Distributed Mininet with Symbiosis

Rong Rong and Jason Liu

School of Computing and Information Sciences
Florida International University
Emails: {rrong001,liux}@cis.fiu.edu

***Abstract*—Mininet is a container-based emulation environment that can study networks with virtual hosts and OpenFlow-enabled virtual switches on Linux. However, it is well-known that experiments using Mininet may lose fidelity for large-scale networks and heavy traffic load. One solution is to use a distributed setup where an experiment constitutes multiple instances of Mininet running on a cluster, each handling a subset of virtual hosts and switches. Such arrangement, however, is still constrained by bandwidth and latency limitations in the physical connection between the instances. In this paper, we propose a novel method of integrating distributed Mininet instances using a symbiotic approach, which extends an existing method for combining real-time simulation and emulation. We use an abstract network model to coordinate the distributed instances, which are superimposed to represent the target network. In this case, one can more effectively study the behavior of real implementation of network applications on large-scale networks, since the interaction between the Mininet instances is only capturing the effect of contentions among network flows in shared queues, as opposed to having to exchange individual network packets, which can be limited by bandwidth or sensitive to latency. We provide a prototype implementation of the new approach and present validation studies to show it can achieve accurate results. We also present a case study that successfully replicates the behavior of a denial-of-service (DoS) attack protocol.**

***Index Terms*—Network emulation; distributed emulation; symbiotic simulation; hybrid network experiments**

I. Introduction

Mininet [1], [2] is a popular container-based emulation environment built for Linux for testing OpenFlow applications [3]. Mininet uses lightweight OS-level virtualization to emulate the network hosts. Each virtual host corresponds to a container attached to a separate Linux network namespace, each containing a virtual network interface assigned 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 of the hosts are connected via virtual Ethernet links to one or multiple instances of the Open vSwitch (OVS) [4], which is a production-quality software switch augmented with OpenFlow capabilities to support experiments of OpenFlow applications.

Using Mininet, one can create network experiments using a set of interconnected virtual hosts and virtual switches as an arbitrary network. Real applications can run directly in virtual hosts. Mininet provides performance limiting and isolation features, using existing Linux tools, such as `cgroups` for scheduling and CPU resource management, and `tc` to control the network link properties. Each virtual host in Mininet can be configured to take only a fraction of the overall system CPU resources. Each network link can also be configured with specific bandwidth, latency, and packet loss probability. For OpenFlow experiments, one can connect an OpenFlow controller with the OVS software switches for a full-fledged software-defined network implementation.

A significant portion of the Mininet's implementation is a python library with a straightforward application programming interface (API) and a command-line interface (CLI) to assist the users to easily create and run tests. The API allows the users to create arbitrary custom network topologies. The CLI allows the users to interactively run and control applications on designated virtual hosts during the experiment. As such, Mininet has become a simple and inexpensive network testbed for developing, testing, and debugging network applications, especially OpenFlow applications, without the need to set up a physical network. For example, for five years in a row, students in Stanford's advanced networking topics course have used Mininet to reproduce results from research papers (either on stand-alone hosts or on Amazon EC2) [5].

Although used popularly in network experiments, Mininet has important limitations. Mininet is an emulation system, and as such it operates in real time. For network-limited experiments, one may have to configure a network with links of smaller bandwidth. Packet forwarding on the virtual network share CPU and memory resources. The aggregate capacity offered by Mininet is typically around a few gigabits depending on the physical machine. For CPU-limited experiments, one would also need to carefully allocate the CPU resources to the virtual hosts. When the size of the target network increases, the CPU resources assigned to each application running a virtual host may become severely under-provisioned and therefore affect the timing accuracy of the experiments.

The performance of Mininet can be unreliable for large networks with heavy traffic load. To allow large-network experiments, MaxiNet extends the container-based emulation approach to run Mininet distributedly in a cluster environment [6]. MaxiNet has a front-end which can control a pool of physical machines, called workers. MaxiNet provides an API for creating the target network, which will be partitioned using spatial decomposition to run on the workers. Each worker runs a separate Mininet instance that only emulates a part of the entire network. At the front end, MaxiNet runs a built-in command-line interface (CLI). Like the traditional Mininet, the CLI allows the users interactively run arbitrary commands on any virtual host. It also supports X forwarding, through which one use graphical user interfaces on both the workers

and the virtual hosts.

