Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

MODEL-DRIVEN NETWORK EMULATION WITH VIRTUAL TIME MACHINE

Jason Liu
Raju Rangaswami
Ming Zhao

School of Computing and Information Sciences
Florida International University
Miami, Florida 33199, USA

ABSTRACT

We present VENICE, a project that aims at developing a high-fidelity, high-performance, and highly-controllable experimental platform on commodity computing infrastructure to facilitate innovation in existing and futuristic network systems. VENICE employs a novel model-driven network emulation approach that combines simulation of large-scale network models and virtual-machine-based emulation of real distributed applications. To accurately emulate the target system and meet the computation and communication requirements of its individual elements, VENICE adopts a holistic machine and network virtualization technique, called virtual time machine, in which the time advancement of simulated and emulated components are regulated in complete transparency to the test applications. In this paper, we outline the challenges and solutions to realizing the vision of VENICE.

1 A HARMLESS NECESSARY CAT

We classify existing experimental networking research testbeds into physical, emulation, and simulation testbeds. Emulation testbeds combine physical testbeds and emulation, and provide the ability to conduct extensive *live* experiments. Although emulation allows a certain level of flexibility, the emulated network conditions are inherently restricted by physical limitations (such as nodal processing speed, link bandwidth and latency) and cannot be scaled up beyond the capabilities of the underlying physical infrastructure. Comparatively, network simulation is effective at capturing high-level design issues, answering what-if questions, and investigating complex system behaviors, such as multi-scale interactions, self-organizing characteristics, and emergent phenomena. However, simulation fairs poorly in other aspects, particularly in its operational realism. Further, simulation development is both labor-intensive and error-prone; reproducing realistic network topology, traffic and diverse operational conditions in simulation is known to be a substantial undertaking.

In this position paper, we describe our ongoing efforts on the VENICE project (*Virtual Emulation Network Infrastructure for Commodity Environments*), which aims to combine the advantages of simulation and emulation techniques to provide a high-fidelity, high-performance, and highly-controllable experimental platform for large-scale network experiments. The central thrust of this project is a novel model-driven network emulation approach for conducting large-scale network experiments. We pursue holistic machine and network virtualization techniques that allow execution of network simulation models and emulation of real implementations of network protocols and distributed applications under a coordinated virtual time-line. VENICE consists of simulated networks connecting to emulated hosts sandboxed inside virtual machines running real implementations of network applications or network protocols (e.g., router software). Traffic generated from the applications is conducted by the virtual network with appropriate delays and losses according to both simulated and emulated network conditions.

Our goal with VENICE is to enable network experimental testbeds that adhere to arbitrary, user-defined "virtual" network and machine infrastructure specifications. To do so, we need to address fundamental challenges related to accuracy, scalability, and controllability. To create a highly accurate virtual network testbed, we propose a unified machine and network virtualization scheme, called *Virtual Time Machine (VTM)*, to both allocate compute resources, regulate the advancement of time within simulated and emulated entities, and control the delivery of network events. These capabilities ensure correct emulation of the target system with specific user-specified computation and communication requirements. VTM involves transparently and dynamically dilating or contracting virtual time experienced by the applications running on individual virtual machine relative to the real passage of time according to the model requirement and resource utilization. For scalability, we combine model-driven network emulation and network simulation, in order to also enable the desired balance with accuracy. Simulation provides an efficient representation of network operations (such as packet forwarding) at scale; emulation implements interactions with real applications and provides the realism by conducting traffic among real applications. For controllability, VENICE implements user-centric experiment workflow creation through network scripting, builds mechanisms for on-demand throttling of simulation/emulation speed, and provides support for configurable fault injection.

To achieve its goals, VENICE must address several fundamental challenges, including identifying a consistent time frame across all entities of the system, dynamically allocating resources to various simulated and emulated entities to maximize resource utilization, supporting full integration of simulation and emulation by performing fine-grained time synchronization, incorporating network models at different levels of detail with real implementation of network applications running unmodified on virtual machines, and streamlining experiment workflow for model configuration, deployment, online monitoring and control, data collection, debugging and analysis. Through the rest of this paper we elaborate upon several of these challenges and the corresponding solutions to address these challenges.

2 ALL THAT GLITTERS IS NOT GOLD

In this section, we review existing approaches to network testbeds, virtual machine resource management, and virtual time management.

