Controller-agnostic SDN Debugging

Ramakrishnan Durairajan
University of Wisconsin-Madison
rkrish@cs.wisc.edu

Joel Sommers
Colgate University
jsommers@colgate.edu

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

ABSTRACT

Complexity in software-defined network (SDN) applications calls for methods and tools that can facilitate comprehensive debugging and analysis. A key challenge in this regard is that SDN configurations interact with network devices that can behave in unexpected ways, depending on factors such as traffic and application mix. In this paper, we describe *OFf*, a debugging and test environment for SDN developers. *OFf* is built on top of the *fs-sdn* simulator, which was developed to offer simple-to-use, accurate and scalable evaluation of OpenFlow-based SDN configurations. *OFf* offers standard debugging features for controller applications such as stepping, breakpoints, and watch variables. It also offers features that provide visibility into network behavior including packet tracing, packet replay and visualization features, and alerts that are triggered when, *e.g.*, configurations change. *OFf* is accessed through a text interface and is designed to interoperate with any standard SDN controller platform. We demonstrate the capabilities of *OFf* through three test scenarios that illustrate its utility and modest performance impact on running applications. Specifically, we show how *OFf* can be used to analyze and fix bugs in a traffic engineering application, and to detect and repair a security vulnerability due to multiple application interaction and unexpected rule expiration.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management; D.2.5 [**Testing and Debugging**]: Testing tools; I.6.3 [**Simulation and Modeling**]: Applications

General Terms

Design, Measurement, Performance, Reliability

Keywords

Debugging; OpenFlow; Simulation; Software-Defined Networks

CoNEXT'14, December 2–5, 2014, Sydney, Australia.

http://dx.doi.org/10.1145/2674005.2674993.

1. INTRODUCTION

Like other software systems, the development of complex SDN configurations requires tools that facilitate debugging. Standard capabilities of debuggers include step-by-step execution, pausing execution, and tracking variables to enable the details of controller applications to be examined. However, the fact that a particular SDN configuration "runs as designed" may be insufficient to ensure that it behaves in a robust and predictable fashion when deployed in a live environment.

SDN deployments must cope with potentially a wide range of operating conditions that include the possibility of unanticipated traffic behavior or interactions between deployed applications. Such conditions can have a range of consequences including degradation of performance, exposure of security vulnerabilities or application failures. The potential severity of these unexpected behaviors calls for robust testing capability that goes beyond standard debugging and includes the ability to assess configurations across a wide range of operating conditions.

In this paper, we describe *OFf* [1], a debugging and test environment for SDN developers. *OFf* is designed for debugging controller applications developed within *any* standard controller environment, such as POX [1], OpenDaylight [2], Ryu [3], Trema [4] and Floodlight [5]. It supports standard debugging capabilities such as stepping, breakpoints, and watch variables through a simple command-line interface. More importantly, it is designed to support comprehensive testing of SDN applications in a representative, controlled and repeatable fashion by providing visibility into network behavior. Key capabilities for testing include packet tracing, packet replay and visualization features, and alerts. *OFf*'s unique capability to simultaneously trace program execution and network state enables unwanted behavior in the network to be tied directly to the control program.

OFf is built on top of *fs-sdn* [6], a simulation environment for SDN that offers the unique capability to accurately assess configurations that might be deployed in large-scale environments such as those found in typical datacenters. *fs-sdn* achieves scalability by using a core abstraction that captures details of *network flows* while avoiding fine-grained packet-level concerns. *OFf* can accommodate any controller platform through a proxy component which has a design similar in spirit to FlowVisor [7], and is situated between a controller under test and the simulated network over which

[1]Source code for *OFf* is openly available to the community and can be found at: `https://github.com/52-41-4d/fs-generic`

the developer has complete control. The proxy translates messages between the controller and the simulated network, and vice versa, and starts a controller in a runtime cradle that masks other differences between the simulated and non-simulated worlds, *e.g.*, simulation time versus wall-clock time. *OFf*s support for fine-grained source code debugging is enabled by simply including a library when coding the application and does not affect program execution unless specifically directed by the developer.

