The system interacts with real applications and real network traffic. Not only does it allow us to study the impact of real application traffic on the virtual network, but also supports studying the behavior of real applications under diverse simulated network conditions.
The challenge is to keep it in real time. Since real applications operate in real time, real-time network simulation must meet real-time requirements. Especially, the performance of a large-scale network simulation must be able to keep up with the wall-clock time and allow real-time interactions with potentially a lot of real applications. A real-time simulator must also be able to characterize the behavior of a network, potentially with millions of network entities and with realistic traffic load at commensurate scale—all in real time. To speed up simulation, on the one hand, we need to apply parallel and distributed discrete-event simulation techniques to harness the computing resources of parallel computers so as to physically increase the event-processing power; on the other hand, we need to resort to multi-resolution modeling techniques using models at high-level of abstraction to reduce the computational demand. We also need to create a scalable emulation infrastructure,

through which real applications can interact with the simulated network and sustain high-level emulation traffic intensity. In this chapter, we review the techniques that allow real-time simulation to model large-scale networks and interact with many real applications under the real-time constraint. We discuss advanced modeling and simulation techniques supporting real-time execution. We describe the emulation infrastructure and machine virtualization techniques supporting the network immersion of a large number of real applications. Through case studies, we show the potentials of real-time simulation in various areas of network science.

2. Background

2.1 Existing Network Testbeds

We classify available network testbeds into physical, emulation, and simulation testbeds. We can further divide physical testbeds into production testbeds and research testbeds (Anderson et al., 2005). Production testbeds, such as CAIRN and Internet2, support network experiments directly on the network itself and thus with live traffic; however, they are very restrictive allowing only certain types of experiments that do not disrupt normal network operations. Comparatively, research testbeds, such as WAIL and PlanetLab, provide far better flexibility. WAIL (Barford and Landweber, 2003) is a research testbed consisting of a large set of commercial networking components (including router, switches, and end hosts) connected to form an experimental network capable of representing typical end-to-end configurations found on the Internet. PlanetLab (Peterson et al., 2002) is a well-known research facility consisting of machines distributed across the Internet and shared by researchers conducting experiments. Most research testbeds, however, can only provide an iconic view of the Internet at large. Also, the underlying facility is typically overloaded due to heavy use, which potentially affects their availability as well as accuracy (Spring et al., 2006).

Many research testbeds are based on emulation. Network emulation adds packet delays and possibly drops packets when conducting traffic between real applications. Examples of emulation testbeds include Ahn et al. (1995); Carson and Santay (2003); Herrscher and Rothermel (2002); Zheng and Ni (2003) and Huang et al. (1999). The traffic modulation function can be implemented at the sender or receiver side, or both. For example, in dummynet (Rizzo, 1997), each virtual network link is represented as a queue with specific bandwidth and delay constraints; packets are intercepted at the protocol stack of the sender and pushed through a finite queue to simulate the time it takes to forward the packet.

Emulation testbeds can be built on a variety of computing infrastructures. For example, ModelNet (Vahdat et al., 2002) extends dummynet, where a large number of network applications can run unmodified on a set of edge nodes and communicate via a virtual network emulated on parallel computers at the core. EmuLab (White et al., 2002) is an experimentation facility consisting of a compute cluster integrated and coordinated to present a diverse virtual network environment. DETER (Benzel et al., 2006) extends EmuLab to support research and development of cyber security applications. Some of the emulation testbeds are built for distributed environments, such as X-Bone (Touch, 2000), VIOLIN (Jiang and Xu, 2004), VNET (Sundararaj and Dinda, 2004), and VINI (Bavier et al., 2006). Other emulation testbeds may require special programmable devices. For example, the Open Network Laboratory (DeHart et al., 2006) uses embedded processors and configures them to represent realistic network settings for experimentation and observation. ORBIT

(Raychaudhuri et al., 2005) is an open large-scale wireless network emulation testbed that supports experimental studies using an array of real wireless devices. The CMU Wireless Emulator (Judd and Steenkiste, 2004) is a wireless network testbed based on a large Field-Programmable Gate Array (FPGA) that can modify wireless signals sent by real wireless devices according to signal propagation models. A major distinction between simulation and emulation is that simulation contains only software modules representing network protocols and network entities, such as routers and links, and mimicking network transactions as pure logic operations to the state variables. Examples of network simulators include Barr et al. (2005); Tyan and Hou (2001) and Varga (2001). The ns-2 simulator (Breslau et al., 2000) is one of the most popular simulators with a rich collection of network algorithms and protocols for both wired and wireless networks. To scale up network simulation, a number of parallel and distributed simulators have also been developed, which include SSFNet (Cowie et al., 1999), GTNets (Riley, 2003), ROSSNet (Yaun et al., 2003), and GloMoSim (Bajaj et al., 1999). Next, we describe parallel and distributed simulation as the enabling technique for real-time simulation.

2.2 Parallel and Distributed Simulation

Parallel and distributed simulation, also known as parallel simulation or parallel discrete-event simulation (PDES), is concerned with executing a single discrete-event simulation program on parallel computers (Fujimoto, 1990). By exploiting the concurrency of a simulation model, parallel simulation can overcome the limitations of sequential simulation in both execution time and memory space. The critical issue of allowing a discrete-event simulation program to run in parallel is to maintain the causality constraint, which means that simulation events in the system must be processed in a non-decreasing timestamp order. This is because an event with a smaller timestamp has the potential to change the state of the system and affect events that happen later (with larger timestamps). Most parallel simulation adopts spatial decomposition: a model is divided into sub-models called logical processes (LPs), each of which maintains its own local simulation clock and can run on a different processor. For network simulation, a simulated network can be partitioned into smaller sub-networks, each handled by a different processor.

The way how the causality constraint is enforced divides parallel simulation into conservative and optimistic approaches. The conservative approach strictly prohibits out-of-order event execution: a processor must be blocked from processing the next event in its event queue until it is safe to do so. That is, it must ensure that no event will arrive from another processor with a timestamp earlier than the local simulation clock. In contrast, the optimistic approach allows events to be processed out of order. Once a causality error is detected—an event arrives at a logical process with a timestamp in the simulated past—the simulation will be rolled back to a state before the error occurs. In order for the simulation to retract and recover from an erroneous execution path, state saving and recovery mechanisms are typically provided. The seminal work for the conservative approach is the CMB algorithm, an asynchronous algorithm proposed independently by Chandy and Misra (1979), and Bryant (1977). The CMB algorithm provides several important observations that epitomize the fundamentals of conservative synchronization. One important concept is lookahead. To avoid deadlock, an LP must determine a lower bound on the timestamp of messages it will send to another LP. In essence, Lookahead is the amount of simulation time that an LP can predict into the simulated future. Extensive performance studies emphasize

the importance of extrapolating lookahead from the model (Fujimoto, 1988,1989; Reed et al., 1988). Nicol (1996) gave a classification of lookahead based on different levels of knowledge that can be extracted from the model. The use of different dimensions of lookahead underscores conservative synchronization protocols. Several models have been shown to exhibit good lookahead properties, such as first-come-first-serve stochastic queuing networks (Nicol, 1988) and continuous-time Markov chains (Nicol and Heidelberger, 1995). In addition, several synchronization protocols have been developed to exploit lookahead for general applications, such as the conditional event approach by Chandy
and Sherman (1989), the YAWNS protocol by Nicol (1991), the bounded lag algorithm by Lubachevsky (1988), the distance-between-objects algorithm by Ayani (1989), and the TNE algorithm by Groselj and Tropper (1988).

The first optimistic synchronization protocol is the Time Warp algorithm (Jefferson, 1985). Since the optimistic approach allows events to be processed out of timestamp order, Time Warp provides mechanisms to "roll back" erroneous event processing. An LP is able to save and later restore the state of the LP and "unsend" any messages it sends to other LPs during an erroneous execution. Since Time Warp requires state saving during event processing, the algorithm must be able to reclaim the memory resource; otherwise, the simulation would soon run out of memory. To accomplish this, the concept of global virtual time (GVT) is introduced as a timestamp lower-bound of all unprocessed or partially processed events at any given time. It serves as a "moving commitment horizon": any message and state with a timestamp less than GVT can be reclaimed and any irrevocable operations (such as I/O) that happen before GVT can be committed. Time Warp needs to overcome several problems in order to maintain good efficiency. These problems have prompted a flood of research in areas of state saving (e.g., Gomes et al., 1996; Lin and Lazowska, 1990; Lin et al., 1993; Ronngren et al., 1996), rollback (e.g., Gafni, 1988; Reiher et al., 1990; West, 1988), GVT computation (e.g., Fujimoto and Hybinette, 1997; Mattern, 1993; Samadi, 1985), memory management (e.g., Jefferson, 1990; Lin and Preiss, 1991; Preiss and Loucks, 1995), and alternative optimistic execution (e.g., Dickens
and Reynolds, 1990; Sokol et al., 1988; Steinman, 1991, 1993).

