Fast, Accurate Simulation for SDN Prototyping

Mukta Gupta
University of Wisconsin
mgupta@cs.wisc.edu

Joel Sommers
Colgate University
jsommers@colgate.edu

Paul Barford
University of Wisconsin
pb@cs.wisc.edu

ABSTRACT

Thorough test and evaluation of new software-defined network (SDN)-based applications and configurations present many challenges. Examples of these challenges include scaling to large networks, accuracy, and efficiency in evaluation along with the ability to easily transition between prototype and test environments. Current methods for test and evaluation include new programming languages and frameworks, debugging and static analysis techniques, and VM- and container-based emulation tools.

In this paper we describe a simulation-based tool called *fs-sdn* that complements and expands upon these existing approaches. Our work is designed to address the problem of prototyping and evaluating new SDN-based applications accurately, at large scale, and in a way that enables easy translation to real controller platforms like POX and NOX. We describe the design, implementation and use of *fs-sdn*, and demonstrate its capability by carrying out a series of experiments using *fs-sdn* and the Mininet platform in nearly identical configurations. We show that the measurements derived from *fs-sdn* are accurate compared with Mininet, but offer significant speed and scalability advantages.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management; I.6.3 [**Simulation and Modeling**]: Applications

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Network flows; OpenFlow; Simulation; Software-Defined Networks

1. INTRODUCTION

Over the past several years, software defined networking (SDN) has emerged as a compelling paradigm for developing and deploying new network capabilities and services. Centralizing the network control plane—the key idea behind SDN—has led to innovative approaches to traffic engineering, reducing network energy consumption, and data center network management, *e.g.*, [6, 18]. As SDN applications and configurations continue to grow in scale and complexity, a primary challenge is how to develop and test new designs so that they behave as expected when deployed in live settings and at scale.

The objective of our work is to develop new capabilities for test and evaluation that complement current tools. Our work focuses on a portion of the design space that we believe is not well served. In particular, we seek to develop a new capability for prototyping and testing SDN applications that is *efficient, scalable, realistic and enables a relatively seamless transition to practice*.

Tools for developing and testing SDN applications have greatly improved recently, but each has certain critical limitations. For example, Mininet [16] has significantly advanced the state-of-the art for developing and testing new controller applications. It is already seeing wide use, as it offers good realism and seamless transition from development to deployment, but as we show in Section 5, it does not yet scale to large networks. Other researchers have used testbed environments for development and evaluation, *e.g.* [5, 14]. These settings certainly offer a high degree of realism, but they also have scaling problems and are typically not available to a broad slice of the research community, thus making replication difficult. To evaluate large-scale designs, other researchers have turned to developing custom fluid-flow simulators, *e.g.*, in [6, 12, 27, 30], or extending existing packet-level simulators [4, 22]. While the fluid-flow simulators scale well, by design they ignore many important network characteristics, and are often "one-off" implementations, which complicates sharing and scientific replication. Furthermore, packet-level simulators generally do not scale well, and the development environment often does not facilitate transition to a "real" controller platform.

In this paper, we describe a simulation-based tool called *fs-sdn*[1] that is designed to facilitate prototyping and evaluating new SDN applications. *fs-sdn* is based on the *fs* [29] simulation platform that was developed for realistic and scalable test and evaluation in standard networks. *fs-sdn* balances realism with scaling and performance by virtue of its core abstraction, which captures details about network flows while remaining at a relatively high conceptual level.

fs-sdn significantly extends the underlying *fs* simulation engine by seamlessly incorporating the POX [3] OpenFlow controller framework and API, and switch components that can be controlled and configured through the OpenFlow control protocol. *fs-sdn* is designed to facilitate controller application prototyping and development by making it easy to transport implementations to POX and similarly-designed controller platforms, thus creating a straightfor-

[1] Source code for *fs-sdn* is available at `https://www.github.com/jsommers/fs`.

ward path to "real" implementation. *fs-sdn* is easy to use and it runs on any system on which Python can run.

We demonstrate the capabilities of *fs-sdn* by running nearly identical network scenarios in both *fs-sdn* and Mininet. We compare traffic measurements and performance and scalability aspects of each platform. By virtue of its implementation, traces generated from Mininet are highly realistic and thus a good benchmark for *fs-sdn*. One of the key results of our tests is that there is a close correspondence between the traffic characteristics extracted from traces generated by each platform in modest configurations. This gives us confidence in the realism of traces generated by *fs-sdn*. The second key result is that simulations in *fs-sdn* run much more quickly and robustly than Mininet. This demonstrates *fs-sdn*'s performance and scalability capabilities.