We illustrate the capabilities of *OFf* through three test scenarios. The experiments are designed to highlight how *OFf* can be used to find bugs that might otherwise lead to inconsistent network state or expose security vulnerabilities. Specifically, our examples show how *OFf* can be used to find and fix *(i) race conditions* and *inconsistent ordering* of rule updates in a traffic engineering application, *(ii)* a *security vulnerability* in a multiple application interaction environment, and *(iii)* a security issue due to *rule expiration* and *rule shadowing*.

2. BACKGROUND

In this section we give a brief overview of the *fs-sdn* tool on which *OFf* is built, and describe prior studies that influence and inform the design and implementation of *OFf*.

2.1 *fs-sdn* overview

fs-sdn [6] is a simulation-based tool that is designed to facilitate prototyping and evaluating new SDN applications. It is based on the *fs* [8] tool that was designed to efficiently generate realistic network measurements such as flow records and SNMP-like counters for use in different types of networking studies. *fs*, and by extension *fs-sdn*, use discrete event simulation techniques to generate network measurements and simulate network conditions. Unlike other simulation-based systems, its core abstractions are based on network flows and as a result it achieves significantly better performance than packet-based simulators. *fs-sdn* extended the *fs* engine by transparently incorporating the POX [1] OpenFlow controller framework and API, including switch components that can be controlled and configured through the OpenFlow control protocol. In this work, we significantly extend *fs-sdn* in order to accommodate *any* standard SDN controller platform.

2.2 Related work

Debugging in an SDN world inherits many of the same challenges of debugging complex software. While "`printf`" debugging is, for better or worse, a common practice, there are more effective techniques for development and testing, including using debuggers such as gdb [9] to trace and modify the state of running programs, techniques such as test-driven development (TDD) [10], and the use of assertions to test invariants. *OFf* is most closely related to work on tools for exposing and tracing program and network state in SDN. In particular, ndb [11] and its successor NetSight [12] offer some similar features as *OFf*. A key difference, however, is that *OFf* offers capabilities not only to trace network state, but also to trace controller program execution state, thus tying together observed network behavior with the control program that induced that behavior. *OFf* is also related to the OFRewind [13] system that enables replaying packets collected in an SDN setting in order to understand the effect of different control programs on traffic flows. Also related is the work by Scott *et al.* to associate "minimal causal sequences" of trace data from tests to an observed bug in order to narrow the focus for debugging source code [14]. Somewhat more distantly related to *OFf* are efforts to verify that certain invariants hold. In particular, the works by Kazemian *et al.* on header space analysis [15, 16], the Anteater system [17], Veriflow [18], and NICE [19] each seek to verify that certain network properties are never violated.

3. SYSTEM DESIGN

In this section we describe the design, implementation, and features provided by *OFf*.

3.1 OFf Overview

OFf is a comprehensive source-level debugger for SDN applications. *OFf* does not require any special effort from the developer before debugging an application: the library can simply be included when coding the application like any other standard library. *OFf* also does not require any additional hardware and does not affect the program execution unless the developer issues a debugging command. Therefore, *OFf* can be used spontaneously and only when needed during the application development.

OFf provides both generic debugging commands as well as features that are specifically designed to meet the needs of SDN application developers. The commands and features are described in more detail later in this section. All commands are source-level em i.e., they operate on symbols defined in the source code of the program.

3.2 OFf Architecture

OFf consists of two parts: the *OFf proxy* and *OFf controller/debugger runtime interfaces* that connect to the *fs-sdn* simulator and the SDN controller platforms. The overall architecture is illustrated in Figure 1, and we describe each part below.

3.2.1 OFf Proxy

The proxy unit provides a bridge between the simulated network in *fs-sdn*, and real controller platforms. Openflow switches in *fs-sdn* are configured with one or more controllers, which communicate with proxy components. The proxy component translates simulated messages from switches to control plane packets to a real controller platform, and vice versa. In response to a switch initiating a control plane connection, the proxy creates a real network connection to the configured controller. One proxy may act on behalf of any number of switches, and thus has complete visibility of network control plane interactions.