The jury is out on which of the two approaches is a better choice. This is because parallel simulation performance largely depends on the characteristics of the simulation model. For network simulation, conservative synchronization is generally preferred as it requires a smaller memory footprint as opposed to the optimistic counterpart that generally needs additional memory for state saving and rollback. An interesting exception is the reverse computation technique (Carothers et al., 1999). Instead of applying state saving, one performs reverse computation to re-create the original state when rollback happens. Recent study shows that, with careful implementation, reverse computation achieves great memory efficiency in simulating large networks (Yaun et al., 2003).

3. Real-Time Network Simulation

Real-time simulation combines the advantages of simulation and emulation by conducting network simulation in real time and interacting with real applications and real network traffic. It allows us to study the impact of real application traffic on the virtual network and study real application behavior under a diverse set of simulated network conditions. Specifically, real-time network simulation provides the following capabilities:

- Accuracy. Real-time network simulation is based on simulation; thus, it is able to efficiently capture detailed packet-level transactions in the network. This is particularly true for simulating packet forwarding on wired infrastructure networks as it is relatively straightforward to calculate the link state with sufficient accuracy (such as the delay for a packet being forwarded from one router to the next). Real-time network simulation can also increase the fidelity of simulation since it can create real traffic conditions generated by real applications. Furthermore, existing implementations, such as routing protocols, can be incorporated directly in simulation rather than using a separate implementation just for simulation purposes. The design and implementation of network protocols, services, and applications is complex and labor-intensive. Maintaining code separately for simulation and for real deployment would have to include costly procedures for verification and validation.
- Repeatability. Repeatability is important to both protocol development and evaluation. In real-time network simulation, an experiment may or may not be repeatable, depending on whether interaction with the applications is repeatable or not. The virtual network in real-time network simulation is controlled by simulation events, and thus can be used to produce repeatable network conditions to test real network applications.
- Scalability. Emulation typically implements packet transmission by really directing a packet across a physical link, although in some cases this process can be accelerated by using special programmable devices (e.g., DeHart et al., 2006). In comparison, network operations in real-time network simulation are handled in software; each packet transmission involves only a few changes to the state variables in simulation that are computationally insignificant compared to the I/O overhead. Furthermore, since packet forwarding operations are relatively easy to parallelize, the simulated network can be scaled up far beyond what could be supported by emulation.
- Flexibility. Simulation is both a tool for analyzing the performance of existing systems and a tool for evaluating new design alternatives potentially under various operating settings. Once a simulation model is in place, it takes little effort to conduct simulation experiments, for example, to explore a wide spectrum of design space. We can also incorporate different analytical models in real-time network simulation. For example, we can use low-resolution models to describe aggregate Internet traffic behavior, which can significantly increase the scale of the network being simulated.

Most real-time network simulators are based on existing network simulators added with emulation capabilities in order to interact with real applications. Examples include NSE (Fall, 1999), IP-TNE (Bradford et al., 2000), MaSSF (Liu et al., 2003), and Maya (Zhou et al., 2004). NSE is an emulation extension of the popular ns-2 simulator with support for connecting with real applications and scheduling real-time events. ns-2 is built on a sequential discrete-event simulation engine, which severely limits the size of the network it is capable of simulating; for real-time simulation, this means that the size of the network has to be kept small to allow realtime processing. IP-TNE is an emulation extension of an existing parallel network simulator. It is the first simulator we know that applies parallel simulation to large-scale network emulations. MaSSF is built on our parallel simulator DaSSF with support for the grid computing environment. Maya is an emulation extension of a simulator for wireless mobile networks. Our real-time network simulator is called PRIME, which stands for Parallel Real-time Immersive network Modeling Environment. The

implementation of PRIME inherits most of our previous efforts in the development of DaSSF, a process-oriented and conservatively synchronized parallel simulation engine designed for multi-protocol communication networks. DaSSF can run on most platforms, including shared-memory multiprocessors and clusters of distributed-memory machines. The DaSSF simulation engine is ultra fast and has been demonstrated capable of handling large network models, including simulation of infrastructure networks, cellular systems, wireless ad hoc networks, and wireless sensor networks. In order to support large-scale simulation, PRIME applies advanced parallel simulation techniques. For example, to achieve good performance on distributed-memory machines, PRIME adopts a hierarchical synchronization scheme to address the discrepancy in the communication cost between distributed-memory and shared-memory platforms (Liu and Nicol, 2001). Further, PRIME implements the composite synchronization algorithm (Nicol and Liu, 2002), which combines the traditional synchronous and asynchronous conservative parallel simulation algorithms. Consequently, PRIME is able to efficiently simulate diverse network scenarios, including those that exhibit large variability in link types (particularly with the existence of low-latency connections), and node types (especially for those with a large degree of connectivity).

PRIME extends DaSSF with emulation capabilities, where unmodified implementations of real applications can interact with the network simulator that operates in real time. Traffic originated from the real applications is captured by PRIME's emulation facilities and forwarded to the simulator. The real network packets are treated as simulation events as they are "carried" on the virtual network and experience appropriate delays and losses according to the run-time state of the simulated network.

4. Supporting Real-Time Performance

Real-time network simulation needs to resolve two important and related issues: responsiveness and timeliness. Responsiveness dictates that the real-time simulator must be able to interact with real applications in time. That is, the system interface must be able to receive and respond to real-time events promptly according to proper real-time deadlines. Timeliness refers to the system's ability to keep up with the wall-clock time. That is, the simulation must be able to characterize the behavior of the networks, potentially with millions of network entities and with a large amount of network traffic flows, in real time. Failing to do so will introduce timing faults, where the simulation fails to process events before the designated deadlines. An elevated occurrence of timing faults will cause the simulator to become less responsive when interacting with real applications. In this section we briefly describe the techniques we developed so far to factor out these issues.

4.1 Hybrid Traffic Modeling

Large-scale real-time network simulation requires simulation be able to characterize the network behavior in real time. To speed up simulation, on the one hand, we apply parallel and distributed simulation techniques to harness the computing resources of parallel computers to physically increase the event-processing power; on the other hand, we resort to multi-resolution modeling techniques mixing models with high level of abstraction (and low resolution) to reduce the computational demand.

Our solution to this problem is to use a hybrid network traffic model that combines a fluid-

based analytical model using ordinary differential equations (ODEs) with the traditional packet-oriented discrete-event simulation (Liu, 2006). The model extends the fluid model by Liu et al. (2004) where ODEs are used to predict the mean behavior of the dynamic TCP congestion windows, the network queue lengths, and packet loss probabilities, as traffic flows through a set of network queues. These network queues are augmented with functions to handle both fluid flows and individual packets, as well as the interaction between them. We briefly describe the functions of these equations below. A detailed discussion of the hybrid model can be found in Liu (2006). We first define the variables in Table 1.

$$\frac{dW_i(t)}{dt} = \frac{1}{R_i(t)} - \frac{W_i(t)}{2} \cdot \lambda_i(t) \tag{1}$$

$$\frac{dq_l(t)}{dt} = \xi_l(t)(1 - p_l(t)) - C_l \tag{2}$$

$$\frac{dx_l(t)}{dt} = \frac{\ln(1-\alpha)}{\delta} x_l(t) - \frac{\ln(1-\alpha)}{\delta} q_l(t) \tag{3}$$

ni	number of (homogeneous) flows in fluid class i
W (t)	congestion window size of fluid class i at time t
Ri (t)	round trip time of fluid class i at time t
Xi (t)	loss rate of fluid class i at time t
qi (t)	instantaneous queue length at link I at time t
pi (t)	packet loss rate at link I at time t
xi (t)	average queue length at link I at time t
11 (t)	aggregate arrival rate at link I at time t
A (t) 4 (t)	arrival rate of fluid class i at link I at time t
D (t)	average packet arrival rate at link I at time t
	departure rate of fluid class i at link I at time t
d\ (t)	cumulative delay of fluid class i at link I at time t
Yl (t)	cumulative loss rate of fluid class i at link I at time t
h	first network queue (traversed by flow class i)
fn	last network queue (traversed by flow class i)
gi (l)	next queue of I for fluid class i
bi (l)	predecessor queue of I for fluid class i
al	propagation delay of link I
Ci	bandwidth of link I
Ni	set of fluid classes passing through link I
qa, qb, Px	RED queue parameters
a	weight used for RED EWMA calculation
	one-way path propagation delay for fluid class i

