Toward Scalable Emulation of Future Internet Applications with Simulation Symbiosis

Jason Liu, Cesar Marcondes[†], Musa Ahmed, and Rong Rong

Florida International University, Miami, Florida, USA

[†]Federal University of Sao Carlos, Sao Carlos, SP, Brazil

***Abstract*—Mininet is a popular container-based emulation environment built on Linux for testing OpenFlow applications. Using Mininet, one can compose an experimental network using a set of virtual hosts and virtual switches with flexibility. However, it is well understood that Mininet can only provide a limited capacity, both for CPU and network I/O, due to its underlying physical constraints. We propose a method for combining simulation and emulation to improve the scalability of network experiments. This is achieved first by applying the symbiotic approach to effectively integrate emulation and simulation for hybrid experimentation. For example, one can use Mininet to directly run OpenFlow applications on the virtual machines and software switches, with network connectivity represented by detailed simulation at scale. We also propose a method for using the symbiotic approach to coordinate separate Mininet instances, each representing a different set of the overlapping network flows. By effectively distributing network emulation among separate machines, one can significantly improve the scalability of the network experiments.**

***Index Terms*—Network emulation; network simulation; symbiotic simulation; openflow; software-defined networking**

I. Introduction

During the last ten years, significant advances have been made in Future Internet Architecture (FIA) design and cyber-infrastructure development. Large-scale coordinated efforts (such as [1]–[4]) with bold ideas, innovative and oftentimes disruptive designs about next-generation networks have been proposed, in order to provide secure, high-performance, and ubiquitous services for applications of the future. These efforts coincide with prominent research trends, such as data center networking (DCN), software-defined networking (SDN), and network function virtualization (NFV), and promise emerging network capabilities, such as deep network programmability, resource slicing and federation.

Essential to the FIA research is the development of network testbeds that can validate key design decisions and expose operational issues at scale. For example, the Global Environment for Network Innovations (GENI) has been a community-based effort for building a collaborative and exploratory network experimentation platform for studying future network applications [5]. Follow-up efforts include various cyber-infrastructure design, development, and build-out projects, such as NSFCloud [6], [7], for building mid-scale cloud-computing testbeds in the U.S. There are similar attempts made in European Union, Japan, Brazil, and other nations.

While all these efforts would pave the way for the network researchers (as well as the network engineers) to validate design and implementation issues directly on the cyber-infrastructure testbeds, one needs to understand the deficiencies of solely relying on real-world implementation and physical deployment in network studies. We illustrate this important issue through a few hypothetical examples:

- A new robust map-reduce algorithm [8] needs to be evaluated for multi-tenant cloud computing environments. The performance of the algorithm depends on the job characteristics (such as the distribution on the number of jobs and the individual job sizes), as well as the configuration and stability of the available resources of the cloud platform. One would find it extremely time-consuming to explore the entire algorithmic parameter space on physical testbeds; let alone the highly diverging cloud configurations. Although, with the availability of NSFCloud, for example, one can study a few real instances of the algorithm at play with specific parameter and configuration settings, it is still difficult to extrapolate the overall robustness of the proposed algorithm.
- A novel enterprise network traffic engineering solution based on OpenFlow [9], which uses opportunistic traffic load balancing and multi-path schemes to increase the throughput of heavy-hitter flows, has been proposed and evaluated on the existing campus cyber-infrastructure build-out. The algorithm is shown to be heavily dependent on an efficient exchange of control-plane information/-knowledge between the participating ISPs. Important questions remain unanswered—for example, whether this algorithm would perform well under strenuous (and oftentimes unpredictable) traffic conditions, whether the algorithm would be resilient for situations caused by partial deployment with varying proportions of non-cooperative entities, and whether the algorithm could scale out to a larger number of ISPs beyond the small-scale cases examined by a physical testbed.
- A data center transport-layer protocol has been proposed (similar to [10]), which is expected to both reduce flow completion time and increase data throughput. The algorithm has been implemented and tested in a small-scale homespun DCN testbed; one needs to know whether it is ready for deployment in a production data center. Before that, however, one would like to investigate the

algorithm's optimal performance conditions for the large data center with high bisection network capacity and also with various traffic loads with known stochastic properties. The challenge is that one cannot test the algorithm directly on the production data center network.

These examples highlight some of the intrinsic limitations of cyber-infrastructure testbeds. However useful, they are limited in scale; it is thus difficult, if at all possible, to reveal scaling properties and robustness issues. They also lack flexibility: it is cumbersome and time-consuming to set up experiments to explore the design and configuration space given the large set of control parameters and system configurations. One would also find it difficult to test algorithms and applications beyond the existing setup of the physical environment. This would in turn limit the researcher's ability to investigate network applications under alternative conditions and ask what-if questions.

The network community obviously needs to overcome these problems in order to achieve meaningful research. Previously we conducted a brief survey of SIGCOMM papers from year 2007 to 2013 [11]. We observed that the use of physical testbeds and simulation accounts for a large proportion of the evaluative work. Although emulation provides the flexibility of subjecting real applications under various test scenarios, it is still limited in scale and in the traffic handling capacity. We also observed that physical testbeds and simulation are often used in complementary roles in the evaluative studies. In a common scenario, a researcher would use simulation (often a simplistic model) to evaluate the key functions under various operating network conditions, and then use a controlled physical testbed for small-scale real-world studies. In another common scenario, a researcher would use a physical testbed to conduct small-scale studies, and then resort to simulation (again with simplified models) for speculating scaling properties. There are two problems associated with the popular approach of using physical testbeds and simulation in isolation.

First, the researchers typically use simplified simulation models for scalability studies and for exploring diverse scenarios and parameter settings. To reduce cost, the algorithm or the protocol under investigation is represented only in essence; and the model is usually polished to remove "unnecessary" implementation details. Simple models usually are not rigorously validated and therefore the results can be questionable. Second, even if the the target network protocol or application is faithfully implemented in simulation, it needs to examined in the context of other coexisting protocols and applications. For example, an enterprise network traffic engineering solution can be heavily dependent upon the behaviors of the users and the characteristics of the prevailing applications. Developing detailed simulation models for all would be prohibitive.

