Probabilistic Profiling of Stateful Data Planes for Adversarial Testing (Extended Abstract)

Qiao Kang Jiarong Xing Yiming Qiu Ang Chen

Rice University

1. Motivation

Our work is motivated by the design of *programmable data planes* [7] in recent switch models, which in turn has sparked significant interest in customizing *program analysis and profiling* techniques for data plane programs written in P4 [39, 32, 15, 33, 12, 11, 37].

Programmable data planes. Traditional network devices are hardwired for a fixed set of protocols. Recently, emerging network hardware is reprogrammable in high-level languages like P4 [7] to customize protocols and processing behaviors for each network and device. For instance, programmable data planes can support customized header fields and protocol types, enabling new protocols to be deployed without hardware upgrades. They can perform sophisticated operations over header fields, enabling task offloading to the switches. They can also implement stateful data structures in hardware, making it possible for the network to adapt its behavior based on past events. Commercial off-the-shelf switch models include Intel FlexPipe [4], Barefoot Tofino [1], Broadcom Trident 4 [2], and Cisco Silicon One [3].

Innovations with programmable data planes have quickly evolved from implementing *forwarding* programs (e.g., IP forwarding, MPLS, or customized network protocols) to supporting far more sophisticated processing inside the network—in this paper, we call these second-generation P4 programs *data plane systems*. Example systems in this category include key/value caching [24], load balancing [26, 21], link failure detection [19], access control [25], covert channel detection [43], likely with many more to come. Data plane systems have more complex program behaviors than forwarding programs, because they perform *stateful* processing. Consider the Blink link failure detector [19]: it randomly samples a set of TCP flows, tracks per-flow retransmissions, keeps a sliding window for monitoring, and activates rerouting to backup paths in a round-robin manner. Historical traffic patterns will directly influence future processing decisions. In contrast, forwarding programs contain no or little state, so their program behaviors do not change based on past traffic patterns.

Program profiling and analysis. Observing that programmable data planes are essentially "complex programs", researchers have been actively adapting program analysis, profiling, and verification techniques [39, 33, 12, 32, 37, 11] for P4 programs. Our work is particularly related to symbolic execution tools for P4 programs [39, 33, 12] . Symbolic execution (symbex) [29] is a principled program profiling technique that aims to systematically explore all program execution paths, analyze which types of inputs would exercise what execution paths, and produce concrete test inputs for validation.

However, existing symbex tools for P4 programs [39, 33, 12] are restricted to *stateless* forwarding program analysis, and they cannot support the second generation of *data plane systems*. As motivated earlier, as the trend of in-network offloading continues, many recent P4 programs have increasingly complex state; exercising these stateful execution paths requires identifying a precise *sequence of packets*. But today's P4 profilers [39, 33, 12] can only generate *single-packet* test cases regardless of program state. This falls short in analyzing stateful program behaviors that are driven by historical traffic inputs. Our work fills this gap and develops support to analyze stateful data plane systems.

Concurrently, our work is driven by another fundamental observation made by the program analysis community—*network behaviors are inherently probabilistic*. Depending on the properties of interest, stochastic traffic patterns, random failures, load balancing decisions, and many other factors would contribute to this inherent non-determinism. In other words, analyzing network programs *as if* all properties are deterministic—as early work in this space did [14, 17, 13]—simply does not match the reality of complex networks. A promising line of work has developed support for *probabilistic* network analysis [41, 16, 36, 35, 38], but so far they only focus on analyzing network configurations [41, 38], or designing new probabilistic network programming languages [16, 36, 35]. Our work shares the same motivation with these work, but it is the first to enable probabilistic profiling of data plane systems.