Table 1. Variables defined in the hybrid model.

$$p(x) = \begin{cases} 0 & 0 \le x < q_a \\ \dfrac{x - q_a}{q_b - q_a} px & q_a \le x < q_b \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

$$A_i^{f_1}(t) = \frac{n_i W_i(t)}{R_i(t)} \tag{5}$$

$$A_i^{g_i(l)}(t + a_l) = D_i^l(t) \tag{6}$$

$$\xi_l(t) = \sum_{i \in N_l} A_i^l(t) + A_P^l(t) \tag{7}$$

$$t_f = t + q_l(t) / C_l \tag{8}$$

$$D_i^l(t_f) = \begin{cases} A_i^l(t)(1 - p_l(t)) & \text{if } \xi_l(t)(1 - p_l(t)) \le C_l \\ \dfrac{A_i^l(t)}{\xi_l(t)} C_l & \text{otherwise} \end{cases} \tag{9}$$

$$d_l^i(t_f) = \begin{cases} \dfrac{q_l(t)}{C_l} & \text{if } l = f_1 \\ d_{b_i(l)}^i(t - a_{b_i(l)}) + a_{b_i(l)} + \dfrac{q_l(t)}{C_l} & \text{otherwise} \end{cases} \tag{10}$$

$$\gamma_l^i(t_f) = \begin{cases} A_i^l(t) p_l(t) & \text{if } l = f_1 \\ \gamma_{b_i(l)}(t - a_{b_i(l)}) + A_i^l(t) p_l(t) & \text{otherwise} \end{cases} \tag{11}$$

$$R_i(t) = d_{f_n}^i(t - \pi_i) + \pi_i \tag{12}$$

$$\lambda_i(t) = \gamma_{f_n}^i(t - \pi_i) / n_i \tag{13}$$

Equation (1) models the additive-increase-multiplicative-decrease (AIMD) behavior of a TCP congestion window during the congestion avoidance stage. The window size and the round-trip time determine the arrival rate at the first router in Equation (5). For UDP flows, we use a constant send rate instead. The arrival rate at subsequent routers is the same as the departure rate at the predecessor router only postponed by the link's propagation delay, as prescribed in Equation (6). Equation (7) sums up the arrivals of both fluid and packet flows. The total arrival rate, together with the loss probability and the link's bandwidth, are used to determine the instantaneous queue length in Equation (2). An average queue length is then calculated in Equation (3), which is derived from the Exponential Weighted Moving Average (EWMA) calculation in network queues with RED (Random Early Detection) queue management. The calculated average queue length contributes to the loss probability as

dictated by the RED policy in Equation (4). The loss probability for drop-tail queues can be calculated directly from projected buffer overflows. Equation (9) describes the departure rate as a function of the arrival rate postponed by the queuing delay calculated using Equation (8). Equations (10) and (11) calculate the cumulative delay and loss since the beginning when the segment of flow is originated from the traffic source. The cumulative delay and loss are used to calculate the round-trip time and the total loss rate in Equations (12) and (13), which in turn are used to calculate the congestion window size.

With proper performance optimization (Liu and Li, 2008), this hybrid traffic model can achieve significant performance improvement, in certain cases, over three orders of magnitude. The hybrid model can also be parallelized to achieve even greater performance.

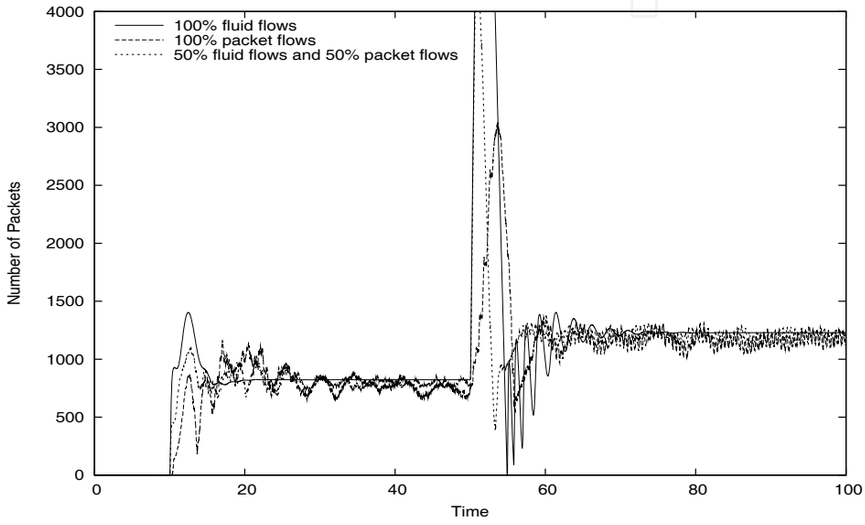

Fig. 1. Instantaneous queue length.

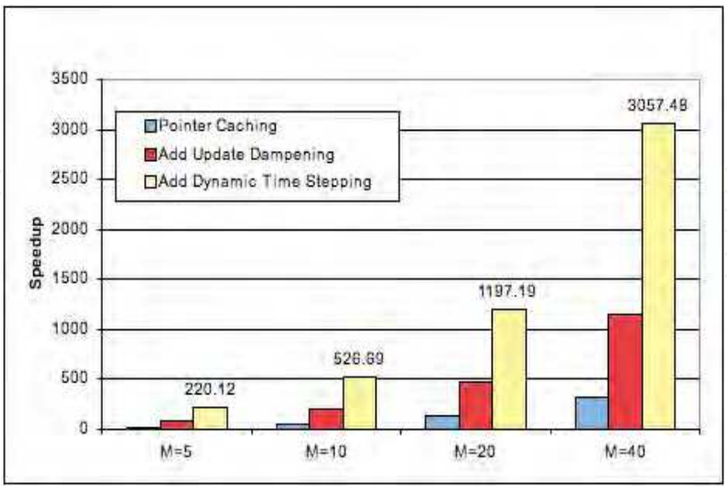

Fig. 2. Speedup over packet simulation.

To illustrate the potential of this approach, here we examine the accuracy and performance of the hybrid model using a simple dumbbell network model. In the experiment, the

dumbbell network contains two routers in the middle connecting N server nodes on one side and N client nodes on the other side. Each server node directs M simultaneous TCP flows to the corresponding client node. All links are set with a propagation delay of 5 ms. The experiments were run sequentially on an Apple Mac Pro with two 3 GHz dual-core Intel Xeon processors and 9 GB of memory. We first set N — 10 and M — 30. Half of the connections are established at time 10 and the rest at time 50. We set the bandwidth of the bottleneck link to be 20 Mb/s. Each server or client node connects to its adjacent router over a 10 Mb/s link.

Figure 1 compares the instantaneous queues lengths at the bottleneck router as predicted by fluid-based and packet-oriented simulations, as well as a hybrid of the two. The result from the fluid-based simulation matches well with that of the packet-oriented simulation in terms of averaged behavior. The hybrid model (with 50% fluid flows and 50% packet flows) produces similar results.

To show the overall performance benefit of our hybrid approach, we use the same dumbbell topology but change the parameters, such as the bandwidth at the bottleneck link, so that the cost of the simulation may increase proportionally as we increase the number of TCP sessions. Specifically, we vary M, the number of simultaneous TCP sessions between each pair of client-server nodes. We set the bandwidth of the link between each client or server node and its adjacent router to be (10 x M) Mb/s. The network queues at both ends of the link has a buffer size of M MB. The link between the two routers has a bandwidth of (10 ×M × N) Mb/s. The corresponding network queues in the two routers have a buffer size of (M × N) MB. All TCP sessions start at time 0 and the experiments are run for 100 simulated seconds. The rest of the parameters are the same as in the previous experiment. Figure 2 shows the speedup of the fluid model over the pure packet simulation with different performance improvement techniques enabled one at a time (see Liu and Li, 2008 for more details about these performance improvement techniques). Here we set N — 100 and M — {5,10,20,40}. We see that, as we turn on all improving techniques in the case of M — 40, we can obtain a speedup as much as 3,057 over packet-oriented simulation. The effective packet-event rate actually reaches over 566 million packet-event per second.