The remainder of this paper is organized as follows. In Section 2, we discuss the context and background of our work, and efforts related to ours. We describe the details of the design and implementation of *fs-sdn* in Section 3. In Section 4 we describe the setup for experiments to evaluate the accuracy and performance of *fs-sdn*, the results of which are presented in Section 5. We conclude and discuss future work in Section 6.

2. BACKGROUND

In this section we provide an overview of the *fs* flow-level simulator and describe prior studies that influence and inform the design and implementation of *fs-sdn*.

2.1 *fs* overview

fs is a Python-based tool developed by two of the co-authors and others for generating network flow records and interface counters à la SNMP [29]. Although it was not primarily designed for the purpose of simulating network activity, it uses discrete-event simulation techniques for synthesizing the network measurements that it produces. We illustrated in earlier work that *fs* not only generates measurements extremely fast compared with identical setups in ns-2 [25], but that the measurements it produces are accurate down to the timescale of 1 second. TCP traffic generation capabilities based on Harpoon [28] (which is in turn, based on SURGE [7]) are built into *fs*. It further leverages existing TCP throughput models (*i.e.,* Mathis *et al.* [24] and Cardwell *et al.* [11]) to simulate individual TCP flows. More generally, *fs* includes the capability to generate a broad range of simulated traffic conditions, which can be easily configured using a declarative specification based either on the Graphviz DOT language [9] or in a JSON format.

Most critically, the key network abstraction it operates on is not the packet, but a higher-level notion called a *flowlet*. A flowlet refers to the volume of a flow emitted over a given time period, *e.g.,* 100 milliseconds, which may be 1 or more packets. By raising the level of abstraction and thus the entity around which most simulator events revolve, *fs* achieves much higher speed and efficiency than existing packet-level simulators, like ns-2 [25] and ns-3 [4]. *fs*'s better scaling properties are particularly relevant to this work, since our longer-term goal is to scale to the size of a large, modern data center.

2.2 Related work

Our efforts are related to the Mininet system [16,23] in the sense that Mininet is also a prototyping and testing platform well-suited for SDN applications. Mininet is based on using lightweight OS containers to emulate hosts and switches in a network. As such, it has much better scaling properties than other systems based on using "full-blown" VMs, *e.g.*, [2]. As we show in later sections, *fs-sdn* offers better scaling and speed for testing and developing SDN applications, and complements the advantages of realism available with Mininet.

Our work is also related to recent efforts to improve the state of the art in debugging and testing the correctness of SDN applications. Handigol *et al.* describe a gdb-like debugger for OpenFlow-based networks in [17]. Their system allows users to obtain information analogous to a stack trace from the network, and to perform other tracing tasks. In a different vein, several research groups have developed various methods for statically analyzing OpenFlow rules in order to identify traffic loops, blackholes, and other problems [10, 20, 21]. Although *fs-sdn* does not currently support dynamic or static analysis of rules, it offers a controller development setting similar to that of an emulated or live network, but in a highly controlled context. Lastly, another thread of research has focused on raising the level of abstraction at the programming level, thereby making SDN-based application development easier. Examples of these efforts include Frenetic [13], Nettle [31], Procera [32], and Pyretic [26].

3. SYSTEM DESIGN

The key design goals for *fs-sdn* are as follows:

- Generate measurements that are *accurate*, in the sense that they are similar to what would be collected in a real or accurately emulated network setting;
- Scale to large networks; and
- Provide an API that enables controller applications to be easily ported to other platforms like POX [3], NOX [15], and Floodlight [1].

The accuracy goal is clearly important. *fs-sdn* can generate SNMP-like counter byte/packet/flow measurements at configurable intervals, as well as flow records. In our prior work [29], we demonstrated that the original version of *fs* produces accurate measurements compared with those generated in a live network setting and in a comparable simulation setting. It must, of course, continue to exhibit high accuracy in a simulated OpenFlow/SDN network setting.

An equally important goal is for *fs-sdn* to exhibit good scaling properties. Our goal is for it to be able to scale to networks of hundreds, if not (many) thousands of switches and hosts. As discussed in Section 2, a central design decision that affects how well *fs-sdn* can scale is that it does not attempt to model packet-level interactions, but rather operates on a higher-level abstraction called a *flowlet*. In our current implementation, *fs-sdn* can scale well to hundreds of switches and hosts, depending on the traffic configuration. We discuss our plans for improving scalability further in Section 6.