2.1 Network Testbeds

Experimental network testbeds are commonly used for prototyping, evaluating, and analyzing new network designs and services. There exist three types of network testbeds: physical, emulation, and simulation testbeds. Physical testbeds, such as WAIL (Barford and Landweber 2003) and PlanetLab (Peterson et al. 2002), provide an iconic version of the network for experimental studies, sometimes even with live traffic. They are limited in user controllability. Also, as shared facilities they can be overloaded due to heavy use (e.g., Spring et al. 2006), potentially affecting availability and accuracy. Some of these testbeds also provide emulation capabilities of "shaping" traffic between real applications. Traffic shaping can be implemented at the sender, the receiver, or both. For example, in dummynet (Rizzo 1997), a virtual network link is represented as a queue with specific bandwidth and delay constraints; packets are intercepted at the protocol stack of the sender host and pushed through the finite queue to simulate packet forwarding with proper packet delays and losses.

An emulation testbed can be built on a variety of computing infrastructures. ModelNet (Vahdat et al. 2002) extends dummynet, where a large number of network applications can run unmodified and communicate via a virtual network emulated on parallel computers. EmuLab (White et al. 2002) is an experimentation facility consisting of many networked computers integrated and coordinated via emulation mechanisms to present a diverse virtual network environment. DETER (Benzel et al. 2006) is an experimental infrastructure that extends EmuLab to support research and development of cyber security technologies. VINI (Bavier et al. 2006) is a virtual network infrastructure that enables realistic and controlled network experiment running on top of PlanetLab. By using virtualization technology, researchers are able to create flexible virtual network topologies to test distributed applications and network services.

There are emulation testbeds based on special hardware platforms or computing architectures to either facilitate faster emulations or accommodate complex communication environments. For example, Open Network Laboratory (DeHart et al. 2006) uses embedded processors within router switches that can be configured 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 real wireless transmissions. The CMU Wireless Emulator (Judd and Steenkiste 2004) is a wireless network testbed using Field-Programmable Gate Array (FPGA) to modify wireless signals sent by real wireless devices according to realistic signal propagation models.

Simulation also plays an important role in network design and evaluation. 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; emulation mainly involves artificially delaying and dropping real packets (Rizzo 1997). Simulation testbeds in general allow better flexibility and controllability. For example, ns-2 (Breslau et al. 2000) (also, the current effort in the ns-3 development) provides a rich collection of network algorithms and protocols. Simulation can also scale up to handle large-scale networks (e.g., (Cowie, Nicol, and Ogielski 1999), (Riley 2003), (Yaun et al. 2003), (Bajaj et al. 1999)).

Real-time simulation is a technique that integrates simulation and real implementation of network applications. Most real-time simulators, such as NSE (Fall 1999), IP-TNE (Simmonds and Unger 2003), MaSSF (Liu, Xia, and Chien 2003), Maya (Zhou et al. 2004), and PRIME (Liu 2008), are based on existing network simulators augmented with emulation capabilities. EmuLab also includes real-time simulation based on NSE. Since real applications operate in real time, real-time network simulation must meet real-time requirements. To speed up simulation, one can apply parallel simulation techniques to harness the computing resources of parallel computers; in conjunction, multi-resolution modeling techniques, such as low-resolution fluid models (e.g., (Liu et al. 2004), (Nicol and Yan 2004), (Liu 2006)) can be employed to reduce the computational demand.

2.2 Virtual Machine Resource Management

Virtual machines have found many interesting applications, including data centers, virtual desktop environments, security, and software distribution (Rosenblum and Garfinkel 2005). Recently, researchers also used virtual machines for network emulation testbeds (Gupta et al. 2006), (Bavier et al. 2006), (Maier, Herrscher, and Rothermel 2007), (Bhatia et al. 2008).

Virtual machine solutions come in variations. *Full-virtualization*, such as VMware (`www.vmware.com`) and User-Mode Linux (Dike 2000), provides complete transparency for running unmodified guest operating systems (OSes); but in doing so incurs considerable performance overhead. *Para-virtualization*, e.g., Xen (Dragovic et al. 2003) and Denali (Whitaker, Shaw, and Gribble 2002), offers partial virtualization for greater efficiency, but requires modification of guest OSes. Some techniques only virtualize network resources, such as OpenVZ (`openvz.org`) and VRF (`http://sourceforge.net/projects/linux-vrf`). In these cases, applications are presented with multiple independent network stacks on the same OS.

Resource management in virtualized environments addresses the problem of automatically deciding a VM's resource needs based on its hosted application's workload demand and QoS requirement. Existing solutions are typically based on using queuing theory to construct analytical performance models for applications executed on VMs (Doyle et al. 2003, Bennani and Menascé 2005), applying control theory to automatically adjust VM resource allocation and achieve the desired application performance (Liu et al. 2005), (Wang, Zhu, and Singhal 2005), (Padala et al. 2008), and using machine learning techniques to automatically learn the resource model for a virtualized system based on data observed from the system (Wildstrom, Stone, and Witchel 2008), (Wood et al. 2008), (Bu, Rao, and Xu 2009), (Xu et al. 2008), (Kundu et al. 2010).