We further extend the hybrid model to represent network background traffic (Li and Liu, 2009a). In real-time network simulation, we can make a distinction between foreground traffic, which is generated by the real applications we intend to study with high fidelity, and background traffic, which represents the bulk of the network traffic that is of secondary interest and does not necessarily require significant accuracy. Nevertheless, background traffic interferes with foreground traffic as they both compete for network resources, and thus determines (and also is determined by) the behavior of network applications under investigation (Vishwanath and Vahdat, 2008).

Our enhanced model enables bi-directional flows and uses heavy-tail distributions to describe the flow durations. To enable bi-directional flows, we assume that the forwarding path of the TCP flows in the fluid class *i* (from the source to the destination) consists of n queues: $f_1, f_2, ..., f_n$, and the reverse path (from the destination to the source) consists of m queues: $r_1, r_2, ..., r_m$. We use Equation (5) to calculate the arrival rate at the first queue f1. For subsequent queues except r_1, i.e., $l \in \{f_2, ..., f_n, r_2, ..., r_m\}$, we use Equation (6) to calculate the arrival rate from the departure rate at the predecessor queue. For queue r_1 (the

first queue on the reverse path), we have:

$$A_i^{r_1}(t) = \alpha_i D_i^{f_n}(t) / \beta_i, \tag{14}$$

where α_1 is the average ACK packet size, and β_i is the average data packet size in fluid class *i*. This equation represents the conversion from the data flows on the forwarding path to the corresponding ACK flows on the reverse path.

To capture traffic burstness, we use the Poisson Pareto Burst process (PPBP) model to predict the aggregate Internet traffic. PPBP is a process based on multiple overlapping bursts, with Poisson arrival and burst lengths following a heavy-tail distribution (Zukerman et al., 2003). We schedule TCP session arrivals using the exponential distribution with a mean arrival rate μ. The durations of the TCP sessions d are independent and identically distributed Pareto random variables with parameters $\delta > 0$ and $1 < \gamma < 2$:

$$P_r(d > x) = \begin{cases} (x/\delta)^{-\gamma} & \text{if } x \geq \delta \\ 1 & \text{otherwise} \end{cases} \tag{15}$$

With the Pareto distributed flow duration, we can regenerate the long range dependence (LRD) characteristic of realistic background traffic in our model, which can be evaluated by a parameter called the Hurst parameter:

$$H = \frac{3-\gamma}{2}. \tag{16}$$

When $0.5 < H < 1$, it implies that the traffic exhibits LRD and is self-similar. In our fluid model, we replace the constant number of homogeneous fluid flows ni with the PPBP process, $N_i(t)$. Specifically, we redefine the equations for calculating the arrival rate at the first queue f_1 (Equation 5), and the end-to-end packet loss rate (Equation 13) as follows:

$$A_i^{f_1}(t) = \frac{N_i(t) W_i(t)}{R_i(t)} \tag{17}$$

$$\lambda_i(t) = \gamma_{r_m}^i(t) / N_i(t) \tag{18}$$

Figure 3 shows the result of an experiment using the same dumbbell model measuring the number of packets per second sent over time for both packet simulation (left plots) and the fluid background traffic model (right plots). From top down we progressively decreasing the sampling time scale, while maintaining the number of samples to be 300. The starting time scale is 1 second; each subsequent plot is obtained from the previous one by concentrating on a randomly chosen sub-interval with a length being one tenth of the previous one.

That is, the time resolution is increased by a factor of 10. To a large extent, the results from

the packet-oriented simulation and from the fluid-based simulation are similar, except for the 10 ms timescale (bottom plots). The fluid model does not capture packet details at sub-RTT level; the RTT for the dumbbell model is at least 10 ms.

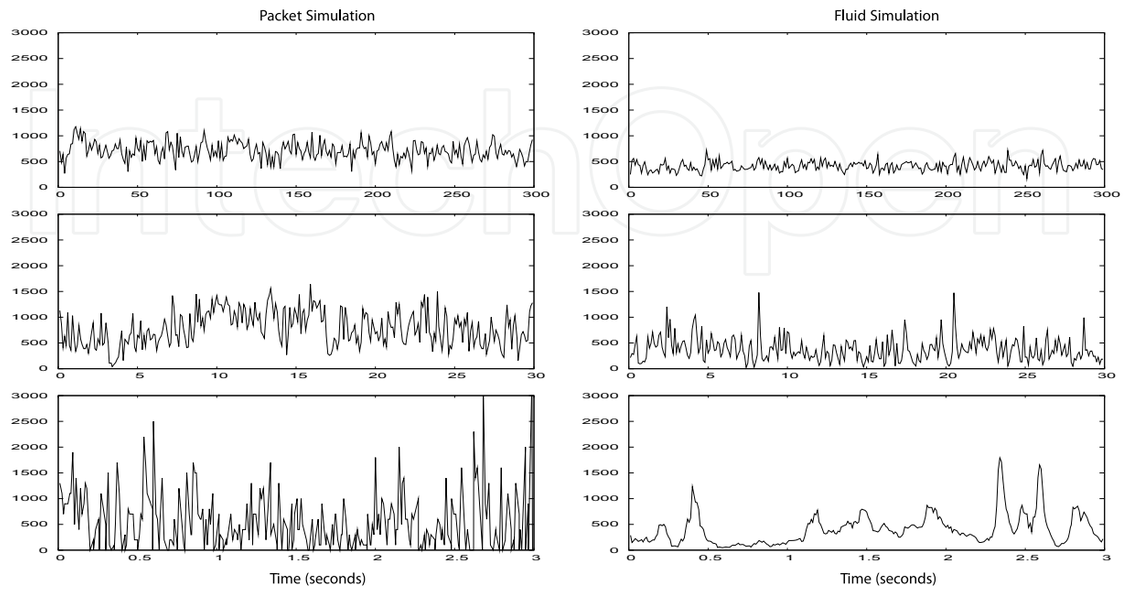

Fig. 3. Traffic burstness.

4.2 Scalable Emulation Infrastructure

A large-scale network simulation must be able to interact with a large number of real applications. The emulation infrastructure, which connects the simulator to the applications, must be able to embed real applications easily in the real-time simulation. There are several ways to incorporate real applications into a simulation environment, the decision of which to use largely depends on where the interactions take place. Several techniques exist that allow running unmodified software, which include using packet capturing techniques (such as libpcap, IP table, and IP tunnel), preloading dynamic libraries, and modifying the binary executables. In certain cases, moderate software modifications are necessary to allow efficient direct execution.

Our first attempt follows an open system approach (Liu et al., 2007). The emulation infrastructure is built on the Virtual Private Network (VPN), which is customized to function as a gateway that bridges traffic between the physical entities and the simulated network (see Figure 4). Client machines run real applications. They establish connection to the simulation gateway as VPN clients (by running an automatically generated VPN configuration scripts). Traffic generated by the applications running on the client machines and destined for the virtual network is directed by the emulation infrastructure to the real-time network simulator. We use an example to show how it works. Suppose two client machines are connected to the simulation gateway (not necessarily the same one) and want to communicate with each other. One client is assigned with the IP address `10.0.0.14` and the other with `10.0.1.2`. Packets

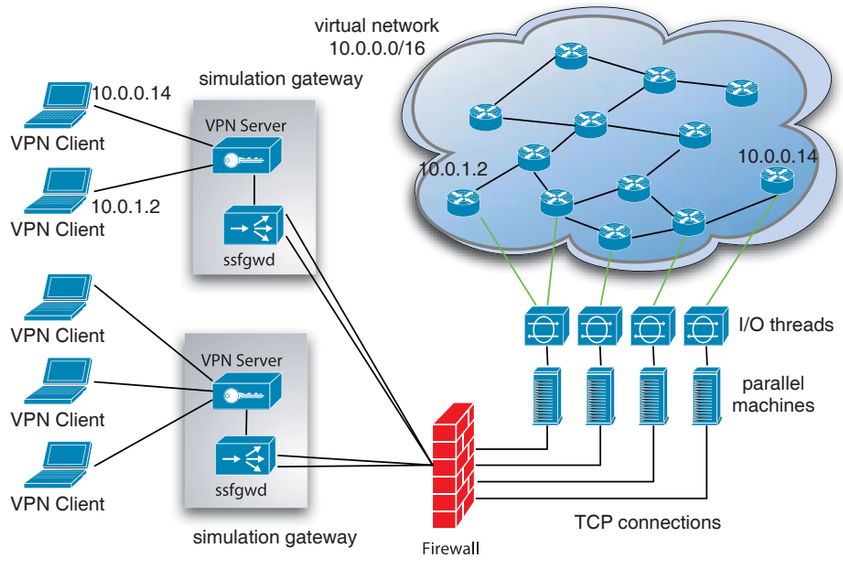

Fig. 4. VPN emulation infrastructure.