For network links connecting switches and hosts assigned to different workers are implemented using GRE tunnels[1]. All packets generated by applications traversing these links must be forwarded across the GRE tunnels accordingly. The bandwidth and latency of the links between the workers therefore depend on the physical network connection, which can be severely limited.

In this paper, we propose an alternative method of decomposing large-scale network experiments among distributed Mininet instances. Our approach extends *symbiotic simulation*, previously proposed by Erazo et al. [7], which integrates real-time simulation and emulation. A target network in symbiotic simulation consists of a simulation system and an emulation system. Simulation represents the entire network of interconnected hosts and routers/switches. The user can select a subset of the hosts to be emulated on virtual machines, on which real implementation of network applications can run. The simulation and emulation systems are synchronized in real time. Emulation provides simulation with flow information of traffic generated by the applications running on the virtual hosts. Simulation provides emulation with periodic updates at the network queues traversed by real application traffic. The emulation system uses the information to calibrate communication (throughput, delay, and packet loss) between the real application instances. The approach minimizes the exchange of information between the simulation and emulation systems, and in doing so allows us to study the behavior of real applications running in diverse simulated network scenarios.

We extends symbiotic simulation and uses a succinct simulation model to coordinate different Mininet instances on a pool of workers, each representing a set of applications exchanging traffic. In its simplest form, the application traffic is contained and isolated in the corresponding Mininet instances and no packet forwarded is needed between the workers. Only synchronization messages (traffic demand and queuing updates) are exchanged periodically (at relatively large time intervals) between the Mininet instances on the workers and the succinct simulation model potentially running on the front-end machine. As such, we can minimize the traffic load over the physical network connecting the distributed machines. Consequently, our approach has the potential to significantly increase the size of the network under study or the traffic load supported by emulation. Our approach can also be combined with the traditional spatial decomposition method used in MaxiNet, where application traffic may span over separate Mininet instances.

We implemented a prototype of the symbiotic approach supporting distributed Mininet. We conducted validation experiments, which show that our approach can generate accurate results. To demonstrate the capability of our distributed emulation approach, we chose to study a particular denial-of-service scenario for real TCP protocols, which cannot be realized using the single-machine Mininet.

[1]Generic Routing Encapsulation (GRE) is a protocol that creates a virtual tunnel between two end points where packets of different network-layer protocols can be encapsulated and sent over an IP network.

II. Related Work

There exist several approaches for conducting network experiments in general, and for developing, testing, and debugging OpenFlow applications in particular.

One approach is to use physical testbeds, where one can directly conduct network experiments using physical machines connected with (OpenFlow) switches. One can also use NetFPGA or network processors to implement new OpenFlow functions. An example of physical testbed is GENI [8], which provides access to hundreds of widely distributed resources (including compute nodes, switches, links, WiMax base stations), for which researchers can obtain exclusive access to a slice of the resources and program for experimentation. Both CloudLab [9] and Chameleon [10] take a similar approach, except using resources from a cluster environment. One important drawback of physical testbeds is that resources are expensive to come by and can be cumbersome to operate, especially for large-scale network experiments.

Simulation (e.g., [11]–[13]) is another approach. Simulation provides high fidelity as it can achieve good timing accuracy through the use of virtual time. Simulation can also be parallelized to run on parallel platforms and model large-scale networks (e.g., [14]–[16]). There exist two major drawbacks, however, in using network simulation. One drawback is that developing simulation models would require significant effort. A case in point is the OpenFlow implementation in the ns-3 simulator. For a long time, ns-3 has only one OpenFlow module [17] that implements an outdated OpenFlow protocol version 0.8.9. This module relies on an externally linked OpenFlow switch library that basically lacks all the advanced features of later OpenFlow versions. Until recently, a new module was introduced to support OpenFlow protocol version 1.3 [18]. This module also relies on an externally linked OpenFlow 1.3 software switch library derived from the CPqD OpenFlow 1.3 Software Switch implementation. Unfortunately, the implementation is still lagging behind the current OpenFlow development. The same argument can be applied to developing models for network applications. The other drawback for using network simulation is related to validation, which would also require extensive efforts. These costs make simulation studies less attractive in many cases. As a result, we see only a few research projects that extensively use detailed network simulation for performance studies.