2.3 Virtual Time Management

Virtual time management is the central theme of parallel discrete-event simulation (Fujimoto 1990). A critical issue for parallel simulation is to maintain *causality constraint*, which requires that simulation events be processed in non-decreasing timestamp order. Most parallel simulation methods can be classified mainly as either conservative synchronization (Chandy and Misra 1979), which strictly prohibits out-of-order event execution, or optimistic synchronization (Jefferson 1985), which allows events to be processed out of order, but provides mechanisms to roll back erroneous events.

For emulation, virtual time management has been proposed for emulating systems superior to those provided by the underlying physical infrastructure. The principal technique employed is *time dilation*, which controls a system's notion of how time progresses (Gupta et al. 2006). This is achieved by enlarging the interval between timer interrupts by what is called a *time dilation factor (TDF)*, and thus slowing down the system clock, so that applications running on the VMs experience a slower passage of time and consequently observe an upgrade in the system resources. For instance, if a VM has a TDF of 10, it means that time, as perceived by the applications running on the VM, will advance at a pace 10 times slower than real time. Meanwhile, the applications will experience a tenfold increase in CPU processing power and networking speed with the slowed timeframe. As an improvement, Bergstrom, Varadarajan, and Back (2006) proposed a modified time dilation mechanism that includes the ability to warp time to handle periods of inactivity during resource undersubscription.

3 OF WISDOM, GRAVITY, PROFOUND CONCEIT

In this section, we elaborate on the architecture and the key design and implementation aspects of VENICE, with the goal of supporting networking researchers to conduct high-fidelity experiments of existing and futuristic network systems on commodity platforms.

3.1 VENICE Architecture

VENICE is designed to operate on a cluster of physical machines and to provide users with the ability to instantiate an arbitrary target network system. The architecture of VENICE is illustrated in Figure 1.

Each physical node of the compute cluster is treated as a *scaling unit*, which also includes a simulation instance (SIM) responsible for a part of the virtual network. The emulated elements (VEs) are assigned to independent virtual machines so they can run unmodified network applications. VEs interact with the SIM on the same scaling unit for communication.

An important design goal of VENICE is to *accurately* reproduce the behavior of the target system, which includes an precise representation of the target machines and networks and a consistent virtual time across the entire distributed

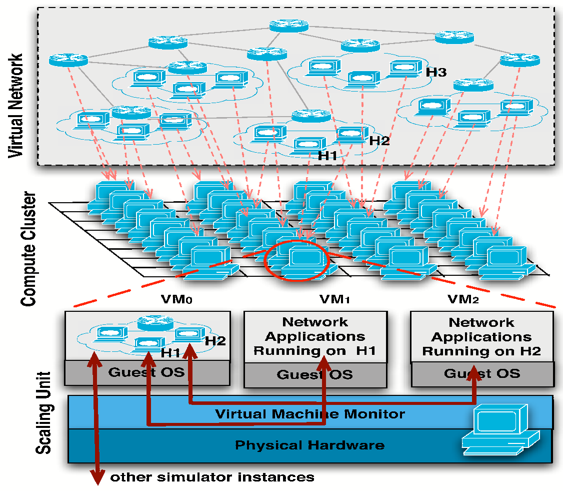

Figure 1: The Overall Architecture of VENICE

infrastructure. This is key for the emulated real applications to run faithfully on VM-based environments, where the modeled resource properties may substantially differ from available physical resources, and to interact correctly with the rest of the simulated and emulated system, which are distributed on VMs across all the participating physical hosts. To achieve these goals, VENICE employs a simulation/emulation co-design, employs time dilation for flexibility in emulating target resource properties using the available physical resources, and optimizes resource utilization using dynamic resource reallocation to simulation/emulation elements and dynamic time dilation. We discuss the major problems and the corresponding solutions in more details in the following sections.

3.2 Simulation/Emulation Co-design

Each physical node within the cluster serves as a basic scaling unit which hosts a simulator instance and a few emulated elements. The simulator simulates a sub-partition of the target network, coordinates with the simulator instances on other scaling units, and serves as the control unit for the virtual machines whereby traffic to and from the emulated elements is conducted by the simulator. Emulated packets (from the real applications) will be processed together with simulated traffic, whereas packet delays and losses will be calculated based on competition for the simulated network resources. Traffic destined to an emulated host or router is sent from the simulator to the corresponding VM and passed on to the application. VENICE consists of both simulation and emulation components that collectively represent the target system for network experimentation.