Currently, one of the most widely used techniques for performing large-scale evaluations of new controller-based algorithms is by developing a custom fluid-flow simulator, *e.g.*, as in [6,12,27,30]. These have typically been designed to address large-scale data center experiments, yet because of limitations in these types of simulators, the profile of traffic generated cannot be easily made to match observed characteristics of data center flows [8,19]. Advantages of *fs-sdn* are that it is a more general and thus reusable platform, and that it not only scales well, but it can easily accommodate measured flow characteristics to drive its traffic generation.

In an effort to make using *fs-sdn* relatively straightforward and to facilitate cross-environment development, controller components developed for the POX [3] platform can be used *directly* in *fs-sdn*. Specifically, controller code developed for *fs-sdn* can be transported without modification for use in POX. The APIs available in

POX are similar to those in NOX and other platforms, and represent a class of controller APIs that are familiar to many SDN developers. As a result of leveraging the existing POX APIs and libraries, *fs-sdn* enables developers and experimenters to prototype, test, and debug an application in simulation, then directly run the same code in other environments.

In terms of implementation, *fs-sdn* and POX are both written in Python, which makes use of POX APIs fairly straightforward. They key challenges to integrating POX are that there are dependencies that rely on some notion of wall-clock time, and that connections between the controller and OpenFlow switches use networking APIs (*e.g.*, the sockets API) to transmit and receive control messages. Moreover, *fs-sdn* operates on *flowlets*, which do not include many low-level details included in standard network packets, and which are important to OpenFlow in general and POX in particular.

To address the timing and network dependencies in POX, we do two things in *fs-sdn* before OpenFlow switch or controller elements are created in simulation. First, we modify the underlying Python `time` module so that requests for the current time return the current *fs-sdn simulation* time, rather than wall-clock time. *fs-sdn* has no notion of wall-clock time, and there were a number of places in POX where the current system time is accessed. Rather than modify POX code, we take advantage of Python features to monkeypatch[2] calls to `time.time`.

To handle network-related dependencies, we modify at runtime a class in POX that is used to encapsulate and abstract an individual connection from the controller to a switch. The original class handled low-level network `send` and `recv`; the new class instead uses the *fs-sdn* API to deliver and receive messages.

We also added new node-level capabilities in *fs-sdn* to model an OpenFlow switch and OpenFlow controller. The new nodes each build on generic *fs-sdn* node capabilities, and add code to bridge the *fs-sdn* and OpenFlow worlds. The new switch node uses many built-in POX capabilities (including the software switch code), and in addition to delegating calls to POX, also performs translation between *fs-sdn* and POX data types. The new controller node also performs delegation to POX, including simulating the arrival of new OpenFlow control messages from a switch via the (overridden) connection class.

There are two key data types that must be translated between *fs-sdn* and POX: flowlets arriving at switches must be converted to a packet representation prior to handling by POX API calls, and packets emitted by POX library calls must be translated to a flowlet representation in order to be properly handled within *fs-sdn*. For arriving dataplane flowlets that are translated to packets, we also cache the original flowlet with the packet object. The reason is that once a forwarding decision is made (either through a match with the local switch flow table, or by the controller), the original flowlet can be forwarded, obviating the need for a second translation. We take a similar approach with packet to flowlet translations, specifically with packets that are emitted by the controller such as LLDP (link-layer discovery protocol) frames.

As part of our modifications to *fs-sdn*, we also added basic notions of MAC addresses and ARP to the new switch class. In our current implementation, some additional work is yet to be done with respect to translation of *all* packet headers, *e.g.*, MPLS shim headers, Ethernet frames besides LLDP and ARP, etc. We intend to make *fs-sdn*'s datatype translations as complete as possible, to ensure no loss of information when translating between the two environments. Lastly, we have as yet only tested a fairly narrow range of POX controller applications. While the built-in hub, layer 2 learning switch, and topology discovery components work "out of the box", we cannot yet not claim a *completely* transparent environment for executing POX controller modules.

[2] `http://en.wikipedia.org/wiki/Monkey_patch`

4. EVALUATION METHODOLOGY

The primary goals of our experiments are to compare the accuracy of *fs-sdn* versus experiments carried out in a non-simulation environment, and to evaluate the performance and scalability of *fs-sdn*. Our basic approach is to set up a series of identical topological and traffic scenarios in both *fs-sdn* and Mininet [16].