Emulation provides a good alternative to simulation. Emulation operates in real time and supports "traffic shaping" by introducing artificial delays and packet losses (e.g. [19]). Mininet is a container-based emulation testbed, where one can create network experiments using a set of virtual hosts (Linux namespaces) and virtual switches (OVS instances) connected as an arbitrary network. OVS is an OpenFlow-enabled software switch that has the capabilities of running SDN applications. Container-based emulators offer functional fidelity, but often introduce significant temporal errors when

the resource demand goes beyond its physical capacity [20]. To improve emulation scalability and fidelity, one method is to introduce virtual time. One such method is to use time dilation, which controls how system time progresses [21]. By increasing the interval between timer interrupts by a given factor, the system clock will be slowed down accordingly. As a result, applications running in the system would experience a slower passage of time and consequently observe an upgrade in the system resources. Such technique has been integrated in several emulators [22]–[24]. Another approach is to modify the virtual machine scheduling mechanism so that the system time of the VMs advances according to virtual time (e.g., [25]–[27]). For example, TimeKeeper [27] applies kernel modification of time-related system calls in Linux to for a lightweight virtual time system. The technique has been applied to build a virtual-time Mininet, called VT-Mininet, for improving emulation timing accuracy [20]. To overcome the capacity constraint, another method is to use distributed emulation that extends the container-based emulators to run on a cluster [6], [28]. Our approach complements the existing distributed approach by providing a novel method for partitioning the potentially large-scale network and synchronizing separate emulation instances without excessive communication overhead.

It is oftentimes advantageous to combine simulation and emulation. One method is *direct-execution simulation*, which directly incorporates protocol implementations in simulation (e.g., [29]–[31]). Another method is *real-time simulation*, which performs simulation in real time so that the virtual network can interact with real applications by exchanging network packets (e.g., [32]–[34]). The third method is *symbiotic simulation*, where the interplay between simulation and emulation underlines a mutually beneficial relationship by the exchange of information for synchronizing the state of the two systems (e.g. [7], [35]). Symbiotic simulation provides an interesting combination of emulation, which provides a realistic execution environment for applications, and simulation, which provides flexible, large-scale network scenarios. Our approach extends symbiotic simulation in that we use a (scaled-down) simulator to coordinate multiple emulation instances in a distributed environment. Our previous work integrates a Mininet instance with a simulation instance using the symbiotic approach [36]. In this paper, we propose to use a succinct simulation model to coordinate distributed Mininet instances with reduced communication overhead. In doing so, our work will allow for large-scale SDN experimentation on parallel platforms.

III. Mininet Symbiosis

We use an example to illustrate the design of Mininet symbiosis for distributed emulation, shown in Fig. 1. The network consists of six hosts, H_1 to H_6, and sixteen routers. During the experiment, there will be three significant flows: from H_1 to H_2, from H_3 to H_4, and from H_5 to H_6. We consider flows as significant if they may cause congestions in the network paths, such as large file transfers from one host to another. For simplicity, we consider these flows as

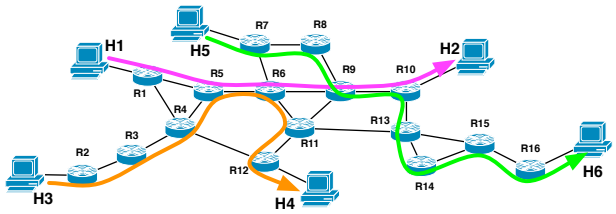

Fig. 1. An emulated network with three traffic flows.

one-directional flows[2]. Our current scheme requires that these three flows have to be specified as part of the experiment configuration. As long as there is a possibility that a significant flow will be created from one host to another during the experiment, we need to include it in the configuration.

The network configuration with the specified flows is then processed to derive a downscaled simulation model. We first partition the virtual network (more specifically, the flows on the network) among the emulation instances. The idea is to keep large flows on the same emulation instances, as opposed to separating them to span across different machines to avoid potential cross-machine traffic. In our example, we assigned each of the three flows to a separate Mininet instance.

Once we partition the network, we identify network links with multiple flows belonging to different emulation instances. We create a queue that corresponds to each of these links. We call such queues as "network pipes"; the network model will use them to handle situations where congestion may occur due to interaction of flows from different emulation instances. In our example, there are two network pipes: one from R_5 to R_6 (q_1), and the other from R_9 to R_{10} (q_2). Note that if a flow is bi-directional, we would need to create two network pipes for each link, one for each direction.