To illustrate, suppose a web client running on VM_1 sends an HTTP request to a web server running on VM_2. The associated packet will be captured at the virtual network device and sent to the simulation instance running on another virtual machine at the same scaling unit (VM_0 in the example). Subsequently, a simulation packet will be sent from virtual node H_1 to H_2 on the virtual network, with a delay and possible loss calculated subject to the conditions of the virtual network. Once the packet "arrives" at virtual node H_2, the simulator will send a real packet to VM_2. The underlying conduit is concealed from the applications running on VM_1 and VM_2, which experience the same behavior in terms of throughput and latency as if on the virtual network.

3.3 Time Dilation

VENICE must be able to support experiments where the total capacity of the target system is far greater than the capacity of the available physical resources. VENICE uses time dilation on the VMs running the emulated hosts and throttle the virtual clock of the simulator accordingly.

The simulation engines in VENICE use simulation time. The speed with which simulation advances its simulation time is dictated by the simulator's event processing speed. For network simulation, this largely depends on the traffic load. Higher level of network traffic results in more simulation events that need to be processed at a given time, and therefore the slower the simulator appears to be. The passage of time within an emulated element, however, may be dilated or contracted relative to real time in case the emulated resource properties differ from available physical resources. For example, if we need to emulate a gigabit network service on a system that can only offer 100 Mb/s connections between VMs, the time on these VMs may need to be dilated (i.e., slowed down) by at least a factor of 10.

To maintain a consistent time frame across the entire system, VENICE employs a *single* time dilation factor for all simulated and emulated elements. Due to nonuniform system properties and dynamic load characteristics of the target network, using a single time dilation factor for all emulation elements during the entire experiment could lead to arbitrary under-utilization of available physical resources. One can extend the time dilation technique, proposed by Gupta et al. (2006), to allow dynamic control over emulated network performance with specific link bandwidths and latencies. Changing the time dilation factor, however, also changes the perceived CPU speed. To allow independent CPU and network scaling, we apply techniques that can flexibly control the speed of a VM's virtual CPU and network interfaces. Specifically, for the scaling of CPU processing speed, we leverage modern VMM's support for dynamic allocation of CPU resource, which can be fine-tuned at the granularity of CPU cycles (e.g., Dragovic et al. 2003). For the scaling of networking speed, we extend para-virtualization-based I/O virtualization to control a VM's bandwidth on the virtual network. In a para-virtualized VM environment, the inter-communications between concurrently hosted VMs can be realized efficiently through shared memory: network packets can be transferred by exchanging memory pages and I/O interrupts can be delivered by setting shared bitmaps in memory. We can effectively control a VM's bidirectional bandwidth by throttling the delivery rate of the virtual I/O interrupts.

3.4 Virtual Time and Resource Management

VENICE adopts a two-level management scheme to ensure that the virtual time progresses at the same speed across the simulated and emulated elements on the same scaling unit, as well as across the subsets of the virtual network modeled on different scaling units.

3.4.1 Global Management

The goal of global virtual time and resource management is two-fold. In terms of accuracy, the progression of virtual time needs to be consistent across the distributed parallel simulation instances which model different subsets of the virtual network. In terms of performance, the modeling load from the simulation and emulation VMs hosted on each physical server should be balanced across the system so that no single host becomes the bottleneck that slows down the entire system's modeling progress and wastes valuable resources.

We can formulate the problem of finding the appropriate time dilation factor as a resource mapping problem. Given an initial configuration of the physical infrastructure and a user-specified model of the target network system, the goal is to achieve an optimal placement of the emulated and simulated elements. We need to provide an optimal subdivision of the target network among the parallel simulation instances, taking into consideration the overall workload and traffic load of the system.