The above problems call for a method to organically integrate physical testbeds and simulation/modeling for network experimentation. Whereas physical testbeds provide the real system environment for evaluating network applications *in-situ* with needed operational realism and with live network traffic conditions, simulation is able to scale to represent large networks and incorporate complex stochastic models with flexibility, including, for example, network-wide traffic characterization, user population and mobility, high-level application behavior and user demand, system failures, and so on.

Previously we proposed a symbiotic approach to combine both simulation and emulation [12]. We developed a prototype that consists of two parts: a simulation system and an emulation system. 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. We use *the emulation system* to inspect the detailed behavior of the real applications, where a number of nodes in the target network can be selected as "emulated" nodes to run unmodified software directly on the virtual machines with specified operating systems, real network stacks, libraries and software tools. In this way, 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 considering real network traffic generated by the unmodified software directly executed on real systems. The emulation system benefits from the simulation system by receiving network updates 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 diverse virtual network settings.

In this paper, we apply the symbiotic approach to improve the capabilities of a specific network emulator, called Mininet [13], [14]. Mininet is a container-based emulation environment built for Linux for testing OpenFlow applications [9]. Using Mininet, one can create network experiments using a set of virtual hosts and virtual switches connected as an arbitrary network. Mininet uses the native Linux namespaces to represent the virtual hosts. It is a lightweight container-based virtualization solution, based on which one can create relatively large virtual networks with hundreds and even thousands of virtual machines on a single physical machine. The containers can be connected to the instances of the Open vSwitch (OVS) [15], which is a production-quality software switch augmented with OpenFlow capabilities for experimentation with SDN applications.

By implementing the symbiotic approach in Mininet, we enable large-scale hybrid SDN/OpenFlow experiments. For example, one can use Mininet to directly run SDN applications using the virtual machines and software switches controlled by real OpenFlow controllers. These virtual machines can be a part of a large-scale network simulated by the network simulator for representation of diverse network scenarios. This would allow us to efficiently and accurately incorporate complex network models, such as different network topologies, network-wide traffic matrices, as well as stochastic models to describe user demands, mobility, and applications behaviors.

Technically, this paper makes two contributions. First, we present a specific design and implementation of the symbiotic construct that can effectively integrate the emulation testbed with a network simulator so that one can conduct hybrid at-scale experiments and test applications and algorithms easily with various system configurations and design parameters. Second, we present a method for using the symbiotic approach to coordinate separate Mininet instances to run (with different virtual machines and switches) on distributed machines with overlapping traffic on shared links of the target virtual network. In this case, we can significantly increase the scalability of the network experiment. We conduct preliminary experiments to mainly assess the feasibility of our proposed approach.

The rest of the paper is organized as follows. In section II, we provide the background and discuss existing work related to physical network testbeds, network emulation and simulation. In section III, we specifically review the symbiotic simulation approach. In section IV, we describe the design of our proposed system for incorporating the Mininet emulator with a high-fidelity real-time network simulator using the symbiotic approach...

II. Background and Related Work

The ability to conduct high-fidelity network experiments and allow easy exploration of design space is crucial for studying future network systems and their complex behaviors. Existing network testbeds offer different capabilities, in terms of *realism*, for reproducing accurate system and network effects; *scalability*, for capturing at-scale network operations; and *flexibility*, for creating diverse network scenarios.

A. Physical Network Testbeds

Physical testbeds provide a real system environment for evaluating network applications directly on the testbeds with the needed operational realism and with live network traffic. Physical testbeds can be further divided into production testbeds and reconfigurable testbeds. Production testbeds (such as Internet2 [16] and ESnet [17]) support live network experiments; however, they allow only "safe" experiments that do not disrupt normal operations, and they provide only a small and iconic version of the entire internet. Comparatively, reconfigurable testbeds provide far better flexibility. PlanetLab [18] is a well-known reconfigurable testbed consisted of machines distributed across internet and shared by researchers simultaneously conducting multiple experiments. An experiment can run on a subset of machines creating an overlay network (called a slice). Similar concepts have been adopted by GENI [5], which is a community-driven research and development effort to build a collaborative and exploratory network experimentation platform. GENI capitalizes on the success of previous efforts, by providing an overarching technology to bring all different network testbeds together as a single platform for building and testing new network designs and new technologies fundamental to future internet.

Recently, there also has been significant investment in cyber-infrastructure development and build-out for the entire network research community (e.g., [6], [7], [19]). Physical testbeds provide realism, but still lack flexibility and scalability. Although reconfigurable, it is difficult to test applications beyond the existing setup and configuration of the underlying physical environment, which is limited in scale and capacity. It would be difficult to realize experiments with the number of nodes significantly larger than the available nodes (either physical or virtual machines), and with the capacity of interconnectivity higher than the available bandwidth.

B. Network Emulation Testbeds

Emulation testbeds support "traffic shaping" by introducing artificial delays and losses to packets to mimic their experience of traversing the network routers and links [20]. An emulation testbed can be built on a variety of computing infrastructures, including dedicated compute clusters (such as ModelNet [21] and EmuLab [22]), distributed platforms (such as VINI [23]), and special programmable devices (such as ONL [24] and ORBIT [25]). Mininet [13] is also an emulation testbed using Linux containers and traffic control (`tc`).

Emulation testbeds provide a good balance between flexibility and realism, whereas real applications can run directly in a designated operating environment, and traffic between them can be "shaped" according to the network delay and bandwidth constraints. However, like physical testbeds, emulation testbeds are also limited in scale and capacity. For example, we observe that there still exists a stringent limitation in the amount of traffic that can be emulated in real time. The aggregate traffic on each physical machine cannot go beyond a certain rate, which depends on the machine type (typically, a few gigabits per second). Mininet has also an extension to allow running on multiple machines in a distributed environment [26]. However, the traffic between the physical machines has to be limited by the available connection bandwidth. For experiments that induce heavy traffic, Mininet cannot produce reliable results.