We create the downscaled simulation model by attaching the simulated hosts to the network pipes with proper delays and bandwidths. Fig. 2 shows the downscaled model of our example (in the middle), in which we have six simulated hosts, h_1 to h_6, that correspond to the emulated hosts, H_1 to H_6. The bandwidth of the link between a host and a network pipe is the minimum bandwidth of all links in the original emulated network, and its delay is the sum of the delays. For example, the link from h_1 to q_1 has the bandwidth set to be minimum between the bandwidth of the link H_1 to R_1, and the bandwidth of the link R_1 to R_6; it has the delay set to be the sum of the link delay from H_1 to R_1 and the link delay from R_1 to R_6. We start the "simulation controller" to run the simulation model (discussed momentarily), and create a TCP server that accepts connections from the distributed Mininet instances to exchange information periodically. As it has been shown that a relatively large synchronization interval, ΔT, can be used without significantly comprising the accuracy of the symbiotic approach [7], we choose one second in our experiments.

We create the distributed Mininet instances, one on each

[2]Note that our method does not preclude bi-directional flows, in which case we simply need to add separate queues to represent potential congestion points in the downscaled simulation model.

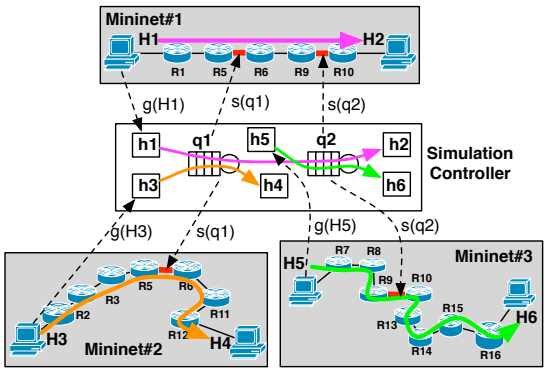

Fig. 2. An abstract simulation model coordinates three Mininet instances.

physical machine. A emulation model consists of a number of virtual hosts (as containers) and switches (as OVS instances). Each Mininet instance connects to the simulation controller as a TCP client to exchange information during the experiment. For each virtual host designated as the source of a significant flow (such as H_3 in Mininet instance #2), we install a "traffic monitor", which periodically collects the traffic demands of the applications running on the virtual host. The traffic demand is represented as $g(H_3)$ in our example; it is sent to the simulation controller.

The simulation controller receives the traffic demand from the Mininet instances, in terms of the number of bytes requested to be sent from H_i to H_j. Upon receiving this information, the simulator simulates a flow with the corresponding demand. If using a packet-oriented simulator, as what we have in our prototype, we initiate a TCP or UDP session between the two simulated hosts and send the desired amount of data. An alternative is to use a fluid traffic model (e.g., [37]) where one can simply modify the send rate in the equation of the corresponding flows.

The simulation controller also collects the state information $s(q_i)$ at each network pipe q_i, which includes the measured packet drop probability p_i, the queuing delay w_i, and the arrival rate of the traffic flows λ_{ki} for all flows k traversing the network pipe q_i. These measurements are collected periodically by the simulator and sent to all Mininet instances that have the corresponding network links.

In each Mininet instance, we also install a "traffic controller" module. The link corresponding to a network pipe is implemented as a virtual Ethernet (`veth`) pair. The traffic controller "shapes" the traffic going through the network link. More specifically, upon receiving the state information $s(q_i)$ for the network pipe q_i, the traffic controller can calculate the bandwidth of the corresponding link as follows (see [7] for derivation in a similar case for symbiotic simulation):

$$\mu_i = \frac{\lambda_i(\Delta T + \hat{w}_i - w_i)}{\Delta T\left(1 + w_i\lambda_i - \sqrt{1 + w_i^2\lambda_i^2}\right)}$$

where λ_i is the sum of λ_{ki} for all flows k belonging to this Mininet instance, and $\hat{w}_i$ is the average packet queuing delay of the network link measured in emulation. The traffic