We describe a simplified problem definition as follows. Consider a target network system $G=(V,E)$, where V is the set of network nodes, and E is the set of links (assuming bidirectional, point-to-point connections here for simplicity). Each link has a bandwidth of $b: E \rightarrow \mathbb{R}$. Assume that, in the user-defined network traffic scenario, each link achieves an average utilization of $\quad : E \rightarrow \mathbb{R}$, which ranges between 0 and 1. The network nodes can be either simulated or emulated. Let $V_E \subseteq V$ be the subset of network nodes that are emulated. Each emulated host, in particular, is configured with a CPU processing power $p: V_E \rightarrow \mathbb{R}$ (in number of instructions per second) and a network capacity of $c: V_E \rightarrow \mathbb{R}$ (in bits per second). Under the same network traffic condition, we derive the network capacity of an emulated host as the sum of its link capacity: $c(v) = \sum_{\forall (v,u) \in E} \quad (v,u)b(v,u)$. Let $\lambda: V_E \rightarrow \mathbb{R}$ denote the average CPU utilization on the emulated host. The virtual CPU's processing power can thus be determined as $\lambda(v)p(v)$. The target network will be distributed across m physical machines. Each physical machine j is configured with a raw CPU processing power of P_j and a network capacity of C_j. Let K be the system-wide time dilation factor we want to get. The problem is to find a partition $\mathscr{P}$ of the network nodes V into m disjoint sets $V_1, \cdots, V_m$, so that the time dilation factor K is minimized, subject to the following conditions:

$$\psi_j(\mathscr{P}) + \sum_{v \in V_j} \lambda(v)p(v)/K \quad \leq \quad P_j \tag{1}$$

$$\sum_{v \in V_j \wedge u \in V \setminus V_j \wedge (v,u) \in E} \quad (v,u)b(v,u)\beta_j/K \quad \leq \quad C_j \tag{2}$$

$$\sum_{v \in V_j \cap V_E \wedge (v,u) \in E} \quad (v,u)b(v,u) \quad \leq \quad \pi_j(\mathscr{P}) \tag{3}$$

for $1 \leq j \leq m$. Equation (1) specifies the constraint on the physical CPU processing power, which is shared among the simulation instance and the time-dilated virtual machines running emulated hosts. $\psi_j(\mathscr{P})$ specifies the amount of work (in terms of number of instructions per second) for simulating the sub-network V_j under the current partitioning scheme. For simplicity, it can be estimated as proportional to the amount of traffic simulated for each (real) second: $\psi_j(\mathscr{P}) = \alpha_j \sum_{\forall v \in V_j \wedge (v,u) \in E} \quad (v,u)b(v,u)/K$, where α_j is a proportion constant. Equation (2) specifies the constraint on the physical bandwidth. The left-hand side specifies the traffic intensity (in terms of bits per second) the simulator uses to conduct traffic between sub-networks across different partitions, which is proportional to the amount of traffic to be sent across the partition on the virtual network (β_j is the proportion constant). Equation (3) specifies the constraint on the simulator's I/O capabilities $\pi_j(\mathscr{P})$, in terms of sending and receiving traffic directly to and from the virtual machines on the same physical machine. This value largely depends on the type of the emulation infrastructure (such as VPN or VLAN) on the physical machine and can be evaluated based on measurements.

This problem can be effectively tackled using graph partitioning to identify partitions that can evenly divide the workload among processors and minimize the cut between the partitions. Previously we proposed a heuristic scheme to transform the network model into a graph (considering only link latencies) and use a graph partitioner (e.g., Karypis and Kumar 1998) to find an optimal partition (Liu and Nicol 2001). Liu and Chien (2003) later proposed a three-stage graph partitioning strategy that takes into account both link latencies and traffic intensity. In their approach, one needs to first run the graph partitioner twice—once with the objective of maximizing the link latency among partitions and once for minimizing the network traffic. Each run produces a network partition and an edge-cut (the number of edges crossing the partitions). The two resulting edge-cuts are then used to derive a graph with new edge weights, calculated from a formula that combines both link latencies and network traffic, and the graph partitioner is run for the third time to obtain a final partition. This algorithm can be extended to a multi-stage multi-constraint partitioning algorithm.

There are several important limitations of the static partitioning technique:

1. The node utilizations $\lambda(v)$ and link utilizations (e) depend upon application traffic and may not be readily known. They may also change during the experiment, sometimes rapidly.
2. The simulator's time complexity $\psi_j(\mathscr{P})$ and communication complexity $\pi_j(\mathscr{P})$ depend on many factors, including the network model (e.g., packet-oriented vs. fluid models), traffic composition (TCP, UDP, etc.), as well as specific simulation design and implementation. Proportionality is a over-simplified approximation.
3. The same dilation factor is applied to all emulated hosts with diverse requirements on resource scaling according to the target network system. Moreover, the CPU processing power and the network capacity may also need to be scaled independently.

To solve the first two problems, we investigate dynamic resource allocation techniques based on measurements of the utilizations and proportionality ratios. We seek a two-level approach. At the global level, dynamic resource management can be achieved through remapping and migration of virtual machines (albeit with substantial overhead). At the local level, we utilize fine-grained resource management and VM time-slice scheduling. The third problem can be solved using local resource control mechanisms, which we describe next.

