PorkFuzz: Testing Stateful Software-Defined Network Applications with Property Graphs

Chaofan Shou
University of California, Santa Barbara
Santa Barbara, CA, USA
shou@cs.ucsb.edu

ABSTRACT

This paper proposes a state-aware fuzzing framework for testing software-defined network applications. It leverages a property graph to store fuzzing results. Application developers can easily express oracles with the graph query language to test their applications. The graph representation also allows the oracles to analyze the fuzzing result efficiently.

CCS CONCEPTS

• **Networks** → **Programmable networks**; • **Software and its engineering** → **Software verification and validation**.

KEYWORDS

coverage-guided SDN fuzzing; network validation

ACM Reference Format:
Chaofan Shou. 2021. PorkFuzz: Testing Stateful Software-Defined Network Applications with Property Graphs. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3468264.3473487

1 INTRODUCTION

Software-defined network (SDN)[14, 19] has been proposed to mitigate the management dilemma of the increase in network complexity. SDN decouples network infrastructure to forwarding plane (i.e., programmable switches) and control plane. Multiple centralized servers serving as the control plane oversee the states of the forwarding plane and modify the states if triggered. For the forwarding plane, DSLs[8, 24] are introduced for its definition. A typical forwarding plane program can be represented as a pipeline with a finite state machine that parses the packet to be structural data, then a few match-action tables that mutate the structural data or state based on these information, and finally a deparser that reassembles the structural data into a packet. The combination of programmable forwarding plane and control plane enables the development of vibrant network applications, like telemetry system[11, 23], DDoS detection[13, 15], and network side-channel protection[17].

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08.
https://doi.org/10.1145/3468264.3473487

Adoption of SDN introduces new attack surfaces. A simple failure or error could lead to significant impact, including allowing arbitrary access to confidential information[3] and network outage[1, 10]. Thus, testing and verifying the SDN applications is of great importance. However, securing a network application is challenging as there are multiple stakeholders and indeterministic factors in the network.

Fuzzing is commonly used for detecting bugs in software [4, 5, 9, 12, 18, 25]. We applied fuzzing for testing SDN applications and propose a state-aware fuzzing framework, which is named as PorkFuzz. PorkFuzz supplies inputs to both control plane and forwarding plane. Then, it records results (e.g., egress packets) and state changes in a property graph. For expressing oracles that could analyze these information, the framework provides an abstraction for developers. Such an abstraction is based on graph query language, which supports relational inference and aggregation.

2 RELATED WORK

Previous works including **p4pktgen**[16] leverage concolic testing techniques to generate packet inputs that could achieve high coverage of the forwarding plane program for testing. This method does not model the relationships between the forwarding plane and control plane. It randomly synthesizes the states that may not be reproducible, leading to false positives.

Fuzzing solutions like **P4RL**[22] and **P6**[21] tests forwarding plane programs using coverage-guided fuzzers. However, their approaches does not consider the state of forwarding plane (e.g., states are never recovered for re-exploration), and disallow aggregation-based oracles.

Other testing works like **P4Fuzz**[7] and **Gauntlet**[18] leverage coverage guided fuzzing to test the SDN application compilers and runtimes of different switches. **P4Consist**[20] is a tool to identify the inconsistency between states of control plane and forwarding plane. This is similar to our work in regards to checking the system as a whole but our work presents a more general fuzzing framework that is state-aware, checks beyond forwarding decisions, and allows efficiently expressing oracles.

3 METHODOLOGY

At the high level, we define a coverage-guided state-aware fuzzing method to test the SDN applications. The test coverage information is collected from the virtual switch runtime and oracles are executed after the fuzzer terminates. In addition to conventional fuzzing strategy, PorkFuzz recovers states based on snapshots. After a packet (i.e., testcase) from the corpus is mutated and sent to either control plane or forwarding plane, a hash of the state is then calculated. When a state hash is unique (i.e., never seen before),

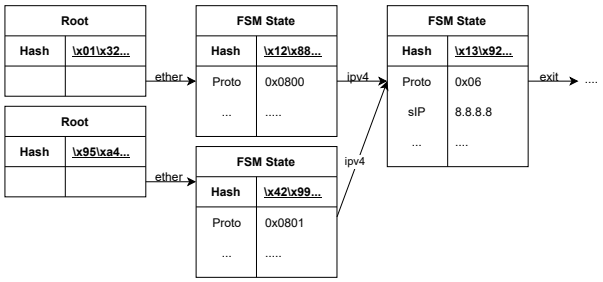

Figure 1: An example of the graph for two TCP packets

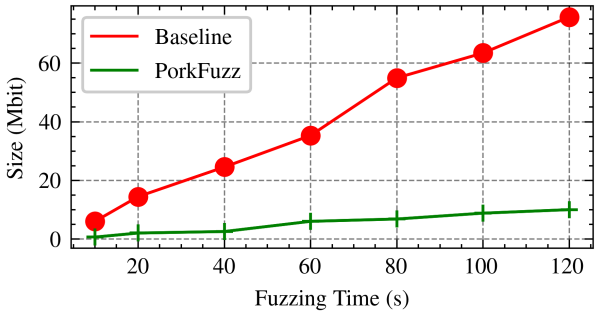

Figure 2: Fuzzing result size of baseline and PorkFuzz

PorkFuzz retrieves all state information required to recover such a state and save it to the priority queue. The state is assigned with a high priority, signifying it is the least explored state. When the coverage of a runtime instance does not increase over a threshold amount of times, the instance is released. The released instance is then recovered with a state having the highest priority in the priority queue, and the priority of that state would be decreased.