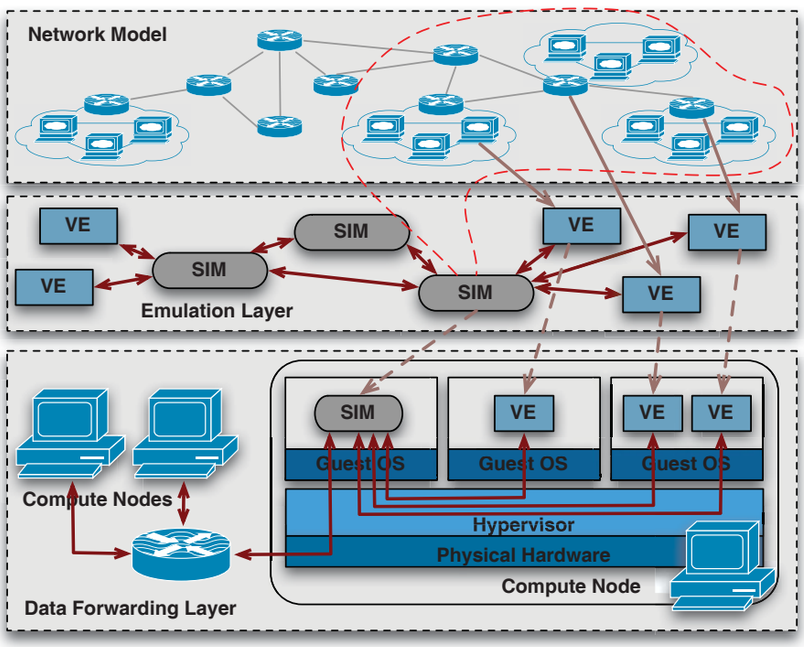

Fig. 5. VM emulation infrastructure.

sent from `10.0.0.14` to `10.0.1.2` are forwarded to the VPN server at the simulation gateway. The VPN server has been altered to forward the packets to a daemon process (ssfgwd), which then sends the packets to the real-time simulator via a dedicated TCP connection. At the simulator, the packets are injected into the simulation event list; the simulator simulates the packets being forwarded on the virtual network as if they were created by the virtual

node with the same IP address `10.0.0.14`. Upon reaching the virtual node `10.0.1.2`, the packets are exported from simulation and travel in the reverse direction via the simulation gateway back to the client machine assigned with the IP address `10.0.1.2`.

One distinct advantage of this approach is that the emulation infrastructure does not require special hardware to set up. It is also secure and scalable, which are merits inherited directly from the underlying VPN implementation. Multiple simulation gateways can run simultaneously. In order to produce accurate results, however, the emulation infrastructure needs a tight coupling between the emulated entities (i.e., the client machines) and the real-time simulator. In particular, the segment between the client machines and the real-time network simulator should consist of only low-latency links. To maintain high throughput, the segment must also provide sufficient bandwidth to carry the emulation traffic. With these constraints, the physical latency between the clients and the simulator can actually be made transparent in the network model (Liljenstam et al., 2005). The idea is to allow an emulation packet in simulation to preempt other simulated packets in the network queues so that the packet can be delivered ahead of its schedule in order to compensate for the physical delays.

We also inspect machine virtualization solutions for an accurate environment of running real applications. Machine virtualization has found a number of interesting applications, including resource management in data centers, security, virtual desktop environments, and software distribution. Recently, researchers have also proposed using virtualization techniques for building network emulation testbeds. We follow the method proposed by Maier et al. (2007) to classify virtual machine (VM) solutions for network emulation. Classical virtual machines, such as VMWare Workstation and User-Mode Linux (Dike, 2000), provide full machine virtualization and can therefore run unmodified guest operating systems. These solutions offer complete transparency (with a complete abstraction of a computer system) to the guest operating system, but in doing so incur a large performance overhead. Light-weight virtual machines, such as Xen (Barham et al., 2003), VMWare ESX Server, and Denali (Whitaker et al., 2002), implement partial virtualization for greater efficiency, but require slight modification of guest OSes.

In addition to virtualizing an entire operating system instance, researchers have proposed virtual network stacks (Bavier et al., 2006; Huang et al., 1999; OpenVZ; Soltesz et al., 2007; Zec, 2003) and virtual routers (Maier et al., 2007; VRF) as alternative solutions. With virtual network stacks, applications running on the same OS instance are presented with multiple independent network stacks, which can be managed individually and control distinct physical devices. With virtual routers, a single OS instance can maintain multiple routing table instances, thereby allowing the co-execution of multiple router software. Since these two techniques only virtualize the network resource, they provide greater efficiency than light-weight VMs. They do not, however, provide a complete isolation of resources (such as CPU); they are also invasive, sometimes requiring substantial modification to the guest OS.

Our work so far has explored the use of light-weight virtual machines and virtual network stacks as candidate emulated elements in a real-time simulation infrastructure. We have built a real-time simulation infrastructure that can seamlessly use light-weight virtual machines to emulate arbitrary network elements including routers and application end-points. We looked into four types of network resources that may be provided by a virtual machine: network sockets, network interfaces, forwarding table, and loopback device. Network sockets (TCP, UDP, and raw sockets) are used by applications to establish connectivity and exchanging information. Network interfaces and the forwarding table are

used by routing protocols to conduct network forwarding. A network loopback device is sometimes used by separate processes to communicate on the same machine. We investigated four popular virtualization technologies: Xen, OpenVZ, Linux-VServer and VRF and found that, while all four types of network resources are provided in Xen and OpenVZ, Linux-VServer and VRF have only partial network virtualization support.
Figure 5 shows a high-level view of our VM-based emulation infrastructure. We view each physical machine as a basic scaling unit, where emulated hosts are mapped to independent virtual machines (or virtual environments) so that they can run unmodified applications. Each instance of the real-time simulator runs on a separate virtual machine of the same physical machine, and processes events associated with a designated sub-network. The simulator instances on different physical machines are synchronized using conservative parallel simulation techniques. Real network traffic generated by the applications is intercepted by the hypervisor (or VM manager) and sent to the virtual machine where the corresponding realtime simulator instance is located. The simulator then processes these packets applying packet delays and losses according to the simulated network conditions.

5. Applications and Case Studies

We have been able to successfully apply real-time simulation to study many applications, including routing algorithms, transport protocols, content distribution services, web services, multimedia streaming, and peer-to-peer networks. In this section, we select several case studies to demonstrate the potentials of real-time simulation.

5.1 Large-Scale Routing Experiments

The availability of open-source router platforms, such as XORP, Zebra, and Quagga, has simplified the task of researchers, who can now prototype and evaluate routing protocols with relative ease. To support experiments on a large-scale network consisting of many routers with multiple traffic sources and sinks, we need to integrate the open-source router platforms with the real-time network simulator.
Since the routers are emulated outside the real-time simulator on client machines where they can run the real routing software directly, every packet traveling along its path from the source to the destination needs to be exported to each intermediate router for forwarding decisions, and subsequently imported back into the simulation engine. Thus, the forwarding operation for each packet at each hop would incur substantial I/O overhead. Consequently, the overall overhead would significantly impact the performance of the emulation infrastructure, especially in large-scale routing experiments. To avoid this problem, we propose a forwarding plane offloading approach, which moves the packet forwarding functions from the emulated router software to the simulation engine so that we can eliminate the I/O overhead associated with communicating bulk-traffic back and forth between the router software and the real-time
simulator (Li et al., 2008).
In our current implementation, we combine XORP with PRIME to provide a scalable platform for conducting routing experiments. We create a forwarding plane plug-in in XORP, which maintains a command channel with the PRIME simulator for transferring forwarding information updates and network interface configuration requests between the XORP instance and the corresponding simulated router.