Our application: Adversarial testing. We discuss several applications in the main paper, but focus on demonstrating one use case in depth: *adversarial testing*. In contrast to basic program testing, adversarial testing distinguishes and specifically focuses on edge cases, as they may likely lead to unexpected behaviors. This is motivated by a long body of work that automates the finding of adversarial inputs for different types of systems and scenarios [34, 8, 42, 28, 31, 5, 23, 30, 20, 22, 18]. The most related work includes a) using machine learning techniques to find adversarial inputs for network protocols [18], and using execution cost aware symbolic execution to find high-cost test packet traces [34]. Compared to these work, P4wn contributes a new approach to adversarial testing that identifies edge cases by their probabilities.

2. The Position of Our Work

The most related to our work is the symbex tools for P4 programs: a) some rely on KLEE [15], a general-purpose symbex engine, and b) others rely on customized symbex

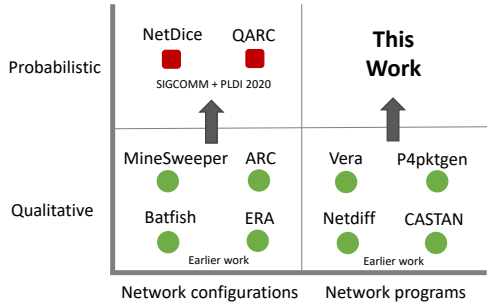

Figure 1: The position of our work in network analysis.

engines [33, 39, 40]. However, neither has considered *stateful* program analysis, and neither can analyze *probabilistic* network behaviors under random traffic distributions. In the design section of our paper, we explain why significant innovation is required to enable these new capabilities. In the evaluation section, we will a) reconfirm that customized symbex engines (e.g., Vera [39]) cannot capture stateful behaviors; [1] and b) show that a naïve use of KLEE scales poorly.

Figure 1 positions our work against a broader set of existing work [41, 38, 14, 6, 17, 39, 33, 12, 34, 13] along two dimensions: a) whether a technique analyzes network configurations or network programs, and b) whether it captures the probabilistic nature of network behaviors. We note that this figure does not comprehensively show all related work: P4 program analysis tools that do not rely on symbex [32, 37], and new probabilistic network programming languages that are designed from the ground up [36, 35, 16], are not shown.

3. Novelty

The design of P4wn involves three main challenges: a) computing probabilities, b) handling stateful analysis, and c) analyzing approximate data structures. We leverage a novel integration of model counting, header space analysis, interactive queries, loop analysis, and abstract interpretation of program behaviors, to address these challenges. A subset of these techniques follow.

In order to *compute the probability* of a program behavior, P4wn draws inspiration from two threads of work: header space analysis (HSA) [27], and model counting [10]. P4wn first relies on traditional symbex to collect header constraints for program execution paths. It then analyzes the resulting header space enclosed by these constraints, using model counting to compute the volume of this multi-dimensional polytope and its ratio to the entire header space. If a concrete network trace is available, P4wn can also issue probability queries to the trace to obtain a representative distribution.

The second key technique, *telescoping*, is designed to handle stateful behaviors that take a very long packet sequence to exercise. Naïvely applying symbolic execution to discover such a trace would lead to state explosion. Our key insight is that stateful network processing tends to have repeatable patterns (e.g., periodic sampling). P4wn probes a complex program using a short packet sequence to detect periodicity, and quickly generalizes to a longer sequence to exercise the target behaviors.

The third technique, *greybox analysis*, abstracts away the internal operations of approximate data structures, such as hash tables, Bloom filters, and sketches, which are very common to data plane systems. Leveraging the insight that these data structures have well-established statistical properties, P4wn creates very compact representations to enable symbex to scale independent of the data structure size.

[1] More concretely, Vera [39, 40] assumes that program state is always embedded in packet headers, so the program itself is always stateless. When given a stateful program, Vera sets all state variables to empty and performs a stateless analysis.

4. Main Artifacts

We have implemented P4wn in 6500 lines of code in C++ as pluggable modules in KLEE [9], which will be released to the community in open source. Our experimental setup and scripts will also be made available in online repositories.

