Rule-Level Data Plane Monitoring With Monocle*

Peter Perešíni
EPFL
Lausanne, Switzerland
peter.peresini@epfl.ch

Maciej Kuźniar
EPFL
Lausanne, Switzerland
maciej.kuzniar@epfl.ch

Dejan Kostić
KTH Royal Institute of Technology
Stockholm, Sweden
dmk@kth.se

ABSTRACT

We present Monocle, a system that systematically monitors the network data plane, and verifies that it corresponds to the view that the SDN controller builds and tries to enforce in the switches. Our evaluation shows that Monocle is capable of fine-grained per-rule monitoring for the majority of rules. In addition, it can help controllers to cope with switches that exhibit transient inconsistencies between their control plane and data plane states.

CCS Concepts

•**Networks** → **Network reliability; Network monitoring;**

1. INTRODUCTION

Ensuring network reliability in today's networks is paramount — customers require tight service-level guarantees such as 99.99% uptime, guaranteed bandwidth, bounded latency, *etc.*. This places network operators in a difficult situation where they need to respond to problems almost immediately. To do so, the network operators constantly monitor their networks for anomalies so as to be able to pinpoint their root causes.

One of the main reasons for faulty networks is the mismatch between the high-level network policy devised by the network operators and the actual data plane configuration in switch hardware which ultimately moves the packets around. Translation between the policy and configuration takes several steps and the desynchronization can happen at any point, starting from parsing the high level network policy down to programming ASIC hardware by the switch firmware.

Ensuring that the last step of this process, *i.e.*, checking if the controller view of the network corresponds to the data plane forwarding, is becoming increasingly important with the growing number of SDN switch vendors and switch models. In particular, failures at this level may range from transient inconsistencies (*e.g.*, switch reporting a rule was updated sooner than it happens in data plane [4,6]), through systematic problems (switches incorrectly implementing the specification, *e.g.*, ignoring priority field in OpenFlow [4]), to hardware failures (*e.g.*, bit-flips, line card not responding, *etc.*) and switch software bugs [8]), neither of which can be reliably detected in the control plane only.

Unfortunately, today, the choice of data plane monitoring tools is rather limited – operators use end-to-end tools (*e.g.*, ping, traceroute, *etc.*) or periodically collect switch forwarding statistics. We argue that these methods are insufficient – while ping/traceroute can detect broken end-to-end connections, it cannot localize the problem if the ICMP traffic is handled differently from TCP flows (*e.g.*, blacklisting based on TCP ports, *etc.*). Similarly, switch statistics do not tell exactly where the problem lies, only that there is something unusual. Finally, while recent research produced a number of new tools (HSA/Netplumber, VeriFlow, ATPG [1–3, 8]), they either work only in the control plane (the first three), or, in case of ATPG, too slow to identify problems as they happen in highly-dynamic SDN networks.

2. PROBLEM STATEMENT

Our goal is to provide fine-grained, per-rule monitoring of the network data plane state, and verify that this state corresponds to its control plane view. In particular, we want to detect all aforementioned error scenarios and precisely localize the problem to a particular switch flow table and problematic rule. As such, our ultimate requirement is to periodically exercise all rules that we expect to be in the data plane. Moreover, the solution should be online (*i.e.*, quickly adapt to the always-changing global forwarding state) and non-invasive (*i.e.*, monitoring should not require big changes to the network hardware, a controller or configuration).

Monocle is our approach to solving the presented problem. To meet our goal of verifying correspondence between control and data planes, our system needs to know the control-plane view of the state. We therefore implement Monocle as a proxy between SDN controller and its switches. This allows Monocle to intercept all rule modifications issued to any switch and maintain the expected global forwarding state installed in the network as well as the (expected) contents of flow tables in each switch. After determining the expected state of a switch, Monocle can compute packet headers that exercise rules on this switch. Finally, Monocle leverages its position as a proxy to inject these generated packets to the network and observe how they are treated (*i.e.*, where they get forwarded and how they are modified). Figure 1 summa-

*Work funded by the ERC project 259110

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2790012

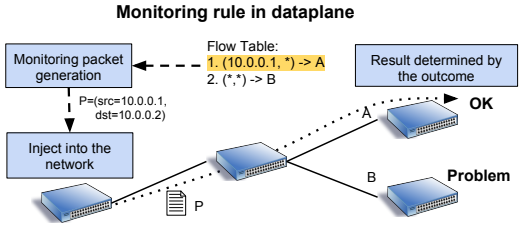

Figure 1: Overview of how data-plane monitoring works.

rizes a high level image of Monocle's operation. To detect if a rule is installed and behaves as expected, the system instructs the upstream switch to send a specially crafted monitoring packet toward the monitored switch. The downstream switch has a special catching rule that redirects the monitoring packet back to Monocle. By positioning our system as a proxy between the controller and the switches we do not require any hardware or software changes to either of them. The only modification is a small number of statically installed catching rules at each switch.