We carried out several experiments using the scalable routing platform. These experiments include an intra-domain routing experiment consisting of a realistic Abilene network model (Li et al., 2008) with the objective of observing the convergence of OSPF and its effect on data traffic. We injected a link failure followed by a recovery between two routers on the network. We were able to measure their effect on the round-trip time and data throughput of end applications. We also conducted realistic large-scale inter-domain routing experiments consisting of major autonomous systems connecting Swedish Internet users with realistic routing configurations derived from the routing registry (Li and Liu, 2009b). We ran a series of real-time security exercises on this routing system to study the consequence of intentionally propagating false routing information on interdomain routing and the effectiveness of corresponding defensive measures.

5.2 Large-Scale TCP Evaluation

The TCP congestion control mechanism, which limits the rate of data entering the network, is essential to the overall stability of the network under traffic congestion and important to the protocol's performance. It has been widely documented that the traditional TCP congestion control algorithms (such as TCP Reno and TCP SACK) have serious problems preventing TCP from reaching high data throughput over high-speed long-latency links. Consequently, quite a number of TCP variants have been proposed to directly tackle these problems. Compared with the traditional methods, these TCP variants typically adopt more aggressive congestion control methods in order to address the under-utilization problem of TCP over networks with a large bandwidth-delay product.

The ability to establish an objective comparison between these high-performance TCP variants under diverse networking conditions and to obtain a quantitative assessment of their impact on the global network traffic is essential to a community-wide understanding of various design approaches. Small-scale experiments are insufficient for a comprehensive study of these TCP variants. We developed a TCP performance evaluation testbed, called SVEET, based on real-time simulation technique using real implementations of the TCP variants, which are evaluated under diverse network configurations and workloads in large-scale network settings (Erazo et al., 2009).

In order for SVEET to accommodate data communications with multi-gigabit throughput performance, we apply time dilation, proportionally slowing down the virtual machines and the network simulator. Using time dilation allows us to provide much higher bandwidths than what can be provided by the physical system and the network simulator at the cost of increased experiment time. We adopt the time dilation technique developed by Gupta et al. (2006), which can uniformly slow the passage of time from the perspective of the guest operating system (XenoLinux). This is achieved primarily by enlarging the interval between timer interrupts delivered to the virtual machines from the Xen hypervisor by a specified factor, called the Time Dilation Factor (TDF). Time dilation can scale the perceived I/O rate and processing power on the virtual machines by the same factor. For instance, if a virtual machine has a TDF of 10, it means that the time, as perceived by the applications running on the virtual machine, will be advanced at a pace 10 times slower than the true wall-time clock. Similarly, the applications would experience a tenfold increase in both network capacity and CPU cycles.

We ported several TCP congestion control algorithms from the ns-2 simulator consisting of thirteen TCP variants originally implemented for Linux. In doing so we are able to conduct

large-scale experiments using simulated traffic generated by these TCP variants. We also customized the Linux kernel on the virtual machines to include these TCP variants so that we can test them using real applications running on the virtual machines to communicate via the TCP/IP stack. We conducted extensive experiments to validate our testbed and investigated the impact of TCP variants on web applications, multimedia streaming, and peer-to-peer traffic.

5.3 Large-Scale Peer-to-Peer Content Distribution Network

We design one of the largest network experiments that involve a real implementation of a peer-to-peer content distribution system under HTTP traffic from a public-domain empirical workload trace and using a realistic large network model (Liu et al., 2009). The main idea behind the content distribution network (CDN) is to replicate content at the edge of the Internet closer to the clients. In doing so, CDN can alleviate both the workload at the server and the traffic load at the network core. We choose to use an open-source CDN system called CoralCDN (Freedman et al., 2004), which is a peer-to-peer web-content distribution network that consists of three parts: 1) a network of cooperative web proxies for handling HTTP requests, 2) a network of domain name servers (DNS) to map clients to nearby web proxies, and 3) an underlying clustering mechanism and an indexing infrastructure to facilitate DNS mapping and content distribution. We statically mapped the clients to nearby Coral nodes to send HTTP requests. Thus we ignore CoralCDN's DNS redirection function and only focus on web-content distribution for the experiment.

We extend the Rocketfuel to build the network model for our study. Rocketfuel (Spring et al., 2004) contains the topology of 13 tier-1 ISPs, derived from information obtained from traceroute paths, BGP routing tables, and DNS. Previously, we created a best-effort Internet topology for large-scale network simulation studies using the Rocketfuel dataset (Liljenstam et al., 2003). Based on this study, we further process the Rocketfuel network topology to improve accuracy and reduce data noise. We choose to use one of the tier-1 ISP networks for our study, which contains 637 routers (out of which 235 are backbone routers) connected by 1,381 links. Attached to the backbone network are medium-sized stub networks, called the campus network. Each campus network consists of 504 end hosts, organized into 12 local area networks (LANs) connected by 18 routers. Four extra end hosts are designated to form a server cluster. Each LAN consists of a gateway router and 42 end-hosts. The entire campus network is divided into four OSPF areas. The campus network is connected to the outside world through a BGP router. We attach 84 such campus networks to the tier-1 ISP network. The entire network thus contains 42,672 end hosts and 3,157 routers.

We place one CoralCDN node within each of the 12 LANs of the 84 campus network (at one of the 42 end hosts in each LAN), thus making a total of 1,008 CoralCDN nodes overall. Each CoralCDN node is emulated in a separate OpenVZ container. The web clients are simulated; they send HTTP requests to the CoralCDN node within the same LAN and subsequently receive data objects from the Coral proxy. PRIME implements a full-fledged TCP model that allows simulated nodes to interact with real TCP counterparts. We attach a stub network to a backbone router in the tier-1 ISP network (located in Paris, France) to run a web server, emulated on a separate compute node.

We select the HTTP trace at the 1998 World Cup web site, which is publicly available (Arlitt and Jin, 1998). The trace is collected with all HTTP requests made to the 1998 World Cup Web site. We select a 24-hour period of this trace (from June 5,1998, 22:00:01 GMT to June

6,1998, 22:00:00 GMT). The segment consists of 5,452,684 requests originated from 40,491 clients. We pre-process the trace to filter out the sequence of requests sent from each client and randomly map the 40,491 clients to the end hosts in our network model for a complete daily pattern of the caching behavior. Through the experiment, we were able to successfully collect three important metrics to analyze the performance the peer-to-peer content distribution network: cache hit rate, web server load, and response time.

6. Conclusions and Future Work

In this chapter we describe real-time simulation of large-scale networks and compare it against other major tools for networking research. We discuss the problems that may prevent simulation from achieving real-time performance and subsequently present our current solutions. We conduct large-scale network experiments incorporating real-time simulation to demonstrate its capabilities.

Future work includes efficient background traffic models for large-scale networks, high-performance communication conduit for connecting virtual machines and the real-time simulator, and effective methods for configuring, running and visualizing network experiments.

Acknowledgments

This chapter significantly extends our previous work (Liu, 2008) with a high-level summary of published results thereafter. Our research reported in this chapter is supported in part by National Science Foundation grants CNS-0546712, CNS-0836408 and HRD-0833093.

7. References

Jong Suk Ahn, Peter B. Danzip, Zhen Liu, and Limin Yan. Evaluation of TCP Vegas: emulation and experiment. In Proceedings of the 1995 ACM SIGCOMM Conference, pages 185-195, August 1995.

Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the Internet impasse through virtualization. Computer, 38(4):34—41, 2005.

Martin Arlitt and Tai Jin. 1998 World Cup web site access logs. Available at: http://www.acm.org/sigcomm/ITA/, August 1998.

Rassul Ayani. A parallel simulation scheme based on the distance between objects. Proceedings of the 1989 SCS Multiconference on Distributed Simulation, 21(2):113-118, March 1989.

Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, and Mario Gerla. GloMoSim: a scalable network simulation environment. Technical Report 990027, Department of Computer Science, UCLA, May 1999.