The fuzzing result is saved as a property graph. Each test case and its corresponding results could be expressed by two classes of trees known as Packet and State. For each test case, a packet is supplied to forwarding plane or control plane test harness. The forwarding plane packet (i.e., a list of bits) is represented as a Packet tree. We introduce a type of child nodes representing the FSM states for parsing the packet, namely FSM state nodes, by tracking the parser of the forwarding plane program. Each node in the tree carries the key value information of the structure extracted by the parser at that state. Each node also has a field with the unique hash of remaining packet content for merging same nodes together (e.g., Figure-1). FSM information is important for oracle construction since it translates a packet in bits to structural data that could be efficient queried by graph query language.

Processing each packet results in either identical or different state for switches. We represent a state as State tree. Root node for this contains hash of the state so as to aggregate packets executing in such state. State of most SDN models could be simply expressed by a set of key-value pairs that could be saved directly in the root node. Some introduce the concept of array, which could be expressed as some child nodes with a parent node containing key information.

Bug #	Bug Cause	P4RL	p4pktgen	PorkFuzz
1[21]	Forwarding Plane	✓		✓
2[21]	Forwarding Plane	✓	✓	✓
3[21]	Runtime	✓	✓	✓
4[2]	Runtime			✓
5[20]	Runtime		✓	✓

Table 1: Comparison of different approaches on discovering bugs after fuzzing applications for 10 minutes

4 EXPERIMENT

We have implemented PorkFuzz in Python with 800 LoCs. Neo4J[6], a widely-used property graph database, and Cypher, the inherent graph query language, are used to query and store the graph.

To test our approach, we fuzzed SDN applications with known bugs. For evaluating the growth of data size of the result produced by PorkFuzz and required by the oracles, we implement a baseline approach that directly output the result in pcap files and raw state dump. The result is shown in Figure-2, in which the growth rate of raw data size is higher than that of the property graph. The difference in growth rate is due to the majority of packets have the same FSM states, which enables property graph to reuse a significant amount of nodes as the fuzzer progresses.

In Table 1[1], PorkFuzz is able to discover a new bug that is not able to be discovered by both P4RL[22] and p4pktgen[16] since their approach can not efficiently re-explore states and do not support control plane testcases. Below, we provide the case study on identifying Bug 1 and Bug 4. Oracles in the case study use CPP and FPP referring to all packets sent to control plane and forwarding plane respectively.

Bug 1 is caused by forwarding plane program not discarding packets that have TTL as 0. By running following oracle, PorkFuzz is able to identify the packets that are not dropped.

$$\forall x \in \text{FPP}, x.\text{ttl} = 0 \rightarrow !x.\text{port}$$

Using the result after fuzzing the program for 10 minutes, the graph based oracle takes 8 seconds to identify all counter cases while directly analyzing the pcap files in Python takes 106 seconds.

Bug 4 is caused by runtime failing to handle a corner case in implementation of longest prefix match (LPM) and exact match. We use following oracle, which states that a packet should be forwarded with respect to the LPM table entries (i.e., select the appropriate forwarding entry with the largest mask).

$$P(x, c) = \forall x \in \text{FPP}, \forall c \in x.\text{table}, x.\text{ip} \in Range[c_s, c_e]$$

$$\forall x \in \text{FPP}, argmax_{c_{\text{mask}}}(P(x, c)) \rightarrow (x.\text{fp} = c.\text{fp})$$

On the same property graphs, this oracles spend 44 seconds for execution and the fuzzer identified two violations. Directly using Python to analyze the pcap files and state dumps, on the other hand, takes 842 seconds.

5 CONCLUSION

We proposed PorkFuzz, a fuzzing solution that leverages property graphs, and demonstrated that it is efficient for testing stateful SDN applications.

[1]We reimplemented P4RL in Python and created a custom oracle inside the virtual switch for p4pktgen

REFERENCES

[1] 2012. Microsoft: misconfigured network device led to Azure outage. https://www.datacenterdynamics.com/en/news/microsoft-misconfigured-network-device-led-to-azure-outage/

[2] 2016. Exposed bug in LPM match unit's add_entry: p4lang/behavioral-model@4323bfb. https://github.com/p4lang/behavioral-model/commit/4323bfb1b9e9331f6dd185c5927d3f015cdd1e85

[3] 2017. Cloud Leak: WSJ Parent Company Dow Jones Exposed Customer Data | UpGuard. https://www.upguard.com/breaches/cloud-leak-dow-jones

[4] 2021. Domato - DOM fuzzer. https://github.com/googleprojectzero/domato original-date: 2017-09-21T15:28:59Z.

