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**.

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)[1, 2] 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[3, 4] 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[5, 6], DDoS detection[7, 8], and network side-channel protection[9].

Adoption of SDN introduces new attack surfaces. A simple failure or error could lead to significant impact, including allowing arbitrary access to confidential information[10] and network outage[11, 12]. 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 [13–18]. 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 (i.e., the egress packets and egress ports) 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**[19] 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**[20] and **P6**[21] tests forwarding plane programs using coverage-guided fuzzers. They define high-level abstractions for expressing oracles. 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**[22] and **Gauntlet**[14] leverage coverage guided fuzzing to test the SDN application compilers for different switches. **P4Consist**[23] 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), the fuzzer would retrieve all state information required to recover such a state and save it to the priority queue. The state would be assigned with a high priority, signifying it is the least explored

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

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473487

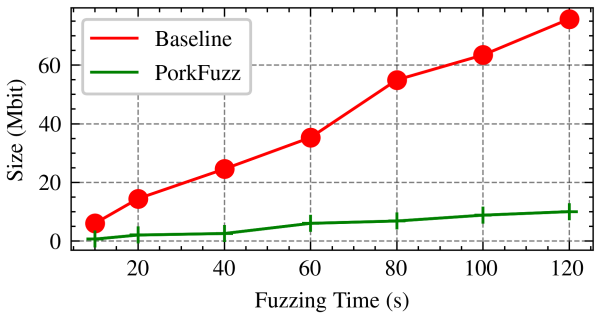

Figure 1: Fuzzing result size of baseline approach and PorkFuzz

state. When the coverage of a runtime instance does not increase over a threshold amount of times, the instance would be released. The released instance would be recovered with a state having the highest priority in the priority queue, and the priority of that state would be decreased accordingly.

The fuzzing result is saved as a property graph. Each test case and its corresponding results could be expressed by three classes of trees known as Input, Result, and State.

For each test case, a packet would be supplied to forwarding or control plane. The forwarding plane packet (i.e., a list of bits) is represented as a Input tree. A root node with field expressing the unique hash of the packet is created to merge same packets together. Additionally, we introduce several types 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 substree carry the key value information of the structure extracted by the parser at that state. Since the node inside property graphs is unable to express a recursive architecture, PorkFuzz uses nodes with concrete information and multiple abstract nodes containing key information, which map to their concrete information at their child nodes. 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. In Result tree, the root node would contain information of the decision of the switch and the hash of the egressing packets. Children of root node would be expressed analogous to Input.

Processing each packet would result 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 based on 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.

4 EXPERIMENT

We have implemented the fuzzer in Python with 800 LoCs. Neo4J[24], 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 on a Xeon Phi (256 vCores) node. For evaluating the growth of data size of the result produced by the fuzzer 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 1, 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.

Bug #	Bug Cause	P4RL	p4pktgen	PorkFuzz
1[21]	Forwarding Plane	✓		✓
2[21]	Forwarding Plane	✓	✓	✓
3[21]	Runtime	✓	✓	✓
4[25]	Runtime			✓
5[23]	Runtime		✓	✓

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

In Table 1[1], PorkFuzz is able to discover a new bug that is not able to be discovered by both P4RL[20] and p4pktgen[19] since their approach can not efficiently re-explore states and mutate state given that they do not support control plane testcases. Additionally, both P4RL and p4pktgen require manual instrumentation in the their implementations so as to provide information for the oracles for Bug 4 and 5. Below, we provide the case study on identifying Bug 1 and Bug 4. Oracles in the case study use CPP as the set of control plane packets and FPP referring to all packets sent to be processed by forwarding plane.

Bug 1 is caused by forwarding plane program not discarding packets that have TTL as 0. By running following oracle on the property graphs, 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 would take 106 seconds.

Bug 4 is due to runtime fails 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 longest prefix).

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

$$\forall x \in \text{FPP}, \mathit{argmax}_{c.\text{prefix}}(P(x, c)) \rightarrow (x.\text{fp} = c.\text{fp})$$

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 and created a custom oracle inside the virtual switch for p4pktgen in Python

Special thanks to Arpit Gupta and Tevfik Bultan.