We consider *fs-sdn* in comparison with Mininet for a few reasons. First, Mininet is currently one of the best and most accessible platforms for carrying out SDN-based experiments. Second, it offers a realistic setting for carrying out experiments, so it is reasonable to treat its measurement results as a basis for comparison with *fs-sdn*. Third, although use of Mininet has many benefits, it does have some limitations, as pointed out on the Mininet wiki itself [3]. Two particular restrictions are that switching capacity is limited to whatever the underlying host system can support, and experiments cannot be run faster than real-time. These issues are not particularly surprising, and they are also not inherent limitations of Mininet. They do, however, make certain types of experiments currently impractical. It is important to note that our motivation for comparing with Mininet is not to point out these limitations, but rather to highlight the particular strengths of *fs-sdn*, and how it can provide a complementary platform for prototyping and evaluating SDN-based applications.

For our experiments, we created four topology sizes in a linear configuration and used a range of traffic profiles. A linear topology was chosen for simplicity, and in order to ensure that all switches are well-exercised. The lengths of the linear chain of switches we used were 1 (tiny), 10 (small), 50 (medium), and 100 (large). We configured the link between each switch with a delay of 5 milliseconds. For each topology, we used constant bit-rate (CBR) UDP traffic at two different loads (10 Mb/s and 100 Mb/s), and heavy-tailed TCP traffic generated using Harpoon [28] at two different loads (average of 5 Mb/s and average of 25 Mb/s). In Mininet, we generated the UDP CBR traffic using iperf. Each experiment was run for 900 seconds. All experiments (both in Mininet and *fs-sdn*) were run using an 8-core 2.4 GHz Intel Xeon E5645-based (Nehalem) system with 12 GB RAM. It was installed with Ubuntu 12.04.2 and a Linux 3.2.0-39 kernel.

To evaluate the accuracy of *fs-sdn*, we compared byte, packet, and flow counts over 1 second intervals from each platform. *fs-sdn* can directly export these measurements if configured to do so. In Mininet, we used the CAIDA Coral Reef toolset [4] to collect these measurements. We collected traffic using Coral Reef at the first switch in the linear topology for each experiment[5]. We also measured CPU and memory utilization on the host system on which we ran our experiments, and measured the wall-clock experiment duration for *fs-sdn*.

Lastly, we note that all *fs-sdn* experiments were run using the PyPy Python interpreter (which includes a just-in-time compiler), version 2.0 beta [6]. Furthermore, in all *fs-sdn* experiments, we used

[3] `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what-are-mininets-limitations`

[4] `http://www.caida.org/tools/measurement/coralreef/`

[5] Traffic measurements collected at different switches in the topology gave qualitatively similar results.

[6] `http://pypy.org`

a simple controller application that computes shortest paths and installs the appropriate OpenFlow rules on demand from switches.

5. RESULTS

In this section, we evaluate the accuracy of *fs-sdn*, as well as its scaling and speed properties.

5.1 Accuracy and Scaling

We first compare traffic volumes forwarded in each of *fs-sdn* and Mininet when subjecting each system to different types of traffic and at different loads.

Figure 1 represents the cumulative distribution functions (CDFs) of byte counts (in thousands of bytes) over 1 second bins with iperf generating UDP CBR traffic. The analysis was done with two different rates, *viz.*, 10 Mb/s (low load) and 100 Mb/s (high load). *fs-sdn* shows a consistent performance across various topologies and rate configurations, but the accuracy of Mininet degrades both with the increase in load from 10 Mb/s to 100 Mb/s, and with an increase in the size of the emulated network. These results illustrate the limits in the Mininet's operating envelope, and highlight *fs-sdn*'s utility as a more scalable and accurate platform to complement the advantages of Mininet. Results from additional topology sizes using UDP CBR traffic are omitted but are consistent with plots shown.