[5] 2021. Fuzzilli - A JavaScript Engine Fuzzer. https://github.com/googleprojectzero/fuzzilli original-date: 2019-03-20T15:32:47Z.

[6] 2021. Neo4J. https://neo4j.com/

[7] Andrei-Alexandru Agape, Madalin Claudiu Danceanu, Rene Rydhof Hansen, and Stefan Schmid. 2021. P4Fuzz: Compiler Fuzzer ForDependable Programmable Dataplanes. In *International Conference on Distributed Computing and Networking 2021* (Nara, Japan) *(ICDCN '21)*. Association for Computing Machinery, New York, NY, USA, 16–25. https://doi.org/10.1145/3427796.3427798

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[9] Tegan Brennan, Seemanta Saha, and Tevfik Bultan. 2020. JVM Fuzzing for JIT-Induced Side-Channel Detection. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1011–1023. https://doi.org/10.1145/3377811.3380432

[10] Net economy. 2012. France seeks influence on telcos after outage. http://blogs.strategygroup.net/wp2/economy/2012/07/11/france-seeks-influence-on-telcos-after-outage/

[11] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-Driven Streaming Network Telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 357–371. https://doi.org/10.1145/3230543.3230555

[12] Kyungtae Kim, Dae R. Jeong, Chung Hwan Kim, Yeongjin Jang, Insik Shin, and Byoungyoung Lee. 2020. HFL: Hybrid Fuzzing on the Linux Kernel. In *Proceedings 2020 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. https://doi.org/10.14722/ndss.2020.24018

[13] Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. 2019. Offloading Real-time DDoS Attack Detection to Programmable Data Planes. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 19–27.

[14] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. https://doi.org/10.1145/1355734.1355746

[15] Francesco Musumeci, Valentina Ionata, Francesco Paolucci, Filippo Cugini, and Massimo Tornatore. 2020. Machine-learning-assisted DDoS attack detection with P4 language. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC40277.2020.9149043

[16] Andres Nötzli, Jehandad Khan, Andy Fingerhut, Clark Barrett, and Peter Athanas. 2018. p4pktgen: Automated Test Case Generation for P4 Programs. *Proceedings of the Symposium on SDN Research* (2018). https://doi.org/10.1145/3185467.3185497

[17] Antônio J. Pinheiro, Paulo Freitas de Araujo-Filho, Jeandro de M. Bezerra, and Divanilson R. Campelo. 2021. Adaptive Packet Padding Approach for Smart Home Networks: A Tradeoff Between Privacy and Performance. *IEEE Internet of Things Journal* 8, 5 (2021), 3930–3938. https://doi.org/10.1109/JIOT.2020.3025988

[18] Fabian Ruffy, Tao Wang, and Anirudh Sivaraman. 2020. Gauntlet: Finding Bugs in Compilers for Programmable Packet Processing. arXiv:2006.01074 [cs.NI]

[19] Myung-Ki Shin, Ki-Hyuk Nam, and Hyoung-Jun Kim. 2012. Software-defined networking (SDN): A reference architecture and open APIs. In *2012 International Conference on ICT Convergence (ICTC)*. 360–361. https://doi.org/10.1109/ICTC.2012.6386859

[20] Apoorv Shukla, Seifeddine Fathalli, Thomas Zinner, Artur Hecker, and Stefan Schmid. 2020. P4Consist: Toward Consistent P4 SDNs. *IEEE Journal on Selected Areas in Communications* 38, 7 (2020), 1293–1307. https://doi.org/10.1109/jsac.2020.2999653

[21] Apoorv Shukla, Kevin Hudemann, Zsolt Vági, Lily Hügerich, Georgios Smaragdakis, Stefan Schmid, Artur Hecker, and Anja Feldmann. 2020. Towards Runtime Verification of Programmable Switches. arXiv:2004.10887 [cs.SE]

[22] Apoorv Shukla, Kevin Nico Hudemann, Artur Hecker, and Stefan Schmid. 2019. Runtime Verification of P4 Switches with Reinforcement Learning. *Proceedings of the 2019 Workshop on Network Meets AI & ML - NetAI19* (2019). https://doi.org/10.1145/3341216.3342206

[23] Dongeun Suh, Seokwon Jang, Sol Han, Sangheon Pack, and Xiaofei Wang. 2020. Flexible sampling-based in-band network telemetry in programmable data plane. *ICT Express* 6, 1 (2020), 62–65. https://doi.org/10.1016/j.icte.2019.08.005

[24] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, and Kunle Olukotun. 2020. Taurus: An Intelligent Data Plane. arXiv:2002.08987 [cs.NI]

[25] Meng Xu, Sanidhya Kashyap, Hanqing Zhao, and Taesoo Kim. 2020. Krace: Data Race Fuzzing for Kernel File Systems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 1643–1660. https://doi.org/10.1109/SP40000.2020.00078