C. Network Simulators

Simulation plays an important role in network design and evaluation. Many network simulators have been developed in the past, e.g., NS-2 [27], NS-3 [28], OPNET [29], and OMNeT++ [30]. Parallel simulation is a technique of running a single discrete-event simulation program in parallel [31]. It can harness the collective power of parallel computers to run complex large-scale models and thus can be successfully applied to increasing the performance and scalability of network simulations, e.g., SSFNet [32], GTNets [33], ROSSNet [34], and GloMoSim [35]. Simulation can be effective at capturing large-scale system design, and answering what-if questions. Using parallel simulation, one is able to handle large-scale models. However, simulation often lacks realism and requires extensive efforts in validation. Developing detailed models is also known to be labor-intensive.

To deal with these issues, there have been two prominent methods. One is to directly incorporating protocol implementations in simulation [36]–[38]. This technique is called *direct-execution simulation*, which includes compile-time techniques (which involve little or only moderate modification to the source code), link-time techniques (such as using linker wrapper functions to replace functions related to communication and timing), and run-time techniques (such as binary code modification, preloading dynamic libraries, or using packet capturing facilities). There are two major issues with this approach. First, reproducing detailed behavior for all network protocols and applications in simulation would be too costly to realize for full-scale network experiments. Second, in cases where one may desire high-level models, such as random traffic generation and stochastic failures, implementing detailed network models does not automatically translate to an accurate representation of high-level behaviors.

Another technique is to allow network simulation to operate in real time so that the virtual network can interact with real network applications. This technique is called *real-time simulation*. Most real-time simulators (e.g., [36], [39]–[45]) are based on existing simulators augmented with emulation capabilities. To support real-time simulation, the simulator is modified to be able to regulate the virtual time advancement; in parallel simulation, the issue becomes an effective scheduling of the logical processes with respect to real time [46]. Although real-time simulation allows hybrid network experiments involving both simulated and physical network components, the scale of the network experiments is constrained by the I/O capacity of the simulator for exchanging network packets with the physical system [47].

III. Symbiotic Simulation

Symbiosis is originally a biological term to describe a type of 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"* [48].

For network experimentation, ROSENET [49] can be seen as the first attempt to promote the symbiotic relationship between simulation and emulation. ROSENET combines the parallel network simulator, GTNetS [33], and the Linux-based network emulator, NIST Net [50], which can run at a different location. During the experiment, the network simulator periodically updates the emulator with the link statistics (such as packet delays, jitters, and packet losses). The network emulator also periodically provides the network simulator with a summary of the real traffic situation. Experiments have shown that ROSENET is capable of emulating a single bottleneck link conducting non-responsive traffic (i.e., UDP traffic) generated by real applications. We also proposed another symbiotic approach to more effectively combine simulation and emulation [12]. In the following, we describe this approach in more detail, which we apply to allow scalable Mininet emulation of future internet applications.

A network experiment consists of a virtual network with an arbitrary topology, potentially with a large number of hosts and routers. For a specific experiment, we can examine a subset of network protocols and applications by directly running them on an emulation testbed. Fig. 1 shows an example where the real applications are running on two physical hosts, H1 and H2. To test them, we specify a simulated network, which contains virtual hosts, h1 and h2, that correspond to the two physical hosts. We "modulate" the real network traffic

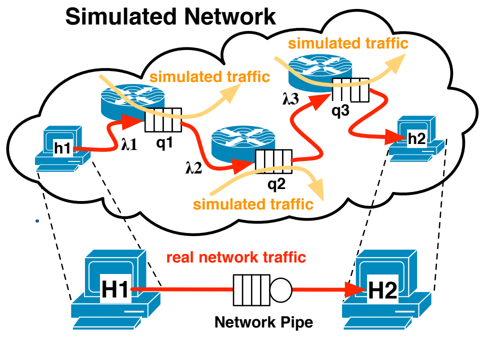

Fig. 1: A symbiotic network experiment.

between the physical hosts using statistics collected from the simulated network. More specifically, we use a facility, called the "network pipe", to represent the sequence of network queues *supposed* to be traversed by the real network traffic if it were placed on the simulated network. The example shows one network pipe consisted of three simulated network queues: q_1, q_2, and q_3. (For brevity, we focus only on the forward traffic from H1 to H2 and ignore the traffic in the reverse direction.) It is important to know that with symbiotic simulation, the network packets generated between the physical hosts, from H1 to H2, do not need to be captured and simulated individually as in real-time simulation. Instead, the symbiotic system captures only the traffic demand at the physical hosts and then sends this information to the simulator so that the simulator can regenerate the same traffic and model its effect over the simulated network (e.g., packet delays and losses).

The network pipe is a mechanism used by the emulator to reflect the traffic conditions in the simulated network so that the packet delays and losses can be applied to the real traffic. In [12], we derived a closed-form solution, for which we only capture the main results below.

In general, let $q_1, q_2, \cdots, q_n$ be the list of network queues in simulation that are supposed to be traversed by the real network traffic. In simulation, we collect three measurements for each queue q_i and periodically report them to the emulator:

1) We measure p_i, which is the average drop probability due to buffer overflow;
2) We measure λ_i, which is the arrival rate of the regenerated emulated network flow; and
3) We measure w_i, the average packet queuing delay.