To enable *fs-sdn* interoperability with multiple controllers, we modified the controller class in *fs-sdn* to behave instead as a proxy, leaving the switch implementation in *fs-sdn* unmodified. We also modified a class in *fs-sdn* that is used to abstract individual connections from the controller to switches. The original class used *fs-sdn*'s API to deliver and receive messages to a fake controller; the new class instead handles connections to real controllers and overrides the *fs-sdn* connection APIs.

Besides its core "bridging" functionality, the proxy is composed of four components. First, a *UI wrapper* provides a text-based interface that dispatches commands from the developer to one of the three other units and prints any output to a display. Second, the *Debugger* component provides

an abstract interface to a language-level debugger in order to associate controller application source code with control plane activity. The debugger component contains separate modules (with enhanced features as described later in the section) to deal with all specialized *OFf* commands such as enable and disable watch points, tracking variables, etc., that are not recognized by language-specific debuggers such as Python's PDB. From an application developer's perspective, simulators producing mere text outputs are less preferred than simulators producing visual output [20]. To that end, we developed the third component—*Trace Replay*—that has the ability to reproduce network activity that has been captured in a trace and replay it later. Finally, the *Diff Report Generator* component helps detect changes in topology, mutations in rules/actions across switches, and performance variations from previous runs (or across configuration changes) of *fs-sdn*, then generates a report to help assess implications of configuration changes.

There are several advantages to our proxy-based approach. First, it completely eliminates the issue of controller compatibility with the simulation environment. That is, the controllers are entirely unmodified and can be used off-the-shelf. Second, the *OFf* proxy acts as a simulation-oriented equivalent of FlowVisor [7], *i.e.*, it acts as a proxy between a real controller and the switches in the network. Instead of the proxy communicating with real switches, however, it communicates via *fs* API calls to simulated switches. This makes the prototyping and debugging generalizable to real networks. Finally, this approach is entirely language and platform agnostic, and opens up interesting avenues for debugging. For instance, since the proxy sees *all* Openflow control messages, it can be used to trace the effects of various high-level operations from the controller code. We also have complete control over the switches, and could augment our current simulated switch implementation to facilitate debugging operations such as tracing packets, observing which rules match (and which one eventually fires) when packets arrive, detect configuration changes, etc.

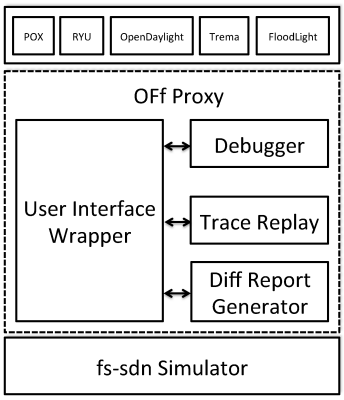

Figure 1: The architecture of OFf.

3.2.2 OFf Controller/Debugger Runtime Interfaces

The *OFf* Proxy described above can be linked to a specific controller platform and language-level debugging environment through an *OFf* runtime interface component. This component handles starting a controller using a library interposition agent [21] that ensures that any time-related calls by the controller will return *fs-sdn* simulation time instead of wall-clock time. This component also provides a translation layer between the generic *OFf* proxy debugging component, and a specific source code debugging environment and command set.

While we have used *OFf* with the OpenDaylight [2], Ryu [3], Floodlight [5], and Trema [4] controllers (shipped as part of [22]), the source-level debugging runtime interface is currently limited to PDB and thus the Python-based controllers. In our ongoing work, we are expanding the debugging runtime interface to accommodate GDB [9], which will enable support of additional controller application source code languages. This effort is primarily one of creating translations between the command set provided in *OFf* (described next) and the equivalent commands available in GDB.