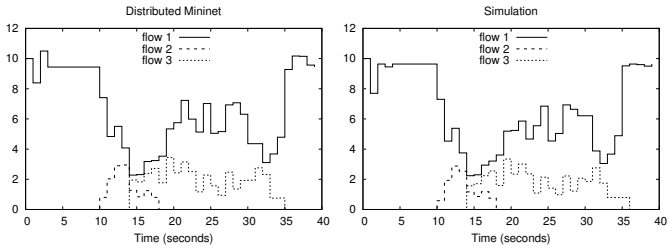

Fig. 3. Comparing TCP throughput (in Mb/s) of the three flows from distributed Mininet vs. simulation.

controller sets the packet loss p_i and bandwidth μ_i of the network link using the Linux traffic control utility, `tc`.

IV. Validation Experiments

We created a prototype implementation of our distributed Mininet with symbiosis. We conducted experiments and compared the results with those obtained from simulation to validate our approach. We used the machines at CloudLab [9] for all our experiments. Each of these machines is equipped with two eight-core Intel Xeon E5-2450 2.1 GHz processors and 16 GB memory; they are connected by 10 Gbps Ethernet. We ran the simulation controller and the Mininet instances on separate machines.

We started with a simple dumbbell model to demonstrate that our symbiotic approach can capture complex network dynamics in real time. The model consists of two routers connecting six hosts (three on either side). We set the bandwidth of the "bottleneck" link between the two routers to be 10 Mb/s, and the link delay is 15 milliseconds. Each router is connected with three hosts via separate "spoke" links. The bandwidth of the spoke link is 1 Gb/s and the delay is 1 ms. In the experiment, we directed three TCP flows with `iperf` (using TCP Reno), one from each host on one side of the dumbbell to a different host on the other side. The first flow was a long-lived flow starting from the beginning of the experiment. The second flow started from 10 seconds and lasted for 8 seconds. The third flow started from 14 seconds and ended at 35 seconds. For distributed emulation, we instantiated three Mininet instances, one for each flow. For comparison, we created the same scenario in simulation. Fig. 3 shows that the throughput from distributed emulation (reported by `iperf`) match well with the simulation output.

In the second experiment, we created a more complicated scenario where flows may interact at multiple locations. The model, shown in Fig. 4, consists of 16 hosts connected by 6 routers organized as a ring. The links connecting the routers have 100 Mb/s bandwidth and 5 milliseconds delay. The links connecting the hosts have 1 Gb/s bandwidth and 1 millisecond delay. We created eight flows during the experiment with different arrive time and duration. Flows 3 and 4 have a random start time. To make it more interesting, we designated flow 0 to be a simulated flow (generated only in simulation controller) with five parallel TCP sessions. The rest of the flows were handled by seven Mininet instances, one for each flow. We

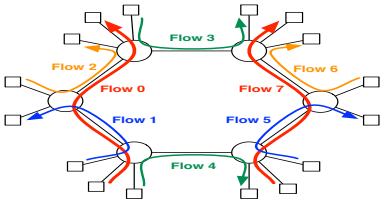

Fig. 4. The ring model.

TABLE I
SEVEN EMULATED FLOWS IN THE RING MODEL

flow	start time	end time	emulation throughput	simulation throughput	error
1	0	20	56.8	56.8	0.0%
2	5	25	51.2	50.7	1.0%
3	0∼10	20∼30	95.3	92.0	3.6%
4	0∼10	20∼30	94.3	91.6	2.9%
5	5	20	48.5	51.1	5.1%
6	20	30	55.3	55.2	0.2%
7	0	25	51.8	51.7	0.2%

compare the results with those obtained from simulation (see table I). The error never exceeds 5.1%.

V. A CASE STUDY

Denial of service (DoS) attacks prevents the target computer from responding quickly to its legitimate users' traffic, or not at all. Shrew is a specific attack pattern where the attacher sends bursts of data at a regular interval to an over-committed bottleneck link [38]. When the attack bursts occur at intervals that synchronize with the minimum retransmission timeout (RTO) of legitimate TCP connections sharing the bottleneck link, they can trigger TCP timeouts and consequently strangle the throughput of those connections. Since the average traffic rate of a shrew attack is low, it can be difficult to be detected. In this section, we use the Shrew attack as a real life example.