Paul Barford and Larry Landweber. Bench-style network research in an Internet instance laboratory. ACM SIGCOMM Computer Communication Review, 33(3):21-26, 2003.

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), 2003.

Rimon Barr, Zygmunt Haas, and Robbert van Renesse. JiST: An efficient approach to simulation using virtual machines. Software Practice and Experience, 35(6):539-576, May 2005.

Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI veritas: realistic and controlled network experimentation. ACMSIGCOMMComputer Communication Review, 36(4):3-14, 2006.

Terry Benzel, Robert Braden, Dongho Kim, Clifford Neuman, Anthony Joseph, Keith Sklower, Ron Ostrenga, and Stephen Schwab. Experience with DETER: A testbed for security research. In Proceedings of 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM'06), March 2006.

Russell Bradford, Rob Simmonds, and Brian Unger. A parallel discrete event IP network emulator. In Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'00), pages 315-322, August 2000.

Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. IEEE Computer, 33(5):59-67, May 2000.

Randal E. Bryant. Simulation of packet communication architecture computer systems. Technical Report MIT-LCS-TR-188, MIT, 1977.

Christopher D. Carothers, Kalyan S. Perumalla, and Richard M. Fujimoto. Efficient optimistic parallel simulations using reverse computation. ACM Transactions on Modeling and Computer Simulation, 9(3):224-253, July 1999.

Mark Carson and Darrin Santay. NIST Net: a Linux-based network emulation tool. SIGCOMM Computer Communication Review, 33(3):111-126, 2003.

K. M. Chandy and R. Sherman. The conditional event approach to distributed simulation. Proceedings of the 1989 SCS Multiconference on Distributed Simulation, 21(2):93-99, March 1989.

K. Mani Chandy and Jayadev Misra. Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering, SE-5 (5):440^52, May 1979.

James Cowie, David Nicol, and Andy Ogielski. Modeling the global Internet. Computing in Science and Engineering, 1(1):42-50, January 1999. DaSSF. Dartmouth Scalable Simulation Framework. http://users.cis.fiu.edu/^liux/research/projects/dassf/index.html.

John DeHart, Fred Kuhns, Jyoti Parwatikar, Jonathan Turner, Charlie Wiseman, and Ken Wong. The open network laboratory. ACM SIGCSE Bulletin, 38(1):107-111, 2006.

Phillip M. Dickens and Paul F. Reynolds. SRADS with local rollback. Proceedings of the 1990 SCS Multiconference on Distributed Simulation, 22(1):161-164, January 1990.

Jeff Dike. A user-mode port of the Linux kernel. In Proceedings of the 4th Annual Linux Showcase & Conference, 2000.

Miguel Erazo, Yue Li, and Jason Liu. SVEET! A scalable virtualized evaluation environment for TCP. In Proceedings of the 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom'09), April 2009.

Kevin Fall. Network emulation in the Vint/NS simulator. In Proceedings of the 4th IEEE Symposium on Computers and Communications (ISCC'99), pages 244-250, July 1999.

Sally Floyd and Vern Paxson. Difficulties in simulating the Internet. IEEE/ACM Transactions on Networking, 9(4):392-403, August 2001.

Michael J. Freedman, Eric Freudenthal, and David Mazieres. Democratizing content publication with Coral. In Proceedings of the 1st USENIX Symposium on Networked Systems

Design and Implementation (NSDI 04), pages 239-252, 2004.

Richard M. Fujimoto. Lookahead in parallel discrete event simulation. In Proceedings of the 1988 International Conference on Parallel Processing, pages 34-41, August 1988.

Richard M. Fujimoto. Performance measurements of distributed simulation strategies. Transactions of the Society for Computer Simulation, 6(2):89-132, April 1989.

Richard M. Fujimoto. Parallel discrete event simulation. Communications ofthe ACM, 33(10): 30-53, October 1990.

Richard M. Fujimoto and Maria Hybinette. Computing global virtual time in shared memory multiprocessors. ACM Transactions on Modeling and Computer Simulation, 7(4):425-446, October 1997.

A. Gafni. Rollback mechanisms for optimistic distributed simulation systems. Proceedings of the 1988 SCS Multiconference on Distributed Simulation, 19(3):61-67, July 1988.

Fabian Gomes, Brian Unger, and John Cleary. Language based state saving extensions for optimistic parallel simulation. In Proceedings of the 1996 Winter Simulation Conference (WSC'96), pages 794-800, December 1996.

Bojan Groselj and Carl Tropper. The time of next event algorithm. Proceedings ofthe 1988 SCS Multiconference on Distributed Simulation, 19(3):25-29, July 1988.

Diwaker Gupta, Kenneth Yocum, Marvin McNett, Alex Snoeren, Amin Vahdat, and Geoffrey Voelker. To infinity and beyond: time-warped network emulation. In Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 06), 2006.

Daniel Herrscher and Kurt Rothermel. A dynamic network scenario emulation tool. In Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN'02), pages 262-267, October 2002.

X. W. Huang, R. Sharma, and S. Keshav. The ENTRAPID protocol development environment. In Proceedings of the 1999 IEEE INFOCOM, pages 1107-1115, March 1999.

David R. Jefferson. Virtual time. ACM Transactions on Programming Languages and Systems, 7 (3):404^25, July 1985.

David R. Jefferson. Virtual time II: Storage management in distributed simulation. In Proceeings ofthe 9th Annual ACM Symposium on Principles ofDistributed Computing, pages 75-89, August 1990.

Xuxian Jiang and Dongyan Xu. VIOLIN: Virtual internetworking on overlay infrastructure.

In Proceedings of the 2nd Internattonai Symposium on Paraiiei and Distributed Processing and Appiications (ISPA'04), pages 937-946, 2004.

Glenn Judd and Peter Steenkiste. Repeatable and realistic wireless experimentation through physical emulation. ACM SIGCOMM Computer Communication Review, 34(1):63-68, 2004.

Ting Li and Jason Liu. A fluid background traffic model. In Proceedings of the 2009 IEEE Internationai Conference on Communications (ICC'09), June 2009a.

Yue Li and Jason Liu. Real-time security exercises on a realistic interdomain routing experiment platform. In Proceedings of the 23rd Workshop on Principies of Advanced and Distributed Simuiation (PADS 09), June 2009b.

Yue Li, Jason Liu, and Raju Rangaswami. Toward scalable routing experiments with real-time network simulation. In Proceedings of the 22nd Workshop on Principies of Advanced and Distributed Simuiation (PADS'08), pages 23-30, June 2008.

Michael Liljenstam, Jason Liu, and David M. Nicol. Development of an Internet backbone topology for large-scale network simulations. In Proceedings of the 2003 Winter Simu-iation Conference, pages 694-702, 2003.

Michael Liljenstam, Jason Liu, David M. Nicol, Yougu Yuan, Guanhua Yan, and Chris Grier. RINSE: the real-time interactive network simulation environment for network security exercises. In Proceedings of the 19th Workshop on Paraiiei and Distributed Simuiation (PADS'05), pages 119-128, June 2005.

Yi-Bing Lin and Edward D. Lazowska. Reducing the state saving overhead for Time Warp parallel simulation. Technical Report 90-02-03, Department of Computer Science, University of Washington, February 1990.

Yi-Bing Lin and Bruno R. Preiss. Optimal memory management for Time Warp parallel simulation. ACM Transactions on Modeiing and Computer Simuiation, 1(4):283-307, October 1991.

Yi-Bing Lin, Bruno Richard Preiss, Wayne Mervin Loucks, and Edward D. Lazowska. Selecting the checkpoint interval in Time Warp simulation. In Proceedings of the 7th Workshop on Paraiiei and Distributed Simuiation (PADS 93), pages 3-10, May 1993.

Jason Liu. Packet-level integration of fluid TCP models in real-time network simulation. In Proceedings of the 2006 Winter Simuiation Conference (WSC'06), pages 2162-2169, December 2006.

Jason Liu. A primer for real-time simulation of large-scale networks. In Proceedings ofthe 41st Annuai Simuiation Symposium (ANSS'08), April 2008.

Jason Liu and Yue Li. On the performance of a hybrid network traffic model. Simuiation Modeiiing Practice and Theory, 16(6):656-669, 2008.