Figure 2 shows cumulative distribution functions (CDFs) of byte volumes (in thousands) produced over 1 second bins using the Harpoon traffic generator. The analysis was done at two different average traffic rates, 5 Mb/s and 25 Mb/s average. Although the Harpoon traffic exhibits burstiness due to the distribution of file sizes, interconnection times and the closed-loop nature of TCP, we observe that for the two smaller topologies (1- and 10-switch) and low traffic load, *fs-sdn* and Mininet exhibit similar results. As the number of switches increases or the traffic load increases, the Mininet results become inaccurate. From observing CPU utilization during Mininet experiments, it is clear that the system switching capacity is saturated. Moreover, it is interesting to observe the significantly more intense effect on system performance that Harpoon has versus using the simplistic UDP CBR traffic source. For the larger 100-switch topology, the kernel Open vSwitch daemon crashed, and we were unable to complete those experiments. We note that in addition to byte count measurements at low loads and small topologies, flow and packet count measurements are consistent between *fs-sdn* and Mininet.

These results suggest that measurements produced by *fs-sdn* at aggregation of 1 second are statistically indistinguishable from Mininet for small topologies with low traffic volumes but for large topologies and traffic volumes *fs-sdn* surely provides an advantage over platforms like Mininet.

5.2 Speed

Table 1 shows the simulation time for carrying out various experiments using *fs-sdn*. Recall that each experiment is designed to take 900 seconds, thus all Mininet experiments took 900 seconds to complete. While the speedup achieved by *fs-sdn* is machine-dependent, on the particular machine configuration that we used for our experimentation it can be concluded that *fs-sdn* offers a significant run-time advantage. For a network of 100 nodes with high load, *fs-sdn* shows speedup by a factor of 11.8 when CBR traffic is used and by a factor of 2.7 when Harpoon is used.

Table 1: Scenario execution times for *fs-sdn* in seconds. The simulated experiment length is 900 seconds in each case.

	UDP CBR			
Load	Tiny	Small	Medium	Large
Low	6	8	33	72
High	4	8	31	76
	Harpoon			
Load	Tiny	Small	Medium	Large
Low	16	33	104	193
High	30	62	194	337

6. SUMMARY, CONCLUSIONS, AND FUTURE WORK

In this paper we describe *fs-sdn*, a scalable and accurate simulation-based tool for prototyping and evaluating new SDN-based applications. *fs-sdn* enables direct use of OpenFlow controller components developed using the POX controller API, enabling researchers and developers to seamlessly move between simulation, emulation, and a live network deployments. As a result, we argue that *fs-sdn* lowers the barrier to experimentation with and testing of new OpenFlow controller applications.

We evaluate *fs-sdn*'s accuracy, scalability, and speed by setting up *fs-sdn* and the popular Mininet platform with a series of identical network and traffic configurations, and comparing network traffic and system-level measurements collected from each. We find that in settings with relatively low traffic volumes and small topologies, network measurements collected in Mininet and *fs-sdn* are nearly identical, but that measurements collected in Mininet become distorted in large topologies, and under more intense traffic conditions. We conclude that while Mininet offers good facilities for developing and testing SDN-based applications in a realistic setting, *fs-sdn* offers advantages of being able to accurately test and prototype new applications at much larger scale and greater speeds. We contend that *fs-sdn* offers a compelling complement to existing tools like Mininet for prototyping and evaluating new SDN applications.

In future work, there are several directions we intend to pursue to improve *fs-sdn*. First, although we have tested *fs-sdn* with network topologies in the hundreds of nodes, we aim to scale it to network topologies comparable to the size of a modern data center. We are currently investigating ways to parallelize *fs-sdn*, in order to vastly improve its scalability.

Lastly, we envision a key use case for *fs-sdn* to be in initial prototyping and debugging of SDN applications. Currently the debugging capabilities in *fs-sdn* include the ability to control log message generation, including logs of OpenFlow control messages. In the future, we intend to implement additional explicit debugging and tracing capabilities in *fs-sdn* to better support and facilitate the initial stages of development and evaluation of new SDN applications.

Acknowledgements

This work was supported in part by NSF grants CNS-1054985, CNS-0831427, CNS-0905186, ARL/ARO grant W911NF1110227 and the DHS PREDICT Project. Any opinions, findings, conclusions or other recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, ARO or DHS.