Once these measurements are propagated to the emulator, we can calculate the packet drop probability for the network pipe:

$$p = 1 - \prod_{i=1}^{n}(1 - p_i) \tag{1}$$

And we can calculate the service rate (i.e., the bandwidth) of the network pipe:

$$\mu = \frac{\lambda_p(\Delta T + W_2 - W_1)}{\Delta T \left(1 + W_1\lambda_p - \sqrt{1 + W_1^2\lambda_p^2}\right)} \tag{2}$$

where $\lambda_p = \min_{1\le i\le n}\{(1-p_i)\lambda_i\}$, which is the minimum effective arrival rate at all queues; ΔT is the sample interval (say, 100ms), which is also the interval at which the simulator updates the emulator with the measurements; $W_1 = \sum_{1\leq i\leq n} w_i$ is the total queuing delay through the n queues measured in simulation; and W_2 is the average packet queuing delay through the corresponding network pipe measured in emulation.

After calculating p and μ, we can apply them at the network pipe in the emulator, which is essentially a first-in-first-out queue installed between the physical hosts. The network pipe will randomly drop packets according to the set probability p and process packets according to the given bandwidth μ, which will effectively add queuing delays to the packets as they go through the network pipe. In this paper, we will show how to implement the network pipe using the Linux traffic control (tc) facility.

In [12], we conducted extensive experiments to show that this symbiotic approach is able to produce accurate results. Using this approach, we can test the real applications running on the physical environment with different network scenarios—such as running on different network topologies, testing with diverse traffic intensity, and using different workload and user demand. In this way, we can enable high-fidelity high-performance large-scale network experiments by combining both simulation and emulation testbed, using simulation for the full-scale detailed network representation and using emulation testbed for directly executing network applications for real. On the one hand, emulation testbeds can execute real applications, operate with real systems, accept real input, produce real output, and respond to real network conditions. They provide the operational realism and fidelity usually unattainable by modeling and simulation. On the other hand, simulation is expedient for constructing and testing models to obtain "the big picture", which would be highly valuable especially when a good understanding of the system's complex behavior is absent. Simulation makes it easy for prototyping, for exploring the design space, for assessing the performance in diverse network settings, and for investigating what-if scenarios.

IV. Mininet Symbiosis

In this section, we discuss our design for integrating the symbiotic approach with Mininet [13], [14].

A. System Overview

Mininet is a popular container-based emulator for testing OpenFlow applications. It uses lightweight OS-level virtualization to emulate the hosts. Each virtual host corresponds to a container attached to a separate network namespace (a mechanism introduced since Linux kernel 2.6.24). Each network namespace can contain a virtual network interface with a distinct IP address along with independent functions of the TCP/IP stack (such as the kernel routing/forwarding table). The virtual network interfaces can be connected via virtual Ethernet links to the software switches (i.e., OVS instances), augmented with OpenFlow capabilities. An OpenFlow controller can be connected to the OpenFlow-enabled

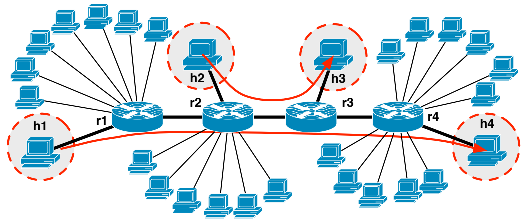

Fig. 2: A target virtual network with emulated traffic identified.

software switches for a full implementation of the software-defined networking experiment. A significant portion of the Mininet implementation is a python library to assist the users to create and maintain the virtual network topology for emulation. Mininet uses `cgroups` for scheduling and resource management so that one can limit the CPU usage for all processes belonging to each container. Mininet also uses `tc`, the Linux traffic control, to control the link properties, such as link bandwidth, packet delay, and packet loss.

A typical procedure for using the symbiotic approach can be shown more easily through an example. Our goal is to execute the target network applications (`iperf` for a simple example) in Mininet containers while creating an illusion that these applications are running on an arbitrary network. Our approach starts by first having the user to specify a network model, which includes a simulated network topology (on which the target real applications are expected to run), as well as network protocols and applications, and how they are engaged during the experiment. For example, one can incorporate complex network topologies with stochastic models for network-wide traffic generation. Fig. 2 shows a simple virtual network with four routers connecting many hosts.

Next, the user can identify a subset of hosts to be emulated in Mininet (we call them *emulated hosts*). They will be instantiated as containers and therefore capable of directly running the target network applications. To reduce overhead, we also ask the user to identify flows that will be generated between the emulated hosts during the experiment (we call *emulated flows*). This can significantly reduce the facilities that need to be maintained for symbiosis. In the example shown in Fig. 2, we specify two emulated flows: one from h1 to h4, and the other from h2 to h3. Here again, for brevity, we only show one-directional traffic. Most flows (such as TCP) would be bi-directional, in which case the user would need to specify the flows for both directions.

Afterwards, we invoke a process, called *downscaling*, in which the original full-scale network simulation model together with the identified emulation traffic is processed to produce an reduced emulation model for Mininet. The downscaling process first prunes the original network model and remove all hosts, routers, and links not traversed by emulated traffic, since they are not needed in emulation involving real traffic. It then compresses the pruned topology by combining the intermediate nodes and links visited by the same set of emulated flows into a single network pipe. For example, the network segment between h1 and r2 in Fig. 2 can be

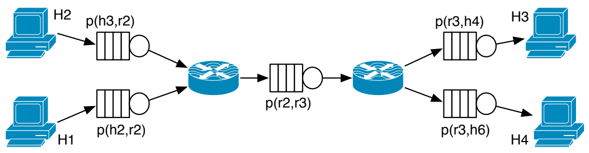

Fig. 3: A downscaled network model to run in Mininet.

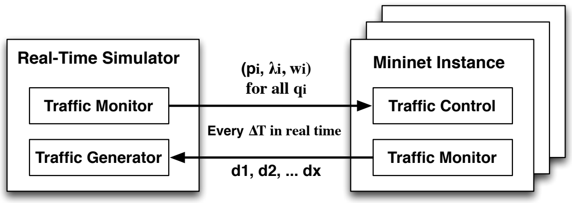

Fig. 4: Mininet symbiosis setup.

