Scalable Emulation of SDN Applications with Simulation Symbiosis

Jason Liu^1 and Cesar Marcondes^2
^{1} Florida International University, Miami, Florida, USA
^{2} Federal University of São Carlos, São Carlos, SP, Brazil

I. Overview

Software Defined Networks (SDN) is an inspiring technology that promises to make networks both easier and cheaper to operate. Therefore, SDN makes networks flexible and programmable, by using a language so simple that average software engineers can comprehend and use to quickly reconfigure centrally the network, without dealing with distributed protocols manually configured that are prone to errors and time-consuming.

The capability of studying and experimenting with large-scale Future Internet SDN applications is of significant importance. 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 [1]. Follow-up efforts include various cyber-infrastructure design, development, and build-out projects, such as NSFCloud [2], [3], 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 [4] 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.
- A novel enterprise network traffic engineering solution based on OpenFlow [5], which uses opportunistic traffic load balancing and multi-path schemes to increase the throughput of heavy-hitter flows, has been proposed. Important questions remain unanswered—for example, whether this algorithm is robust under various traffic conditions, whether the algorithm would perform well due to partial deployment with varying proportions of non-cooperative entities, and whether the algorithm could scale out to a larger number of ISPs.
- A data center transport-layer protocol has been proposed (similar to [6]), 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.

These examples highlight some of the intrinsic limitations of cyber-infrastructure testbeds. Another popular method is to use emulation. For example, the current widespread use of container-based emulation combined with a software switch SDN capable have change the way research is conducted in SDN. Mininet is a popular container-based emulation environment built on Linux for testing OpenFlow applications. Using Mininet, one can create network experiments using a set of virtual hosts and virtual switches connected as an arbitrary network. With Mininet, it is relatively easy to build and execute experiments. However, it is well-known that Mininet only provides a limited capacity for both CPU and network I/O. Consequently, it does not work well on large scenarios and topologies with large volume of traffic, even if used in a cluster environment.

We propose a method for applying a symbiotic approach that combines simulation and emulation for improving the scalability of network experiments. The essential aspect of our approach is to remove a large portion of traffic from emulation. Rather than re-creating each network packet generated from applications, we can capture in real time the aggregate traffic demand of these applications and simulate the corresponding effect on the network queues (effective bandwidths, packet loss, and packet delays), that can affect other applications.

With the symbiotic approach, we achieve two objectives. First, we can effectively integrate emulation and simulation so that one can conduct hybrid network experiments with flexibility and scale. For example, we can use Mininet to directly run SDN applications using the virtual machines and software switches, whereas the connections between the virtual machines can be integrated with large-scale network models performed in simulation. In this case, the SDN applications can

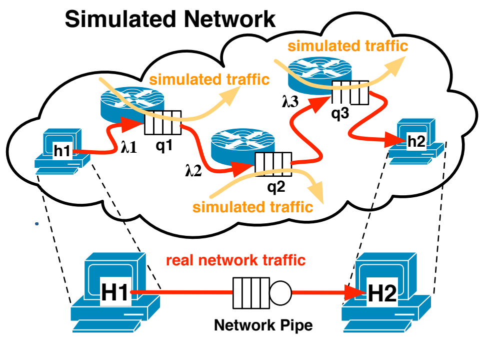

Fig. 1. A symbiotic network experiment.

be tested under diverse virtual network scenarios. Second, the symbiotic approach can be used to coordinate separate Mininet instances, each representing a set of different yet possibly overlapping network flows. In this case, we can significantly increase the scalability of the network emulation experiments in a cluster environment.

II. Symbiotic Simulation

We propose a symbiotic approach that can effectively combine simulation and emulation [7]. 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 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 [7], 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 from individual loss drop probabilities of the constituent queues. And we can calculate the service rate (i.e., the bandwidth) of the network pipe:

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

where $\lambda_p = \min_{1\leq i\leq n}\{(1-p_i)\lambda_i\}$, 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 [7], 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.

III. Mininet Symbiosis

We describe our design for integrating the symbiotic approach with Mininet. A detailed discussion can be found in [8]. Our goal is to execute the target network applications 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.

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.

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.

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 [9] for our prototype implementation), and the the emulation system consists of one or more Mininet instances, potentially running on separate machines (see Fig. 2). 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.

Note that such symbiotic approach can support distributed emulation, where multiple Mininet instances can operate in parallel, each handling a different set of emulated flows. In this case, the state of common network pipes is mirrored on different instances, which will be controlled by the simulator with the identical link properties.

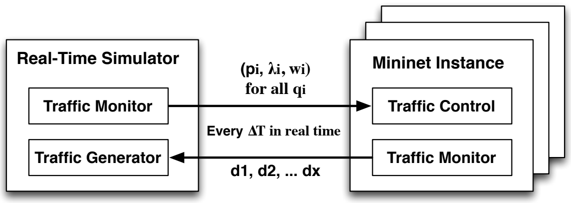

Fig. 2. Mininet symbiosis setup.

REFERENCES

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] J. Liu, C. Marcondes, M. Ahmed, and R. Rong, "Toward scalable emulation of future internet applications with simulation symbiosis," in *Proceedings of the 19th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2015, to appear.

[9] 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.