5. Key Results and Contributions

Our key results and contributions are:

- The first program profiler for *stateful* data plane programs, and it can characterize their *probabilistic* behaviors for a particular traffic distribution.
- A set of *new techniques* customized for data plane programs that enables scalable symbolic execution.
- A *new approach to adversarial testing* that directly identifies edge cases using their probabilities.

6. Why ASPLOS

We believe that ASPLOS is the ideal venue for our work, because of its multidisciplinary nature:

1. Programmable data planes are an emerging hardware *architecture* for network devices, and they are programmed using domain-specific programming languages like P4. This sets the context of our work.

2. Customizing *program analysis* techniques for network programs has gained increased attention recently. Our work significantly improves the state of the art.

3. Adversarial testing of complex programs is closely related to *security* research. Our work represents a new approach to this goal.

7. Citation for Most Influential Paper Award

"P4wn was the first stateful and probabilistic profiler for then-recent programmable data planes. Its draws inspiration from a wide range of program analysis techniques, and customizes them to complex data plane programs. The resulting tool enables a comprehensive profiling of data plane programs that prior tools cannot support."

References

[1] Barefoot Tofino. https://www.barefootnetworks.com/technology/#tofino.

[2] Broadcom Trident 4. https://www.broadcom.com/blog/trident4-and-jericho2-offer-programmability-at-scale.

[3] Cisco Silicon One. https://www.cisco.com/c/en/us/solutions/service-provider/innovation/silicon-one.html.

[4] Intel FlexPipe. https://www.intel.com/content/www/us/en/products/network-io/ethernet/switches.html.

[5] Radu Banabic, George Candea, and Rachid Guerraoui. Automated vulnerability discovery in distributed systems. In *Proc. HotDep*, 2011.

[6] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *Proc. SIGCOMM*, 2017.

[7] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR*, 44(3), 2014.

[8] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. Using Frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *Proc. IEEE Security and Privacy*, 2014.

[9] Cristian Cadar, Daniel Dunbar, and Dawson R Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proc. USENIX OSDI*, 2008.

[10] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *Proc. CP*, 2013.

[11] Dragos Dumitrescu, Radu Stoenescu, Lorina Negreanu, and Costin Raiciu. bf4: towards bug-free P4 programs. In *Proc. SIGCOMM*, 2020.

[12] Dragos Dumitrescu, Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Dataplane equivalence and its applications. In *Proc. USENIX NSDI*, 2019.

[13] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. Efficient network reachability analysis using a succinct control plane representation. In *Proc. OSDI*, 2016.

[14] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. A general approach to network configuration analysis. In *Proc. NSDI*, 2015.

[15] Lucas Freire, Miguel Neves, Lucas Leal, Kirill Levchenko, Alberto Schaeffer-Filho, and Marinho Barcellos. Uncovering bugs in P4 programs with assertion-based verification. In *Proceedings of the Symposium on SDN Research*, page 4. ACM, 2018.

[16] Timon Gehr, Sasa Misailovic, Petar Tsankov, Laurent Vanbever, Pascal Wiesmann, and Martin Vechev. Bayonet: Probabilistic inference for networks. In *Proc. PLDI*, 2018.

[17] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. Fast control plane analysis using an abstract representation. In *Proc. SIGCOMM*, 2016.

[18] Tomer Gilad, Nathan H. Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. Robustifying network protocols with adversarial examples. In *Proc. HotNets*, 2019.

[19] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. Blink: Fast connectivity recovery entirely in the data plane. In *Proc. USENIX NSDI*, 2019.

[20] Md. Endadul Hoque, Hyojeong Lee, Rahul Potharaju, Charles E. Killian, and Cristina Nita-Rotaru. Adversarial testing of wireless routing implementations. In *Proc. WiSec*, 2013.