3.3 OFf Commands

Commands in *OFf* are entered through a command-line interface similar to PDB, GDB, and other debuggers. *OFf* supports both basic commands provided by these debuggers, and new features to enhance the SDN application development and debugging experience.

3.3.1 Basic Commands

OFf directly exports the most commonly used commands that are provided by PDB including the ability to pause, continue, and step through execution. It also adds many features to the basic PDB command set by providing the ability to *(i)* longlist and shortlist source code during debugging, *(ii)* pretty print expressions, *(iii)* hide and unhide hidden code frames during debugging, *(iv)* interact (via interpreter) with all variables in scope, *(v)* track, watch, or unwatch variables, *(vi)* edit source files during debugging, *(vii)* enable or disable break points on the fly, and *(viii)* sticky mode visualize code during debugging session.

3.3.2 Additional Features

OFf provides the following additional new features to enhance the SDN application development experience. In what follows, we give a general overview of these additional features, but omit many details due to space constraints.

- **Trace packet through the network.** *fs-sdn* provides the ability to model Openflow switches and controllers. *OFf* can interact directly with these components, providing the ability to trace *fs-sdn flowlets* as they pass through switches and cause Openflow events to be generated (*e.g.*, PacketIn) and controller components and code to be invoked. As a result *OFf* enables a *holistic* view of every network flow, how flows and controller code interact, and provides opportunities to comprehensively trace network events with no additional hardware support needed for data collection and processing (*e.g.*, as in [12]).
- **Packet replay.** Motivated by OFRewind [13], *OFf* includes a light-weight packet replay feature that enables network administrators to reproduce and locate software errors in network configuration. Features in *OFf* differ from OFRewind in the following way: *(a)* there is no explicit record component that *intercepts* all the messages from controller, and the traces are generated in such a way to balance efficiency and accuracy, and *(b)* no additional Openflow protocol-level support is needed

to enable network event tracing and capture of device state (*e.g.*, flow table dump) or to reset network devices to known states (*i.e.*, to clear previously installed rules that led to incorrect behavior). Since *OFf* has an oracular view of the (simulated) network, it does not need any special protocol or hardware support in devices.

- **Detect configuration changes.** Debugging SDN programs is complicated by the fact that program behavior is affected by controller program state, switch state, and network traffic patterns. To combat this complexity, we believe that one way to improve the experience of debugging SDN programs is to provide capabilities for detecting configuration and network changes across multiple executions of a controller program. To that end, *OFf* contains a novel capability via the *Diff Report Generator* to produce an account of differences across several runs of the same control program and network traffic scenario. In particular, it highlights changes in network topology, differences in rules and actions across network switches, and performance variations, and displays these differences to developer in a meaningful way.

4. OFf IN ACTION

In this section, we demonstrate how *OFf* can be used to find logical bugs in the source code that can lead to poor performance or transient outages, and eventually to inconsistent network state or security vulnerabilities. We developed three test scenarios based on ideas from existing literature like [23, 24]. Our approach is to run a series of identical topological simulations in *fs-sdn* with and without *OFf* and identify those bugs that lead to transient outages and losses.

4.1 Scenario 1: Incorrect Ordering of Updates

A testing process with *OFf* can include careful evaluation of traffic before and after rule installation. In this scenario, we used a simple traffic engineering application in the network topology shown in Figure 2. The scenario premise is that a developer has coded an application to install rules in switches in the following order: switch B and C are updated simultaneously (*barrier* messages are not used), then switch A is updated. The developer has not considered latency differences between the controller and switches, leading to a potential race condition. Specifically, rules may be installed at switches C and A, and data packets may start flowing before the appropriate rules are installed at switch B. This inconsistent state will lead to poor performance for traffic flows until switch B is updated, and could result in *e.g.*, initial discontinuity in a VoIP call or high jitter at the start of a video stream.

To resolve this problem, the rules must be installed in the order of switch C, switch B, then switch A. For example, *barrier* messages could be introduced to ensure a sequential ordering of updates and consistent network state.