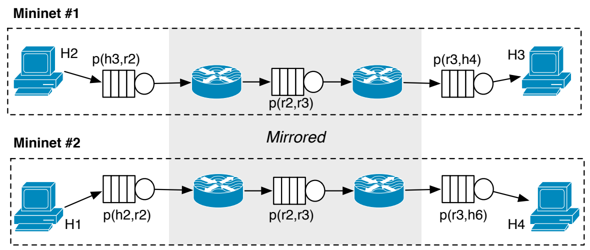

Fig. 5: Downscaled models for two Mininet instances.

compressed into one network pipe. The downscaled emulation model (only forwarding portion) of the same example is shown in Fig. 3, which consists of the four emulated hosts and two switches, connected by five network pipes.

Our symbiotic system consists of a simulation system and an emulation system running side by side. The simulation system is a real-time network simulator (we use PrimoGENI [45] for our prototype implementation), and the the emulation system consists of one or more Mininet instances, potentially running on separate machines (see Fig. 4). Communication between the real-time network simulator and the Mininet instances is achieved via TCP connections, whereas the simulator functions as the server and each Mininet instance as a client. The real-time network simulator runs the original full-scale network; as such, it needs to implement necessary network elements (such as routers, hosts, network interfaces and links) and common network protocols (such as IP, TCP, UDP, and others). In addition, two components are added to the simulator to facilitate synchronization with the Mininet instances: a traffic monitor and a traffic generator. The traffic monitor is used to collect measurements at each queue q_i traversed by the emulated flows, which include the packet drop probability p_i, the arrival rate of emulated flows λ_i, and the queuing delay w_i. These measurements are collected periodically every ΔT units of time and then sent to the corresponding Mininet instances. The traffic generator receives information from Mininet about the traffic demand d_k from applications for each emulated flow k in terms of the number of bytes requested to be sent during the last interval. Upon receiving this information, the simulator generates the emulated flows by initiating the corresponding TCP or UDP sessions in simulation with the same demand size accordingly.

In Mininet, the emulated hosts are instantiated as Linux containers with separate network namespaces, and the switches are represented by OVS instances. The virtual Ethernet (`veth`) pairs are used to represent the links augmented with the Linux traffic control (`tc`) for managing the link properties. Linux `tc` is a set of tools (included since kernel 2.2) to allow users to have fine-grained control over the packet transmission. Linux `tc` consists of different queuing mechanisms, easily composable for handling more complex situations (including packet mangling, IP firewalling, and bandwidth metering). We use `tc` for setting the link bandwidth, the packet delay, and the random packet loss probability. More specifically, we statically set the link delay as the cumulative propagation delay of the links between the consecutive queues that constitute the network pipe. We modify the packet loss probability and the link bandwidth dynamically during the experiment using the measurements from simulation (Equations 1 and 2).

Note that our symbiotic approach can easily support distributed emulation, where multiple Mininet instances can operate in parallel, each handling a different set of emulated flows. For the example shown in Fig. 2, the flow from h2 to h3 can be emulated in a separate Mininet instance from the one used for emulating the flow from h1 to h4. The downscaled models for the two Mininet instances are shown in Fig. 5. Note that the state of the network pipe, $p(r2, r3)$, is mirrored on both instances; that is, they will be controlled by the simulator with the identical link properties.

In the following sections, we discuss the detailed design and implementation of the symbiotic constructs.

B. *Regenerate Emulated Flows in Simulation*

A unique aspect of our symbiotic approach, different from the traditional real-time network simulation method, is that real network packets in the emulated system that need to be simulated on the full-scale network do not need to be captured individually to reproduce the same traffic effect (in order to calculate their packet delays and packet losses accordingly). Instead, the symbiotic approach reproduces the effect of the real traffic flows in simulation by having the emulation system to capture the interval-based traffic demand at the traffic source (preferably at the application/transport interface) and then reproduce the demand traffic using the corresponding simulated TCP or UDP. In this case, we can minimize the synchronization overhead between the simulator and the physical system.

There are several ways to collect traffic demand agnostic of specific application behaviors. A somewhat complicated method involves creating a wrapper to a socket library and collect the read/write and send/receive calls from the applications right before they invoke the kernel functions. Another possibility is to monitor the state of a TCP connection using the `tcpprobe` kernel module. One can monitor the `SND.NXT` pointer, which represents the sequence number of the first unsent byte of user data and then calculates the difference between consecutive packets to estimate the demand over an interval. The drawback of this approach, however, is that this demand (taken from consecutive packet departures) represents a transmission that has already taken place from the perspective of the transport layer. With zero lookahead for reproducing the traffic, the system would be sensitive to the latency between the simulator and the Mininet instances.

Our traffic monitor on Mininet uses a simple and lightweight solution to capture the traffic demand at each Linux container (emulated host). We chose to use a tracing tool for the Linux system calls, called `strace`. One can use `strace` to collect the traffic demand at the interface between the applications and the transport layer. Network system calls—such as `connect`, `accept`, `read` and `write`, and others—invoked by applications running inside the containers can be captured and parsed continuously to arrive at the application traffic behavior. The following shows a snippet of the `strace` output for running `iperf` data transfer inside a container. We can see that the `connect` system call from process 15742 (which is the `iperf` process) established a TCP connection with another container with the IP address 10.0.0.2. The subsequent system calls to `write` indicate the request to send 131,072 bytes of data each time via the TCP connection.

```
[pid 15742] connect(3, {sa_family=AF_INET,
    sin_port=htons(5001), sin_addr=
    inet_addr ("10.0.0.2")}, 16) = 0
[pid 15742] write(3, ... 131072 <unfinished ...>
[pid 15742] <... write resumed> ) = 131072
[pid 15742] write(3, ... 131072) = 131072
[pid 15742] write(3, ... 131072 <unfinished '...>
```

All socket-related system calls can be captured in this way. For certain system calls, such as `write` , we need to distinguish the calls handling data transmissions over sockets from those handling regular file IOs. This can be achieved by checking the state of the file descriptor of a process in the Linux /proc system. For example, the information for the `iperf` process can be located at `/proc/15742/fd/3`. Each container in Mininet starts with a bash shell. To speed up the process, we can cache the lookups for child processes spawned from the container's bash process, so that one can quickly identify the connections used by the applications running by a child. The use of `strace` is indeed lightweight. In our prototype, we found that the overhead is only around 1% CPU per container.