3.4.2 Local Management

The goal of local virtual time and resource management is also two-fold. For accuracy, each VM-based emulator needs to be provisioned with the proper amounts of virtual resources to truthfully represent the hardware configuration required by the network application, and the concurrently hosted emulators need to be synchronized with the simulator so that network events are delivered in order as they should. For performance, resources need to be allocated to the simulation and emulation VMs based on their workload demands so that they are not over-provisioned to any VM and no single VM becomes the bottleneck to slow down the entire physical host's modeling progress. Although the workload for the simulator and the emulated hosts changes over time, the virtual time progression must be consistent across the entire system. Static resource allocation mechanisms can be extremely inefficient. To address this challenge, we investigate two solutions, one driven by load and the other driven by deadline.

The load-driven resource management solution is based on online prediction of VM resource demand and dynamic adjustment of resource allocation and time dilation factor. With this approach, the local resource manager can continuously monitor a VM's resource consumption and apply forecasting techniques to predict its future resource needs. Based on the resource estimation from all the concurrent VMs, it is able to determine whether the available resources on the physical host are sufficient to satisfy all the resource demands. If so, it will allocate the resources to the VMs according to the predicted values; otherwise, it will trigger an increase of TDF. This whole process is performed periodically in order to capture the latest trend of a VM's resource usage and correct the possible errors in the prediction. The key to the success of this approach is the forecasting method that can predict a VM's resource usage patterns. We can apply a fuzzy-logic based forecasting method, which can effectively handle the dynamics and uncertainties in a VM's resource usage behaviors and accurately maps the observations of its recent resource usage to its future resource needs (Xu et al. 2008).

The deadline-driven resource management solution is based on fine-grained VM scheduling to satisfy the VM resource needs and meet the virtual time progression target. With this approach, the local resource manager can periodically calculate the deadline for all the concurrent VMs in real time for the progression of a unit of virtual time using the current TDF. Based on this deadline, the task of resource management is to schedule a VM with the necessary resources so that it can make the required progress on its virtual time before the deadline arrives. The scheduling can be done in a fashion similar to the Earliest Deadline First algorithm (Stankovic et al. 1998), but using the distance between a VM's current virtual time and the target virtual time to determine its scheduling priority. If the amount of work that a VM needs to process in the next period of time is known or can be predicted, then a schedulability test can be performed in advance to determine whether the available resources on the physical host are sufficient to meet this deadline. If such a test is not possible, then the insufficiency of resources will reflect on the deadline misses of the VMs. Either way, when the deadline cannot be met by all the VMs, it will trigger the increase of TDF to slow down the modeling speed. Compared to the above load-driven solution, this approach can operate at a finer granularity to provide a tighter synchronization with the progression of virtual time across all VMs; but the approach introduces more overhead from frequent VM preemptions and context switches.

4 LET IT SERVE FOR TABLE TALK

We conclude this paper with a discussion of several important aspects of the VENICE approach that fundamentally differentiate it from other network emulation testbeds.

Live versus synthetic traffic. The ability to perform live experiments is a major attraction for systems like PlanetLab and GENI (`www.geni.net`), and is unparalleled by other methods. Conducting tests under real network traffic can reveal both design and operational problems before the ultimate deployment. However, the ability of live testing sacrifices controllability. After all, one cannot control live traffic to reveal every potential problem that the target system would encounter under all kinds of traffic conditions. VENICE uses network simulation to model network transactions and therefore can easily incorporate synthetic traffic models. Synthetic traffic cannot replace live traffic, but it provides a controllable diversity. For example, synthetic traffic can be used to represent various congestion levels of the network queues. It is possible to run VENICE on distributed platforms (such as an overlay network) so that it can incorporate live traffic situations. It is important, however, to accurately integrate live traffic with emulation traffic from time-dilated systems.

Common versus special devices. Simulated packet forwarding is faster than explicit I/O operations, but is slower in terms of processing packets than on specialized hardware architectures, such as FPGAs and network processors (e.g., DeHart et al. 2006). VENICE uses time dilation to proportionally increase the perceived networking speed to compensate for this deficiency and to accommodate future network capabilities. It is possible to integrate model-driven network emulation with other emulation testbeds supported by special devices for sake of efficiency.