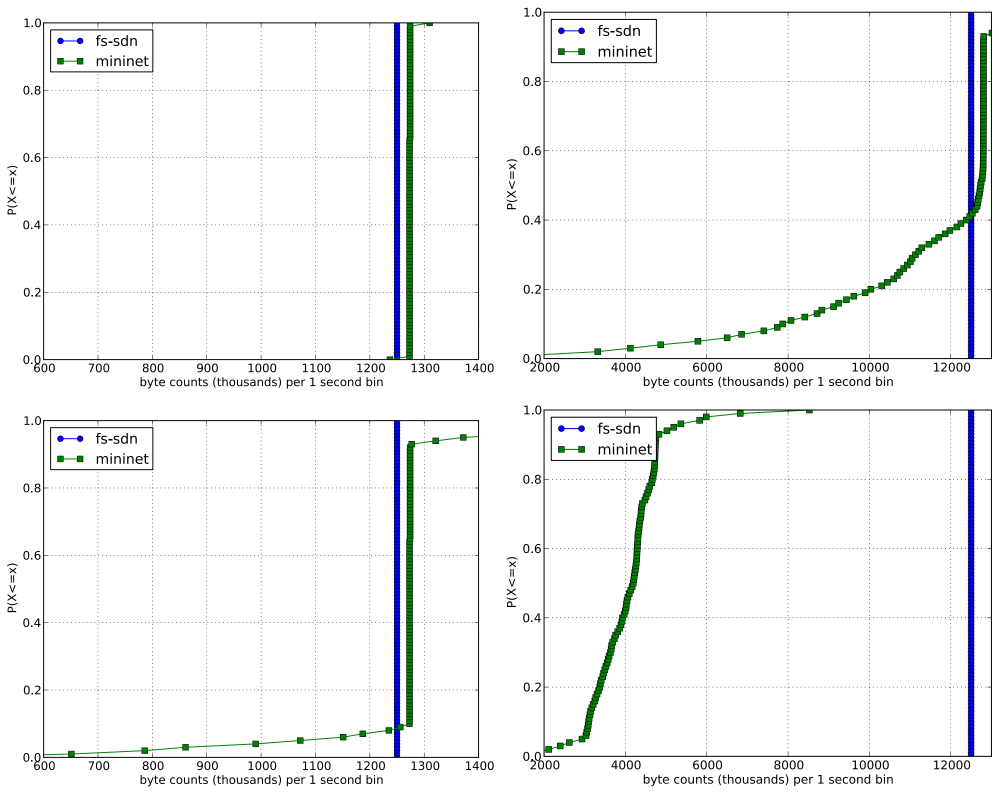

Figure 1: Bytes forwarded per 1 second intervals (in thousands) in Mininet and *fs-sdn* using UDP CBR traffic. Results shown for the small topology (top) and large topology (bottom) using low traffic load (left) and high traffic load (right).

7. REFERENCES

[1] Floodlight, Java-based OpenFlow Controller. `http://floodlight.openflowhub.org/`.

[2] Openflow virtual machine simulation. `http://www.openflow.org/wk/index.php/OpenFlowVMS`.

[3] POX, Python-based OpenFlow Controller. `http://www.noxrepo.org/pox/about-pox/`.

[4] ns-3: Openflow switch support. `http://www.nsnam.org/docs/release/3.13/models/html/openflow-switch.html`, 2013.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of ACM SIGCOMM '08*, August 2008.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of USENIX NSDI '10*, 2010.

[7] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of ACM SIGMETRICS*, Madison, WI, June 1998.

[8] T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of ACM Internet Measurement Conference, '10*, pages 267–280, 2010.

[9] A. Bilgin, J. Ellson, E. Gansner, Y. Hu, Y. Koren, and S. North. Graphviz-Graph Visualization Software. `http://www.graphviz.org`, 2013.

[10] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. A NICE way to test OpenFlow applications. In *Proceedings of USENIX NSDI '12*, April 2012.

[11] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, March 2000.

[12] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: scaling flow management for high-performance networks. In *Proceedings of ACM SIGCOMM '11*, 2011.

[13] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, Story A., and D. Walker. Languages for software-defined networks. *Communications Magazine, IEEE*, 51(2):128–134, 2013.

[14] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of ACM SIGCOMM '09*, August 2009.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[16] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of ACM CoNeXT '12*, pages 253–264, 2012.