Although seemingly simple, the bulk of Monocle's complexity lies in generating the monitoring packets for arbitrary rules in switch flow tables in an online manner. Monocle leverages its knowledge of the flow tables to formulate a set of constraints that the monitoring probe has to satisfy. Without going into details, the constraints take into account: (*i*) structure of rule overlaps, including rule priorities, and (*ii*) what actions are associated with each rule (including header rewriting, packet dropping or forwarding to multiple ports).

The important part of Monocle design is the choice of converting constraints relevant for a given monitored rule into a form understood by off-the-shelf satisfiability (SMT/SAT) solvers and generating monitoring packets for each rule independently. This allows us to quickly solve the monitoring packet constraints for a single rule (less than 5 ms), in contrast to ATPG which uses header space analysis to compute monitoring packets for all rules in a single run (taking minutes to hours). The speed with which Monocle can generate monitoring packets enables new use-cases, namely monitoring rule updates as they happen in data plane as well monitoring networks undergoing frequent updates.

3. Monocle

Finally, Monocle applies a packet generation library to create real packets based on the solver's output. To collect monitoring outcomes, Monocle requires all downstream switches to recognize the monitoring packets. Since we aim to monitor all switches separately, each of them is being monitored and simultaneously collecting monitoring outcomes of all its neighbors. To prevent catching rules from interfering with monitoring, the catching rules have to differ between switches. To reduce the number of these additional rules per switch, we formulate the catching-rule selection as a well-known vertex coloring problem. As a result, our solution requires less than 10 additional rules per switch in a network containing over 10000 switches.

4. EVALUATION

There are two important questions that we need to confirm that a system like Monocle is feasible. First, can it find probes that monitor a significant fraction of rules in the network, and second, can it generate them at a rate high enough to keep up with the network reconfiguration speed? Our results show that both answers are positive. Using two real data sets: Stanford backbone router "yoza" configuration [2] and ACL rules from a large-scale campus network [7], we show that Monocle finds monitoring packets for 89% and 97% of all rules respectively and needs on average 1.44 and 4.13 ms to generate a single probe. Moreover, we verify that sending probes at a rate of 1000/s has no negative impact on rule modification rate of hardware switches.

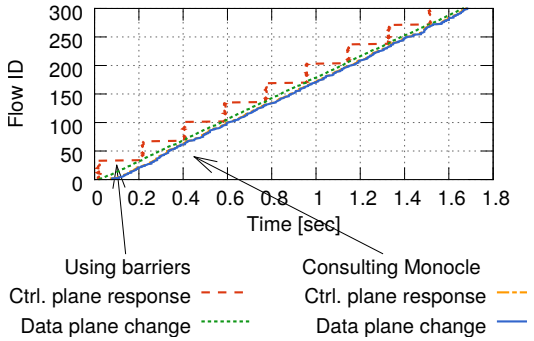

Figure 2: Finally, to demonstrate usefulness of Monocle and its online nature, we showcase how it can cooperate with the controller to provide reliable network updates. In particular, while network controllers may try to perform consistent updates of flows [5], some existing hardware switches allow a transient period during which there is inconsistency between what is reported by the control plane, and what happens in the data plane This effectively ruins the consistency guarantees as the controller might modify an upstream switch prematurely and thus send the traffic to a not-yet-ready rule (causing a temporary black hole in our experiment). By monitoring rule updates, Monocle helps the controller to reliably identify when the rules are ready in the data plane and avoids transient inconsistencies.

5. REFERENCES

[1] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real Time Network Policy Checking using Header Space Analysis. In *NSDI*, 2013.

[2] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *NSDI*, 2012.

[3] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *NSDI*, 2013.

[4] M. Kuźniar, P. Perešíni, and D. Kostić. What You Need to Know About SDN Flow Tables. In *PAM*, 2015.

[5] R. Mahajan and R. Wattenhofer. On Consistent Updates in Software Defined Networks. In *HotNets*, 2013.

[6] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. OFLOPS: An Open Framework for OpenFlow Switch Evaluation. In *PAM*, 2012.

[7] D. E. Taylor. Survey and Taxonomy of Packet Classification Techniques. *ACM Comput. Surv.*, 37(3):238–275, Sept. 2005.

[8] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown. Automatic Test Packet Generation. In *CoNEXT*, 2012.