Jason Liu and David M. Nicol. Learning not to share. In Proceedings of the 15th Workshop on Paraiiei and Distributed Simuiation (PADS'01), pages 46-55, May 2001.

Jason Liu, Scott Mann, Nathanael Van Vorst, and Keith Hellman. An open and scalable emulation infrastructure for large-scale real-time network simulations. In Proceedings of 2007 IEEE INFOCOM MiniSymposium, pages 2471-2475, May 2007.

Jason Liu, Yue Li, and Ying He. A large-scale real-time network simulation study using PRIME. In Proceedings ofthe 2009 Winter Simuiation Conference (WSC 09), December

2009. To appear.

Xin Liu, Huaxia Xia, and Andrew A. Chien. Network emulation tools for modeling grid behavior. In Proceedings of3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03), May 2003.

Yong Liu, Francesco Presti, Vishal Misra, Donald Towsley, and Yu Gu. Scalable fluid models and simulations for large-scale IP networks. ACM Transactions on Modeling and Computer Simulation (TOMACS), 14(3):305-324, July 2004.

Boris D. Lubachevsky. Bounded lag distributed discrete event simulation. Proceedings ofthe 1988 SCS Multiconference on Distributed Simulation, 19(3):183-191, July 1988.

Steffen Maier, Daniel Herrscher, and Kurt Rothermel. Experiences with node virtualization for scalable network emulation. Computer Communications, 30(5):943-956, 2007.

Friedemann Mattern. Efficient distributed snapshots and global virtual time approximation. Journal of Parallel and Distributed Computing, 18(4):423-434, August 1993.

David M. Nicol. Parallel discrete-event simulation of FCFS stochastic queueing networks. ACM SIGPLAN Notices, 23(9):124-137, September 1988.

David M. Nicol. Performance bounds on parallel self-initiating discrete-event simulations. ACM Transactions on Modeling and Computer Simulation, 1(1):24-50, January 1991.

David M. Nicol. Principles of conservative parallel simulation. In Proceedings ofthe1996 Winter Simulation Conference (WSC 96), pages 128-135, December 1996.

David M. Nicol and Philip Heidelberger. A comparative study of parallel algorithms for simulating continuous time Markov chains. ACM Transactions on Modeling and Computer Simulation, 5(4):326-354, October 1995.

David M. Nicol and Jason Liu. Composite synchronization in parallel discrete-event simulation. IEEE Transactions on Parallel and Distributed Systems, 13(5):433-446, May 2002. OpenVZ. http://openvz.org/.

Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the Internet. HotNets-I, October 2002.

Bruno Richard Preiss and Wayne Mervin Loucks. Memory management techniques for Time Warp on a distributed memory machine. In Proceedings ofthe 9th Workshop on Parallel and Distributed Simulation (PADS 95), pages 30-39, June 1995.

PRIME. http://www.primessf.net/.

Quagga. http://www.quagga.net/.

D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In Proceedings ofthe IEEE Wireless Communications and Networking Conference (WCNC 05), March 2005.

Daniel A. Reed, Allen D. Malony, and Bradley McCredie. Parallel discrete event simulation using shared memory. IEEE Transactions on Software Engineering, 14(4):541-53, April 1988.

P. L. Reiher, R. M. Fujimoto, S. Bellenot, and D. Jefferson. Cancellation strategies in optimistic execution systems. Proceedings ofthe 1990 SCS Multiconference on Distributed Simulation, 22(1):112-121, January 1990.

George F. Riley. The Georgia Tech network simulator. In Proceedings ofthe ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research (MoMe-Tools 03), pages 5-12, August 2003.

Luigi Rizzo. Dummynet: a simple approach to the evaulation of network protocols. ACM SIGCOMM Computer Communication Review, 27(1):31-41, January 1997.

Robert Ronngren, Michael Liljenstam, Rassul Ayani, and Johan Montagnat. Transparent incremental state saving in Time Warp parallel discrete event simulation. In Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS'96), pages 70-77, May 1996.

Behrokh Samadi. Distributed simulation, algorithms and performance analysis. PhD thesis, Department of Computer Science, UCLA, 1985.

L. M. Sokol, D. P. Briscoe, and A. P. Wieland. MTW: A strategy for scheduling discrete simulation events for concurrent execution. Proceedings of the 1988 SCS Multiconference on Distributed Simulation, 19(3):34^2, July 1988.

Stephen Soltesz, Herbert Potzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson.
Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systemsof (EuroSys'07), March 2007.

Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring isp topologies with rocketfuel. IEEE/ACM Transactions on Networking, 12(1):2-16, 2004.

Neil Spring, Larry Peterson, Andy Bavier, and Vivek Pai. Using PlanetLab for network research: myths, realities, and best practices. ACM SIGOPS Operating Systems Review, 40(1):17-24, 2006.

Jeff S. Steinman. SPEEDES: Synchronous parallel environment for emulation and discrete event simulation. Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation, SCS Simulation Series, 23(1):95-103, January 1991.

Jeff S. Steinman. Breathing Time Warp. In Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'93), pages 109-118, May 1993.

Ananth I. Sundararaj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In Proceedings of the 3rd USENIX Conference on Virtual Machine Technology (VM'04), pages 14-14, 2004.

Joe Touch. Dynamic Internet overlay deployment and management using the X-Bone. In Proceedings of the 2000 International Conference on Network Protocols (ICNP'00), pages 59-68, 2000.

Hung-ying Tyan and Jennifer Hou. JavaSim: A component based compositional network simulation environment. In Proceedings of the Western Simulation Multiconference, January 2001.

Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. Scalability and accuracy in a large scale network emulator. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02), pages 271-284, December 2002.

Andrs Varga. The OMNeT++ discrete event simulation system. In Proceedings of the European Simulation Multiconference (ESM 01), June 2001.

Kashi Venkatesh Vishwanath and Amin Vahdat. Evaluating distributed systems: Does background traffic matter. In Proceedings of the 2008 USENIX Technical Conference, pages 227-240, May 2008.

VMWare ESX Server. http://www.vmware.com/products/vi/esx/.

VMWare Workstation. http://www.vmware.com/products/desktop/workstation.html.

VRF. Linux Virtual Routing and Forwarding. http://sourceforge.net/projects/ linux-vrf/.

Darrin West. Optimizing Time Warp: Lazy rollback and lazy re-evaluation. Master's thesis, Department of Computer Science, University of Calgary, January 1988.

A. Whitaker, M. Shaw, and S. Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In Proceedings of the USENIX Annual Technical Conference,

June 2002. Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 02), pages 255-270, December 2002.

XORP. http://www.xorp.org/.

Garrett Yaun, David Bauer, Harshad Bhutada, Christopher Carothers, Murat Yuksel, and Shiv-kumar Kalyanaraman. Large-scale network simulation techniques: examples of TCP and OSPF models. ACM SIGCOMM Computer Communication Review, 33(3):27-41, 2003.

Zebra. http://www.zebra.org/.

Marko Zec. Implementing a clonable network stack in the FreeBSD kernel. In Proceedings of the 2003 USENIX Annual Technical Conference, June 2003.

Pei Zheng and Lionel M. Ni. EMPOWER: a network emulator for wireline and wireless networks. In Proceedings of the 2003 IEEE INFOCOM, volume 3, pages 1933-1942, March/April 2003.

Junlan Zhou, Zhengrong Ji, Mineo Takai, and Rajive Bagrodia. MAYA: integrating hybrid network modeling to the physical world. ACM Transactions on Modeling and Computer Simulation (TOMACS), 14(2):149-169, April 2004.

Moshe Zukerman, Timothy D. Neame, and Ronald G. Addie. Internet traffic modeling and future technology implications. In Proceedings ofthe2003 IEEE INFOCOM, 2003.

Parallel and Distributed Computing

Edited by Alberto Ros

ISBN 978-953-307-057-5

Hard cover, 290 pages

Publisher InTech

Published online 01, January, 2010

Published in print edition January, 2010

The 14 chapters presented in this book cover a wide variety of representative works ranging from hardware design to application development. Particularly, the topics that are addressed are programmable and reconfigurable devices and systems, dependability of GPUs (General Purpose Units), network topologies, cache coherence protocols, resource allocation, scheduling algorithms, peertopeer networks, largescale network simulation, and parallel routines and algorithms. In this way, the articles included in this book constitute an excellent reference for engineers and researchers who have particular interests in each of these topics in parallel and distributed computing.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen