Symbiotic Network Simulation and Emulation

MIGUEL A. ERAZO, RONG RONG, and JASON LIU, Florida International University

A testbed capable of representing detailed operations of complex applications under diverse network conditions is invaluable for understanding the design and performance of new protocols and applications before their real deployment. We introduce a novel method that combines high-performance large-scale network simulation and high-fidelity network emulation, and thus enables real instances of network applications and protocols to run in real operating environments and be tested under simulated network settings. Using our approach, network simulation and emulation can form a symbiotic relationship, through which they are synchronized for an accurate representation of the network-scale traffic behavior. We introduce a model downscaling method along with an efficient queuing model and a traffic reproduction technique, which can significantly reduce the synchronization overhead and improve accuracy. We validate our approach with extensive experiments via simulation and with a real-system implementation. We also present a case study using our approach to evaluate a multipath data transport protocol.

Categories and Subject Descriptors: I.6.3 [**Simulation and Modeling**]: Applications; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms: Algorithms, Performance, Experimentation

Additional Key Words and Phrases: Network simulation, network emulation, symbiotic simulation, DDDAS, online simulation.

ACM Reference Format:
M. A. Erazo, R. Rong, and J. Liu, 2014. Symbiotic network simulation and emulation. *ACM Trans. Model. Comput. Simul.* X, Y, Article Z (2015), 25 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The ability to conduct high-fidelity high-performance network experiments is crucial for studying future network systems and their complex behaviors. Existing network testbeds offer different capabilities in terms of providing *controllability*, for creating diverse network scenarios, *scalability*, for capturing large-scale network operations, *realism*, for reproducing important system and network effects, and *performance*, for supporting high-throughput high-capacity data transport.

— *Physical testbeds* (such as PlanetLab [Peterson et al. 2002] and WAIL [Barford and Landweber 2003]) provide a realistic operational environment for testing network applications. They can directly test the applications *in-situ* with the needed operational realism and with live network traffic. However, physical testbeds lack controllability; it is difficult, if not impossible, to test applications not supported by the prescribed setup of the physical environment. They are also limited in scale, which makes it

This research is supported in part by NSF grants CNS-0836408, CCF-0937964, and HRD-0833093, and by a subcontract from the GENI Project Office at Raytheon BBN Technologies (CNS-1346688).
Author's addresses: M. A. Erazo, (current) Amazon.com, 1800 9th Ave, Seattle, WA 98101; R. Rong, and J. Liu, School of Computer and Information Science, Florida International University, Miami, Florida 33199.

DOI: http://dx.doi.org/10.1145/0000000.0000000

impractical for studying certain important aspects, such as application scaling and robustness issues against diverse environments, and asking what-if questions.
— *Simulation* (e.g., NS-2 [Breslau et al. 2000] and OPNET [Chang 1999]) can be effective at capturing overall design aspects, answering what-if questions, and revealing complex system characteristics, such as multi-scale interactions, self-organizing behaviors, and emergent phenomena. Parallel simulation has also demonstrated its capability of dealing with large-scale detailed models by harnessing the collective power of parallel computing platforms. However, simulation often lacks a certain level of realism—reproducing realistic network traffic and operational conditions in simulation is labor-intensive and error-prone.
— *Emulation* (e.g., ModelNet [Vahdat et al. 2002] and EmuLab [White et al. 2002]) provides a good balance between controllability and accuracy, whereas real applications can run directly in a native operating environment. However, like physical testbeds, its scale and capability is limited by the physical limitations of the underlying platform, such as the processing power, and the network bandwidth and latency.

A testbed capable of performing large-scale experiments, providing diverse network scenarios and network conditions, maintaining accurate representation of the operation of the target applications, and supporting high-throughput high-capacity network transactions remains elusive.

Broadly defined, simulation and emulation differ in the scope of the network functions being examined. Simulation consists of software modules necessary for representing the network elements (such as links, switches, and end hosts) and the transactions between them (such as packets and protocols). In contrast, emulation provides a runtime environment for conducting network experiments with unmodified applications running on virtual machines or with virtual network stacks, and interacting with real operating system interfaces and libraries. Only the network traffic between the applications is modulated to represent the target network conditions. We observe that both simulation and emulation provide a good level of controllability and reproducibility: one can specify the detailed configuration of the target network (such as the network topology, and bandwidth and delay of individual links) relatively easily, and conduct simulation or emulation experiments in a repeated fashion. However, their expected capabilities differ significantly.

In general, simulation is used to construct high-level network models with protocols that may not be fully developed (such as those at the physical and link layers). As such, simulation is desirable for obtaining "the big picture", which is especially valuable when a complete understanding of the system's complex behavior is absent. Simulation offers good flexibility and scalability, but may not provide the necessary accuracy for describing detailed behavior of the network or the execution environment. In contrast, emulation allows one to execute unmodified applications directly on a real system, which accepts application data as input and produces detailed responses as output. It provides the operational realism, but may not be able to handle all elements of a large-scale network experiment due to resource constraints. It is also more difficult to set up an emulation environment capable of representing diverse network topologies and arbitrary traffic conditions.

To allow high-fidelity high-performance large-scale experiments, in this article, we propose a method to combine both simulation and emulation. We use *the simulation system* to run the full-scale network model in real time with detailed network topology and protocols for a close representation of a target network. To simulate potentially large-scale network systems, we also adopt parallel simulation and advanced traffic modeling techniques. We use *the emulation system* to inspect the detailed behavior of the real applications. We select a number of nodes in the target network as "emulated"

and run unmodified software directly on specified operating systems, with real network stack, libraries and software tools.

In our approach, simulation and emulation forms a symbiotic relationship through which each can benefit from the other. Both systems evolve in real time. The simulation system benefits from the emulation system by incorporating real network traffic generated by the unmodified software running on the real platforms. The emulation system benefits from the simulation system by receiving up-to-date information of the global network behavior and traffic conditions, and using it to calibrate communication between the real applications. As a result, the symbiotic approach allows us to test and analyze applications by *embedding* them seamlessly in target virtual networks with diverse network conditions.

In this article, we present the symbiotic approach that exploits the mutually beneficial relationship between simulation and emulation. Specific contributions of this paper can be summarized as follows:

— We introduce a model-downscaling technique that can significantly reduce the complexity of large-scale network models, in terms of the number of modeling elements (e.g., network nodes, links, and queues) needed for accurately representing the flows and their interactions. In doing so, we are able to improve the computational efficiency of the emulation system, and reduce the synchronization overhead between the simulation and emulation systems, both operating in real time.
— We propose a queuing model for the downscaled emulation system that can efficiently represent the transient behavior of a large-scale simulated network. In doing so, it can accurately capture the interaction between the emulated applications and the simulated traffic at the network queues.
— We propose a technique for reducing the synchronization overhead for the emulation system to update the simulation system on the state of the emulated traffic. Rather than measuring and transferring detailed device-level statistics, we propose to collect the traffic demand at the transport layer in emulation and use the same transport-layer protocols implemented in simulation to reproduce the flows.

The rest of this article proceeds as follows. In Section 2, we present an overview of our symbiotic approach. In the next three sections, we introduce specific techniques of our symbiotic simulation and emulation approach. More specifically, in Section 3, we present the model-downscaling technique to achieve effective emulation. In Section 4, we present the queuing model for capturing the transient behavior of simulated network traffic on emulated applications. In Section 5, we introduce the technique for reproducing the emulated traffic in simulation. We conducted extensive experiments to validate our approach. In Section 6, we describe experiments for validating the queuing model using a simulator of the symbiotic system. In Section 7, we describe a prototype implementation of the symbiotic system and the real-system validation results. We also present a test case of a multipath data transport protocol showcasing the utility of our system in Section 8. Finally, we describe related work in Section 9 and provide our conclusions in Section 10.

2. THE SYMBIOTIC APPROACH

A network experiment consists of a target network network with a detailed specification of the network topology, potentially connecting a large number of hosts and routers running various network protocols and applications. Directly running the applications and protocols on real machines brings at least two advantages. First, implementing complex distributed applications and network protocols in simulation can be difficult. As such, the modelers typically implement in simulation only a subset of the functions of the applications considered as essential. In certain cases, however, it is not imme-

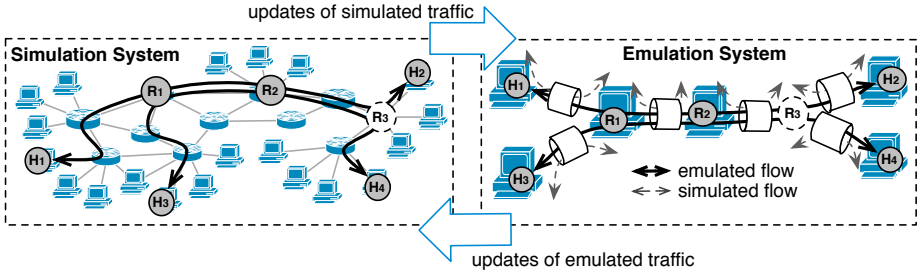

Fig. 1: The symbiotic approach.

diately clear which part of the applications shall be rendered in simulation with great accuracy. Second, directly running the applications in real systems offers a level of operational realism unavailable in simulation; for example, it is often impossible to investigate software configuration issues, system dependencies, deployment constraints, and other system-related artifacts in simulation. Using our symbiotic approach, while it is expected that most of these protocols and applications are simulated, some hosts and routers can be real—they can run unmodified applications and real instances of network protocols on real machines.

Let us first standardize the terminologies used to describe our approach. Specifically, we call the network that one investigates in the experiment as the *target virtual network*, and we call the applications that are expected to run on the target virtual network as the *target applications*. To differentiate from the simulated components, we name the real machines that run the target applications as the *emulated hosts* or *emulated routers*, and we name the traffic between the emulated hosts and routers as either *emulated traffic* or *emulated flows*. Note that the emulated components, including the emulated hosts, routers, and emulated flows, also need to be represented in simulation. One should be able to distinguish the physical realization of the emulated components from their simulated representation easily from the context.

Figure 1 (the top portion) shows an example of a target virtual network consisting of both simulated and emulated components. In particular, the network contains four emulated hosts, H_1 to H_4, and two emulated routers, R_1 and R_2, all marked with solid circles. The rest of the target virtual network will be simulated. The emulated hosts and routers will be instantiated and run as individual machines (either physical machines or virtual machines) in the emulation system. The emulated hosts will run target applications, such as web clients, web servers, and peer-to-peer applications. The emulated routers will typically run routing software as target applications (e.g., [Handley et al. 2005; Kohler et al. 2000; Open vSwitch 2013]).

During the experiment, the target applications at the emulated hosts and routers may engage in communication with one another over the simulated network. As shown in this example, there are two emulated flows, one between H_1 and H_2, and the other between H_3 and H_4. Note that the two flows digress at R_1 and R_3, where R_1 is an emulated router and R_3 is a simulated router. Since the emulated flows are mixed with the simulated ones, one must be able to accurately capture the effect of the simulated flows on the emulated flows, and vice versa.

Figure 1 presents a schematic view of the architecture of the symbiotic system, which consists of a simulation system and an emulation system in a closed loop. The simulation system runs the full-scale network model with a detailed specification of the network topology, traffic and protocols. The network model can be partitioned and mapped onto a parallel computing platform for parallel simulation in order to represent detailed transactions of a large-scale network in real time. The emulation system

consists of a set of emulated hosts and routers as physical or virtual machines that run the target applications. The emulated hosts and routers are connected via a set of active "pipes", which are software constructs responsible for conducting and modulating the real network traffic between the emulated hosts and routers so that the packets probabilistically experiences the same delays and losses as if the target applications were directly connected by a real full-scale network.

The symbiotic approach allows us to run real network applications and protocols directly in the real machine environments. We can test these target applications by embedding them in a large-scale network setting, and evaluating them using diverse network configurations and traffic conditions created by simulation. Of course, the effectiveness of the symbiotic approach lies in its ability to efficiently and accurately modulate the emulated traffic in accordance with the simulated network conditions. Due to the interdependency of the simulation and emulation systems, the two systems must be effectively synchronized.

More specifically, we need to address two issues related to the synchronization of the simulation system and the emulation system. One issue is that the two systems must be time synchronized so that they can communicate using the same timeframe. This problem has been dealt with before using real-time simulation [Liu 2008]. The same technique can also be applied to parallel simulation by augmenting the simulator with functions that regulate the clock advancement in logical processes in accordance with the wall-clock time [Liu 2013].

The other issue is that the two systems need to communicate so that in combination they can represent the true state of the target virtual network. That is, the two systems must effectively exchange their state. One existing approach is to directly "inject" the network packets generated by the target applications into the simulator. That is, we generate a simulation event to represent each packet that traverses a simulated link. Similarly, when a simulated packet reaches an emulated host or router, a real packet must be created in the emulation system. Although this approach has been demonstrated capable of achieving accurate results, the overhead is significant. As such, the throughput of the the emulated flows is bounded by the I/O capacity of the connection between the two systems, which can severely compromise the accuracy of the system, especially when the emulated traffic load approach the capacity limitation.

We note that the emulated traffic (i.e., the flows of real packets sent between the target applications running on the real machines) will affect the simulated traffic (i.e., the traffic between the network entities in the simulation model). The reverse is also true. Both types of traffic compete for the network resources, including the buffer space at the network queues and the bandwidth of the communication links. For example, a sudden increase in emulated traffic may can cause congestion to happen at a link in the target virtual network. When it happens, it can affect (or simply "strangle") all traffic traversing the same link (including both simulated and emulated flows) because of congestion control, which in turn would affect other parts of the target virtual network traversed by these flows, and therefore cause a ripple effect.

We observe, however, unless the traffic is between a simulated host and an emulated host[1], both systems only need to consider the reciprocal effect of the simulated and emulated traffic—in this case, there is no need to exchange network packets between the two systems! The simulation system only needs to update the emulation system with the *effects* of the simulated traffic at the corresponding pipes. Similarly, the emulation system only needs to update the simulation system about the state of the emulated

[1]The case for interactive simulation, which allows directly exchanging traffic between emulated hosts and simulated ones, has been explored in our previous studies, e.g., [Erazo et al. 2009] and [Liu et al. 2009].

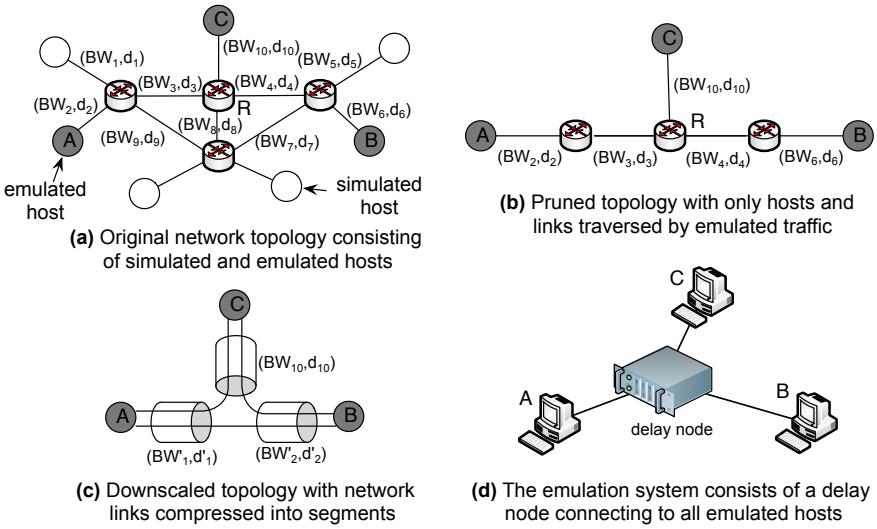

Fig. 2: An example using the model downscaling method.

traffic between the target applications. In subsequent sections, we elaborate on the individual challenges and solutions of our symbiotic approach.

3. EMULATION MODEL DOWNSCALING

As mentioned previously, the symbiotic approach aims to address the discrepancy in capabilities between simulation and emulation by combining them, using simulation to handle network models at scale and using emulation to directly test target applications. Therefore, one does not need to initiate the full-scale network model in the emulation system as long as the same network experience can be rendered for the target applications when they communicate with one another. In this section, we present a model downscaling method for reducing emulation complexity.

The downscaled model needs to be functionally equivalent to the full-scale simulated network model in terms of determining the end-to-end packet delays and packet losses. More specifically, the downscaled model must be able to capture two important network effects:

(1) *The cross-traffic effect*. The full-scale network will be populated with traffic originated from both simulated and emulated hosts. When calculating the end-to-end delays and packet losses between a pair of emulated hosts, we need to consider the effect of other traffic traversing the same segment of network links.
(2) *The multi-bottleneck effect*. The emulated traffic may traverse several intermediate routers where congestion can happen. The effect of multiple bottlenecks is location-dependent and time-varying, and may be highly dependent upon the simulated network conditions.

Our model downscaling method consists of two steps. Figure 2 shows an example with a simple network topology (with user-defined link bandwidths and delays) to illustrate the steps taken by our method. In the first step, the algorithm takes the original network topology as input and prunes the topology by removing the network nodes and links not traversed by emulated flows. The simulated hosts and routers are not instantiated in the emulation system; therefore, the hosts and links that carry only simulated traffic are not needed in emulation for modulating emulated traffic. Note

that this step assumes that network routing can be determined statically. This can be achieved, for example, via "spherical routing", which calculates static network forwarding tables before simulation starts, according to either shortest paths or simple policy specifications [Van Vorst et al. 2011b]. In cases where one needs to simulate detailed routing protocols to handle dynamic routing behaviors, we have to be conservative and instead preserve all those links that can potentially be visited by emulated flows. The pruning here is for efficiency; it does not affect correctness.

The network topology can be further pruned if we have information *a priori* about the communication pattern between the emulated hosts. In particular, if we know that two emulated hosts will *never* contact each other during the experiment, we do not need to maintain the path between the two emulated hosts. In the worst case, we can assume all emulated hosts are capable of contacting all other emulated hosts. Figure 2(a) shows the original model, where A, B, and C are emulated hosts, and all other nodes are simulated. Figure 2(b) shows the pruned topology, only with the emulated hosts and links potentially carrying emulated traffic between them.

In the second step, the algorithm takes the pruned topology as input and compresses the set of links traversed by the same emulated flows into one network segment. We will use a single entity in emulation to represent a network segment, because the emulated flows traversing the same segment would experience similar network conditions: they visit the same set of queues with similar queue lengths, share the same available bandwidths, and interact with the same set of simulated flows. Figure 2(c) shows the results of the path compression step. For example, the network path between the two emulated hosts A and B consists of two segments—one between host A and router R, and the other between router R and host B—since the two segments carry different emulated flows[2].

The resulting downscaled topology will be mapped onto the emulation system. The emulated hosts and routers will be instantiated on individual machines and run designated target applications. The network segments will be represented as "pipes" on a separate delay node, as shown in Figure 2(d). The emulated packets flowing through these pipes can be dropped or added with artificial delays to reflect the simulated network conditions. The state of the pipes will be updated constantly in real time by the simulation system using the statistics collected at the network interfaces corresponding to the segments. In the next section, we describe the queuing model through which we can efficiently update the state of the pipes and ensure that the downscaled emulation model can accurately represent the simulated network conditions.

4. EMULATING NETWORK PATHS

A network segment consists of one or more links traversed by the same set of emulated flows. Here, we propose a queuing model to estimate the packet delays and packet losses for the emulated flows traversing the same network segment. In particular, we use a single M/D/1 queue to model the network effect at a segment in the emulation system. We adopt the M/D/1 queuing model because it is simple and easy to manipulate in a closed form. We later show empirically that the model is not limited to Poisson arrivals; it can produce good results for almost all types of traffic we have experimented with so far.

In our model, the service rate and the drop probability of the M/D/1 queue are calibrated periodically using simple measurements from simulation, in order to calculate the *effect* of the simulated traffic at the corresponding segments. It has been ob-

[2]For the sake of simplicity, the example treats the path as bidirectional; in reality, there needs to be a separate path for each direction (which is not necessarily traversing the same set of nodes), and each path needs to be compressed separately.

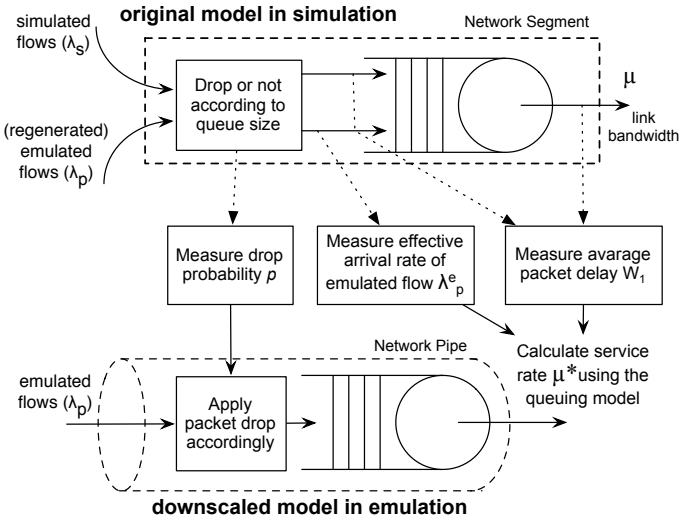

Fig. 3: Queuing model for a single-link segment.

served that network traffic characteristics change insignificantly within small time intervals. For example, it is safe to assume that the packet loss rate, the packet delay, and the throughput would remain relatively constant for at least one minute [Zhang and Duffield 2001]. Consequently, one can expect that our queuing model can be applied at regular time intervals large enough to overcome the synchronization overhead between the simulation and emulation systems.

In the discussion to follow, we first deal with the situation that the network segment consists of only one link; we develop a closed-form solution only for steady state. We then extend the model to deal with multi-link segments. After that, we complete the model by capturing the transient behavior of both network segment types.

4.1. Steady-State Queuing Model for Single-Link Segments

Figure 3 depicts our queuing model for a segment with only one link. In simulation, a segment is traversed by both simulated flows and emulated flows. The emulated flows are regenerated in simulation; we discuss its method in the next section.

Let λ_s be the arrival rate of all simulated flows entering the segment, and λ_p be the arrival rate of all emulated flows entering the segment. Let μ be the link bandwidth. Upon a packet arriving at a network queue (at a network interface), if the buffer is full, the packet will be dropped. Let p be the drop probability due to buffer overflow. Let λ_s^e and λ_p^e be effective arrival rate of the simulated flows and emulated flows, respectively. We have $\lambda_s^e = \lambda_s(1-p)$ and $\lambda_p^e = \lambda_p(1-p)$. We use W_1 to denote the average packet delay (including both simulated and emulated traffic) through the segment, which can be measured easily in simulation.

A segment in simulation is represented in emulation as a pipe and is modeled as an M/D/1 queue (with infinite buffer). We drop packets with probability p before they enter the queue. In emulation, we have only emulated flows. We aim to set the service rate of the queue, μ^*, so that, once an emulated packet enters the queue, it will experience the same delay as it would in simulation.

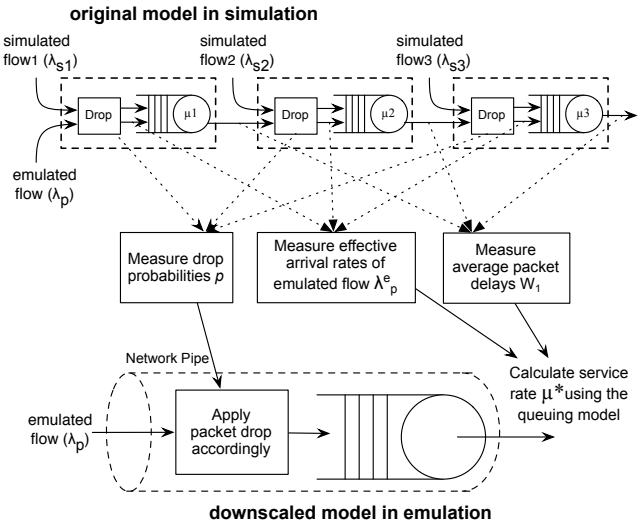

Fig. 4: Queuing model for a multi-link segment.

For M/D/1, the average number of jobs in the system, L, can be obtained in a closed form:

$$L = \rho + \frac{\rho^2}{2(1-\rho)} \tag{1}$$

where ρ is the server utilization. We can then calculate W_2, the average time a job spent in the system:

$$W_2 = \frac{L}{\lambda_p^e} = \frac{\rho + \frac{\rho^2}{2(1-\rho)}}{\lambda_p^e} \tag{2}$$

We set $W_1 = W_2$, since it is expected that the original model and the downscaled model should generate the same average packet delay. We know that $\rho = \lambda_p^e/\mu^*$. We can calculate the service rate of the queue:

$$\mu^* = \frac{\lambda_p^e}{1 + W_1\lambda_p^e - \sqrt{1 + W_1^2(\lambda_p^e)^2}} \tag{3}$$

Note that μ^* is simply a function of the average packet delay (W_1) and the effective arrival rate of the emulated flows (λ_p^e), both of which can be obtained easily from simulation. We also measure the packet loss probability in simulation, and impose the same probability to drop packets before they enter the queue.

4.2. Steady-State Queuing Model for Multi-Link Segments

The previous queuing model can be readily extended to represent a network segment with multiple links. Figure 4 depicts our method with a segment consisted of three links. We measure the drop probability, the effective arrival rate, and the average packet delays of all emulated flows for each link.

We can still use Equation (3) to calculate the service rate. In this case, W_1 is the average delay of packets traversing the whole segment, and λ_p^e is the minimum of the

effective arrival rates at all network interfaces (i.e., the effective arrival rate at the bottleneck link). It is the same as the throughput achieved by the emulated traffic. We can calculate the overall packet drop probability as: $p = (1 - \prod_{i=1}^{n}(1 - p_i))$, where p_i is the drop probability at the i^{th} link, assuming that the segment consists of n links.

4.3. Handling Transient Behavior

The steady state solution yields good results when the network congestion level is low. However, the model fails to predict accurate packet delays when congestions occur. This is because the model represents the averaged long-term behavior when the physical system reaches the steady state for each the update interval. This could be far from the truth if the congestion level is high when the packets experience significant queuing delays.

It is a non-trivial task to find a general closed-form solution for the transient behavior. Our solution to this problem is a practical one. We observe that, when network congestion happens, the packet delays measured in simulation will be different from the predicted values calculated by the steady-state queuing model due to the transient behavior. To remove such discrepancies, we can adjust the packet processing speed in emulation so that the delays match the simulation results.

Let ΔT be the interval at which the simulation system updates the emulation system with its measurements. At the beginning of each update interval, say, at time t, let $p(t)$ be the measured drop probability, let $\lambda_p^e(t)$ be the effective arrival rate of the emulated flows, and let $W_1(t)$ be the average packet delay through the network segment. All these measurements are collected during the last interval in simulation.

Let $W_2(t)$ be the average packet delay through the corresponding pipe in emulation, which we can measure during the same period. The difference between $W_1(t)$ and $W_2(t)$ indicates the effect from the transient behavior. We can calculate such difference in the number of packets in the queuing system during this period:

$$\Delta L(t) = \mu^*(t)(W_2(t) - W_1(t)) \tag{4}$$

where $\mu^*(t)$ is the steady-state service rate calculated using Equation (3). Now we can compute the excess (or deficit) service rate to compensate for the transient effect:

$$\Delta\mu^*(t) = \frac{\Delta L(t)}{\Delta T} = \frac{\mu^*(t)(W_2(t) - W_1(t))}{\Delta T} \tag{5}$$

Finally, we can add the adjustment to the steady-state service rate to arrive at the final service rate we use for the queue in emulation during the next update interval:

$$\begin{aligned} \hat{\mu}(t) &= \mu^*(t) + \Delta\mu^*(t) \\ &= \frac{\lambda_p^e(\Delta T + W_2(t) - W_1(t))}{\Delta T(1 + W_1(t)\lambda_p^e(t) - \sqrt{1 + W_1^2(t)\lambda_p^e(t)^2})} \end{aligned} \tag{6}$$

The adjustment effectually forces the emulation system to "track" the simulated network conditions at each update interval. Experiment results, which we show in section 6, confirm that using the adjusted service rate the emulation system is able to match with the simulated network behavior even during extremely heavy congestions.

5. REPRODUCING EMULATED FLOWS IN SIMULATION

Our queuing model is based on the assumption that the emulated flows—those between the target applications in the emulation system—can be faithfully reproduced in simulation. In Figure 3 we show that the same arrival rate is used for the emulated flows entering the network segment in simulation and entering the corresponding pipe in emulation.

To do that, one approach is to directly "inject" the packets generated by the target applications into the simulator. That is, we generate a simulation event to represent the arrival of each emulated packet in emulation. Similarly, when a simulated packet reaches an emulated host or router, a real packet will be created in emulation. This is the approach taken by real-time simulation [Liu 2008], which has demonstrated capable of achieving pretty accurate results. The problem with this approach is that the throughput of the emulated flows is bounded by the I/O capacity of the system; the accuracy of the model can be severely compromised if the emulated traffic load reaches its limitation [Liu et al. 2009].

We propose a different approach. Rather than directly injecting the emulated packets generated by the target applications into simulation, we make the emulated hosts report only the traffic demand of the target applications to simulation as metadata. In particular, the emulated hosts only need to capture the number of bytes (`appBytes`) requested by the target applications to be sent through the transport layer. In simulation, we implement the same transport layer to generate the simulated packets accordingly and recreate the same traffic load as in the emulation system.

This method obviously can scale better as it does not require exchanging individual network packets between the simulation and emulation systems; however, it depends on a careful implementation of the TCP/IP stack in simulation so that it can create the same traffic behavior as in the real systems. For example, the simulator needs to include various TCP flavors used commonly by the real systems. Previously, we implemented and validated fourteen TCP variants in our network simulator, which are found commonly in use today, including New Reno, BIC, CUBIC, and others. We ported code specific to the congestion control mechanism of of these TCP variants directly from the Linux TCP implementation [Erazo et al. 2009].

There are several methods to capture the data size sent by the target applications. One way is to use `/proc` on Linux or similar facilities. This approach is most straightforward; however, it only supports polling, which would generate noticeable overhead if a small polling interval is needed to achieve accuracy. Another approach is to replace the transport-layer functions in the communication library with one added with a callback feature whenever the send functions are invoked. This can be achieved without modification to the application source code, through either static or dynamic linking (e.g., [Liu et al. 2003]). It is also possible to develop a kernel module to achieve the same function.

As a proof of concept, here we simply rewrite the target applications, such as `iperf`, and let them report, through inter-process communication, to another program running on the same machine, which we call the *"traffic sensor"*. The traffic sensor is expected to update the simulator for each application data transfer request, which includes the transfer size, and the source and destination addresses. Upon receiving this information, the simulator invokes the the same transport layer protocol to generate the packets accordingly.

6. MODEL VALIDATION

In this section we first validate the queuing model. Here we use a simulator of the symbiotic system so that we can conveniently explore various network settings and stay clear from the potential system-related artifacts in a real implementation. (We present real-system validation in the next section.) In particular, we aim to find out whether our queuing model is robust and can accurately capture the interaction between simulated and emulated traffic in the reduce model as in the full-scale model. For that, we start with a simple single-link segment and then extend it to multi-link segment. We experiment with different packet arrival processes (including those from real packet traces) and with different mixtures of simulated and emulated traffic.

6.1. Single-Link Validation

We first conduct experiment to validate the queuing model assuming that a network segment contains only one link. We examine whether the queuing model can be generalized and used for different packet arrivals with different mixture of simulated and emulated flows. In particular, we test the model with different packet inter-arrival time distributions: exponentially distributed, a combination of constant and exponentially distributed, and also from real packet traces. We test whether the model can produce the same packet delays in the downscaled model in the emulation system as those in the full-scaled model in the simulation system.

In the experiments, we designate two flows at a network interface: one as "simulated flow" and the other as "emulated flow", each with an independent input process. At each second (i.e., we set the update interval $\Delta T = 1$ second), we measure the drop probability, the effective arrival rate of the emulated flow, and the average packet delay of both emulated and simulated flows, at the network interface. The measurements are collected in a trace file and later used by a subsequent simulation of the emulation system that has only the "emulated flow" with exactly the same packet arrivals.

Poisson Arrivals. We first set the packet inter-arrival time of both simulated and emulated flows to be exponentially distributed. We set the bandwidth of the network interface to be 10 Mbps and the queue length to be 1.5 MB. We also fix the packet size to be 1500 bytes in this study. We vary the aggregate arrival rate of both simulated and emulated flows to be 10%, 50%, and 90% of the bandwidth for different service levels, and we vary the proportion of the emulated flow to be 20%, 50%, and 80%, respectively. We examine the accuracy of the model by comparing the packet delays between the original and the downscaled system.

Figures 5 shows the average packet delays measured at each second during the experiment for low, mid, and high service utilization scenarios. In all cases, the results match quite well. The difference is almost negligible: around 10 μs for 10% utilization, 40 μs for 50% utilization, and below 1 μs for 90% utilization.

Mixed Poisson and Constant Arrivals. Next, we set the packet inter-arrival time of the emulated flow to be constant and that of the simulated flow to be exponentially distributed, and vice versa. We use the same settings as that in the previous experiment. We also vary the aggregate arrival rate of both simulated and emulated flows to be 10%, 50%, and 90% of the total bandwidth, and we vary the proportion of the emulated flow to be 20%, 50%, and 80%, respectively.

Figure 6 shows the average packet delays for different service utilizations when the packet inter-arrival time of the emulated flow is constant and that of the simulated flow is exponentially distributed, and the portion of the emulated flow is fixed at 20%. Figure 7 shows when the emulated flow is exponentially distributed and the simulated flow is constant. Similar results (not shown) are obtained for different mixture of the simulated and emulated flows. In all cases, the model matches well with differences below 1 μs.

Real Packet Traces. The simulator is able to replay packet traces generated by `tcpdump` so that we can reproduce similar traffic demand, with the same packet sizes and inter-arrival times, as in the real system. We also add functions in the simulator to either dilate or contract packet inter-arrival times by a constant factor in order to artificially adjust the traffic intensity if needed.

For this experiment, we use a packet trace from *the CAIDA Anonymized Internet Traces 2011 Dataset* [CAIDA 2011]. The trace is collected at an OC-192 link (9953 Mbps), consisting of over twenty million packets. We replay the CAIDA trace as the simulated flow. Here we show the results of two experiments: one with a link band-

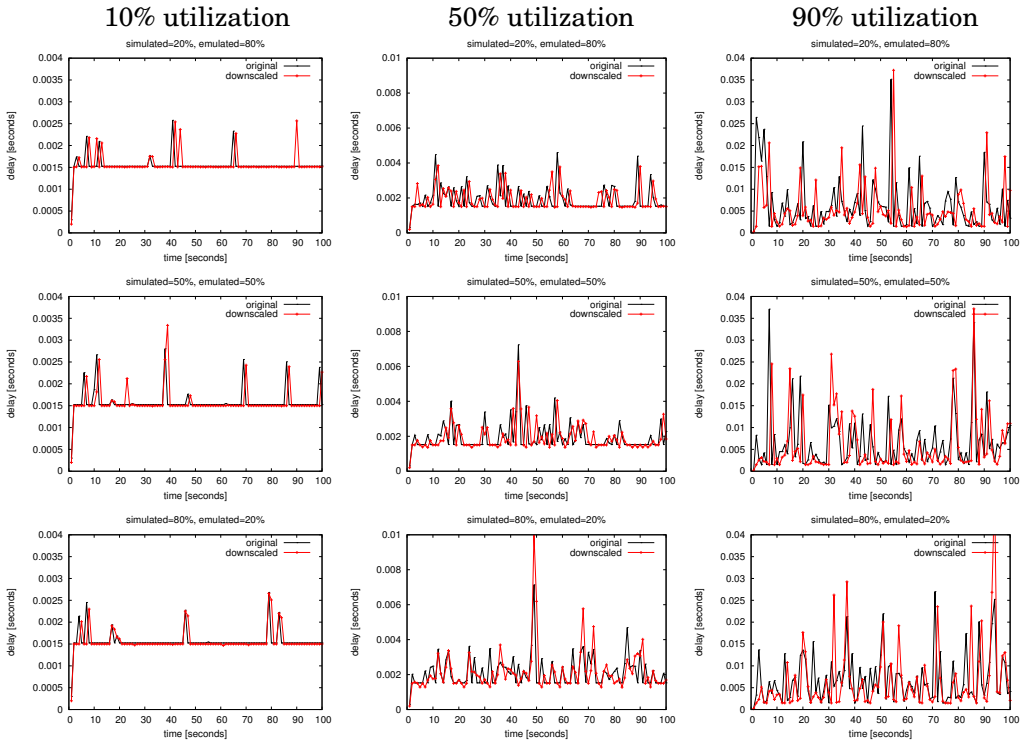

Fig. 5: Packet delays for Poisson arrivals at different utilization levels.

width of 1 Gbps and the other with 100 Mbps. For the 1 Gbps experiment, we dilate the packet trace (i.e., multiply the packet inter-arrival time) by a factor of 30; for the 100 Mbps experiment, we dilate the packet trace by a factor of 100. In this case, we get a lower traffic intensity for the first scenario than the second. In both scenarios, we limit the buffer size to be 12.5 MB. We play the trace as the simulated flow and then cut it off at around 30 seconds to create a burst at the beginning of the experiment. The emulated flow is exponentially distributed and its arrival rate is set at 50% of the bandwidth. The emulated flow lasts for the entire experiment.

Figure 8 compares the average packet delays measured in the original system and the downscaled system. In the first scenario (with 1 Gbps bandwidth), there is rarely any congestion; there is no significant queuing delay, and the average end-to-end delay stays around 3 milliseconds. In the second scenario (with 100 Mbps bandwidth), during the first 5 seconds, the queue builds up quickly until packets start to get dropped. The packet delay is much higher in this case than the previous one. The congestion persists until 33 seconds before the average packet delay drops down resulted from the cutoff of the simulated flow. For both scenarios, the original and the downscaled model produce very similar results.

Exploring Various Network Settings. In order to quantitatively evaluate the effectiveness of our model to emulate a single link, with and without congestion, we run a batch of experiments using different bandwidths and buffer sizes. In particular, we set the link bandwidth to be either 1, 10, or 100 Mbps, and the buffer size to be 100, 500, or 1000 packets. We fix the packet size to be 1500 bytes. For each distinct bandwidth and buffer size setting, we independently set the packet arrival rate of the emulated

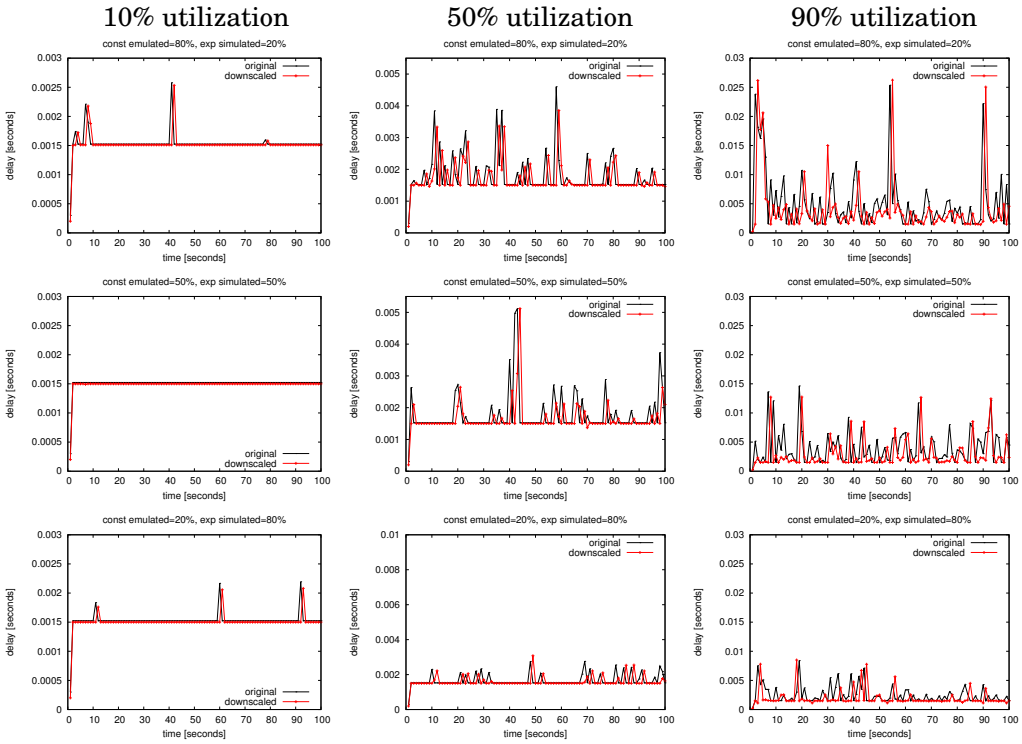

Fig. 6: Packet delays for exponentially distributed simulated flows and constant emulated flows at different utilization levels.

Table I: Errors from a single-link segment

Bandwidth Mb/s	Buffer Size pkts	Packet Delays msecs (%)	Packet Lost pkts (%)	Packet Received pkts (%)
1	100	8 (7.96%)	2.06 (0.22%)	3.25 (0.07%)
1	500	8 (7.63%)	5.93 (0.56%)	7.12 (0.16%)
1	1000	7 (7.57%)	2.93 (0.49%)	3.00 (0.07%)
10	100	2 (8.83%)	18.56 (0.13%)	21.37 (0.04%)
10	500	2 (7.93%)	34.18 (0.29%)	32.12 (0.07%)
10	1000	2 (7.78%)	24.43 (0.22%)	21.00 (0.04%)
100	100	0.3 (5.04%)	141.50 (0.62%)	56.75 (0.01%)
100	500	0.4 (3.90%)	183.12 (0.17%)	98.56 (0.02%)
100	1000	0.4 (3.67%)	209.00 (0.19%)	113.12 (0.02%)

flow and that of the simulated flow to be 12, 25, 50, or 75% of the link bandwidth. Consequently, we conduct a total of 16 experiments (with separate settings for the packet arrival rate for the emulated flow and for the simulated flow) for each distinct bandwidth and buffer size setting. Each experiment is run for 100 seconds.

Table I shows the difference in the average packet delays, the average number of dropped packets, and the average number of received packets between the original and the downscaled system—the numbers are averaged across the 16 experiments for each combination of link bandwidth and buffer size.

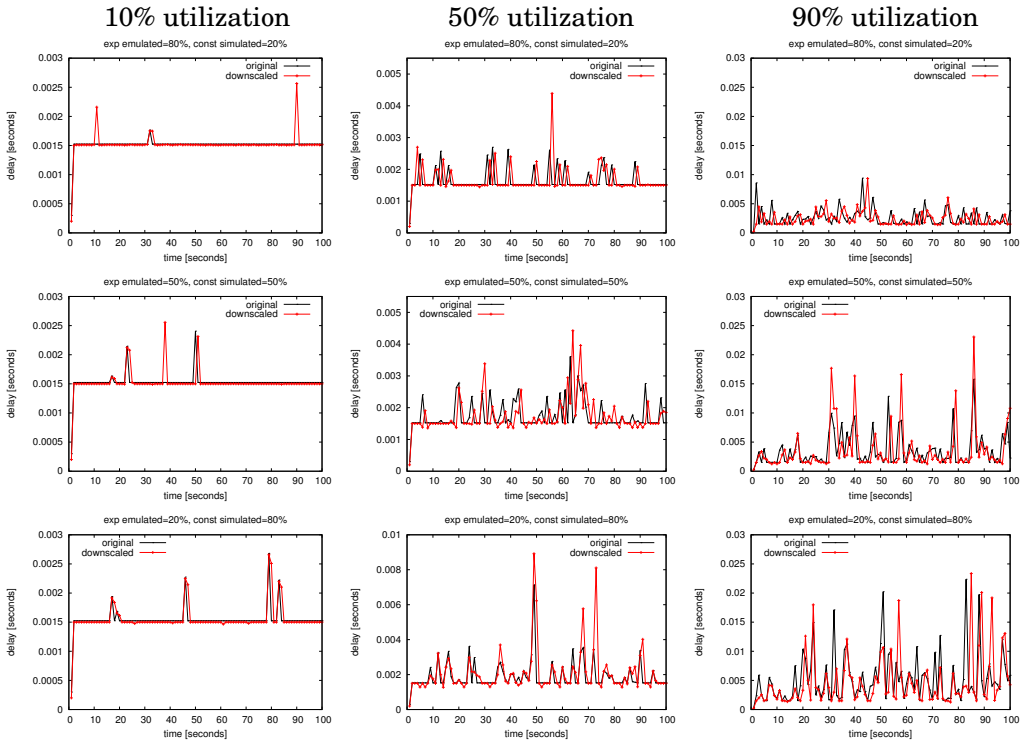

Fig. 7: Packet delays for constant simulated flows and exponentially distributed emulated flows at different utilization levels.

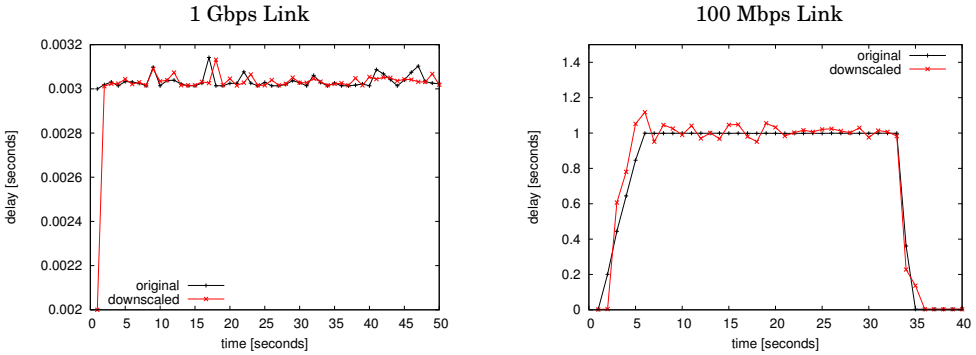

Fig. 8: Results from using the CAIDA trace.

The average percentage error (shown in the parentheses) for the number of dropped packets and the number of received packets is very small, indicating a good match between the two systems. The packet-delay error for emulating a link with a bandwidth of 100 Mbps is less than 1 millisecond. It goes up to 2 milliseconds for a 10 Mbps link, and 8 milliseconds for a 1 Mbps link. This is mainly due to the packet transmission time. Such error would be mostly imperceptible for high-bandwidth high-latency links. Overall, the results show conclusively that our queuing model provides a good approximation when emulating a single link regardless of the congestion level.

6.2. Multiple-Link Validation

In the previous section, we show the validation results for a segment consisted of only a single link. In this section we examine the model behavior when a segment contains multiple links. Our model downscaling technique compresses the links traversed by the same set of emulated flows into one network segment and uses a single queue in emulation to calculate the packet delays and losses. However, different links may involve different simulated flows. We need to find out whether the queuing model in this case is able to accurately capture the traffic behavior in terms of end-to-end packet delays and packet losses in the presence of cross-traffic and multiple bottlenecks.

We use a network that consists of three nodes connected in tandem with distinct bandwidths. The setup is similar to the one shown in Figure 4. We designate an emulated flow to traverse all three network interfaces. We also direct three simulated flows, each entering at a different network interface, but all existing at the last one. The simulated and emulated flows have independent input sources with potentially different packet arrival rates.

Like before, we conduct two back-to-back simulations for each experiment setting—one for the original system and the other for the downscaled system. During the simulation of the original system, we measure the drop probability and the effective arrival rate of the emulated flow at each of the three interfaces. We also record the end-to-end delays through the network segment. These measurements are collected and stored in a trace file and used by a subsequent simulation of the downscaled system.

Staggered Arrivals. In this experiment, we set the bandwidth of the three links to be 1, 10, and 100 Mbps, respectively. We fix the packet size to be 1500 bytes. The buffer size of all network queues is set to be 200 packets. The packet arrivals for all flows are Poisson. We set the arrival rate of the emulated flow to be 50 packets/s, that is, 600 Kbps. We set the arrival rate of the three simulated flows to be 600 Kbps, 9.6 Mbps, and 99.6 Mbps, respectively.

We start the emulated flow at the beginning of the experiment and let it persist through the whole experiment, which lasts for 120 seconds. We start the simulated flows in reverse order. The one entering the third network interface starts at the beginning of the experiment and lasts for 40 seconds. The one entering the second interface starts at 30 seconds and ends at 50 seconds. The one entering the first interface starts at 60 seconds and ends at 80 seconds. In this way, we manage to create three separate congestion points in the network segment.

Figure 9 shows the packet delay, the cumulative packet loss, and the throughput of the emulated flow over time. The delay and the throughput are average values measured at each second. We see clearly from the delay plot that congestion happens between 30 and 50 seconds, and between 60 and 100 seconds. And we see that it takes some time before a congestion can be cleared up—the figure shows that the average delay waits until 106 seconds to come back to normal. The first link is the slowest; as expected, longer packet transmission time at the first queue causes longer packet delays during the congestion (between time 60 and 106).

The packet loss is also significant between 60 and 100 seconds; this is because the emulated and simulated flows are mixed up with the same proportion at the first queue and therefore share the loss equally. At the other two queues the simulated flow dominates the emulated flow, the losses are there but much more insignificant. In all cases, the results from the original system and the downscaled system match well.

Exploring Various Network Settings. In order to quantitatively evaluate the effectiveness of our queuing model for emulating a segment with multiple links, we run a batch of experiments using different bandwidths for the links. To assess accuracy,

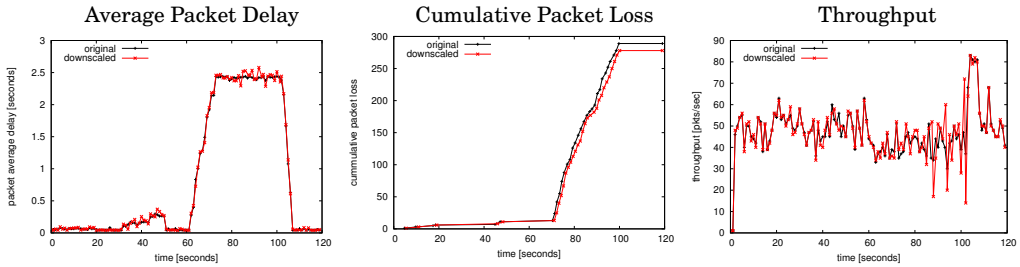

Fig. 9: Results from staggered arrivals.

Table II: Errors from a multi-link segment

μ_1	μ_2	μ_3	Packet Delay	Packet Losses	Throughput
10	10	10	1.72%	0.30%	0.28%
10	10	100	1.96%	0.59%	0.28%
10	100	10	1.91%	2.47%	0.21%
10	100	100	0.85%	0.40%	0.15%
100	10	10	0.54%	0.32%	0.99%
100	10	100	0.55%	0.18%	0.79%
100	100	10	0.19%	0.37%	0.97%
100	100	100	1.8%	0.23%	0.10%

we again compare the average packet delay, the number of dropped packets, and the number of received packets, between the original system and the downscaled system.

Like in the previous experiment, we fix the packet size to be 1500 bytes, the buffer size to be 200 packets, and we use Poisson arrival for all flows. Unlike the previous experiment, here we independently set the bandwidth of the three links to be either 10 or 100 Mbps. Since we have three links, each with two possible settings, there are eight combinations of the link bandwidth settings. For each of the eight settings, we conduct six separate experiments, by setting the packet arrival rate of all flows to be 12%, 25%, 50%, 100%, 125%, or 150% of the bandwidth of the link where the respective flow is entering the system. For example, if the bandwidth of the first link (μ_1) is 10 Mbps, and the bandwidth of the second and third link (μ_2 and μ_3) is 100 Mbps, and if the arrival rate of all flows is 50%, it means that the packet arrival rate of the emulated flow and the first simulated flow would be 5 Mbps, and the packet arrival rate of the second and the third simulated flow would be 50 Mbps.

We run the simulations for 500 seconds. All flows start at time zero and last for the entire duration of the experiment. Table II shows the differences between the original and the downscaled systems, in packet delays, packet losses, and throughput of the emulated flow. Each row represents a different permutation of the link bandwidth setting and shows the result averaged among the six experiments with different traffic intensity. Again the two systems match well. The errors are all below 2%.

7. REAL SYSTEM IMPLEMENTATION

We implemented a network testbed, called *symbiosim*, as a prototype realization of our proposed symbiotic approach. In this section, we briefly describe the design and implementation of the system, and present results from preliminary experiments involving real applications and protocols.

7.1. Prototype Implementation

The prototype consists of several components, including a simulator, an emulator using physical machines for emulated hosts and delay nodes, and several utility programs for connecting the simulator and the emulator.

We use PRIME [2013] for high-performance network simulation. PRIME can run the full-scale network model *in real time* on parallel platforms. The simulator also has detailed models for various congestion control algorithms of common TCP variants, which have been validated extensively [Erazo et al. 2009].

We use dummynet [Rizzo 1997] as the the network emulator, which functions as the "delay node". The network is represented as a set of pipes with specific delay constraints. We apply our queuing model, and set the bandwidth and packet drop probability of the pipes using the statistics collected by the simulator to control the real network packets being pushed through them.

We instantiate PRIME, dummynet, and the emulated hosts and routers on individual physical machines on ProtoGENI [2013]. ProtoGENI allows us to allocate an experiment slice, which can be configured to contain a set of machines with specific operating systems and network connections on an EmuLab cluster citeEMULAB.

We create three utility programs to facilitate communication between simulation and emulation. The first utility program is called the *"data gatherer"*, which is expected to run side-by-side with the simulator instance[3]. The program takes as input the raw statistics exported by the simulator at the end of each update interval for all simulated network interfaces traversed by emulated flows. It puts together the packet drop probabilities and calculates the service rates for the corresponding network segments, and then sends the update information to dummynet. The same program is also responsible for gathering information about the traffic demand from the emulated hosts (in the emulation system), and informing the simulator to regenerate the flows in simulation accordingly.

We run the second utility program, called the *"actuator"*, at the machine which runs dummynet (i.e., the delay node). The program receives the state information from the data gatherer and then updates the parameters of the corresponding pipes in dummynet, including the drop probabilities and the service rates.

At each emulated host, we run third utility program, called the *"traffic sensor"*, which collects the target applications' traffic demand in number of bytes. As a prototype, we simply rewrite the target applications and have them report to the co-located traffic sensor program through inter-process communication. The program collects the information and then sends an update to the "data gatherer" at the simulator site, which informs the simulator to regenerate the traffic in simulation.

7.2. Preliminary Experiments

We conduct experiments to validate the accuracy of the symbiotic approach using real applications and real protocols. We compare the delay and throughput produced by the real applications running in *symbiosim* with those produced by applications with similar characteristics running entirely in simulation. We make sure that simulated applications behaves similar to their real counterparts, such as using the same TCP variant with the same configuration parameters. The results would demonstrate the feasibility of the symbiotic approach in handling real-world scenarios.

We conduct this test using a simple yet representative network topology, called "parking lot network", as shown in Figure 10. The network consists of six end hosts connected by four routers. The network is a classic topology often used in literature for

[3]If the simulator is run in parallel, there will be one data gatherer associated with each parallel simulation instance.

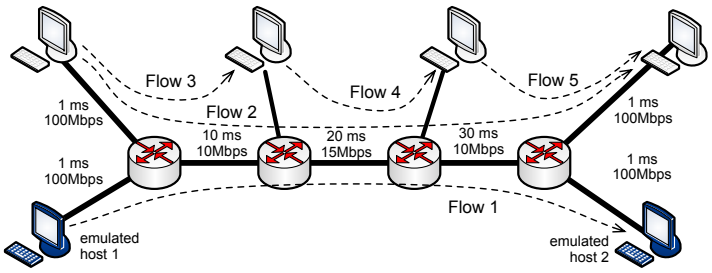

Fig. 10: Baseline network for validation.

testing various network congestion control protocols. As described momentarily, one can easily create multiple bottlenecks in the network and cause cross-traffic interactions between different flows.

For the experiment, we designate two end hosts (at the bottom on either side) to be the emulated hosts. In the downscaled topology, the two emulated hosts are connected by one network segment consisted of five links. The emulation system is instantiated on three physical machines: one for each of the two emulated hosts, and a third as a delay node running dummynet for link emulation and connecting the other two machines running as emulated hosts. The three machines are allocated using ProtoGENI, which are specified to run the latest Linux OS directly on the physical nodes. The emulated hosts use the default TCP version (CUBIC) to communicate with each other. Since the simulation workload is relatively moderate for this experiment, we instantiate the simulation system directly on the same delay node. We make sure the network simulator uses the same TCP version for communication between the end hosts.

In the experiment, we direct five TCP flows, each containing multiple simultaneous TCP sessions. *Flow 1* is an emulated flow, which has only one TCP session generated by a pair of Java client/server applications that use HTTP to transfer a large data file over the network. The data transfer will span the entire experiment, which lasts for 60 seconds. The other four flows are all simulated flows. We carefully set the start time and the size of the data transfers so that we can create separate and diverse congestion scenarios in the network throughout the experiment. *Flow 2* contains 5 simultaneous TCP sessions each transferring 0.5 MB of data and all starting at 10 seconds. At 30 seconds, *Flow 3* starts with another 5 TCP sessions, each transferring 2 MB of data. Before the transfers finish, we start *Flow 4*, which contains 10 TCP sessions each transferring 1 MB of data. We intentionally make *Flow 3* and *Flow 4* overlap and thus create two bottlenecks in the network at the same time. Finally, at 50 seconds, *Flow 5* starts with another 5 TCP sessions each transferring 1MB of data.

Figure 11 shows the measured round-trip times between the two emulated hosts, both from *symbiosim* and from pure simulation. For *symbiosim*, we also run `ping` on the emulated hosts together with the HTTP application so that we have the round-trip time at each second during the experiment. For pure simulation, we simply extract the round-trip time from the trace files generated by the simulated TCP protocol. The plot shows the *symbiosim* results from running the same experiment fifteen times. We also plot the average round-trip time from *symbiosim* and from pure simulation. We see that the delays from *symbiosim* closely follows what is expected from the simulation. The average round-trip time from simulation is 148 ms and that from *symbiosim* is 152 ms; the error is about 3%. We also observe that, at the congestion points we created in the simulation, *symbiosim* is able to react promptly as we can see the delays for the emulated flows jump accordingly due to the congestion.

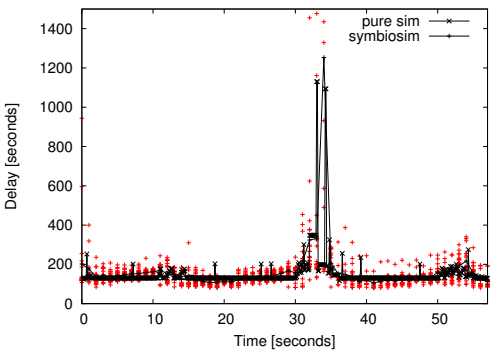

Fig. 11: Round-trip delay.

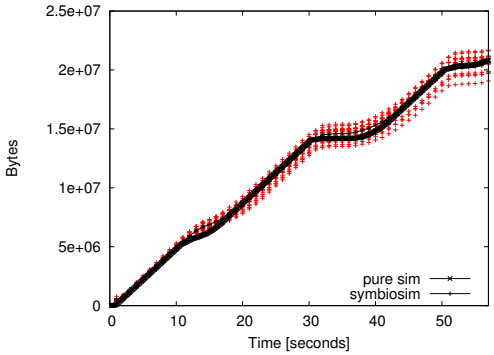

Fig. 12: Received data.

Figure 12 plots the sequence numbers (in bytes) of the received TCP segments by the emulated host over time. Again, we show the *symbiosim* results from the fifteen runs, along with the averages from both *symbiosim* and pure simulation. We see that the average sequence number history from *symbiosim* is almost indistinguishable from that of simulation. The maximum sequence number we collect at the end of the experiment in simulation is 20,843,961; *symbiosim* gets 20,748,458. The error is only 0.5%. These results confirm that our symbiotic approach can accurately emulate the communication path between the target applications.

8. A CASE STUDY

Recent research shows that multipath routing can be used to improve data transport over the Internet by taking advantage of its path diversity [Han et al. 2006; Raiciu et al. 2009]. In this section, we present a performance study using *symbiosim* to test a multipath protocol, as an example to demonstrate the use of the symbiotic approach for evaluating new network protocols and applications.

Our protocol uses source routing. Each end host can choose multiple paths to route traffic from source to destination based on information from dedicated access routers. These dedicated access routers provide multipath services to subscribed end hosts, by informing them the availability of multiple paths to a given destination, and providing periodic updates about these paths, including the current bandwidth, round-trip time (RTT), and loss rate. This information will be used by the end host to determine how to transport data to maximize its target utility function.

An example is shown in Figure 13. A sender is subscribed to the multipath service and therefore receives periodic updates about the three alternative paths to a chosen receiver. Each alternative path is depicted in the figure as a separate cloud, which represents the set of network links on the path from the sender's access router to the receiver's access router. Our multipath transport protocol at the end host is implemented as a Java application using TCP. At any moment, the sender can establish multiple TCP sessions with the receiver, each using a different network path. There are different ways to implement source routing, including using label switching or OpenFlow. For simplicity, in our implementation, we only emulate the effect of source routing by designating separate receivers with distinct IP addresses and routing paths.

We conducted an experiment to investigate the effectiveness of multipath selection algorithms—whether they can quickly find a high-capacity alternative path when failure occurs. We use a simple network topology with one sender and one receiver with three distinct paths between them (as shown in Figure 13). The three alternative paths all have the same bandwidth of 100 Mbps. Without other traffic, the round-trip times

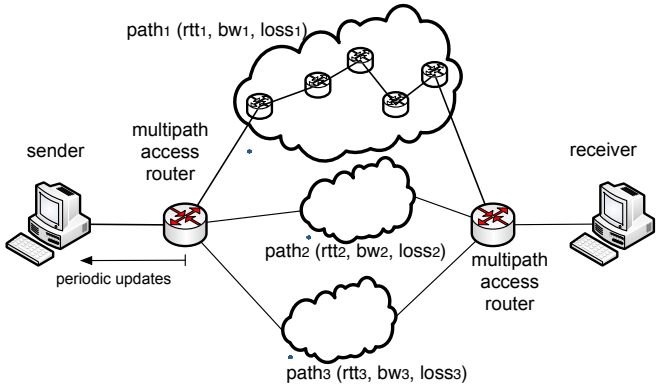

Fig. 13: Multipath routing infrastructure.

for the three alternative paths are 5 ms, 10 ms, and 50 ms, respectively. The access link between the sender and its access router is set to have 50 Mbps bandwidth and 1 ms delay. The access link for the receiver has infinite bandwidth and zero delay. The emulation system is instantiated on three physical machines: two emulated hosts for the sender and the receiver, and one delay node running dummynet connecting the two emulated hosts.

We evaluate three multipath selection algorithms. For the experiment, the sender sends a large data file to the receiver over two TCP connections simultaneously. All algorithms chooses the first two paths with the least RTTs in the beginning. The first algorithm is static; the selection does not change throughout the data transmission. The second algorithm dynamically selects the two paths with the least loss rate. The third algorithm selects the two paths with the most available bandwidth.

The data transfers start immediately at the beginning of the experiment. At 20 seconds, we artificially create network congestion by initiating 10 TCP flows on each of the three paths in simulation. Figure 14 shows the instantaneous throughput (measured at one second intervals) for the three multipath selection algorithms together with 95% confidence intervals corresponding to 20 trials. We observe that the static path selection algorithm is not able to adapt to the changes in the network condition. The throughput drops significantly starting at 20 seconds due to the congestion. Both loss-based and bandwidth-based path selection algorithms can improve the situation. The loss-based algorithm achieves less throughput and does not seem to reach stability (which persists beyond the 40 seconds shown in the figure). After investigation, we found that the loss-based algorithm constantly changes its decision on second data path for data forwarding. Path switching introduces overhead due to TCP ramping up during slow start.

Our study here is preliminary. One would create more realistic network topologies and use more complex network background traffic to carefully test the applications. However, it is sufficient to show that our symbiotic simulation and emulation approach can provide the mechanism for one to evaluate the implementation of new protocols and applications under diverse simulated network conditions.

9. RELATED WORK

In Biology, symbiosis is defined as the mutually beneficial relationship between two or more different organisms. Symbiotic simulation can be defined as “one that interacts with the physical system in a mutually beneficial way” [Fujimoto et al. 2002].

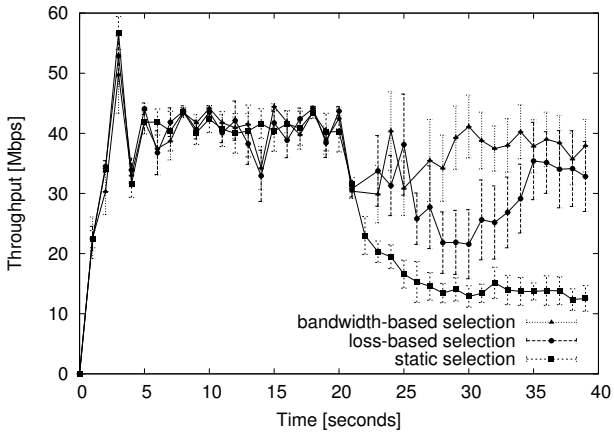

Fig. 14: Instantaneous throughput of the multipath algorithms.

There are two promising areas that combine network simulation and emulation. *Online simulation* uses simulation as an integrated service for real-time network management with the goal of improving network performance, via network planning, monitoring, parameter tuning, and traffic engineering (e.g., [Szymanski et al. 2002; Ye et al. 2001]). *Real-time simulation* performs simulation in real time so that the target virtual network can interact with real network entities (e.g., [Fall 1999; Simmonds et al. 2000; Zhou et al. 2004; Liljenstam et al. 2005; Ahrenholz et al. 2008; Liu et al. 2009; Nicol et al. 2011]).

ROSENET [Gu 2007] is an early attempt to promote the symbiotic relationship between simulation and emulation. It combines a high-performance simulator and a low-fidelity emulator running at separate locations. The simulator continuously updates the emulator with link statistics, including packet delay, jitter, and loss. The emulator also continuously updates the simulator with a summary of the real traffic. ROSENET achieved its initial success, which has inspired our work. However, it is shown to be capable of emulating only a single bottleneck link and also only applications that generate non-responsive traffic (i.e., UDP applications). Our work has improved over ROSENET both in terms of handling large and complex networks of arbitrary topology, and in terms of dealing with elastic TCP flows.

Network emulators, such as dummynet [Rizzo 1997], ModelNet [Vahdat et al. 2002], NIST Net [Carson and Santay 2003], and EmuLab [White et al. 2002], test real applications with well orchestrated network conditions. For example, dummynet works by forwarding the network packets through a set of pipes that approximate the behavior of the corresponding network queues. Each pipe focuses only on a single link. Our work extends the network emulators to deal with network-wide behaviors. Also, our work incorporates simulation which brings the flexibility of including different abstract models.

Topology downscaling is based on the observation that only congested links introduce sizable queuing delays and packet losses [Barakat et al. 2002; Fraleigh et al. 2003a; Fraleigh et al. 2003b; Papagiannaki et al. 2002]. In other words, uncongested links, especially those with capacities large enough to simultaneously carry many flows, are somewhat transparent to the packets traversing them [Eun and Shroff 2003]. Papadopoulos et al. [2006] proposed a method that aims at downscaling network topology by removing the uncongested links, retaining only the congested ones, and compensating for the removed links with additional delays. The problem with

their approach is that the bottleneck links have to be known in advance. Although static analysis may help reveal the potential congestion points in the network, the selection can be too general for emulation to achieve an effective model reduction.

Sanaga et al. [2009] proposed a method to approximate the entire network path with a single link, by modeling the link capacity and the available bandwidth separately. The method calculates the available bandwidth using non-responsive traffic (with constant bit rate). Although this approach can give a good approximation of the average behavior, it does not provide sufficient granularity to accurately capture the interaction between the simulated and emulated flows as in our case. Modeling the traffic intensity simply as non-responsive flows is also unfair to TCP.

Symbiotic simulation is also referred as DDDAS (Dynamic Data-Driven Application System), in a larger context that has broadly applied in the areas of manufacturing, business, system engineering, civil engineering, biology, social science, and many other disciplines. In DDDAS, simulation and the physical system form a symbiotic feedback control system, whereas a simulation can dynamically incorporate data from the physical system so that it can improve the measurement process or exercise more precise control of the physical system [DDDAS 2014].

10. CONCLUSIONS

In this article, we propose a symbiotic simulation and emulation approach, which provides a new method for evaluating complex network systems, where large-scale network transactions can be modeled using simulation, and real application instances can run directly in real-machine environments and communicate through emulated paths reflecting the simulated traffic conditions of the large-scale network. More specifically, we propose a model downscaling technique that can significantly reduce the computational complexity of the original large-scale model in order to enable high-capacity traffic emulation. In order to efficiently synchronize the full-scale simulated network model and the downscaled emulated network model, we introduce a queuing model so that emulation can reproduce the same simulated traffic conditions between the target applications. We also introduce a technique for efficiently regenerating the same emulated traffic behavior in simulation, by capturing the target applications' traffic demand and then creating the traffic flows at the corresponding hosts in simulation. Extensive experiment results using simulation and with a prototype implementation of the symbiotic approach show that our approach is able to produce accurate results.

It is important to recognize the limitations of our current approach. First, the model downscaling method assumes fixed network topologies as well as stable traffic routing and forwarding. As such, it would limit this method to studies of infrastructure networks where dynamic routing cannot the be primary objective of the studies. The method also cannot be easily extended to studying wireless networks where network connectivity may constantly change as a result of node mobility or frequent shifts in the wireless channel properties. Second, our network queuing model is based on first-come-first-serve (FCFS) scheduling, which may be questionable for today's popular network switches that adopt flow-level QoS or fair queuing policies. In this aspect, the recent work by Jin and Nicol [2010] may provide a valuable direction for incorporating more diverse and realistic switch scheduling policies in our symbiotic approach. Third, the current symbiotic approach focuses only on flow rates and their effect on network queuing (such as throughput, delay, and packet loss). In particular, it cannot deal with the content. Incorporating content-based traffic models may provide significant value for cyber-security applications, such as studying distributed intrusion detection techniques. We defer that to future work.

For immediate future work, we are currently developing a full-scale implementation of the *symbiosim* testbed. We will investigate methods for further scaling up the

system, such as instantiating the emulation system on virtual machines, similar to what has been done for real-time network simulation [Liu et al. 2009; Van Vorst et al. 2011a]. Once we have a full-scale implementation, we plan to conduct extensive evaluations on the performance and capabilities of our approach in the context of real large-scale network applications.

REFERENCES

J. Ahrenholz, C. Danilov, T.R. Henderson, and J.H. Kim. 2008. CORE: A real-time network emulator. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*. 1–7.

Chadi Barakat, Patrick Thiran, Gianluca Iannaccone, Christophe Diot, and Philippe Owezarski. 2002. A flow-based model for Internet backbone traffic. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment (IMW'02)*. 35–47.

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

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

CAIDA. 2011. The CAIDA Anonymized Internet Traces 2011 Dataset. (2011). http://www.caida.org/data/passive/passive_2011_dataset.xml. Last accessed: December 2013.

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

Xinjie Chang. 1999. Network simulations with OPNET. In *Proceedings of the 1999 Winter Simulation Conference*, Vol. 1. 307–314.

DDDAS. 2014. Dynamic Data-Driven Application Systems Info Cybernetics. (2014). http://www.dddas.org/. Last access: August 2014.

Miguel A. Erazo, Yue Li, and Jason Liu. 2009. 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 & Communities and Workshops (TRIDENTCOM'09)*. 1–10.

Do Young Eun and N.B. Shroff. 2003. Simplification of network analysis in large-bandwidth systems. In *INFOCOM'03*.

Kevin Fall. 1999. Network emulation in the Vint/NS simulator. In *Proceedings of the fourth IEEE Symposium on Computers and Communications*. 244–250.

C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. C. Diot. 2003a. Packet-level traffic measurements from the Sprint IP backbone. *Netwrk. Mag. of Global Internetwkg.* 17, 6 (2003), 6–16.

C. Fraleigh, F. Tobagi, and C. Diot. 2003b. Provisioning IP backbone networks to support latency sensitive traffic. In *INFOCOM'03*.

R. Fujimoto, D. Lunceford, E. Page, and A. M. Uhrmacher. 2002. *Grand challenges for modeling and simulation*. Technical Report 350. Schloss Dagstuhl.

Yan Gu. 2007. *ROSENET: A remote server-based network emulation system*. Ph.D. Dissertation. Georgia Institute of Technology.

Huaizhong Han, S. Shakkottai, C.V. Hollot, R. Srikant, and D. Towsley. 2006. Multi-path TCP: A joint congestion control and routing scheme to exploit path diversity in the Internet. *IEEE/ACM Transactions on Networking* 14, 6 (2006), 1260–1271.

Mark Handley, Eddie Kohler, Atanu Ghosh, Orion Hodson, and Pavlin Radoslavov. 2005. Designing extensible IP router software. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI'05)*. 189–202.

Dong Jin and David M. Nicol. 2010. Fast simulation of background traffic through fair queueing networks. In *Winter Simulation Conference*. 2935–2946.

E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. 2000. The Click modular router. *ACM Transactions on Compauter Systems* 18, 8 (2000), 263–297.

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

Jason Liu. 2008. A primer for real-time simulation of large-scale networks. In *Proceedings of the 41st Annual Simulation Symposium (ANSS'08)*. 85–94.

Jason Liu. 2013. Real-time scheduling of logical processes for parallel discrete-event simulation. In *Proceedings of the 2013 Winter Simulation Conference (WSC'13)*. 2959–2971.

Jason Liu, Yue Li, Nathanael Van Vorst, Scott Mann, and Keith Hellman. 2009. A real-time network simulation infrastructure based on OpenVPN. *Journal of Systems and Software* 82, 3 (2009), 473–485.

Xin Liu, Huaxia Xia, and Andrew 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)*.

David M. Nicol, Dong Jin, and Yuhao Zheng. 2011. S3F: the scalable simulation framework revisited. In *Winter Simulation Conference*. 3288–3299.

Open vSwitch. 2013. An Open Virtual Switch. (2013). http://openvswitch.org/. Last accessed: December 2013.

F. Papadopoulos, K. Psounis, and R. Govindan. 2006. Performance preserving topological downscaling of Internet-like networks. *IEEE J. Sel. Areas Commun.* 24, 12 (2006), 2313–2326.

K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot. 2002. Analysis of measured single-hop delay from an operational backbone network. In *INFOCOM'02*.

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

PRIME. 2013. Parallel Real-time Immersive network Modeling Environment. (2013). https://www.primessf.net/prime/. Last access: December 2013.

ProtoGENI. 2013. ProtoGENI. (2013). http://www.protogeni.net/. Last access: December 2013.

Costin Raiciu, Damon Wischik, and Mark Handley. 2009. *Practical congestion control for multipath transport protocols*. Technical Report. University College of London.

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

Pramod Sanaga, Jonathon Duerig, Robert Ricci, and Jay Lepreau. 2009. Modeling and emulation of Internet paths. In *Proceedings of the 6th conference on Networked Systems Design & Implementation (NSDI'09)*. 199–212.

Rob Simmonds, Russell Bradford, and Brian Unger. 2000. Applying parallel discrete event simulation to network emulation. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'00)*. 15–22.

Boleslaw K. Szymanski, Adnan Saifee, Anand Sastry, Yu Liu, and Kiran Madnani. 2002. Genesis: a system for large-scale parallel network simulation. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS'02)*. 89–96.

Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. 2002. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.* 36, SI (2002), 271–284.

Nathanael Van Vorst, Miguel Erazo, and Jason Liu. 2011a. PrimoGENI: Integrating real-time network simulation and emulation in GENI. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS'11)*. 1–9.

Nathanael Van Vorst, Ting Li, and Jason Liu. 2011b. How low can you go? Spherical routing for scalable network simulations. In *Proceedings of the 19th IEEE Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'11)*. 259–268.

Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet 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.

Tao Ye, Shivkumar Kalyanaraman, David Harrison, Biplab Sikdar, Bin Mo, Hema Tahilramani, Ken Vastola, and Boleslaw Szymanski. 2001. Network management and control using collaborative on-line simulation. In *Proceedings of the IEEE International Conference on Communications (ICC'01)*.

Yin Zhang and Nick Duffield. 2001. On the constancy of Internet path properties. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW'01)*. 197–211.

Junlan Zhou, Zhengrong Ji, Mineo Takai, and Rajive Bagrodia. 2004. MAYA: integrating hybrid network modeling to the physical world. *TOMACS* 14, 2 (2004), 149–169.

Received MONTH YEAR; revised MONTH YEAR; accepted MONTH YEAR