Once the demands are received, in order to generate the same amount of simulated traffic, we instantiate a "symbiosis application" at each of the emulated hosts at the start of the simulation. For each emulated flow, the symbiosis application at the sender host creates a socket connection (either a TCP or UDP session) with the receiver also at the start of the simulation. During the experiment, upon receiving an updated traffic demand from the emulator, the simulator simply issues a send command with the same size for the corresponding session at the sender host. Note that in order to preserve the same traffic behavior, the real-time network simulator must support the same set of TCP variants commonly used in the physical platform. PrimoGENI contains fourteen TCP variants that can be found commonly in use today, including New Reno, BIC, CUBIC, and others. These TCP congestion control mechanisms have been previously ported from the Linux implementation and have been tested extensively [51].

C. Actuate Network Pipes

As mentioned earlier, the simulator is instrumented to generate the queuing statistics at the simulated network interfaces that constitute the network pipes, including the packet loss probability, the packet arrival rate, and the average queuing delay. These measurements are distributed periodically to the corresponding Mininet instances that handle the network pipes.

The network pipes are created with Linux `tc` using the specific token bucket queuing disciplines. The delay of a network pipe is fixed at the the system configuration; the value is the total propagation delay of the network links that constitute the network pipe in the simulation model. The packet drop probability and service rate need to be changed during the experiment. It is important that, once the simulation measurements reach the Mininet instances periodically (say every 100ms), it is necessary to change the corresponding `tc` link properties immediately so that the real traffic flows can reflect the traffic conditions in the simulated network. In Mininet, we created a separate thread to receive the periodic updates from the simulator: (p_i, λ_i and w_i), for each simulated queue q_i traversed by the emulated flows.

The packet drop probability can be applied directly using the `replace` primitive in `tc`. Using `tc replace` is fast and convenient. To verify its effectiveness, we tested by executing the `tc show` command immediately after applying `replace` primitive. We did not notice any degradation in traffic performance for all experiments we performed even with update small update intervals.

In order to apply Equation (2) to calculate the new service rate, we need to measure the average packet queuing delay, W_2, through the network pipe. Directly measuring the packet queuing delay by packet can be costly. Instead, we can estimate the average queuing delay by sampling the instantaneous queue length gathered from the `tc` statistics. We created a collector mechanism that obtains the relevant values from the kernel. Since these values are constantly monitored for the Linux queues in any case, the collector presents no additional overhead. In particular, we capture instantaneous queue lengths in bytes at smaller sample intervals (say, one tenth of the update interval used to synchronize simulation and emulation). We accumulate the samples and average them over the update period. The resulted average queue size is then divided by the service rate to produce the estimated average queuing delay W_2. Finally, we can apply Equation (2) to calculate the new service rate. Again, we use `tc replace` to update the network pipe.

V. Preliminary Experiments

We conducted experiments to test our design using a preliminary implementation. We first study the effectiveness of the mechanisms for reproducing the emulated traffic demand in simulation. In particular, we aim to examine whether the real traffic demand from the virtual machines can be captured accurately by our traffic monitor in Mininet, and whether the new traffic generator module in the simulator can faithfully reproduce the same flows in a timely fashion.

We used a simple dumbbell model, similar to the one shown in Fig. 3. We set the bandwidth of the "bottleneck" link connecting the two routers to be 10 Mbps, and all other "spoke" links to be 1 Gbps. The bottleneck link has a propagation delay of 15 ms while the spoke links all have a propagation delay of 1 ms. We ran the real-time simulator and the Mininet instances on separate machines connected via a gigabit network.

In the first experiment, we manually created two TCP flows using `iperf` one after another with only a few seconds in-between. The two flows were generated from the same emulated host on one side of the dumbbell to a fixed host on the opposite side (thus traversing the bottleneck link). We ran `tcpdump` to capture the packets at both the sender and the receiver, and therefore used the TCP sequence numbers to measure the traffic situation in Mininet. We compare them against the corresponding traffic regenerated in simulation.

We started by using one second as the interval for synchronizing the simulator and the emulator; it's at least one order of magnitude higher than the network latencies one would normally observe over the wide-area network. The result is shown in the left plot of Fig. 6. The staircase behavior of the simulated traffic is due to the large synchronization interval. The traffic demand from Mininet is only reported to the simulator once every second. As a result, the simulator tried to replay the entire one second worth of traffic at the beginning of each interval. Despite this artifact, however, the simulated traffic is shown to be able to track the real traffic quite well.

Next we reduced the synchronization interval from one second to 100 ms and performed the same experiment. The result is shown in the right plot of Fig. 6. The previous staircase behavior of the simulated traffic is no longer apparent. We observe that the simulated traffic can still match with the real traffic, however with a slight decrease in its transfer rate. This is due to an issue with the simulator's traffic generator. In the original design, we extended a simple server-client model in the simulator, where a request message has to be sent from the client to the server, which would cause a slight delay before the data transfer can be effectuated. There are also additional overhead related to the choice of using a smaller segment size for TCP. We are redesigning the simulation traffic generator to remove these problems.

In the next experiment, we studied the effectiveness of our traffic control mechanism in Mininet. In this experiment, we started a long-term TCP flow between two virtual machines using `iperf`. The two virtual machines were connected directly through a virtual Ethernet pair (veth). We used the `tc` commands to regulate the bandwidth of the link in-between by randomly selecting a bandwidth from a set of values: 1 Mbps, 10 Mbps, 100 Mbps, and 1 Gbps.