Commodity versus shared platforms. The VENICE's approach to increasing availability of the network testbed is to adopt commodity platforms (so that everybody would be able to run the software on existing computing systems). This is in contrast to the philosophy behind PlanetLab, EmuLab, and GENI, where resource sharing is of critical importance. Virtualization techniques are used in the latter case primarily to reduce interference between experiments sharing the same hardware platform. In VENICE, virtualization techniques are used primarily to represent the target system independent of the underlying execution environment, and to incorporate virtual time management according to resource utilization and time synchronization. VENICE complements and strengthens existing network testbed approaches; it can help smooth the transition from controlled studies (with diverse execution and communication scenarios) to live experimentation and deployment (e.g., on the GENI platform).

ACKNOWLEDGMENTS

This research is supported in part by the National Science Foundation grants CNS-0836408, CNS-0821345, CCF-0937964, and HRD-0833093.

REFERENCES

Bajaj, L., M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. 1999. GloMoSim: a scalable network simulation environment. Technical Report 990027, Department of Computer Science, UCLA.

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

Bavier, A., N. Feamster, M. Huang, L. Peterson, and J. Rexford. 2006. In VINI veritas: realistic and controlled network experimentation. In *Proceedings of the 2006 ACM SIGCOMM Conference*, 3–14.

Bennani, M. N., and D. A. Menascé. 2005. Resource allocation for autonomic data centers using analytic performance models. In *Proceedings of the International Conference on Autonomic Computing (ICAC'05)*, 229–240.

Benzel, T., B. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. 2006. Experience with DETER: A testbed for security research. In *Proceedings of the 2nd IEEE Conference on testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM'06)*.

Bergstrom, C., S. Varadarajan, and G. Back. 2006. The distributed open network emulator: Using relativistic time for distributed scalable simulation. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*.

Bhatia, S., M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. 2008. Hosting virtual networks on commodity hardware. Technical Report GT-CS-07-10, Georgia Tech, Department of Computer Science.

Breslau, L., D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. 2000. Advances in network simulation. *IEEE Computer* 33 (5): 59–67.

Bu, X., J. Rao, and C.-Z. Xu. 2009. A reinforcement learning approach to online web systems auto-configuration. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems (ICDCS'09)*, 2–11.

Chandy, K. M., and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* SE-5 (5): 440–452.

Cowie, J. H., D. M. Nicol, and A. T. Ogielski. 1999. Modeling the global Internet. *Computing in Science and Engineering* 1 (1): 42–50.

DeHart, J., F. Kuhns, J. Parwatikar, J. Turner, C. Wiseman, and K. Wong. 2006. The open network laboratory. *ACM SIGCSE Bulletin* 38 (1): 107–111.

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

Doyle, R. P., J. S. Chase, O. M. Asad, W. Jin, and A. Vahdat. 2003. Model-based resource provisioning in a web service utility. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*.

Dragovic, B., K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. 2003. Xen and the art of virtualization. *Proceedings of the ACM Symposium on Operating Systems Principles*.

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

Fujimoto, R. M. 1990. Parallel discrete event simulation. *Communications of the ACM* 33 (10): 30–53.

Gupta, D., K. Yocum, M. McNett, A. Snoeren, A. Vahdat, and G. Voelker. 2006. To infinity and beyond: time-warped network emulation. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI'06)*.

Jefferson, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7 (3): 404–425.

Judd, G., and P. Steenkiste. 2004. Repeatable and realistic wireless experimentation through physical emulation. *ACM SIGCOMM Computer Communication Review* 34 (1): 63–68.

Karypis, G., and V. Kumar. 1998. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0. http://glaros.dtc.umn.edu/gkhome/views/metis/.

Kundu, S., R. Rangaswami, K. Dutta, and M. Zhao. 2010. Application performance modeling in a virtualized environment. In *16th IEEE International Symposium on High-Performance Computer Architecture (HPCA-16)*.

Liu, J. 2006. Packet-level integration of fluid TCP models in real-time network simulation. In *Proceedings of the 2006 Winter Simulation Conference (WSC'06)*, 2162–2169.

Liu, J. 2008. A primer for real-time simulation of large-scale networks. In *Proceedings of the 41st Annual Simulation Symposium*, 85–94.

Liu, J., and D. M. Nicol. 2001. Learning not to share. In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation (PADS'01)*, 46–55.

Liu, X., and A. A. Chien. 2003. Traffic-based load balance for scalable network emulation. In *Proceedings of the 2003 Supercomputing Conference (SC'2003)*.

Liu, X., H. Xia, and A. A. Chien. 2003. Network emulation tools for modeling grid behavior. In *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03)*.

Liu, X., X. Zhu, S. Singhal, and M. F. Arlitt. 2005. Adaptive entitlement control of resource containers on shared servers. In *Integrated Network Management*, 163–176.