To establish the baseline, we start with the same experiment setup with a simple topology as in [38]. There is one pair of good sender and receiver (the victim flow) and another pair of bad sender and receiver (the attack flow); they share the same bottleneck link. The good pair are separated by two routers, and the bad pair by three routers. The shared bottleneck link has 10 Mb/s bandwidth and 20 ms delay. All the other links have 100 Mb/s bandwidth and 2 ms delay. We ran one Mininet instance to emulate the good data transfer (using `iperf`) and simulate the attack flow using UDP. We set the burst rate to be 10 Mb/s and the length of each burst to be 100 ms. We ran experiments with different inter-burst period (the time between consecutive bursts) from 0.9 to 5 seconds.

The results are shown in Fig. 5. The y-axis is the normalized throughput, which is the throughput of the victim flow divided by the bandwidth of the bottleneck link. Our results, marked as "TCP Reno (Emulation)", are comparable with "TCP Reno (Simulation)", the results reported in the original paper [38]. We also tried other TCP versions (Vegas and BIC are shown in the plot); we got similar results. As expected, we see that the Shrew attack can significantly lower the throughput of the victim flow (in this case when the inter-burst period is at around 1 second).

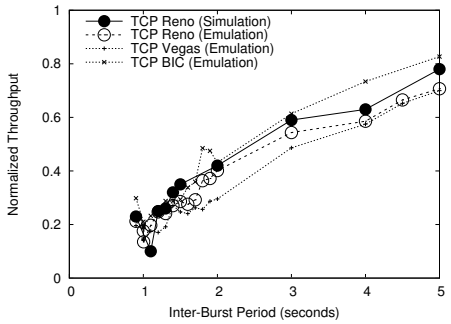

Fig. 5. The effect of the Shrew attack.

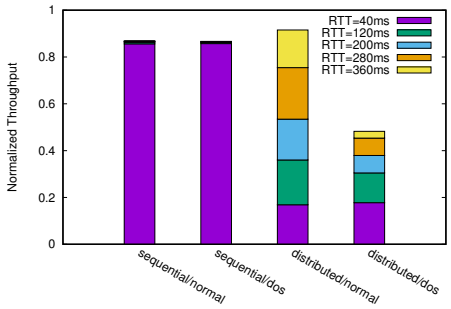

Fig. 6. Sequential vs. distributed Mininet runs.

We conducted another experiment to study the effect of the Shrew attack against a group of victim TCP flow with different RTTs. We used the dumbbell topology, where we have five pairs of good senders and receivers and one pair of bad sender and receiver. We set the bandwidth of the bottleneck link to be 100 Mb/s and the other links to be 1 Gb/s. We set the victim flows to have different round-trip times (RTTs): 40, 80, 160, 280 and 360 ms. The attack flow has an RTT of 440 ms. The attack flow has the same burst and length as in the previous example. We fix the inter-burst period to be 1.0003 seconds.

We use distributed emulation with six Mininet instances, one for each flow. We compare the results obtained from the distributed Mininet with those from running a single Mininet instance. In Fig. 6, one can see that running a single Mininet instance, the Shrew DoS attack basically has no effect on the throughput of the flows. In fact, it generates incorrect results: the flow with the smallest RTT got the whole share of the throughput. Using our distributed Mininet, we observe that not only the aggregate throughput is reduced by the DoS attack, but also the flows react differently: the throughput degrades more significantly for flows with higher RTTs.

VI. CONCLUSIONS

We present a distributed emulation method using a symbiotic approach, which has been used early to combine real-time simulation and emulation. In our approach, the simulator acts as a coordinator for the distributed emulation instances by capturing the effect of contention among the network flows potentially belonging to different distributed instances. Our approach provides a novel method for partitioning the virtual network among the emulation instances and can be used in accordance with the traditional spatial decomposition method. Through experiments using a distributed Mininet

implementation, we show that the symbiotic approach can generate accurate results, and it can be readily used in studies involving high traffic load scenarios.

For future work, we would like to first integrate our symbiotic approach with the traditional spatial decomposition. In this case, a robust partitioning algorithm is needed to be able to handle different scenarios. Second, our current method requires that significant flows be identified during experiment configuration. This can be an unnecessary burden if the system can dynamically identify these flows and create network pipes on demand. Third, our current design has but one centralized simulation controller. A distributed approach is needed to avoid the potential bottleneck for a large number of emulation instances. Last, we would like to explore other more efficient simulation abstractions (such as fluid models) which can further reduce the cost of the controller.