We changed the bandwidth every second or every 100 ms and measured the average TCP throughput at the corresponding time intervals. Fig. 7 shows the results from a randomly chosen time period during the experiment. The left plot shows the results for changes at one second intervals. The average TCP throughput responds well to the bandwidth changes, except for a few instances (at time 33 and 36 seconds) when the bandwidth is drastically reduced from 1 Gbps to 1 Mbps. `tc` uses token buckets for regulating the packet transmission over the link; the higher than expected throughput is probably due to the backlog. The right plot of Fig. 7 shows the results for changes at 100 ms intervals. The TCP throughput does not seem to track the bandwidth changes as well as in the previous case. This means that regulating the bandwidth at the 100 ms time scale may introduce nontrivial inaccuracies.

VI. Conclusion and Future Work

Symbiotic simulation provides a promising tradeoff, by combining the emulation testbeds, which can feature a more realistic environment for running network applications, and simulation, which can provide more flexible, large, and complex network scenarios. In this paper, we outline a specific design of combining instances of a popular network emulator, called Mininet, with a real-time simulator, called PrimoGENI. We provide a detailed account on the use of low-level mechanisms for implementing the symbiotic approach in the Linux environment.

This paper provides a feasibility study while we currently undergo a full-scale implementation. Our immediate future work is to study the overall effectiveness of our symbiotic method. In particular, the simulator shall be able to connect with multiple Mininet instances to support large-scale experiments. We would like to apply the symbiotic approach to studying bandwidth-intensive OpenFlow applications, which would otherwise be difficult to realize in the traditional simulation or emulation testbeds.

References

[1] MobilityFirst Future Internet Architecture Project, http://mobilityfirst.winlab.rutgers.edu/.

[2] Named Data Networking (NDN) Project, http://www.named-data.net/.

[3] eXpressive Internet Architecture (XIA) Project, http://www.cs.cmu.edu/~xia/.

[4] NEBULA Project, http://nebula.cis.upenn.edu/.

[5] NSF Global Environment for Network Innovations (GENI), http://www.geni.net/.

[6] CloudLab, https://www.cloudlab.us/.

[7] Chameleon - A configurable experimental environment for large-scale cloud research, https://www.chameleoncloud.org/.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'04)*, 2004.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

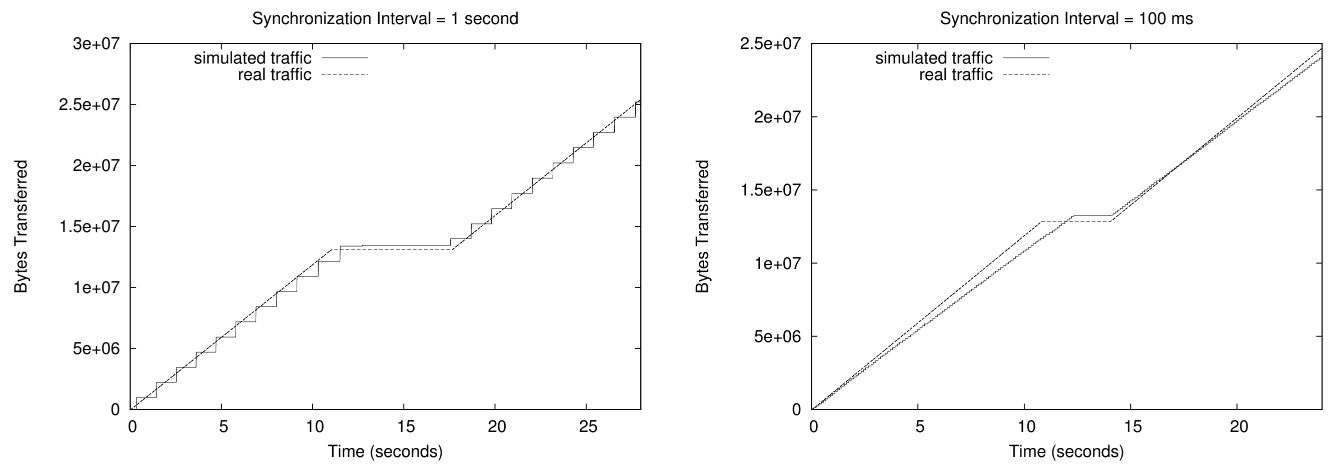

Fig. 6: Reproducing real traffic in simulation.