[17] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. Where is the debugger for my

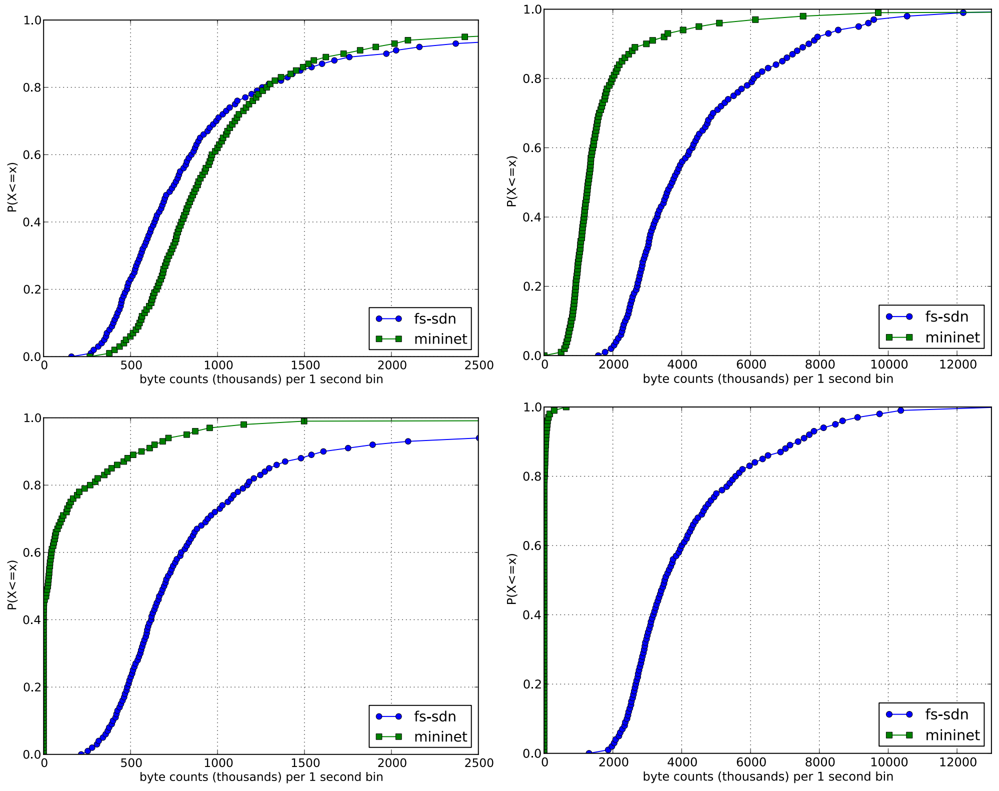

Figure 2: Bytes forwarded per 1 second intervals (in thousands) in Mininet and *fs-sdn* using Harpoon to generate traffic. Results shown for the small (top) and large (bottom) topologies using low traffic load (left) and high traffic load (right).

software-defined network? In *Proceedings of ACM HotSDN '12*, pages 55–60, 2012.

[18] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: saving energy in data center networks. In *Proceedings of USENIX NSDI '10*, pages 17–17, 2010.

[19] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of ACM Internet Measurement Conference, '09*, pages 202–208, 2009.

[20] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proceedings of USENIX NSDI '12*, April 2012.

[21] A. Khurshid, W. Zhou, M. Caesar, and P.B. Godfrey. VeriFlow: verifying network-wide invariants in real time. In *Proceedings of ACM HotSDN '12*, pages 49–54, 2012.

[22] D. Klein and M. Jarschel. An OpenFlow Extension for the OMNeT++ INET Framework. In *6th International workshop on OMNeT++*, March 2013.

[23] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of ACM HotNets '10*, page 19, 2010.

[24] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):82, 1997.

[25] S. McCanne, S. Floyd, K. Fall, K. Varadhan, et al. Network Simulator ns-2, 1997.

[26] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing Software Defined Networks. In *Proceedings of USENIX NSDI '13*, April 2013.

[27] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of ACM SIGCOMM '11*, pages 266–277, 2011.

[28] J. Sommers and P. Barford. Self-Configuring Network Traffic Generation. In *Proceedings of ACM Internet Measurement Conference*, Taormina, Italy, October 2004.

[29] J. Sommers, R. Bowden, B. Eriksson, P. Barford, M. Roughan, and N. Duffield. Efficient network-wide flow record generation. In *Proceedings of INFOCOM '11*, pages 2363–2371, April 2011.

[30] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter. PAST: scalable ethernet for data centers. In *Proceedings of ACM CoNeXT '12*, pages 49–60, 2012.

[31] A. Voellmy and P. Hudak. Nettle: Taking the Sting Out of Programming Network Routers. In *Proceedings of PADL*, pages 235–249, 2011.

[32] A. Voellmy, H. Kim, and N. Feamster. Procera: a language for high-level reactive network control. In *Proceedings of ACM HotSDN '12*, pages 43–48, 2012.