Liu, Y., F. L. Presti, V. Misra, D. F. Towsley, and Y. Gu. 2004. Scalable fluid models and simulations for large-scale IP networks. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 14 (3): 305–324.

Maier, S., D. Herrscher, and K. Rothermel. 2007. Experiences with node virtualization for scalable network emulation. *Computer Communications* 30 (5): 943–956.

Nicol, D. M., and G. Yan. 2004, July. Discrete event fluid modeling of background TCP traffic. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 14 (3): 211–250.

Padala, P., K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. 2008. Automated control of multiple virtualized resources. Technical Report HPL-2008-123R1, Hewlett Packard Laboratories.

Peterson, L., T. Anderson, D. Culler, and T. Roscoe. 2002. A blueprint for introducing disruptive technology into the Internet. In *Proceedings of the 1st Workshop on Hot Topics in Networking (HotNets-I)*.

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

Riley, G. F. 2003, August. The Georgia Tech network simulator. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research (MoMeTools'03)*, 5–12.

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

Rosenblum, M., and T. Garfinkel. 2005. Virtual machine moniors: Current technology and future trends. *IEEE Computer* 38 (5).

Simmonds, R., and B. W. Unger. 2003. Towards scalable network emulation. *Computer Communications* 26 (3): 264–277.

Spring, N., L. Peterson, A. Bavier, and V. Pai. 2006. Using PlanetLab for network research: myths, realities, and best practices. *ACM SIGOPS Operating Systems Review* 40 (1): 17–24.

Stankovic, J. A., M. Spuri, K. Ramamritham, and G. C. Buttazzo. 1998. *Deadline scheduling for real-time systems - EDF and related algorithms*. Kluwer Academic Publishers.

Vahdat, A., K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. 2002. Scalability and accuracy in a large scale network emulator. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*.

Wang, Z., X. Zhu, and S. Singhal. 2005. Utilization and SLO-based control for dynamic sizing of resource partitions. In *Proceedings of the 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'05)*, 133–144.

Whitaker, A., M. Shaw, and S. Gribble. 2002, June. Denali: Lightweight virtual machines for distributed and networked applications. In *Proceedings of the USENIX Annual Technical Conference*.

White, B., J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. 2002. An integrated experimental environment for distributed systems and networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, 255–270.

Wildstrom, J., P. Stone, and E. Witchel. 2008. Carve: A cognitive agent for resource value estimation. In *Proceedings of the International Conference on Autonomic Computing (ICAC'08)*, 182–191.

Wood, T., L. Cherkasova, K. M. Ozonat, and P. J. Shenoy. 2008. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Middleware Conference (Middleware'08)*, 366–387.

Xu, J., M. Zhao, J. A. B. Fortes, R. Carpenter, and M. S. Yousif. 2008. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing* 11 (3): 213–227.

Yaun, G. R., D. Bauer, H. L. Bhutada, C. D. Carothers, M. Yuksel, and S. Kalyanaraman. 2003. Large-scale network simulation techniques: examples of TCP and OSPF models. *ACM SIGCOMM Computer Communication Review* 33 (3): 27–41.

Zhou, J., Z. Ji, M. Takai, and R. Bagrodia. 2004. MAYA: integrating hybrid network modeling to the physical world. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 14 (2): 149–169.

AUTHOR BIOGRAPHIES

JASON LIU is an Assistant Professor at the School of Computing and Information Sciences, Florida International University. His research focuses on parallel simulation and high-performance modeling of computer systems and communication networks. He received a B.A. degree from Beijing Polytechnic University in China in 1993, an M.S. degree from College of William and Mary in 2000, and a Ph.D. degree in from Dartmouth College in 2003. He served as a WSC proceedings editor in 2006. His email address is <liux@cis.fiu.edu>.

RAJU RANGASWAMI is an Associate Professor of Computer Science at the Florida International University in Miami. He received a B.Tech. degree in Computer Science from the Indian Institute of Technology, Kharagpur, India. He obtained M.S. and Ph.D. degrees in Computer Science from the University of California at Santa Barbara. His research interests include operating systems, storage systems, virtualization, security, and real-time systems. His email address is <raju@cis.fiu.edu>.

MING ZHAO is an Assistant Professor at the School of Computing and Information Sciences, Florida International University. His research interests include virtualization, operating systems, distributed computing, high-performance computing, and autonomic computing. He received his B.S. and M.S. degrees from Tsinghua University, Beijing, China. He obtained his Ph.D. degree in Electrical and Computer Engineering from the University of Florida. His email address is <ming@cis.fiu.edu>.