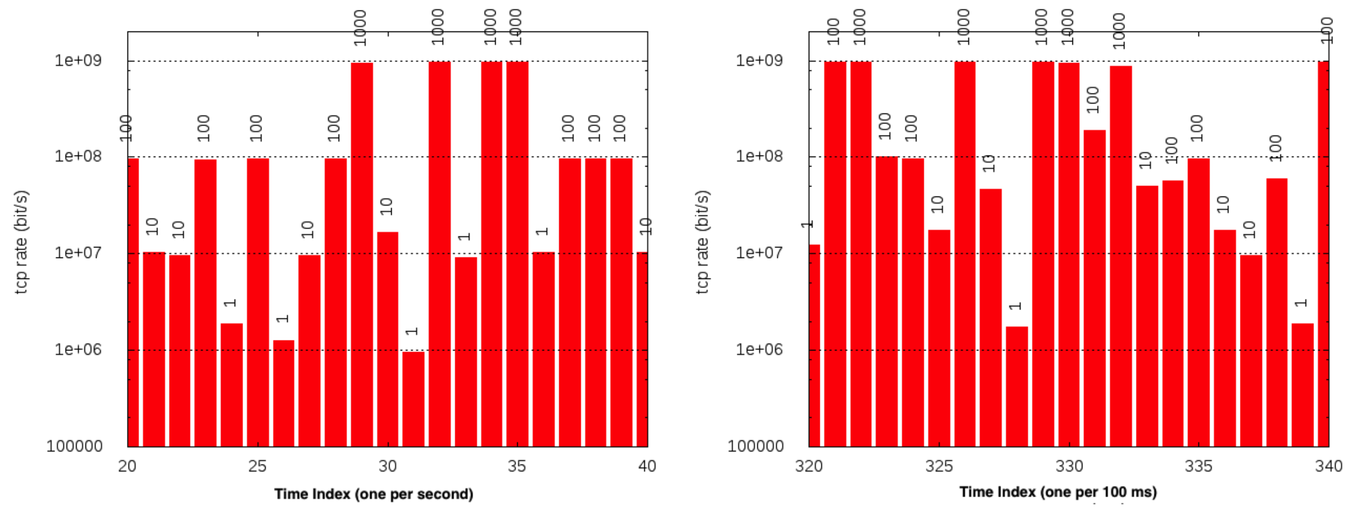

Fig. 7: Controlling traffic in Mininet.

[10] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. Liu, and F. Dogar, "Friends, not foes - synthesizing existing transport strategies for data center networks," in *Proceedings of the 2014 ACM SIGCOMM Conference (SIGCOMM)*, 2014.

[11] T. Li and J. Liu, "Cluster-based spatiotemporal background traffic generation for network simulation," *ACM Trans. Model. Comput. Simul.*, vol. 25, no. 1, pp. 4:1–4:25, 2014.

[12] M. A. Erazo and J. Liu, "Leveraging symbiotic relationship between simulation and emulation for scalable network experimentation," in *Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS)*, 2013, pp. 79–90.

[13] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM Workshop on Hot Topics in Networks*, 2010, pp. 19:1–19:6.

[14] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *CoNEXT*, 2012, pp. 253–264.

[15] "Open vSwitch," http://openvswitch.org/.

[16] Internet2, http://www.internet2.edu/.

[17] ESnet: Energy Sciences Network, http://www.es.net/.

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

[19] GENI Racks, http://groups.geni.net/geni/wiki/GENIRacksHome.

[20] L. Rizzo, "Dummynet: a simple approach to the evaulation of network protocols," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, January 1997.

[21] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, "Scalability and accuracy in a large scale network emulator," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002, pp. 271–284.

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

[23] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *SIGCOMM*, 2006, pp. 3–14.

[24] J. DeHart, F. Kuhns, J. Parwatikar, J. Turner, C. Wiseman, and K. Wong, "The open network laboratory," *ACM SIGCSE Bulletin*, vol. 38, no. 1, pp. 107–111, 2006.

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

[26] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks,"

in *Proceedings of the 2014 IFIP Networking Conference*, 2014, pp. 1–9.

[27] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, May 2000.

[28] The ns-3 Project, http://www.nsnam.org/.

[29] OPNET Technologies, Inc., http://www.opnet.com/.

[30] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'01)*, June 2001.

[31] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[32] J. Cowie, D. Nicol, and A. Ogielski, "Modeling the global Internet," *Computing in Science and Engineering*, vol. 1, no. 1, pp. 42–50, January 1999.

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

[34] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: a high-performance, low memory, modular time warp system," in *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'00)*, May 2000, pp. 53–60.

[35] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "GloMoSim: a scalable network simulation environment," Department of Computer Science, UCLA, Tech. Rep. 990027, May 1999.

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

[37] J. Liu, Y. Yuan, D. M. Nicol, R. S. Gray, C. C. Newport, D. Kotz, and L. F. Perrone, "Simulation validation using direct execution of wireless ad-hoc routing protocols," in *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, May 2004, 7-16.

[38] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous, "Direct code execution: Revisiting library OS architecture for reproducible network experiments," in *CoNEXT*, 2013, pp. 217–228.

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

[40] R. Simmonds and B. W. Unger, "Towards scalable network emulation," *Computer Communications*, vol. 26, no. 3, pp. 264–277, February 2003.

[41] J. Zhou, Z. Ji, M. Takai, and R. Bagrodia, "MAYA: integrating hybrid network modeling to the physical world," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 2, pp. 149–169, April 2004.

[42] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier, "RINSE: the real-time immersive network simulation environment for network security exercises (extended version)," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 82, no. 1, pp. 43–59, 2006.

[43] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A real-time network emulator," in *MILCOM*, 2008, pp. 1–7.

[44] D. Nicol, D. Jin, and Y. Zheng, "S3F: the scalable simulation framework revisited," in *WSC*, 2011, pp. 3288–3299.

[45] N. V. Vorst, M. Erazo, and J. Liu, "PrimoGENI for hybrid network simulation and emulation experiments in GENI," *Journal of Simulation*, vol. 6, no. 3, pp. 179–192, 2012.

[46] J. Liu, "Real-time scheduling of logical processes for parallel discrete-event simulation," in *Proceedings of the 2013 Winter Simulation Conference (WSC'13)*, 2013.

[47] J. Liu, Y. Li, N. V. Vorst, S. Mann, and K. Hellman, "A real-time network simulation infrastructure based on OpenVPN," *Journal of Systems and Software*, vol. 82, no. 3, pp. 473–485, March 2009.

[48] R. Fujimoto, D. Lunceford, E. Page, and A. M. Uhrmacher, "Grand challenges for modeling and simulation," Schloss Dagstuhl, Tech. Rep. 350, 2002.

[49] Y. Gu, "ROSENET: A remote server-based network emulation system," Ph.D. dissertation, Georgia Institute of Technology, 2007.

[50] M. Carson and D. Santay, "Nist net: A linux-based network emulation tool," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 111–126, 2003.

[51] M. A. Erazo, Y. Li, and J. Liu, "SVEET! a scalable virtualized evaluation environment for TCP," in *Proceedings of the 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops (TRIDENTCOM'09)*, 2009, pp. 1–10.