ACKNOWLEDGMENT

This research is supported in part by the NSF grant CNS-1563883, the DOD grant W911NF-13-1-0157, the DOE LANL/LDRD Program, and a USF/FC2 SEED grant.

REFERENCES

[1] 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.

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

[3] 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.

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

[5] "Reproducing network research: network systems experiments made accessible, runnable, and reproducible," https://reproducingnetworkresearch.wordpress.com/.

[6] 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.

[7] M. A. Erazo, R. Rong, and J. Liu, "Symbiotic network simulation and emulation," *ACM Trans. Model. Comput. Simul.*, vol. 26, no. 1, pp. 2:1–2:25, 2015.

[8] GENI Project Office, "The Global Environment for Network Innovations (GENI)," http://www.geni.net.

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

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

[11] NS-3 Project, "ns-3," http://www.nsnam.org/index.html.

[12] http://www.opnet.org.

[13] András Varga, "OMNeT++ Network Simulation Framework," http://www.omnetpp.org/.

[14] D. M. Nicol, J. Liu, M. Liljenstam, and G. Yan, "Simulation of large-scale networks using SSF," in *Proceedings of the Winter simulation Conference*, 2003, pp. 650–657.

[15] 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)*, 2003, pp. 5–12.

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

[17] B. Hurd, "GSoC 2010 OpenFlow," https://www.nsnam.org/wiki/GSOC2010OpenFlow.

[18] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "Ofswitch13: Enhancing ns-3 with openflow 1.3 support," in *Proceedings of the Workshop on Ns-3 (WNS3'16)*, 2016, pp. 33–40.

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

[20] J. Yan and D. Jin, "Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR'15)*, 2015, pp. 27:1–27:7.

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

[22] D. Gupta, K. V. Vishwanath, M. McNett, A. Vahdat, K. Yocum, A. Snoeren, and G. M. Voelker, "Diecast: Testing distributed systems with an accurate scale model," *ACM Trans. Comput. Syst.*, vol. 29, no. 2, pp. 4:1–4:48, 2011.

[23] 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.

[24] A. Grau, S. Maier, K. Herrmann, and K. Rothermel, "Time jails: A hybrid approach to scalable network emulation," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS'08)*, 2008, pp. 7–14.

[25] D. M. Nicol, D. Jin, and Y. Zheng, "S3F: the scalable simulation framework revisited," in *Proceedings of the Winter Simulation Conference*, 2011, pp. 3288–3299.

[26] E. Weingärtner, F. Schmidt, H. V. Lehn, T. Heer, and K. Wehrle, "Slicetime: A platform for scalable and accurate network emulation," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*, 2011, pp. 253–266.

[27] J. Lamps, D. M. Nicol, and M. Caesar, "Timekeeper: A lightweight virtual time system for linux," in *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS'14)*, 2014, pp. 179–186.

[28] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dot: Distributed openflow testbed," in *Proceedings of the 2014 SIGCOMM Conference*, 2014, pp. 367–368.

[29] X. Liu, H. Xia, and A. A. Chien, "Validating and scaling the microgrid: A scientific instrument for grid dynamics," *J. Grid Comput.*, vol. 2, no. 2, pp. 141–161, 2004.

[30] J. Liu, Y. Yuan, D. M. Nicol, R. S. Gray, C. C. Newport, D. Kotz, and L. F. Perrone, "Empirical validation of wireless models in simulations of ad hoc routing protocols," *SIMULATION*, vol. 81, no. 4, pp. 307–323, 2005.

[31] 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," *CoNEXT*, pp. 217–228, 2013.

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

[33] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim, "Core: A real-time network emulator," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, 2008, pp. 1–7.

[34] 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, 2009.

[35] Y. Gu and R. Fujimoto, "Performance evaluation of the rosenet network emulation system," *SIMULATION*, vol. 85, no. 5, pp. 319–333, 2009.

[36] 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'15)*, 2015, pp. 68–77.

[37] Y. Liu, F. L. Presti, V. Misra, D. F. Towsley, and Y. Gu, "Scalable fluid models and simulations for large-scale ip networks," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 3, pp. 305–324, 2004.

[38] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proceedings of the 2003 SIGCOMs Conference*, 2003, pp. 75–86.