[21] Kuo-Feng Hsu, Ryan Beckett, Ang Chen, Jennifer Rexford, Praveen Tammana, and David Walker. Contra: A programmable system for performance-aware routing. In *Proc. NSDI*, 2020.

[22] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. LTEinspector: A systematic approach for adversarial testing of 4G LTE. In *Proc. NDSS*, 2018.

[23] Samuel Jero, Xiangyu Bu, Hamed Okhravi, Cristina Nita-Rotaru, Richard Skowyra, and Sonia Fahmy. BEADS: Automated attack discovery in OpenFlow-based SDN systems. In *Proc. RAID*, 2017.

[24] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. NetCache: Balancing key-value stores with fast in-network caching. In *Proc. SOSP*, 2017.

[25] Qiao Kang, Lei Xue, Adam Morrison, Yuxin Tang, Ang Chen, and Xiapu Luo. Programmable in-network security for context-aware BYOD policies. In *Proc. USENIX Security*, 2020.

[26] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. HULA: Scalable load balancing using programmable data planes. In *Proc. SOSR*, 2016.

[27] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *Proc. SIGCOMM*, 2012.

[28] Charles Killian, Karthik Nagara, Salman Pervez, Ryan Braud, James W. Anderson, and Ranjit Jhala. Finding latent performance bugs in systems implementations. In *Proc. FSE*, 2010.

[29] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[30] Hyojeong Lee, Jeff Seibert, Endadul Hoque, Charles Killian, and Cristina Nita-Rotaru. Turret: A platform for automated attack finding in unmodified distributed system implementations. In *Proc. ICDCS*, 2014.

[31] Hyojeong Lee, Jeff Seibert, Charles Killian, and Cristina Nita-Rotaru. Gatling: Automatic attack discovery in large-scale distributed systems. In *Proc. NDSS*, 2012.

[32] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Cascaval, Nick McKeown, and Nate Foster. p4v: Practical verification for programmable data planes. In *Proc. ACM SIGCOMM*. ACM, 2018.

[33] Andres Nötzli, Jehandad Khan, Andy Fingerhut, Clark Barrett, and Peter Athanas. P4pktgen: Automated test case generation for P4 programs. In *Proc. ACM SOSR*, 2018.

[34] Luis Pedrosa, Rishabh Iyer, Arseniy Zaostrovnykh, Jonas Fietz, and Katerina Argyraki. Automated synthesis of adversarial workloads for network functions. In *Proc. ACM SIGCOMM*, pages 372–385. ACM, 2018.

[35] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets scott: Semantic foundations for probabilistic networks. In *Proc. POPL*, 2017.

[36] Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. Scalable verification of probabilistic networks. In *Proc. PLDI*, 2019.

[37] Hardik Soni, Myriana Rifai, Praveen Kumar, Ryan Doenges, and Nate Foster. Composing dataplane programs with μp4. In *Proc. SIGCOMM*, 2020.

[38] Samuel Steffen, Timon Gehr, Petar Tsankov, Laurent Vanbever, and Martin Vechev. Probabilistic verification of network configurations. In *Proc. SIGCOMM*, 2020.

[39] Radu Stoenescu, Dragos Dumitrescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Debugging P4 programs with Vera. In *Proc. ACM SIGCOMM*, 2018.

[40] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Symnet: scalable symbolic execution for modern networks. In *Proc. SIGCOMM*, 2016.

[41] Kausik Subramanian, Anubhavnidhi Abhashkumar, Loris D'Antoni, and Aditya Akella. Detecting network load violations for distributed control planes. In *Proc. PLDI*, 2020.

[42] Max von Hippel, Cole Vick, Stavros Tripakis, and Cristina Nita-Rotaru. Automated attacker synthesis for distributed protocols. In *Proc. SafeComp*, 2020.

[43] Jiarong Xing, Qiao Kang, and Ang Chen. Netwarden: Mitigating network covert channels while preserving performance. In *Proc. USENIX Security*, 2020.