REFERENCES

[1] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *2012 International Conference on ICT Convergence (ICTC)*, 2012, pp. 360–361.

[2] 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, p. 69–74, Mar. 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[4] T. Swamy, A. Rucker, M. Shahbaz, and K. Olukotun, "Taurus: An intelligent data plane," 2020.

[5] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 357–371. [Online]. Available: https://doi.org/10.1145/3230543.3230555

[6] D. Suh, S. Jang, S. Han, S. Pack, and X. Wang, "Flexible sampling-based in-band network telemetry in programmable data plane," *ICT Express*, vol. 6, no. 1, pp. 62–65, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959519302358

[7] C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 19–27.

[8] F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-assisted ddos attack detection with p4 language," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[9] A. J. Pinheiro, P. Freitas de Araujo-Filho, J. de M. Bezerra, and D. R. Campelo, "Adaptive packet padding approach for smart home networks: A tradeoff between privacy and performance," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3930–3938, 2021.

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

[11] "Microsoft: misconfigured network device led to Azure outage." [Online]. Available: https://www.datacenterdynamics.com/en/news/microsoft-misconfigured-network-device-led-to-azure-outage/

[12] N. economy, "France seeks influence on telcos after outage," Jul. 2012. [Online]. Available: http://blogs.strategygroup.net/wp2/economy/2012/07/11/france-seeks-influence-on-telcos-after-outage/

[13] T. Brennan, S. Saha, and T. Bultan, "Jvm fuzzing for jit-induced side-channel detection," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1011–1023. [Online]. Available: https://doi.org/10.1145/3377811.3380432

[14] F. Ruffy, T. Wang, and A. Sivaraman, "Gauntlet: Finding bugs in compilers for programmable packet processing," 2020.

[15] "Fuzzilli - A JavaScript Engine Fuzzer," Apr. 2021, original-date: 2019-03-20T15:32:47Z. [Online]. Available: https://github.com/googleprojectzero/fuzzilli

[16] "Domato - DOM fuzzer," Apr. 2021, original-date: 2017-09-21T15:28:59Z. [Online]. Available: https://github.com/googleprojectzero/domato

[17] K. Kim, D. R. Jeong, C. H. Kim, Y. Jang, I. Shin, and B. Lee, "HFL: Hybrid Fuzzing on the Linux Kernel," in *Proceedings 2020 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2020. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24018.pdf

[18] M. Xu, S. Kashyap, H. Zhao, and T. Kim, "Krace: Data Race Fuzzing for Kernel File Systems," in *2020 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2020, pp. 1643–1660. [Online]. Available: https://ieeexplore.ieee.org/document/9152693/

[19] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, and P. Athanas, "p4pktgen: Automated test case generation for p4 programs," *Proceedings of the Symposium on SDN Research*, 2018.

[20] A. Shukla, K. N. Hudemann, A. Hecker, and S. Schmid, "Runtime verification of p4 switches with reinforcement learning," *Proceedings of the 2019 Workshop on Network Meets AI ML - NetAI19*, 2019.

[21] A. Shukla, K. Hudemann, Z. Vági, L. Hügerich, G. Smaragdakis, S. Schmid, A. Hecker, and A. Feldmann, "Towards runtime verification of programmable switches," 2020.

[22] A.-A. Agape, M. C. Danceanu, R. R. Hansen, and S. Schmid, "P4fuzz: Compiler fuzzer fordependable programmable dataplanes," in *International Conference on Distributed Computing and Networking 2021*, ser. ICDCN '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 16–25. [Online]. Available: https://doi.org/10.1145/3427796.3427798

[23] A. Shukla, S. Fathalli, T. Zinner, A. Hecker, and S. Schmid, "P4consist: Toward consistent p4 sdns," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, p. 1293–1307, 2020.

[24] "Neo4j." [Online]. Available: https://neo4j.com/

[25] "Exposed bug in LPM match unit's add_entry: p4lang/behavioral-model@4323bfb." [Online]. Available: /p4lang/behavioral-model/commit/4323bfb1b9e9331f6dd185c5927d3f015cdd1e85