To identify this bug, the developer could initiate *OFf*'s packet replay feature and find that packets are dropped at switch B because of rules not being correctly preinstalled. The developer might set a *break point* at the rule installation logic and restart the simulation in *sticky mode*. On *stepping* through the execution and *watching* flow table entries and relevant variables at switch B, the developer might observe that there is an *ordering problem* because of the race

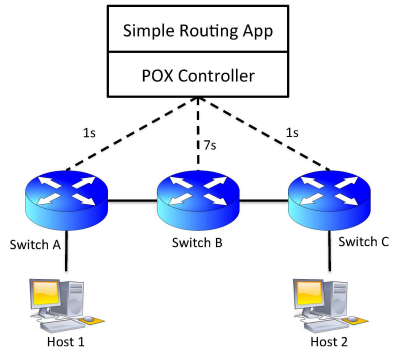

Figure 2: Topology used to evaluate incorrect ordering of updates scenario.

condition between switches B and C. After fixing the bug, the developer could invoke *OFf*'s *trace replay* and *diff report* capabilities to analyze if the expected rules fired in the expected order.

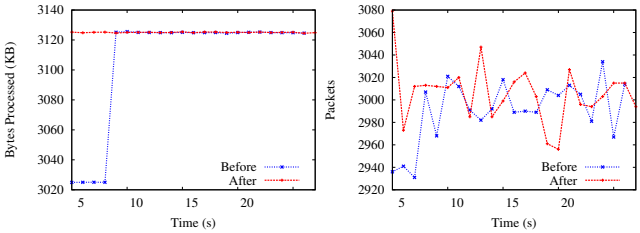

Figure 3: (left) Bytes processed and (right) Packets processed at switch C before and after fixing the bug in scenario 1.

Figure 3 shows the number of packets and bytes processed at switch C before and after the bug is fixed, and generated via *OFf* diff report. In the first few seconds of the connection, before the bug is fixed, there is reduction in the number of packet and bytes processed. Once the bug is fixed (at around 8 sec), there is a noticeable increase in both the number of packets and bytes processed over that same time span.

4.2 Scenario 2: Bad Multi-app Interaction

SDN controllers typically permit multiple, logically separate control programs to be run simultaneously, each of which can receive network events and modify switch state. It is well known that inconsistent network state can arise when SDN programs are not *composed* correctly. We call this problem *Multi-app Interaction Inconsistency* (MAII).

To evaluate this test scenario, we used the topology in Figure 4 (from the *dynamic-flow tunneling* example in [24]). A firewall written by developer 1 installs a rule to block traffic from an external host (10.0.0.1) to an internal web server (10.0.0.4). At the same time, a routing application written by developer 2 is deployed, and at a certain time implements three rules within the controller application. The first rule changes the source IP address of a packet to 10.0.0.2 if the packet is delivered from 10.0.0.1. The second rule changes the destination IP address of a packet to 10.0.0.4 if the packet is destined to 10.0.0.3. The final rule simply

allows packet forwarding from 10.0.0.2 to 10.0.0.3. Unfortunately, now any packet arriving from 10.0.0.1 destined to 10.0.0.3 can bypass the firewall and reach the web server.

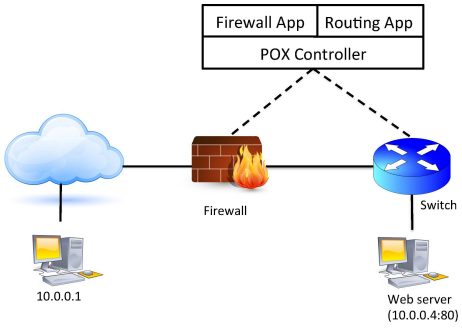

Figure 4: Topology used to evaluate bad multi-app interaction scenario.

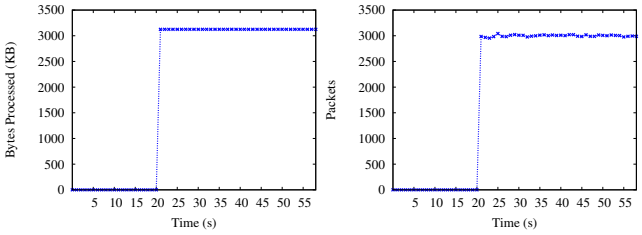

Figure 5: (left) Bytes processed and (right) Packets processed at switch before the Multi-app Interaction Inconsistency bug is fixed in scenario 2.

The MAII problem can be addressed with *OFf* as follows. First, before deploying the routing application, developer 2 can prototype the application and collect network traces using *fs-sdn*. Figure 5 shows the number of packets and bytes seen at the switch after the routing application is prototyped (and before MAII is identified). Once deployed in an environment with the firewall application, network traces can be collected again. At this point, the problem can be easily resolved using *OFf*s *diff reports* as it can report changes across configurations and alert the developer that the routing rule set conflicts with the existing firewall rule set. To verify if the bug is fixed, the developer can then invoke *trace replay* and *diff reports* to analyze if the firewall invariants are maintained. These capabilities of *OFf* highlight its potential to act as a *proactive invariant generator* for tools like VeriFlow [18], which we intend to investigate in future work.

4.3 Scenario 3: Unexpected Rule Expiration

It is well known that unexpected interactions can occur when wildcarded rules overlap, or specific *microflow* rules are shadowed by wildcard rules. If the actions associated with the rules differ, unexpected behavior can occur if the more specific rule is removed while traffic is still flowing.

In this scenario, we extended the topology from scenario 2 to include an additional switch (switch D, attached to switch B), as depicted in Figure 6. Assume for this scenario that switch B sees two types of flows: *trusted* traffic, which is forwarded to switch D, and *untrusted* traffic, which is forwarded to switch C. A wildcarded rule (*e.g.*, for 10.0.0.0/8) is installed at switch B for untrusted traffic, and a separate rule for 10.5.0.0/16 exists for forwarding trusted traffic. Note that the trusted rule is *shadowed* by the untrusted rule (*i.e.*, the untrusted rule is more general). Lastly, assume that the rule for trusted traffic expires due to a hard timeout (at t=30 sec), causing all trusted traffic to erroneously be forwarded to switch C and exposing a potential security vulnerability.

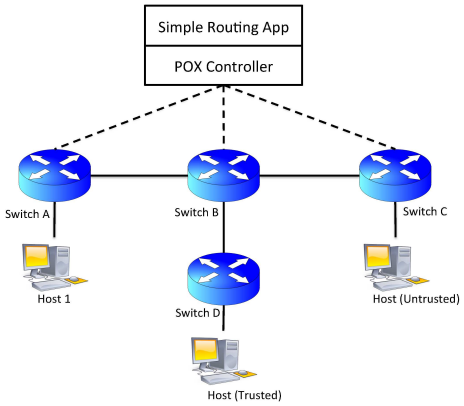

Figure 6: Topology used to evaluate unexpected rule expiration scenario.

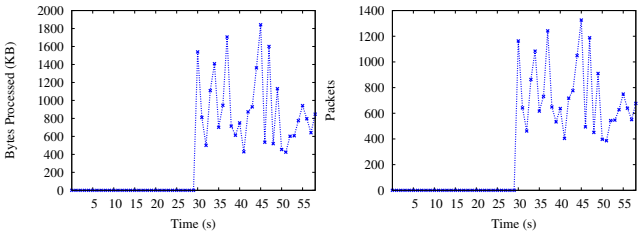

Figure 7: (left) Bytes processed and (right) Packets processed at switch C before rule expiration bug in scenario 3 is fixed.

A developer might use *OFf* in the following way to evaluate and debug this problem. First, after prototyping the application in *fs-sdn*, flows are captured and replayed using *OFf*s *trace replay* facilities and the problematic leak of flows to switch C is identified. Figure 7 shows the number of packets and bytes processed at switch C; observe that no traffic is seen prior to t=30 when the more specific rule expires and the wrong forwarding rule is applied. At this point, the developer could trace the flows and the specific rules that are applied to observe that the wrong rule is triggered. The developer could then change the timeout behavior of the rule for forwarding trusted traffic, or use some other approach for fixing the problem. Once fixed, the developer could invoke *trace replay* and *diff reports* to verify that trusted traffic is never forwarded to switch C.

4.4 Performance Impact of OFf

For the tests described above, we examined the performance impact of *OFf* by enabling between 1 and 3 breakpoints, and between 1 and 3 *watch* variables and comparing with an *OFf*-less simulation. Table 1 shows execution

times for scenarios consisting of either 60 or 180 simulated seconds. We see from the table that breakpoints have a relatively small impact on runtime compared with watched variables, especially for shorter simulations. We also observe that overheads of *OFf* are amortized for longer simulations and that performance impacts are, overall, modest.

Table 1: Wall-clock execution times for an SDN simulation scenario with and without *OFf* capabilities enabled.

	60 simulated seconds		180 simulated seconds	
	w/o *OFf*	with *OFf*	w/o *OFf*	with *OFf*
Break 1	2.254	3.419	6.568	7.411
Break 2	2.254	3.765	6.568	7.700
Break 3	2.254	4.005	6.568	7.942
Watch 1	2.254	8.261	6.568	10.883
Watch 2	2.254	8.843	6.568	11.010
Watch 3	2.254	9.677	6.568	11.318

5. SUMMARY AND FUTURE WORK

Ensuring that SDN configurations behave as expected is predicated on careful and comprehensive debugging and testing. In this paper, we describe *OFf*, a controller-agnostic SDN debugging and testing tool that provides standard debugging capabilities such as stepping and watch variables, as well as SDN-specific capabilities to assess details of network interactions and changes over iterations of the same program. *OFf* is built on top of *fs-sdn*, which provides accurate and scalable simulation of OpenFlow-based SDN configurations. We highlight the capabilities of *OFf* through three test scenarios, and demonstrate the tool's utility and modest performance impact on running applications. We show that *OFf* can be used to identify and eliminate bugs in a traffic engineering application, and to identify and remove a security vulnerability that is exposed by the interaction of multiple applications, and unexpected rule expiration. *OFf* is openly available to the community and development of additional features and capabilities is ongoing.

In future work, we plan to focus on the following extensions and applications of *OFf*:

- **Application-level consistency.** We intend to evaluate several applications for which the source code is available by running failure scenarios across configurations to identify consistency issues. In particular, we plan to select applications based on problems identified in prior work, and where a tool like *OFf* would be helpful. For instance, a key finding in [25] is that middleboxes experience a variety of misconfigurations and that failover between middlebox replicas can be ineffective due to configuration bugs. As another example, OF.CPP [26] presents two classes of bugs related to consistency issues, and we believe that capabilities in *OFf* may provide ways to completely avoid such bugs. We also plan to investigate the potential for *OFf* to generate *invariants* for efforts like SIMPLE [27], HotSwap [28], and PyResonance [29].
- **Switch implementation models.** The work by Huang *et al.* [30] argues that vendor-specific emulation of SDN switches are required, and describes a method to record the switch behavior of various vendors. The *OFf* proxy component is a natural place for implementing such fingerprinting methods and could be used to evaluate such models by comparing multiple switch models simultaneously. To that end, we intend to add a benchmarking capability in *OFf* to record various vendor-specific switch implementation models and establish ideal switch models for a wide variety of configurations.
- **Distributed controller simulations.** To provide reliability and scalability, SDN controllers should be logically centralized and physically distributed. A natural requirement for such distributed controllers (*e.g.,* [31]) is correct and consistent behavior across distributed configurations. Since multiple controllers are supported by *fs-sdn*, *OFf* could be used as an orchestration layer to test distributed controllers and ensure correctness by identifying errors across distributed configurations.

6. ACKNOWLEDGMENTS

We thank the reviewers for their invaluable feedback. This material is based upon work supported by the National Science Foundation under grants CNS-1054985, CNS-0905186, ARL grant W911NF1110227, DHS BAA 11-01 and AFRL grant FA8750-12-2-0328. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF, ARL, DHS or AFRL.

7. REFERENCES

[1] POX, Python-based OpenFlow Controller. `http://www.noxrepo.org/pox/about-pox/`.
[2] The OpenDaylight Controller. `http://www.opendaylight.org`.
[3] The Ryu Framework. `http://osrg.github.io/ryu/`.
[4] The Trema Controller. `http://trema.github.io/trema/`.
[5] The Floodlight Controller. `http://www.projectfloodlight.org/floodlight/`.
[6] M. Gupta, J. Sommers, and P. Barford. Fast, Accurate Simulation for SDN Prototyping. In *Proceedings of ACM HotSDN*, 2013.
[7] R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Proceedings of OSDI*, 2010.
[8] J. Sommers, R. Bowden, B. Eriksson, P. Barford, M. Roughan, and N. Duffield. Efficient network-wide flow record generation. In *Proceedings of INFOCOM*, 2011.
[9] GDB: The GNU Project Debugger. `http://www.gnu.org/software/gdb/`.
[10] S. Vissicchio D. Lebrun and O. Bonaventure. Towards test-driven software defined networking. In *Proceedings of IEEE NOMS*, May 2014.
[11] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown. Where is the Debugger for My Software-defined Network? In *Proceedings of ACM HotSDN*, 2012.
[12] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*, 2014.

[13] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann. OFRewind: Enabling Record and Replay Troubleshooting for Networks. In *Proceedings of the 2011 USENIX Annual Technical Conference*, 2011.

[14] C. Scott, A. Wundsam, B. Raghavan, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, H. B. Acharya, K. Zarifis, and S. Shenker. Troubleshooting SDN Control Software with Minimal Causal Sequences. In *Proceedings of the ACM SIGCOMM*, 2014.

[15] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[16] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real Time Network Policy Checking Using Header Space Analysis. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, 2013.

[17] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King. Debugging the Data Plane with Anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011.

[18] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying Network-wide Invariants in Real Time. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, 2013.

[19] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford. A NICE Way to Test Openflow Applications. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[20] E. Tufte. The Visual Display of Quantitative Information.

[21] Timothy W. Curry and Sun Microsystems Inc. Profiling and tracing dynamic library usage via interposition. In *Proc. of the USENIX Summer 1994 Technical Conf*, pages 267–278, 1994.

[22] All-in-one SDN App Development Starter VM. `http://sdnhub.org/tutorials/sdn-tutorial-vm-64-bit/`.

[23] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent Updates for Software-defined Networks: Change You Can Believe in! In *Proceedings of the ACM HotNets*, 2011.

[24] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A Security Enforcement Kernel for OpenFlow Networks. In *Proceedings of the ACM HotSDN*, 2012.

[25] R. Potharaju and N. Jain. Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters. In *Proceedings of the ACM Internet Measurement Conference*, 2013.

[26] P. Perešíni, M. Kuzniar, N. Vasić, M. Canini, and D. Kostiū. OF.CPP: Consistent Packet Processing for Openflow. In *Proceedings of ACM HotSDN*, 2013.

[27] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of the ACM SIGCOMM*, 2013.

[28] L. Vanbever, J. Reich, T. Benson, N. Foster, and J. Rexford. HotSwap: Correct and Efficient Controller Upgrades for Software-defined Networks. In *Proceedings of ACM HotSDN*, 2013.

[29] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark. Resonance: Dynamic Access Control for Enterprise Networks. In *Proceedings of the ACM Workshop on Research on Enterprise Networking*, 2009.

[30] D. Y. Huang, K. Yocum, and A. C. Snoeren. High-fidelity switch models for software-defined network emulation. In *Proceedings of ACM HotSDN*, 2013.

[31] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of USENIX OSDI*, 2010.